ЭФФЕКТИВНАЯ РАБОТА В СИСТЕМЕ LINUX
Работа с текстами

1998 год

e-mail: Andrei.Chevel@pnpi.spb.ru
Дата формирования текста
29 ноября 1999 г.

ЭФФЕКТИВНАЯ РАБОТА В СИСТЕМЕ LINUX
Работа с текстами

Андрей Е. Шевель

Начато 5 сентября 1998

Оглавление

Список иллюстраций

Список таблиц

Замечания по использованию шрифтов в данной книге

В данной книге мы будем стараться придерживаться следующих соглашений об использовании различных шрифтов.

- Имена серверов, ftp и http адреса будут печататься наклонным шрифтом, например, *http://www.pnpi.spb.ru*. Используется псевдофонт `\HSname{}` (Host name).
- Имена переменных окружения будут печататься другим шрифтом, например, $EDITOR. Таким же шрифтом будут обозначаться названия протоколов, например, протокол FTP. Используется псевдофонт `\VSname{}` (Variable name).
- Имена переменных, на месте которых следует использовать подходящие значения мы будем обозначать наклонным стилизованным шрифтом *variable*. Используется псевдофонт `\PSname{}` (Parameter name).
- Имена файлов, конкретных программ, а также сами исходные тексты программ и скриптов будут печататься машинописным шрифтом, например, `cvs` или `modules`. Используется псевдофонт `\FSname{}` (Constant or character name).
- Названия программных систем, имена команд и параметров будем печатать жирным шрифтом, например, система **CVS** или команда **commit**. Используется псевдофонт `\SSname{}` (System name).
- Ещё один фонт – это термины. Используется псевдофонт `\TSname{}` (Term name).

- Наконец, компьютерные сообщения или команды. Используется псевдофонт (как телетайп) `\CSname{}` (Computer line).

Фактически, из-за бедности выбора фонтов на моей установке, изображение части фонтов могут совпадать.

Глава 1

Введение

1.1 Об этой книге

Автор впервые услышал про Linux в 1992 году, когда он пытался разыскать подходящую бесплатную операционную платформу типа **UNIX** для установки на персональный компьютер. Позже, когда Виктором Балашовым из ОИЯИ (Дубна) на **Linux** была установлена прикладная библиотека программ CERN, **Linux** довольно быстро стал распространяться среди студентов и научных сотрудников, вовлечённых в эксперименты по физике высоких энергий.

Автор обратил внимание на скорость распространения **Linux**, когда принял участие в работе физической коллаборации HERMES в немецком ядерном исследовательском центре DESY (Гамбург). Так, если в 1995 году в книжном магазине в Гамбурге были 2-3 книги про **Linux**, то в 1997 году в том же магазине продавалось уже около 10 книг про **Linux**.

Естественно, что все книги о **Linux**, которые попались мне в Гамбурге были на немецком и английском. На русском таких книг немного, хотя **Linux** - одна из популярных операционных систем.

Нужна ли книга о **Linux** на русском языке? Этот вопрос возникает у практиков, которые знают, что значительная часть программного обеспечения, особенно в небольших компаниях, используется совершенно без лицензий. Зачем изучать **Linux**, когда можно установив **MS Windows** пользоваться всеми пакетами, которые разработаны для **MS Windows**?

Действительно, часть программных пакетов, изготовленных для **MS Windows**, являются принятыми стандартами в ряде организаций: **MS Word**, **AutoCAD**, другие сложные приложения, составляющие существо вашего бизнеса. Если вы ориентируетесь лишь на использование таких программных пакетов, то вам, возможно, **Linux** и не слишком нужен. Тогда вам надо приготовиться платить за очередные версии программного обеспечения, или быть готовым к судебным тяжбам за незаконное использование того или иного программного продукта. А такие тяжбы последуют, если ваш бизнес будет успешным. Ведь деловой успех привлекает не только тех, кто хотел бы вам помогать, но и злопыхателей. Если же ваша организация не является коммерческой или строго зарегулированной, как военные или правительственные конторы, то вам прямая дорога в **Linux**.

А зачем вообще нужна какая-то книга, если Интернет полон всякой информации про все, что вам угодно, в том числе и про **Linux**? Автор должен сказать, что книга и информация в Интернет не заменяют друг друга, а лишь отлично друг друга дополняют. А как известно, недостаток одного компонента невозможно компенсировать избытком другого.

В практическом плане **Linux** имеет целый ряд преимуществ перед коммерческими операционными системами.

Во-первых, **Linux** поставляется свободно, значит используется массой лиц, которые не имеют средств на покупку коммерческих операционных систем. Обычно такие люди весьма изобретательны и часто хотят продемонстрировать свою изобретательность всем желающим, что приводит к потоку свободного программного обеспечения (или за символическую плату) на платформе **Linux**.

Во-вторых, если у вас возникли проблемы в системе, вы можете воспользоваться советами большого количества экспертов, которые уже используют **Linux**. Например, воспользовавшись news группами по **Linux**, например, `comp.os.linux.{announce,hardware,misc,networking,setup}` вы сможете как задать вопрос и в течение 24/48 часов (или быстрее) получить ответ, так и найти сразу ответ на ваш вопрос.

В-третьих, **Linux** весьма стабильно работающая многозадачная многопользовательская операционная система, которая успешно работает не только на платформе **Intel**, но и на многих других, если не всех известных платформах.

В-четвертых, все больше компаний-производителей прикладного программного обеспечения сообщают, что их продукты могут быть использованы в **Linux**.

Все это вместе взятое позволяет полагать, что **Linux** становится заметной альтернативой для ноутбуков, настольных компьютеров, серверов средней производительности и становится одной из важных операционных платформ наравне с AIX, IRIX, HPUX, NT, Solaris.

А надо ли вообще глубоко изучать устройство **Linux**, чтобы им пользоваться? Совсем не обязательно всем становиться авто механиками, но иметь общие представления об устройстве автомобиля и его возможностях очень полезно, чтобы успешно водить машину в разных условиях или иметь возможность сравнить различные марки автомобилей.

С использованием операционной платформы **Linux** можно выполнять любую работу, однако здесь мы обратимся в основном к работе с текстами: описаниями, исходными текстами программ на любом языке, письмами, романами, руководствами, рефератами, наконец, просто заметками для себя. Нетрудно догадаться, что работа с каким-то текстом всегда имеет место в любой деятельности связанной с компьютером.

Хотелось бы обратить внимание отдельно, что как правило тексты создаются группой лиц, которые могут быть рассредоточены в географическом смысле. Даже в тех случаях, когда вы создаёте текст в одиночестве, вы пользуетесь какими-то текстовыми материалами из Интернет, Интранет или в локальной базе данных на вашем компьютере. Иначе, создание новых текстов есть творческий процесс группы лиц по преобразованию имеющихся текстов. Большая часть средств, рассматриваемая автором, является свободно распространяемыми продуктами, которые были разработаны под эгидой проекта **GNU** (*h*ttp://www.gnu.org). Итак, перед вами книга о том как анализировать и преобразовывать текстовые файлы в системе **Linux**.

Настоящая книга адресована широкому кругу читателей: как новичку, так и человеку с опытом.

Глава 2

Краткая история Linux

2.1 Основные факты

История **Linux** изложена во множестве книг, а также в Интернет; здесь мы ограничимся лишь основными событиями. **Linux** есть один из вариантов или диалектов популярной операционной системы **UNIX**, которая была разработана в начале 70-ых. Популярность **UNIX** легко объяснить, если иметь в виду свойство РАЗВЕРТКИ **UNIX**, т.е. возможность установки **UNIX** на машине любой архитектуры. Модульность и структурированность системы позволяла, переделав относительно небольшое количество составляющих программных компонентов, запустить систему на совершенно новой машине за относительно небольшое время. Это свойство оказалось решающим в 80-ые при выборе операционных платформ для процессоров типа RISC: почти на всех процессорах общего назначения были установлены различные варианты **UNIX**. . Так, компания **IBM** ввела свой **UNIX** под именем **AIX**, **SGI - IRIX**, **CRAY - UNICOS**, **HP - HPUX**, **SUN - Solaris**, **SCO SCO Unix** и так далее.

Кстати об имени **UNIX** (произносится "юникс"). Название **UNIX** никак не переводится, оно появилось в противовес названию проекта грандиозной операционной системы того времени **MULTIX** (конец 60-х). Как правило, это название пишется латинскими буквами и воспринимать его следует как иероглиф.

Ядро операционной системы **Linux** для процессора **Intel 80386** было разработано финским студентом по имени Linus Torvalds (Линус Торвальдс), который продолжает развивать ядро с помощью многих десятков добровольных помощников со всего мира. Нетрудно догадаться, что остальная часть системы: утилиты, языки программирования, сервисные подсистемы были взяты из проекта **GNU**. Таким образом, всё это вместе составило новую свободно распространяемую операционную систему.

Нетрудно видеть, что название **Linux** (читается "линукс"[1]) образовано по аналогии с **UNIX**, поэтому оно также воспринимается как иероглиф.

2.2 Кому принадлежат права на Linux

Таким образом, права на ядро системы принадлежит Линусу Торвальдсу, который объявил, что его ядро может распространяться на условиях **GNU** GENERAL PUBLIC LICENSE (**GNU GPL** - основная общественная лицензия GNU). В соответствии с этой лицензией ядро может свободно копироваться, использоваться в любых целях (модификация, продажа, прочее), но вы не имеете права накладывать ограничения на распространение продуктов, которые вы образовали на основе ядра, раз вы получили его в соответствии с **GNU GPL**. Иными словами, если кто-то купил у вас модифицированное ядро, он также имеет право продавать этот продукт, не спрашивая вашего разрешения. Итак, любой человек может свободно копировать любой доступный вариант **Linux** и поступать с ним так, как считает нужным.

Полностью детали копирования можно прочесть в файле `COPYING`, который является частью операционной системы **Linux**.

2.3 Различные варианты Linux

В действительности, под именем **Linux** имеется в виду большое семейство операционных систем с близкими вариантами ядра (KERNEL), но с различным составом прикладного программного обеспечения. Мы перечислим несколько наиболее популярных компаний, которые поставляют **Linux**.

RedHat (URL: *http://www.redhat.com/*) - компания, которая поставляет один из распространенных вариантов Linux. Linux на основе варианта **RedHat** используется в ряде крупнейших ядернофизических исследовательских центрах мира: CERN (Швейцария), FNAL (США) и других. В составе **RedHat** можно получить библиотеку **Motif**, клиентскую часть **CDE**, другие программные пакеты, а также варианты **Linux** для различных аппаратных платформ: Intel, Alpha, Sparc. **RedHat** внедрила пакет **R**PM для распространения программного обеспечения, который сейчас используется во многих вариантах **Linux**.

Caldera (URL: *http://www.caldera.com/*) поставляет **OpenLinux**, где уже имеются такие системы как **Wabi** (позволяет запускать **MS Windows**

---

[1] Некоторые произносят иначе. Если вы хотите услышать непременно от Линуса Торвальдса как это произносится, вы можете списать звуковой файл ENGLISH.AU или SWEDISH.AU с сервера `ftp.funet.fi` из директории (каталога) `/pub/Linux/PEOPLE/Linus/SillySounds`.

приложения в **Linux**), **StarOffice** (настольный офис), NetWare для **Linux** и прочие полезные приложения.

S.u.S.e. (URL: *http://www.suse.de/e/linux.html*) - немецкая компания, которая поставляет **Linux** вместе с описаниями на нескольких языках (немецкий, английский, французский). В состав пакета из 6 CD дисков входит довольно много разнообразной информации о Linux и программ из Интернет. Поставляется СУБД **ADABAS**, комплект настольный офис Applixware, а также **Wabi**. **Slackware** - популярный вариант **Linux**, который поддерживается энтузиастом *Patrick Volkerding*. Она легко устанавливается и поддерживается. Скопировать **Slackware** можно во многих местах.

Debian (URL: *http://www.debian.org/*) - позволяет скопировать **Debian Linux**.

UrbanSoft (URL: *http://www.usoft.spb.ru/*) - российская компания в С.Петербурге (Россия), которая поставляет **Linux** и другое открытое программное обеспечение.

Как показывает опыт выбор варианта **Linux** определяется, как правило, вашими рабочими контактами. Так, если все ваши коллеги используют, например, SuSe, то использовать другой вариант нет смысла без серьезных на то причин. Если вы устанавливаете **Linux** дома и у вас нет особенных предпочтений, то можно ставить любой. Драматических отличий вы не увидите.

Имеется довольно других компаний и часто появляются новые компании, которые начинают поставлять тот или иной вариант **Linux**. Читатель сможет сам в этом убедиться, если попробует разыскать список поставщиков **Linux** в Интернет.

Глава 3

Информация о Linux

3.1 Linux и родственная информация в пространстве WWW

3.1.1 Замечание о www

В пространстве www имеется море информации обо всем, в том числе о **Linux**. Однако, www пространство весьма переменчиво: то что вчера было доступно, сегодня сменило адрес или вообще исчезло из поля зрения. Поэтому автору традиционной книги приходится указывать несколько источников (URL адресов), поскольку часть из них может вообще сменить ориентацию, а часть файлов могла быть удалена с серверов: из-за смены системного менеджера или просто по недосмотру. Чем больше времени прошло с момента написания книги, тем менее вероятно, что вы что-то найдете по указанным адресам в www пространстве. Вероятно - это жизнь. Тем не менее попробуйте, говорят, информацию невозможно уничтожить.

3.1.2 Несколько популярных сайтов о Linux

Операционная система Linux довольно богато представлена в мировой паутине www. Только перечисление даже известных адресов займет немало страниц. Здесь мы кратко охарактеризуем лишь несколько известных адресов.

Страница *http://hepwww.ph.qmw.ac.uk/HEPpc/*

- Linux Resources for High Energy Physics (Ресурсы Linux для физики высоких энергий) - один из наиболее привлекательных адресов для студентов и исследователей, работающих в области физики высоких энергий. Однако, любой "юниксист"найдет на этих страницах полезную информацию.

Страница *http://www.linux.org/*

- является одной из самых известных по всем вопросам программного обеспечения в Linux.

Страница *http://sunsite.unc.edu/mdw/linux.html*

- **Linux Documentation Project (LDP)** - проект **Документация для Linux**. На этой странице можно найти ссылки на всевозможные документы, описания и, вообще, любые тексты, имеющие отношения к Linux. Нетрудно догадаться, что подавляющая часть документов написана на английском или на техническом жаргоне, который похож на английский.

Полезная информация имеется также на www серверах компаний, которые продают Linux или программные продукты для **Linux**. Один из способов найти что-то конкретное о **Linux** - это использовать одну из поисковых систем в Интернет, например, *http://www.metacrawler.com/*.

В качестве ключей для поиска вводите "Linux network card", если вас интересуют проблемы и программное обеспечение сетевых адаптеров под Linux. В ответ вы получите одну или много страниц со списком адресов, где можно найти информацию о сетевых адаптерах под Linux. Воспользовавшись указанной поисковой машиной и введя упомянутые ключи для поиска, я получил 4–го сентября 1998 года список адресов на трех страницах.

Кроме www страниц, полезно пользоваться news группами по Linux, часть из них приведены в главе "Введение".

3.1.3 Описания компонентов Linux и родственная информация на русском языке

Страница *http://www.linux.org/books/basic.html*

На этой странице можно найти ссылки на страницы описаний (man pages) по Unix (Linux) на русском языке. Многие из переводов выполнены Виктором Вислобоковым (*victor_v@permonline.ru*). На сентябрь 1998 было переведено уже около 200 описаний команд.

Страница *http://xtalk.price.ru/linux/*

Страница содержит разнообразную информацию о Linux и указатели на таковую на русском.

Страница *http://www.nevod.ru/linux/doc/*

Содержит переводы разных пособий по программам в **UNIX**, таким как **sed** и **awk**. Кроме того, с этой страницы доступен перевод книги М. Уелш

"ИНСТАЛЛЯЦИЯ LINUX И ПЕРВЫЕ ШАГИ"(M. Welsh "Linux Installation and Getting Started").

Можно также взглянуть страницу "Библиотека Мошкова" *http://kulichki-koi.rambler.ru/moshkow/*. На странице *http://koi.aha.ru/ agb/* информация и ссылки на неё, в том числе и на русском. Среди прочего указаны серверы IRC (канал #ruslinux) *irc.mo.net*, *irc.emory.edu*, *irc-w.primenet.com*, *irc-e.primenet.com*, *irc.sprynet.com*). Имеется в виду, что в канале #ruslinux, на указанных IRC серверах можно поговорить с русскоязычными коллегами о Linux. На этой же странице имеется список русских ресурсов Linux.

По русским ресурсам Linux Вы можете также попробовать список URL, приведенный ниже.

```
http://cclib.nsu.ru/projects/gnudocs/
http://hh.demos.su/~slackl/
http://knot.pu.ru/faq/xfaq.html
http://m66.nevod.perm.su/service/linux/doc/
http://nexus.odessa.ua/linux/
http://t37.nevod.perm.su
http://win.www.netclub.ru/Russian/linux.html
http://www.aha.ru/~agb/
http://www.dkd.ot.lt/hompag/linux/default.htm
http://www.linux.inf.ru
http://www.nevod.perm.su/linux/doc/
http://www.uco.ru/~garris/
http://www.usoft.spb.ru
http://xtalk.price.ru/linux/
```

3.2 ftp серверы с информацией о Linux

Кроме других источников полезно попробовать ftp серверы. Поиск по ftp серверам можно произвести с помощью поисковой машины *http://ftpsearch.ntnu.no/*. Если задать в качестве ключа поиска слово "Linux", то получите список ftp серверов размером во много сотен строк. К счастью, есть возможность ограничивать количество найденных серверов. В качестве примера я также попробовал найти ftp серверы и ограничить их список тридцатью строками. Результат внизу.

```
?Case insensitive substring search? for ?linux?

----------------------------------------------------------------------

ftp.doc.ic.ac.uk     /Mirrors/ftp.tex.ac.uk/tex-archive/tools/zip/info-zip/UNIX/LINUX
ftp.doc.ic.ac.uk     /Mirrors/ftp.cdrom.com/pub/infozip/OLD/UNIX/LINUX
```

```
ftp.doc.ic.ac.uk    /Mirrors/ftp.cdrom.com/pub/infozip/UNIX/LINUX
ftp.doc.ic.ac.uk    /Mirrors/ftp.ncsa.uiuc.edu/HDF/contrib/LINUX
ftp.kreonet.re.kr   /.3/CTAN/tools/zip/info-zip/UNIX/LINUX
ftp.kreonet.re.kr   /.3/tools/compress/infozip/OLD/UNIX/LINUX
ftp.kreonet.re.kr   /.3/tools/compress/infozip/UNIX/LINUX
ftp.kreonet.re.kr   /.3/tools/compress/infozip-nonUS/UNIX/LINUX
ftp.kreonet.re.kr   /.3/tools/compress/infozip-nonUS/OLD/UNIX/LINUX
ftp.in-chemnitz.de  /afs/tex/tools/zip/info-zip/UNIX/LINUX
ftp.ncsa.uiuc.edu   /HDF/contrib/LINUX
ftp.cis.nctu.edu.tw /Vendors/TekrAm/SCSI/LINUX
ftp.cs.columbia.edu /.archives/networking/uunet/archival/zip/UNIX/LINUX
ftp.univ-evry.fr    /.01/archiving/infozip/UNIX/LINUX
ftp.univ-evry.fr    /.01/archiving/infozip/OLD/UNIX/LINUX
ftp.chg.ru          /.3/TeX/CTAN/tools/zip/info-zip/UNIX/LINUX
ftp.iij.ad.jp       /TeX/CTAN/pub/tex-archive/tools/zip/info-zip/UNIX/LINUX
ftp.jaist.ac.jp     /.cached2/lang/CTAN/tools/zip/info-zip/UNIX/LINUX
ftp.cdrom.com       /.1/tex/ctan/tools/zip/info-zip/UNIX/LINUX
ftp.univ-evry.fr    /.04/text/TeX/CTAN/tools/zip/info-zip/UNIX/LINUX
ftp.cv.nrao.edu     /aips/15APR98/LINUX
ftp.cv.nrao.edu     /aips/BIN/LINUX
ftp.cdrom.com       /.12/infozip/UNIX/LINUX
ftp.cdrom.com       /.12/infozip/OLD/UNIX/LINUX
ftp.doc.ic.ac.uk    /Mirrors/quest.jpl.nasa.gov/pub/UNIX/LINUX
ftp.doc.ic.ac.uk    /Mirrors/quest.jpl.nasa.gov/pub/LINUX
ftp.cv.nrao.edu     /aips/DA00/LINUX
ftp.tekram.com      /SCSI/LINUX
ftp.stalker.ru      /Pub/HardSoft/SoftForHard/Net/SureCom/EP325/LINUX
ftp.vislist.com     /SHAREWARE/ENVIRONMENTS/LINUX

Next
------------------------------------------------------------------------
30 reported hits
0.014 seconds prospero
0.017 seconds HTTP
0 partial writes.
DONE
------------------------------------------------------------------------
Hardware by FAST ASA, ITEA and IDI. Network by UNINETT.
NTNU - Norwegian University of Science and Technology
This server runs FreeBSD and is located in Trondheim, Norway
FAST FTP Search, Copyright (C) 1998 FAST ASA.
```

3.3 Прикладные программные пакеты для Linux

Естественно, одним из первых полезных мест, где можно посмотреть необходимые пакеты - это www-сервер проекта **GNU** *http://www/gnu.org*. Как известно, **GNU's not Unix**, т.е. **GNU** - это не **Unix**, такой шутливой рекурсией определяется аббревиатура **GNU**. Этот проект посвящён реализации свободно распространяемого программного обеспечения. Более подробные сведения о проекте и его продуктах можно найти на сервере **GNU** и на многих других серверах в мире.

Одним из полезных мест в WWW-пространстве является Scientific Applications on Linux (SAL) – коллекция научных прикладных программ *http://SAL.KachinaTech.COM/index.shtml*. Термин научные программы не является эквивалентом неприменимых в обычной жизни. Просто это один из больших списков полезных программных продуктов, работающих под **Linux**, которые подходят и для научных исследований. Эти программы имеются на многих серверах в Интернет. Вы можете посмотреть наиболее близкий к вам сервер из нижеследующего перечисления:

`Australia`	`http://sal.rising.com.au/`
`Austria`	`http://nswt.tuwien.ac.at/scicomp/sal/`
`Colombia`	`http://linuxSITE.univalle.edu.co/sal/`
`Finland`	`http://sal.jyu.fi/`
`France`	`http://www-sor.inria.fr/mirrors/sal/`
`Germany`	`http://ftp.llp.fu-berlin.de/lsoft/`
`Italy`	`http://chpc06.ch.unito.it/linux/`
`Japan`	`http://ec.tmit.ac.jp/koyama/linux/SAL/`
`New Zealand`	`http://nix.tmk.auckland.ac.nz/SAL`
`Poland`	`http://www.tuniv.szczecin.pl/linux/doc/other/SAL`
`Portugal`	`http://www.idite-minho.pt/SAL/`
`Russia (Moscow)`	`http://www.sai.msu.su/sal/`
`Russia (Novosibirsk)`	`http://www.siblug.org/SAL/`
`South Africa`	`http://web.ee.up.ac.za/sal/`
`South Korea`	`http://bioinfo.bioneer.com/sal`
`Spain`	`http://ceu.fi.udc.es/SAL/`
`Turkey`	`http://sal.raksnet.com.tr`
`United Kingdom`	`http://www.ch.qub.ac.uk/SAL/`
`USA`	`http://SAL.KachinaTech.COM`

Другие интересные прикладные пакеты появляются на странице *http://www.redhat.com/linux-info/linux-app-list/linapps.html*, которую поддерживает RedHat.

3.4 На каких аппаратных платформах доступен Linux

Ответ очень прост - на всех широко известных и еще на нескольких, которые известны лишь узкому кругу специалистов. Тем не менее, автор хотел бы указать несколько конкретных источников по Linux для ряда платформ, исключая Intel.

Информацию о Linux на платформе PowerPC можно найти на *http://www.linuxppc.org/*. Информация о Linux/m68k на платформах Моtorola 68020, 68030, 68040, 68060 и близких к ним может быть почерпнута на *http://www.linux-m68k.org/*. FAQ на эту же тему: *http://www.linux-m68k.org/faq/faq.html*.

Довольно богато представлена информация по Linux на процессорах Alpha (Digital) в news группе *comp.os.linux.alpha*. Информация о переносе Linux на платформу 64-bit DEC Alpha/AXP имеется на *http://www.azstarnet.com/ axplinux/*. По поводу платформы MIPS можно взглянуть *ftp://ftp.fnet.fr/linux-mips*, а также *ftp://ftp.linux.sgi.com/pub/mips-linux*. Информация о Linux на платформе SunSparc имеется во многих местах, например, можно посмотреть *http://www.redhat.com*. Имеются архивы по Sun Linux *http://www.geog.ubc.ca/sparclinux.html*, а также *ftp://vger.rutgers.edu/pub/linux/Sparc*.

Для обсуждения конкретных деталей переноса Linux можно пользоваться списком рассылки по ядру Linux, который поддерживается на *vger.rutgers.edu*. Довольно полный список платформ, на которые перенесён Linux, а также список различных вариантов Linux приведён на странице *http://www.linuxhq.com/dist-index.html*, а также на странице *http://www.ctv.es/USERS/xose/linux/linux_ports.html*

3.5 Книги по Linux

Что касается литературы по Linux, то мы найдем много книг по этой теме и самого разного назначения. Например, если взглянуть на страницу *http://www.linux.org/books/basic.html*, то можно найти список, состоящий из кратких аннотаций. Пример такой аннотации ниже.

- Using Linux
    - Автор: Bill Ball
    - ISBN: 0789716232
    - Издатель: Que Education and Training
    - Дата публикации: июль 1998
    - Объем: 650 страниц

    Аннотация: Книга "Using Linux"написана, чтобы помочь совсем новичку или имеющему небольшой опыт пользователю Linux решать задачи управления системой более эффективно. Большинство книг в этой категории нацелены на установку и конфигурирование Linux, а

> не на решение ежедневных проблем, возникающих у менеджера. Эта книга содержит сведения обо всех сторонах использования системы, упомянутых выше. Кроме этого читатель найдет полезные индексы и другую справочную информацию по Linux варианта Red Hat.

Далее приведён список из 30 или 40 названий книг разных издательств, направлений и объёма. Объем колеблется от 160 до 1600 страниц. Цена может варьироваться от $19 до $70. Таким образом, есть из чего выбирать. Но, если читатель не намерен скупать книги десятками названий, а нуждается лишь в одном или двух толковых руководствах, то не стоит останавливаться на книгах менее 500 страниц. Все-таки книга должна быть не только кратким руководством по конкретной системе, но и полезным справочником по родственным вопросам. Полезно обратить внимание на следующие книги [1, 2].

Кроме книг имеется ряд газет и журналов посвященных разным аспектам применения Linux. Объем печатной продукции по Linux на английском языке явно превышает возможности обычного человека прочесть все это даже за целую жизнь. Этот факт лишний раз подчеркивает популярность этой операционной системы.

3.5.1 Книги по Linux на русском языке

Если взглянуть на сервер `http://books.ru`, то нетрудно отыскать несколько книг по **Linux**. Я посмотрел 10 сентября 1998 года и обнаружил 4 (четыре) названия связанных с **Linux** (два CD-шных издания и две традиционные книги, сопровождаемые CD). Все книги переводные. Обращает на себя внимание двухтомник Петерсена "Руководство по Linux"[3].

Поскольку **Linux** есть один из вариантов **UNIX**, то возникает естественный вопрос, а можно ли использовать книги по **UNIX** на русском языке для работы в **Linux**? Во многих случаях можно. Если вы начали пользоваться системой недавно, то на уровне пользовательского интерфейса есть много общего и вы не допустите серьезных ошибок. Что касается пакетов **GNU**, то они просто одинаковы во всех вариантах операционных систем и определяются лишь версией самого пакета.

Среди сравнительно новых книг по **UNIX** можно отметить [4], которая даёт полезные представления не только о командах, но и об архитектуре системы **UNIX**.

3.6 Другие источники информации по Linux

Существует отличный комплект англоязычных документов по **Linux**, собранный под заголовками HOWTO (как ...), например, "как сконфигурировать принтер". На большинство практических вопросов можно найти ответы в файлах HOWTO. Эти файлы имеются на многих серверах, например, *http://sunsite.unc.edu/pub/Linux/docs/HOWTO/*. Ниже приведен список HOWTO, который получен с упомянутого сервера 13 сентября 1998 года.

```
Index of /pub/Linux/docs/HOWTO
 [   ]  3Dfx-HOWTO             10-May-98 00:13    73k
 [   ]  AX25-HOWTO             03-Nov-97 22:00   137k
 [   ]  Access-HOWTO           30-Mar-97 15:53    63k
 [   ]  Alpha-HOWTO            02-Sep-97 22:18    22k
 [   ]  Assembly-HOWTO         26-Apr-98 01:20    54k
 [   ]  Benchmarking-HOWTO     26-Apr-98 01:20    43k
 [   ]  BootPrompt-HOWTO       03-Feb-98 00:57    81k
 [   ]  Bootdisk-HOWTO         05-Jul-98 10:34    84k
 [   ]  Busmouse-HOWTO         27-Aug-98 19:57    22k
 [   ]  CD-Writing-HOWTO       29-Jan-98 22:36    34k
 [   ]  CDROM-HOWTO            26-Apr-98 01:21    71k
 [   ]  Chinese-HOWTO          30-Jun-98 23:28   112k
 [   ]  Commercial-HOWTO       23-Aug-98 02:32   227k
 [   ]  Config-HOWTO           10-May-98 00:13    39k
 [   ]  Consultants-HOWTO      27-Aug-98 20:14   339k
 [   ]  Cyrillic-HOWTO         26-Apr-98 01:22    74k
 [   ]  DNS-HOWTO              28-Aug-98 00:25    69k
 [   ]  DOS-Win-to-Linux-HOWTO 10-May-98 00:13    55k
 [   ]  DOS-to-Linux-HOWTO     26-Apr-98 01:22    52k
 [   ]  DOSEMU-HOWTO           26-Apr-98 01:23    57k
 [   ]  Danish-HOWTO           05-Jul-98 12:28    31k
 [   ]  Distribution-HOWTO     20-Aug-98 03:02    35k
 [   ]  ELF-HOWTO              26-Apr-98 01:23    53k
 [   ]  Emacspeak-HOWTO        26-Apr-98 01:24    65k
 [   ]  Esperanto-HOWTO        24-Jul-98 00:40    23k
 [   ]  Ethernet-HOWTO         26-Jul-98 19:45   224k
 [   ]  Finnish-HOWTO          14-Mar-96 12:15    30k
 [   ]  Firewall-HOWTO         13-Nov-96 23:25    53k
 [   ]  French-HOWTO           23-Feb-98 23:16    70k
 [   ]  Ftape-HOWTO            29-Mar-97 19:46    52k
 [   ]  GCC-HOWTO              26-Apr-98 01:25    60k
 [   ]  German-HOWTO           30-Mar-97 14:44    55k
 [   ]  Glibc2-HOWTO           18-Apr-98 22:49    35k
 [   ]  HAM-HOWTO              31-Mar-97 23:26    76k
 [   ]  HOWTO-INDEX            31-Aug-98 07:19    37k
 [   ]  Hardware-HOWTO         20-Aug-98 00:31   100k
 [   ]  Hebrew-HOWTO           14-Mar-96 12:17    21k
 [   ]  IPCHAINS-HOWTO         31-Aug-98 02:42    72k
```

```
[   ]  IPX-HOWTO               18-Apr-98 23:25    78k
[   ]  IR-HOWTO                28-Aug-98 00:34    29k
[   ]  ISP-Hookup-HOWTO        30-Aug-98 12:04    32k
[   ]  Installation-HOWTO      30-Aug-98 19:20    57k
[   ]  Intranet-Server-HOWTO   26-Apr-98 01:25    40k
[   ]  Italian-HOWTO           26-Apr-98 02:22    92k
[   ]  Java-CGI-HOWTO          16-Dec-96 00:10    24k
[   ]  Kernel-HOWTO            26-May-97 11:48    55k
[   ]  KickStart-HOWTO         31-Aug-98 04:20    27k
[CMP]  Linux-HOWTOs.tar.gz     31-Aug-98 09:11   2.2M
[   ]  MGR-HOWTO               02-Jun-96 19:24    26k
[   ]  MILO-HOWTO              15-Nov-97 18:03    44k
[   ]  Mail-HOWTO              28-Aug-98 19:58    70k
[   ]  Multi-Disk-HOWTO        26-Apr-98 01:27   171k
[   ]  Multicast-HOWTO         10-May-98 00:07    72k
[   ]  NET-3-HOWTO             28-Aug-98 00:42   201k
[   ]  NFS-HOWTO               03-Nov-97 22:09    38k
[   ]  NIS-HOWTO               05-Jul-98 12:01    42k
[   ]  Optical-Disk-HOWTO      26-Apr-98 01:28    16k
[   ]  Oracle-HOWTO            27-Aug-98 21:06    36k
[   ]  PCI-HOWTO               30-Mar-97 14:27    92k
[   ]  PCMCIA-HOWTO            27-Aug-98 23:55   115k
[   ]  PPP-HOWTO               16-Apr-97 22:21   155k
[   ]  Pilot-HOWTO             17-Aug-97 23:33    15k
[   ]  Polish-HOWTO            28-Aug-98 22:49    12k
[   ]  Portuguese-HOWTO        28-Aug-98 22:57    67k
[   ]  PostgreSQL-HOWTO        30-Jul-98 17:05   236k
[   ]  Printing-HOWTO          05-Jul-98 11:47    50k
[   ]  Printing-Usage-HOWTO    26-Apr-98 01:28    21k
[   ]  Quake-HOWTO             28-Aug-98 23:23    85k
[   ]  RPM-HOWTO               06-Jul-97 21:27    32k
[   ]  Reading-List-HOWTO      09-May-98 18:30    19k
[   ]  Root-RAID-HOWTO         09-May-98 18:30    82k
[   ]  SCSI-Programming-HOWTO  18-May-96 20:26   128k
[   ]  SMB-HOWTO               18-Aug-96 20:48    32k
[   ]  SRM-HOWTO               02-Sep-97 22:18    23k
[   ]  Security-HOWTO          03-May-98 13:06   109k
[   ]  Serial-HOWTO            11-Jul-98 13:36    71k
[   ]  Shadow-Password-HOWTO   05-Jun-96 19:25    67k
[   ]  Slovenian-HOWTO         06-Nov-96 20:29    30k
[   ]  Sound-HOWTO             26-Apr-98 01:29    78k
[   ]  Sound-Playing-HOWTO     20-Aug-98 03:06    30k
[   ]  Spanish-HOWTO           25-Aug-96 20:39    50k
[   ]  TeTeX-HOWTO             19-Apr-98 00:37    85k
[   ]  Text-Terminal-HOWTO     05-Jul-98 12:15   168k
[   ]  Thai-HOWTO              20-Aug-98 01:31    42k
[   ]  Tips-HOWTO              27-Aug-98 21:54    36k
[   ]  UMSDOS-HOWTO            14-Mar-96 12:22    22k
[   ]  UPS-HOWTO               19-Nov-97 23:13   138k
[   ]  UUCP-HOWTO              28-Aug-98 22:33    26k
[   ]  User-Group-HOWTO        27-Apr-98 22:45    56k
```

```
[   ] VAR-HOWTO              28-Aug-98 23:30    34k
[   ] VME-HOWTO              27-Aug-98 22:23    28k
[   ] VMS-to-Linux-HOWTO     10-May-98 00:14    51k
[   ] Virtual-Services-HOWTO 30-Aug-98 15:12    71k
[   ] WWW-HOWTO              19-Nov-97 23:08    57k
[   ] WWW-mSQL-HOWTO         15-Nov-97 18:03    42k
[   ] XFree86-HOWTO          29-Aug-98 08:16    26k
```

Кроме того, имеется несколько групп новостей (news group), посвященных различным аспектам **Linux**.

comp.os.linux.announce является контролируемой группой. Вам необходимо читать эту группу, если вы планируете использовать **Linux**. Она содержит информацию о состоянии программного обеспечения для Linux (изменения, появление новых программных пакетов, совещания пользователей Linux и прочее). Объявления в эту группу должны направляться по адресу: *linux-announce@news.ornl.gov.* Имеются и другие полезные группы, перечисленные ниже:

```
comp.os.linux.setup
comp.os.linux.hardware
comp.os.linux.networking
comp.os.linux.x
comp.os.linux.development.apps
comp.os.linux.development.system
comp.os.linux.advocacy
comp.os.linux.misc
```

3.7 Как найти информацию о системе или описания в установленной системе Linux?

Найти какое-то описание в уже работающей системе **Linux** можно разными способами. Рассмотрим наиболее важные.

3.7.1 Команда *uname*

Команда **uname** поможет вам узнать какой вариант Linux установлен на вашей машине:

```
uname -a
```

В ответ система напечатает что-то в духе: `Linux pcfarm.pnpi.spb.ru 2.0.33 #18 Thu Jun 4 11:54:03 MSD 1998 i686 unknown` Это означает:

операционная система - **Linux**, имя машины в сети - *pcfarm.pnpi.spb.ru*, версия ядра операционной системы - 2.0.33, версия обновления ядра и дата создания ядра системы - #18 Thu Jun 4 11:54:03 MSD 1998, тип машины - `i686`, тип процессора - `unknown`.

3.7.2 Как прошла загрузка системы

Информацию о сообщениях во время загрузки ядра системы может быть получена из протокола, который выдаёт программа **dmesg**.

3.7.3 Программа *procinfo*

С помощью программы `procinfo` можно получить массу полезной информации (как статической, так и динамической) о работающей системе. Что именно выдаёт `procinfo` можно узнать с помощью команды:

```
procinfo -h
```

3.7.4 Команда *man*

Чтобы получить краткое описание какой-то команды или термина в системе можно использовать команду **man**. Если вас интересует информация о термине МОДЕМ, то удобно использовать команду

```
man -k modem
```

Здесь k"обозначает, что далее следует ключевое слово, о котором нужна информация. В ответ Вы получите список команд и функций, которые как-то связаны с заданным ключевым словом. Далее вы сможете снова воспользоваться командой **man**, чтобы узнать, что означают элементы полученного вами списка. Так, в RedHat в ответ, на вышеприведенную строку Вы получите:

```
efax (1)             - send/receive faxes using Class 1 or 2 fax modems
mgetty (8)           - smart modem getty
rx, rb, rz (1)       - XMODEM, YMODEM, ZMODEM (Batch) file receive
sendfax (8)          - send group 3 fax files (G3 files) with a class 2 faxmodem
statserial (1)       - display serial port modem status lines
sx, sb, sz (1)       - XMODEM, YMODEM, ZMODEM file send
zplay (1)            - modem utility to record and play voice files
XF86VidModeQueryExtension, XF86VidModeQueryVersion,
XF86VidModeGetModeLine, XF86VidModeGetAllModeLines, XF86VidModeD
eleteModeLine, XF86VidModeModModeLine, XF86VidModeValidateModeLine,
XF86VidModeSwitchMode, XF86VidModeSwitchToMode,
XF86VidModeLockModeSwitch, XF86VidModeGetMonitor, XF86VidModeGetViewPort,
XF86VidModeSetViewPort (3) - XFree86-VidMode extension interface functions
```

Вариант программы **man**, которая использует средства **X-Window**, называется **xman**.

3.7.5 Команда *info*

Очень удобна команда **info**, которая реализована на базе средств редактора **emacs**. Однако, в моем варианте **Linux** команда не обнаружила подходящей информации:

```
File: dir       Node: Top       This is the top of the INFO tree

  This (the Directory node) gives a menu of major topics.
  Typing "q" exits, "?" lists all Info commands, "d" returns here,
  "h" gives a primer for first-timers,
  "mEmacs<Return>" visits the Emacs topic, etc.

  In Emacs, you can click mouse button 2 on a menu item or cross reference
  to select it.

* Menu:

Texinfo documentation system
* Texinfo: (texinfo).           The GNU documentation format.
* install-info: (texinfo)Invoking install-info. Updating info/dir entries.
* texi2dvi: (texinfo)Format with texi2dvi.      Printing Texinfo documentation.
* texindex: (texinfo)Format with tex/texindex.  Sorting Texinfo index files.
* makeinfo: (texinfo)makeinfo Preferred.        Translate Texinfo source.

Miscellaneous
* Autoconf: (autoconf).         Create source code configuration scripts.
* Finding Files: (find).        Listing and operating on files
                                that match certain criteria.
* GIT: (git).         GNU Interactive Tools
* Gdb: (gdb).                     The GNU debugger.
* Gdb-Internals: (gdbint).      The GNU debugger's internals.
* Gettext Utilities: (gettext).         GNU gettext utilities.
* Libg++: (libg++).             The g++ class library.
* Mtools: (mtools).        Mtools: utilities to access DOS disks in Unix.
* Shar utilities: (sharutils).          GNU shar utilities.
* Text utilities: (textutils).          GNU text utilities.
* cat: (textutils)cat invocation.               Concatenate and write files.
* cksum: (textutils)cksum invocation.           Print POSIX CRC checksum.
* comm: (textutils)comm invocation.             Compare sorted files by line.
* csplit: (textutils)csplit invocation.         Split by context.
* cut: (textutils)cut invocation.               Print selected parts of lines.
* expand: (textutils)expand invocation.         Convert tabs to spaces.
* fmt: (textutils)fmt invocation.               Reformat paragraph text.
* fold: (textutils)fold invocation.             Wrap long input lines.
* gzip: (gzip).                 The GNU compression utility.
```

```
* head: (textutils)head invocation.          Output the first part of files.
-----Info: (dir)Top, 162 lines
--Top-----------------------------------------------------------------------
There is no menu item "modem" in this node.
```

3.7.6 Команда *apropos*

Команда **apropos** также помогает найти информацию среди man-файлов, имеющихся на вашей машине. Если вы попробуете выдать команду

`apropos modem` ,

то заметите, что она сработала примерно также как и команда `man -k`

```
efax (1)            - send/receive faxes using Class 1 or 2 fax modems
mgetty (8)          - smart modem getty
rx, rb, rz (1)      - XMODEM, YMODEM, ZMODEM (Batch) file receive
sendfax (8)         - send group 3 fax files (G3 files) with a class 2 faxmodem
statserial (1)      - display serial port modem status lines
sx, sb, sz (1)      - XMODEM, YMODEM, ZMODEM file send
zplay (1)           - modem utility to record and play voice files
XF86VidModeQueryExtension, XF86VidModeQueryVersion,
XF86VidModeGetModeLine, XF86VidModeGetAllModeLines, XF86VidModeD
eleteModeLine, XF86VidModeModModeLine, XF86VidModeValidateModeLine,
XF86VidModeSwitchMode, XF86VidModeSwitchToMode,
XF86VidModeLockModeSwitch, XF86VidModeGetMonitor, XF86VidModeGetViewPort,
XF86VidModeSetViewPort (3) - XFree86-VidMo
de extension interface functions
```

3.7.7 Команда *helptool*

По команде `helptool` появляется графическое окно, в котором вы сможете задать интересующий вас термин. Команда просматривает все файлы документов (вы можете сконфигурировать какие документы следует просматривать при поиске). По завершении поиска вам будет выдан список файлов, где встречается данный термин. Если кликнуть мышкой на элементе списка, то появится дополнительное окно, в котором будет отображаться выбранный вами файл. При этом файл будет отображаться в том формате, в котом он хранится на вашей машине: страницы `info`, страницы `man` и др.

3.7.8 Команда *locate*

Наконец, может оказаться полезной команда **locate**. Например, вы собираетесь найти файлы, которые содержать слово `modem`

`locate modem`

вы получите

```
/usr/bin/modemtool
/usr/doc/util-linux-2.7/README.modems-with-agetty
/usr/farm/gcc-2.7.2.3/modemap.def
/usr/lib/rhs/control-panel/modemtool.init
/usr/lib/rhs/control-panel/modemtool.xpm
/usr/lib/rhs/control-panel/python/modem.py
/usr/lib/rhs/control-panel/python/modem.pyc
/usr/share/usernet/1.0.5/modem.xpm
```

Легко видеть, что в полученном списке имеется лишь один исполняемый файл: `/usr/bin/modemtool`. Можно попробовать **man modemtool** или просто вызвать данную программу с параметром `-help`. В данном случае это оказалась диалоговая конфигурационная программа для модема.

3.7.9 Команда *rpm*

rpm – это специальный пакет, который используется в большинстве вариантов **Linux** для установки, удаления, модернизации других программных пакетов на конкретной машине. **rpm** ведет свою базу данных установленных на машине пакетов, поэтому удобно опрашивать командой rpm о всех установленных на вашей машине пакетах. Система **rpm** довольно обширна по своим возможностям. Мы отметим лишь несколько её свойств, которые помогут понять какое программное обеспечение установлено на локальном сервере (рабочей станции).

Общий вид команды получения информации о пакетах:

`rpm -q` *query-options*

Имеется два подмножества параметров команды **rpm** для получения информации: выбор пакета(ов) и выбор типа информации, которая будет напечатана. Например, для того, чтобы получить весь список установленных программных пакетов на вашей машине, можно использовать:

```
rpm -q -a
```

В ответ будет напечатан длинный список программных пакетов имеющихся на машине.

Общий вид команды **rpm** для получения информации следующий:

rpm *query-options*

Информация о возможных значениях *query-options* приведена в таблицах 3.1 и 3.2.

Таким образом, если вас интересует установлен ли какой-то вариант пакета LaTeX на машине, можно выдать команду:

```
rpm -q -a | grep -i latex
```

На своей машине я получил:

`tetex-latex-0.4pl8-9`

Более подробное описание пакета можно получить командой:

`rpm -q -i tetex-latex-0.4pl8-9`

В ответ вы получите что-то вроде нижеследующего:

```
Name        : tetex-latex                 Distribution: Hurricane
Version     : 0.4pl8                            Vendor: Red Hat Software
Release     : 9                             Build Date: Wed Oct 22
23:36:05 1997
Install date: Fri Jan 16 17:01:13 1998   Build Host: porky.redhat.com
Group       : Applications/Publishing/TeX   Source RPM:
tetex-0.4pl8-9.src.rpm
Size        : 9911252
Packager    : Red Hat Software <bugs@redhat.com>
URL         : http://www.tug.org/teTeX/
Summary     : LaTeX macro package
Description :
LaTeX is a TeX macro package. The LaTeX macros encourage writers to
think about the content of their documents, rather than the form.  The
ideal, very difficult to realize, is to have no formatting commands
(like ``switch to italic'' or ``skip 2 picas'') in the document at
all; instead, everything is done by specific markup instructions:
``emphasize'', ``start a section''.
```

Таблица 3.1: Параметры выбора пакетов из базы **rpm**

Выбор пакетов	
Параметр	Описание
-**whatrequires** *service*	показать список пакетов, которые требуют *service*.
-**whatprovides** *virtual*	показать список пакетов, которые обеспечивают сервис *virtual*.
-f *file*	показать пакет, которому принадлежит файл с именем *file*.
-p *file*	Показать пакет, которому принадлежит файл с именем *file*, даже если пакет был удален.
-a	Показать все пакеты в базе данных **rpm**.

Таблица 3.2: ПАРАМЕТРЫ ВЫБОРА ВИДА ИНФОРМАЦИИ ИЗ БАЗЫ `rpm`

Выбор вида информации	
Параметр	Назначение
-i *пакет*	печатает информацию о пакете.
-**changelog** *пакет*	печатает информацию об изменениях в программном пакете.
-l *пакет*	печатает список файлов пакета.
-s *пакет*	печатает список файлов пакета вместе с информацией о статусе каждого файла.
-d *пакет*	печатает список файлов документов указанного пакета.
-c *пакет*	печатает список конфигурационных файлов пакета.
-**dump**	печатает всю проверяемую информацию для каждого файла; должно использоваться с параметрами `-c, -d, -s, -l`.
-**provides** *пакет*	печатает свойства, которые обеспечивает данный пакет.
-**requires** *пакет*	печатает список файлов и каталогов, которые необходимы для установки данного пакета.
-R *пакет*	то же самое, что `-requires`.
-**scripts** *пакет*	печатает скрипты, которые используются в данном пакете для установки или для удаления пакета.

3.7.10 Примеры получения информации от системы rpm

Узнать, какому пакету принадлежит файл `/usr/share/texmf/tex/latex/misc/umlaute.sty`? Ниже приведена команда и ответ на неё.

```
$ rpm -q -f /usr/share/texmf/tex/latex/misc/umlaute.sty
tetex-latex-0.9-17
```

Т.е. файл входит в пакет с именем **tetex-latex-0.9-17**.

Какие пакеты требуют наличия сервиса TETEX? Ниже приведена команда запроса и ответ на неё системы **rpm**.

```
$ rpm -q --whatrequires tetex
OB_Java-1.0-1
PSCyr-0.3-3
tetex-afm-0.9-17
tetex-dvilj-0.9-17
tetex-dvips-0.9-17
tetex-latex-0.9-17
tetex-russian-cyrplain-2.1-4
tetex-russian-cyrsam-2.1-4
tetex-t2-2.1-4
```

Какие каталоги нужны для установки ситемы **a2ps**? Ответ – ниже.

```
$ rpm -q --requires a2ps
/sbin/install-info
/bin/sh
/sbin/ldconfig
ld-linux.so.2
libc.so.6
libm.so.6
/bin/sh
/usr/bin/perl
libm.so.6(GLIBC_2.1)
libc.so.6(GLIBC_2.1)
libc.so.6(GLIBC_2.0)
```

3.8 На каком языке говорит Linux (локализация)

Локализация означает приспособление программы или операционной системы к кодировке и стилям печати времени, даты, денежных единиц принятых в данной стране. Стандарт описан, например, в документах **POSIX 1.c**. Здесь мы отметим важные особенности.

Многие системные программы (быть может большинство) в **Linux** для определения вида кодировки и языка сообщений используют установленные переменные окружения.

Таблица 3.3: ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ ДЛЯ ЛОКАЛИЗАЦИИ

Имя переменной	Описание

Таблица 3.3: Переменные окружения для локализации

LC_ALL	Если установлена, то эта переменная определяет всё сразу: язык сообщений, кодировку, вид даты и т.п.
LANGUAGE	Эта переменная используется в основном программами из проекта **GNU**.
LC_CTYPE	Эта переменная, если не установлена LC_ALL, определяет кодировку символов. В отсутствии LC_CTYPE и LC_ALL, для определения типа кодировки используется переменная LANG.
LC_COLLATE	Эта переменная используется в отсутствии LC_ALL для определения алгоритма сортировки. В отсутствии LC_COLLATE для этого используется переменная LANG.
LC_MONETARY	Эта переменная используется, если отсутствует LC_ALL, для определения вида денежной единицы. Если LC_ALL и LC_MONETARY не установлены, то используется переменная LANG.
LC_NUMERIC	Эта переменная окружения используется, если не установлена переменная LC_ALL, для определения национального формата печати чисел (например, с плавающей точкой или запятой). Если не установлены LC_ALL и LC_NUMERIC, то используется переменная окружения LANG.
LC_TIME	Данная переменная окружения используется, если не установлена LC_ALL для определения формата даты и времени. Если не установлены переменные LC_ALL и LC_TIME, то используется переменная LANG.
LANG	Эта переменная окружения используется для всего, что не установлено ранее переменными вида LC_*.

Чтобы увидеть какие переменные из вышеописанных и как установлены можно использовать программу `locale` из комплекта утилит языка **perl**

```
[shevel@pcfarm BOOK]$ locale
LANG=ru_SU
LC_CTYPE="ru_SU"
LC_NUMERIC="ru_SU"
LC_TIME="ru_SU"
```

```
LC_COLLATE="ru_SU"
LC_MONETARY="ru_SU"
LC_MESSAGES="ru_SU"
LC_ALL=ru_SU
```

Какие имеются файлы для локализации на вашей машине, можно посмотреть в каталоге `/usr/share/locale/`, а также посмотреть результат выполнения команды

`locale -a`.

Общий вид строк, которые присваиваются переменным окружения, должен быть примерно таким:

`язык_территория_кодировка`

Например, `ru_RU_koi8r`. Однако, как видно выше, на моей установке имеются другие файлы для кодировки `koi8r`, которые приведены в выводе программы `locale`. Следует заметить, что часто одни и те же кодировки называются чуть по-разному. Например, автору приходилось видеть следующие обозначения: `koi8`, `koi8r`, `koi8-r` или `KOI8-R`. Хотя последний вид обозначения встречается нечасто.

Полезные сведения о локализации можно почерпнуть в

`man perllocale`.

Глава 4

Файловая система Linux

Поскольку операционная система **Linux** есть вариант (диалект) системы **UNIX**, то общая организация файловой системы одна и та же. Есть и отличия, так **Linux** имеет псевдокаталог с именем `/proc`, который не является обычным каталогом в смысле операционной системы. При чтении отдельных файлов каталога на самом деле вызываются соответствующие программы ядра системы, которые и выдают запрошенную информацию.

По состоянию на 1999 год **Linux** является 32-битной операционной системой. Этот факт означает, что ни при каких условиях в **Linux** невозможно адресовать пространство памяти более, чем можно указать в 32-битном адресе. При этом максимальный размер одного файла в системе не может превышать 2 GB. Предполагается, что 64-битная версия **Linux** появится в 2000-ом году.

Здесь мы кратко рассмотрим организацию файловой системы и назначение отдельных файлов.

4.1 Иерархия системных файлов

Linux имеет целый ряд каталогов специального назначения, наиболее важные из них обсуждаются в данном разделе.

Таблица 4.1: ВАЖНЫЕ ФАЙЛЫ СИСТЕМЫ LINUX

Имя файла	**Назначение**
`/`	Это корневой каталог. Здесь начинаются все каталоги системы включая каталоги, созданные пользователями.

Продолжение таблицы на следующей странице

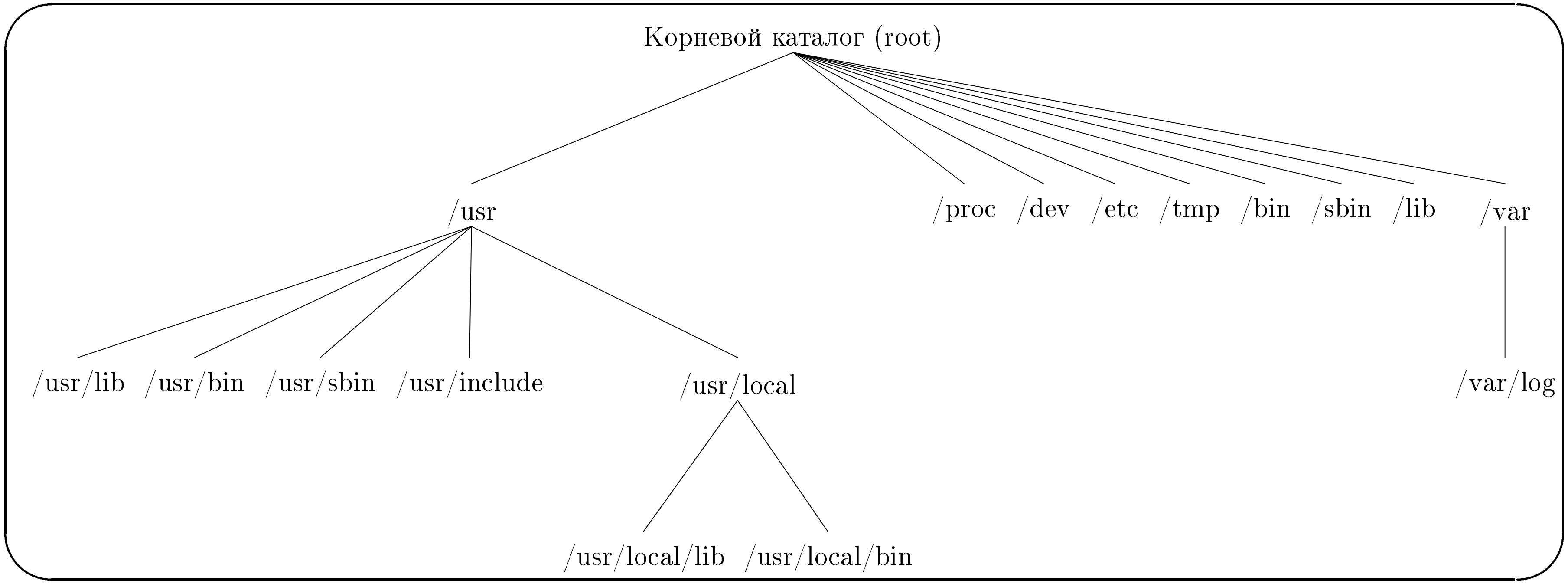

Рис. 4.1: Структура каталогов Linux

Файлы Linux (продолжение таблицы 4.1)	
`/bin`	Этот каталог содержит в основном готовые к исполнению программы, большинство из которых необходимы в однопользовательском системном режиме во время старта системы (или отладки).
`/boot`	Содержит постоянные файлы для загрузчика системы. Файлы из этого каталога нужны только во время загрузки системы. План расположения файлов и различные конфигурационные файлы находятся в `/sbin` и `/etc`.
`/dev`	Каталог специальных файлов или файлов устройств. Дополнительная информация содержится на странице `mknod(1)`.
`/dos`	Если на компьютере запускается поочередно Linux и MS DOS, то этот каталог обычно используется, чтобы монтировать файловую систему MS DOS.
`/etc`	Каталог содержит конфигурационные файлы, которые являются локальными для этой машины. Крупные прикладные пакеты, например, X11, могут иметь свой каталог для конфигурационных файлов ниже по файловой иерархии. Конфигурационные файлы общие для группы машин помещаются обычно в каталог `/usr/etc`. Тем не менее, часть конфигурационных файлов может находится и в `/etc` и в `/usr/etc`.

Продолжение таблицы на следующей странице

Файлы Linux (продолжение таблицы 4.1)	
`/etc/skel`	Когда создаётся новый account, то файлы из этого каталога копируются во вновь созданный каталог пользователя.
`/etc/X11`	Каталог для конфигурационных файлов подсистемы X11.
`/home`	Обычно в этом каталоге находятся каталоги пользователей.
`/lib`	Этот каталог должен содержать разделяемые библиотеки, которые необходимы для загрузки операционной системы и для выполнения команд в корневой файловой системе.
`/mnt`	В этом каталоге обычно содержатся точки монтирования для временно смонтированных файловых систем.
`/proc`	Это точка монтирования для файловой системы `proc`, которая обеспечивает информацию о выполняющихся процессах, ядре, оборудовании вычислительной установки и т.д. Это псевдо-файловая система, подробности о которой можно найти в `proc(5)`.
`/sbin`	Подобно каталогу `/bin` содержит в основном программы, необходимые для загрузки операционной системы.
`/tmp`	Каталог для временных файлов. В любой момент пользовательские файлы из этого каталога можно удалить без большого ущерба для остальных пользователей. Однако, не стоит удалять файлы из этого каталога, если вам не стало ясно, что конкретный файл или группа файлов мешает продолжению продуктивной работе на машине.

Продолжение таблицы на следующей странице

Файлы Linux (продолжение таблицы 4.1)	
`/usr`	Этот каталог обычно содержит библиотеки или данные предназначенные лишь для чтения. Каталог `/usr` на вашей машине может быть смонтирован на других **Linux** машинах посредством **NFS**.
`/usr/X11R6`	Файлы относящиеся к системе X-Window версии 11, ревизия 6.
`/usr/X11R6/bin`	Готовые к исполнению программы системы X-Window версии 11, ревизия 6.
`/usr/X11R6/lib`	Файлы и библиотеки связанные с системой X-Window.
`/usr/X11R6/lib/X11`	Каталог содержит различные файлы, необходимые для работы системы X-Window.
`/usr/X11R6/include/X11`	Содержит файлы типа **include** необходимые для компилирования программ, которые используют библиотеки системы X-Window.
`/usr/bin`	Готовые к исполнению программы, которыми часто вызывают обычные пользователи.
`/usr/bin/X11`	Обычное место для расположения готовых к исполнению программ из X-Window в **Linux**. Часто это символический линк к `/usr/X11R6/bin`.
`/usr/dict`	Этот каталог содержит файлы со словарным запасом для программ проверки корректности написания слов.
`/usr/etc`	Здесь содержатся конфигурационные файлы для группы машин или для всей организации. Однако, команды и программы должны смотреть в каталог `/etc`, в котором должны быть линки к файлам в каталоге `/usr/etc`.

Продолжение таблицы на следующей странице

Файлы Linux (продолжение таблицы 4.1)

`/usr/include`	Файлы типа **include** для программ на языке **C**.
`/usr/include/X11`	Файлы типа **include** необходимые для трансляции программ на **C**, которые используют систему X-Window. Обычно это символический линк к каталогу `/usr/X11R6/include/X11`.
`/usr/include/asm`	Файлы типа /SSnameinclude для ряда функций ассемблера.
`/usr/include/linux`	Этот каталог содержит файлы, которые могут меняться от версии к версии **Linux**. Часто это имя является символическим линком к каталогу `/usr/src/linux/include/linux`. Отсюда **Linux** получает информацию специфическую для системы. Но, например, в **Debian Linux** получение информации реализовано по иному.
`/usr/include/g++`	Каталог содержит include-файлы для использования в GNU C++.
`/usr/lib`	В данном каталоге содержится объектные библиотеки подпрограмм, динамические библиотеки, некоторые готовые к исполнению программы, которые не вызываются непосредственно. Сложные программные системы могут иметь свои подкаталоги.
`/usr/lib/X11`	Обычное место для помещения файлов связанных с **X-Window**, а также конфигурационных файлов самой системы **X-Window**. В **Linux** это обычно символический линк к каталогу `/usr/X11R6/lib/X11`.
`/usr/lib/gcc-lib`	Содержит готовые к исполнению программы и файлы типа **include** для компилятора **GNU C** (**gcc**).
`/usr/lib/groff`	Файлы для системы форматирования текстов **groff**.

Продолжение таблицы на следующей странице

Файлы Linux (продолжение таблицы 4.1)	
`/usr/lib/uucp`	Файлы для **UUCP**.
`/usr/lib/zoneinfo`	Файлы для определения временной зоны (часового пояса). Смотрите также страницы руководства `named-xfer (8)`, `tzfile (5)`, `tzselect (8)`, `zdump (8)`, `zic (8)`.
`/usr/local`	Обычно здесь помещают программы и подкаталоги, которые являются локальными (уникальными) для данной машины.
`/usr/local/bin`	Обычно здесь помещают готовые к исполнению программы, которые являются локальными (уникальными) для данной машины.
`/usr/local/doc`	Локальная документация обычно помещается здесь.
`/usr/local/etc`	Конфигурационные файлы для локально установленных программ.
`/usr/local/lib`	Библиотеки и файлы для локально установленных программ и систем.
`/usr/local/info`	Страницы описаний, которые просматриваются посредством программы **info**, для локально установленных программ.
`/usr/local/man`	Страницы описаний, которые просматриваются посредством программы **man**, для локально установленных программ.
`/usr/local/sbin`	Локальные программы системного администратора.
`/usr/local/src`	Локального значения исходные тексты программ, установленных на данной машине.
`/usr/man`	Страницы руководств. Полезно посмотреть страницы `man`.
`/usr/man/<locale>/man[1-9]`	Эти каталоги содержать страницы руководств в исходной форме. Системы, которые используют один язык и один кодовый набор могут не использовать подстроку `<locale>`.

Продолжение таблицы на следующей странице

Файлы Linux (продолжение таблицы 4.1)	
`/usr/sbin`	Этот каталог содержит готовые к исполнению программы для системного администрирования, которые не используются во время загрузки.
`/usr/src`	Исходные тексты различных частей **Linux**.
`/usr/src/linux`	Исходные тексты для ядра **Linux**.
`/usr/tmp`	Ещё одно место для хранения временных файлов. Это символический линк к каталогу `/var/tmp`. Не рекомендуется использовать.
`/var`	Этот каталог содержит файлы, которые могут сильно изменяться по размеру, например, протоколы (логи), временные файлы и т.д.
`/var/adm`	Этот каталог, если есть, должен быть символическим линком к каталогу `/var/log`.
`/var/backups`	Этот каталог используется, чтобы сохранить резервную копию важных системных файлов.
`/var/catman/cat[1-9]`	Этот каталог используется чтобы хранить уже сформированные страницы руководств в соответствии с номером главы.
`/var/lock`	Здесь содержатся управляющие файлы системы, которые используются для резервирования использования тех или иных ресурсов системы.
`/var/log`	Различные файлы протоколов (логи).
`/var/preserve`	Здесь редактор **vi** сохраняет сессии редактирования при ненормальном завершении выполнения. Таким образом, они могут быть восстановлены позже.
`/var/run`	Переменные файлы времени выполнения различных программ. Они содержат идентификаторы процессов (PIDs) и записывают текущую информацию (utmp). Файлы в этом каталоге обычно очищаются во время загрузки системы.
Продолжение таблицы на следующей странице	

Файлы Linux (продолжение таблицы 4.1)	
`/var/spool`	Файлы различных программ поставленные в очередь на обслуживание к разным системам: на печать, на передачу по электронной почте.
`/var/spool/at`	Файлы заданий, запущенных посредством команды **at**.
`/var/spool/cron`	Файлы системы **cron**.
`/var/spool/lpd`	Файлы ожидающие вывода на печать.
`/var/spool/mail`	Пользовательские почтовые ящики.
`/var/spool/news`	Файлы системы **news**.
`/var/spool/uucp`	Файлы системы **uucp**.
`/var/tmp`	Временные файлы.

4.2 Важные конфигурационные файлы Linux

4.2.1 Что это такое

Конфигурационные файлы в **Linux** – это специального назначения текстовые файлы с описанием параметров и свойств всей системы или отдельных компонентов. Ниже приведен список важных конфигурационных файлов, которые находятся в системном каталоге с именем `/etc`. Имеется несколько вариантов суффиксов имён файлов: `*.conf`, `*.config`, `*.cfg`, наконец имеется каталог с конфигурационными файлами системы `/etc/sysconfig`. Всё большее количество авторов свободно распространяемых продуктов рекомендуют помешать конфигурационные файлы для конкретных программ и систем в каталог `/etc`.

Для прикладных программ, например, **pine** или **a2ps**, в `/etc` помещаются, как правило, конфигурационные файлы общие для в всех пользователей. Однако, любой пользователь может установить свою персональную конфигурацию, которая описывается соответствующим файлом в каталоге $HOME.

Если вам потребовалось поэкспериментировать с содержанием конфигурационного файла, не делайте этого в каталоге `/etc`! Перед внесением любых изменений в любой конфигурационный файл в каталоге `/etc` убедитесь, что вы вполне точно понимаете, что будет происходить в системе или как будет себя вести прикладная программа.

Автор не задался целью подробно и детально перечислить все без исключения мыслимые конфигурационные файлы и их форматы. Такая задача представляется малопродуктивной из-за относительно частого изменения состава программ и систем, а также форматов самих файлов. Основная идея состоит в том, чтобы дать сведения лишь об основных конфигурационных файлах.

4.2.2 Инициализационные файлы

Это те же конфигурационные файлы, но они обычно прочитываются однажды за время выполнения программы – во время старта программы, которая их использует. Довольно часто они располагаются в домашнем каталоге пользователя $USER (каталог $HOME) и имеют вид *.name*`rc`, где *name* есть имя программы, для которой предназначен инициализационный файл. Этот файл прочитывается программой *name* лишь тогда, когда она вызывается пользователем с именем $USER.

В то же время имеется каталог `/etc`, в котором обычно находятся главные конфигурационные и инициализационные файлы. Эти файлы прочитываются программой вне зависимости от того, какой пользователь вызвал конкретную программу. Конфигурационные файлы из каталога `/etc` прочитываются раньше файлов из домашнего каталога пользователя. Иными словами, значения параметров установленные пользователем в свём каталоге оказываются более приоритетными по сравнению с каталогом `/etc`.

4.3 Конфигурационные файлы в каталоге /etc

4.3.1 adm.conf

Файл `/etc/amd.conf` используется программой автоматического монтирования файловых систем – **amd**. Программа **amd** является демоном, который автоматически монтирует требуемую файловую систему, если есть обращение к ней и размонтирует её, если никто смонтированную систему не использует. Детали можно посмотреть с помощью `man adm.conf`, `man amd`, `man amq`.

4.3.2 apcupsd.conf

Файл `/etc/apcupsd.conf` представляет собой конфигурационный файл для демона, который следит за состоянием вашего устройства бесперебойного питания – УБП (UPS).

4.3.3 dosemu.conf

Файл `/etc/dosemu.conf` представляет собой конфигурационный файл для эмулятора MS DOS **dosemu**. Подробности следует искать в документах сопровождающих пакет **dosemu**. Первичную информацию о пакете можно получить с помощью `rpm -q -i dosemu`.

4.3.4 gated.conf

Файл `/etc/gated.conf` представляет собой конфигурационные данные для работы демона **gated**, который предназначен для маршрутизации многих протоколов **RIP**, **BGP**, **EGP**, **HELLO**, **OSPF**. Подробности могут быть получены посредством `man gated`.

4.3.5 gpm-root.conf

Файл `/etc/gpm-root.conf` представляет собой конфигурационные данные для программы **gpm-root**, которая предназначена для управления *мышкой* в Linux. Более подробная информация может быть получена посредством `man gpm-root.conf`, `man gpm`, `info gpm`.

4.3.6 group

Файл `group` представляет собой системный конфигурационный файл, который содержит описания групп, к которым принадлежат пользователи. Каждая строка файла имеет следующий формат

```
group_name:passwd:GID:user_list
```

Более подробная информация может быть получена на страницах `group` и `groupadd`.

4.3.7 host.conf

Файл `/etc/host.conf` представляет собой конфигурационные данные для комплекта программ **resolver**, которые обеспечивают нахождение имени хоста, под которым он известен в Интернет. Среди прочего, в файле должен быть описан порядок нахождения (разрешения) имени. Подробности могут быть найдены на странице руководства `host.conf`.

4.3.8 inetd.conf

Файл `/etc/inetd.conf` представляет собой конфигурационные данные для программы **inetd**. Файл `/etc/inetd.conf` часто называют базой данных интернетовских серверов, которые могут использоваться на данной машине. все строки данного файла должны содержать следующие элементы или поля:

- имя сервиса, который описан в файле `/etc/services`;
- тип сокета;
- имя протокола;
- применимо только к дейтаграммным сокетам (wait/nowait), остальные должны иметь в этом поле **nowait**;
- user[.group];
- имя программы сервера;
- аргументы программы сервера.

Примеры:

```
login  stream  tcp    nowait  root    /usr/sbin/tcpd  in.rlogind
talk   dgram   udp    wait    root    /usr/sbin/tcpd  in.talkd
ftp    stream  tcp    nowait  root    /usr/sbin/tcpd  in.ftpd -l -a
telnet stream  tcp    nowait  root    /usr/sbin/tcpd  in.telnetd
auth   stream  tcp    nowait  nobody  /usr/sbin/in.identd in.identd
```

Подробности можно получить посредством `man inetd.conf`, `man inetd`.

4.3.9 issue

Файл `/etc/issue` представляет собой текстовый файл, который содержит сообщение или системную идентификацию вашей версии Linux, которая выводится на экран до приглашения программы **login**. Смотрите также страницу `issue`.

4.3.10 ld.so.conf

Файл `/etc/ld.so.conf` представляет собой конфигурационные данные для сборщика-загрузчика. Он содержит список каталогов, в которых следует искать библиотеки. Подробнее смотрите `man ld.so` и `man ldconfig`.

На основании содержания файла `/etc/ld.so.conf` строится другой файл `/etc/ld.so.cache`, который содержит упорядоченный список библиотек, найденных в каталогах, имена которых перечислены в файле `/etc/ld.so.conf`. Файл `/etc/ld.so.cache` можно посмотреть посредством команды `ldconfig -p`.

4.3.11 lilo.conf

Файл `/etc/lilo.conf` – конфигурационный файл для программы **lilo**, требуется во время загрузки системы. Подробная информация может быть получена посредством `man lilo.conf`, `man lilo`.

4.3.12 logrotate.conf

Файл `/etc/logrotate.conf` – конфигурационный файл для программы **logrotate**, который позволяет администрировать файлы сообщений, генерируемые различными программами. Программа **logrotate** позволяет автоматически начать файл сообщений заново, удалить, сжать файл сообщений, послать его по электронной почте. Такие действия можно делать ежедневно, еженедельно, ежемесячно или когда файл достигнет определённого размера. Обычно **logrotate** запускается как задание из демона **cron**.

Подробности могут быть найдены в `man logrotate`.

4.3.13 ltrace.conf

Файл `/etc/ltrace.conf` – конфигурационный файл для программы **ltrace**, которая является трассировщиком обращений к программам из динамической библиотеки. Подробнее смотрите страницы руководства `ltrace`.

4.3.14 man.config

Файл `/etc/man.config` – является конфигурационным файлом для программы **man**. Подробнее смотрите руководство `man`.

4.3.15 mtools.conf

Файл `/etc/mtools.conf` – является конфигурационным файлом для пакета программ **mtools**, который предназначен для обеспечения доступа к дискетам, записанным в формате MS DOS. Подробности могут быть почерпнуты в `man mtools`.

4.3.16 named.conf

Файл `/etc/named.conf` является конфигурационным файлом для программного сервера **named** (Internet domain name server – сервер имён машин вашей организации, под которыми они известны в Интернет). Подробности в `man named`.

4.3.17 nscd.conf

Файл `/etc/nscd.conf` является конфигурационным файлом для сервиса Name Service Cache - Сервис Кеширования Имён. Программа **nscd** кеширует имена, которые получены посредством **lookup**. Кеширование серьёзно сокращает время ожидания при повторных запросах в системах **NIS+** и DNS. Информация о программе имеется вместе с самой программой `rpm -q -i nscd`.

4.3.18 nsswitch.conf

Файл `/etc/nsswitch.conf` является конфигурационным файлом для целой группы библиотечных программ, которые связаны с получением имён хостов и использованию этих имён локально (обычно в пределах вашей организации). Подробнее можно посмотреть в `man nsswitch`.

4.3.19 ntp.conf

Файл `/etc/ntp.conf` является конфигурационным файлом для средств реализующих Network Time Protocol. Обмениваясь сообщениями в соответствии с данным протоколом хост устанавливают одинаковое время в компьютерной сети. Смотрите также `man ntp`.

4.3.20 nwserv.conf

Файл `/etc/nwserv.conf` является конфигурационным файлом для эмулятора Netware для Linux. Другие документы можно найти посредством `apropos netware`.

4.3.21 paper.config

Файл `/etc/paper.config` описывает форматы бумаги на принтере.

4.3.22 pine.conf

Файл `/etc/pine.conf` конфигурационный файл для программы чтения электронной почты **pine**. Этот файл позволяет установить умолчания для всех пользователей компьютера. Каждый пользователь может сам установить свои умолчания в своём домашнем каталоге. Детальнее можно познакомиться в `man pine`.

4.3.23 syslog.conf

В файле `/etc/syslog.conf` содержится конфигурационная информация для демона **syslogd**, который поддерживает запись диагностических сообщений системы. Более подробная информация о протоколировании системных сообщений содержится в `man syslog.conf`, `man syslogd`, **man sysklogd**.

4.3.24 passwd

Файл `/etc/passwd` содержит список пользователей компьютера вместе с дополнительной информацией. Подробности содержатся в `man -section 5 passwd`.

4.3.25 pwdb.conf

Файл `/etc/pwdb.conf` является конфигурационным файлом для библиотеки **libpwdb**. Библиотека состоит из набора функций, которыми авторы библиотеки хотели бы объединить разные файлы с информацией о пользователе (`passwd`, `shadow`, group и т.п) в виде единых средств доступа к этой информации. Описание функций данной библиотеки можно поискать на вашей рабочей станции, например, в `usr/doc/pwdb-0.58/html/pwdb-1.html` или в другом месте, которое будет подсказано ответом команды

```
locate pwdb
```

4.3.26 shadow

Файл `/etc/shadow` содержит зашифрованные пароли пользователей вместе с дополнительной информацией. Подробности содержатся в `man -section 5 shadow`.

4.3.27 services

Файл `/etc/services` содержит конфигурационную информацию о видах интернетовских сервисов на данной машине: имена, номера портов, типы протоколов. Каждая строка файла имеет вид:

```
service-name port/protocol [aliases ...]
```

Подробности в `man services`.

4.3.28 smb.conf

Файл `/etc/smb.conf` содержит конфигурационную информацию для пакета **Samba**. Этот пакет является файл-сервером **Windows SMB/CIFS** для **Unix/Linux**. SMB – Server Message Block. CIFS – Common Internet File System.

Подробности могут быть получены в `man smb.conf` и `man samba`.

4.3.29 hosts

В файле `/etc/hosts` содержится информация об окружающих ваш компьютер хостах. Эта информация может оказаться очень полезной в момент загрузки, когда ещё не запущен сервер **named**. Она может использоваться и позже в отсутствии сервера имён.

4.3.30 hosts.equiv

Файл `/etc/hosts.equiv` содержит информацию о хостах и пользователях, которым доверяют логироваться на данном хосте с использованием **rlogin** без ввода пароля, а также выполнять другие операции типа **r**: **rcp**, **rsh** и т.п. Смотрите `man hosts.equiv`.

4.3.31 motd

В файле `/etc/motd` содержится системное сообщение, которое выдаётся на экран при логировании в систему. Смотрите `man motd`.

4.3.32 hosts.access, hosts.deny

Файлы `/etc/hosts.access` и `/etc/hosts.deny` содержат описания правил доступа к вашему хосту. Язык описания правил позволяет весьма гибко описать кому разрешён доступ, отсекая тем самым все попытки несанкционированного доступа.

Смотрите `man hosts_access` и `man hosts_options`.

4.3.33 at_allow, at_deny

Файлы `/etc/at_allow` и `/etc/at_deny` используются для конфигурирования правил, кто из пользователей может запускать задания в пакете (команда **at**), а кто нет.

4.3.34 updatedb.conf

Файл `/etc/updatedb.conf` содержит конфигурационную информацию для команды **updatedb**.

Для получения дополнительной информации смотрите содержимое самого файла `/etc/updatedb.conf`, а также используйте

```
man updatedb
```

4.3.35 ypserv.conf

Файл `/etc/ypserv.conf` является ASCII конфигурационным файлом, который определяет ряд переменных для программного сервера **ypserv**, который является частью системы **NIS** вашей организации.

Можно смотреть `man ypserv.conf` и `man ypserv`.

4.3.36 yp.conf

Файл `/etc/yp.conf` является конфигурационным файлом для программы **ypbind**, которая является частью системы **NIS** вашей организации.

Можно смотреть `man ypbind`.

4.4 Конфигурационные файлы некоторых прикладных систем

4.4.1 a2ps-site.cfg, a2ps.cfg

Программа **a2ps** (преобразование обычного текста в **PostScript**) имеет два конфигурационных файла `/etc/a2ps-site.cfg` и `/etc/a2ps.cfg`. С их содержанием и описанием форматов можно познакомиться в

`info a2ps`.

4.4.2 enscript.cfg

Файл `/etc/enscript.cfg` является конфигурационным файлом для программы **enscript**, предназначенной для преобразования текста в **PostScript**. Подробнее можно прочесть в

```
info enscript
```

Глава 5

Языки программирования в Linux

Языки программирования (а точнее их реализации) условно разделяются на группу компиляторов и группу интерпретаторов. Таким способом подчеркивается характер порождаемых кодов, которые получаются в результате процесса трансляции. Если исходный текст языка сначала проверяется и транслируется целиком, а затем порождаемые коды являются в основном кодами машинного языка, то такой исходный язык является компилятором. Если строка исходного языка исполняется сразу после ввода, то такой язык зовётся интерпретатором. Оба типа имеют свои преимущества в определённых ситуациях. Так компиляторы, обычно, дают более эффективные по времени выполнения программы. Однако, стоимость компилятора и сопутствующих программ высока, а затраты на приготовление сложной программы (компиляция, сборка), которая часто меняется, оказываются неприемлемыми. Интерпретаторы удобны тем, что строки языка исполняются сразу. Таким образом, сразу видны недочеты или ошибки при написании программы. Кроме того, во многих случаях неважно, будет программа выполнена за 1 секунду или за 0.0001 секунды, например, программа диалога с оператором.

Естественно, что между этими типами имеется масса переходов и промежуточных вариантов. Примерами компиляторов являются **C**, **Fortran**, другие процедурно ориентированные языки. Примерами интерпретаторов являются **APL** (впервые появился на машинах **IBM**), **ИНФ** (на машинах типа **Днепр**), **REXX** (**IBM 370**), **bash**, **tcsh**, **perl**, **phython**, **Java**.

Со временем языки программирования стали все более склонны к интерпретации. Это заметно упрощает транслятор (что в свою очередь, уменьшает вероятность ошибок), увеличивает возможности переноса языка на машины с другой архитектурой, сокращает весь цикл приготовления программы: от разработки до получения программного продукта.

5.1 Фортран

Фортран – один из самых старых языков программирования, чистый компилятор. Первый вариант транслятора с языка **Фортран** появился в 1952 году. Название происходит от английского термина FORMULA TRANSLATION – транслятор формул. **Фортран** – весьма простой и широко распространенный язык для научных и инженерных вычислений. Он имеется на всех без исключения аппаратных платформах. На фортране написано много программ, которые используются до сих пор. Высказывается мнение, что стоимость используемых программ на фортране примерно равна или превышает стоимость установленных компьютеров, предназначенных для вычислений.

На **Фортран**е легко писать программы связанные с расчётами, поскольку язык специально ориентирован для этих целей. Обработку текста выполнять на **Фортран** не так удобно, как строить вычислительные программы.

Информацию о **Фортран** можно найти на странице *http://www.fortran.com/fortran/metcalf.html*.

5.1.1 f2c

На современных рабочих станциях часто не используют официальные трансляторы с языка **Фортран**, а пользуются бесплатными перекодировщиками из фортрана в **C**. Перекодировщик **f2c** FORTRAN TO C является одним из них.

5.1.2 g77

Компиляторы **C** и **F77** интегрированы в версии **GNU**; **g77** - это программа, которая вызывает **gcc** с возможностями распознавания текстов, написанных на **фортране** (ANSI FORTRAN 77, который часто называется просто F77). Компилятор gcc обрабатывает вводные файлы в несколько (от одной до четырех) последовательных стадий: макрообработка, компиляция, ассемблирование и сборка. Полное описание продукта **g77** следует смотреть документацию по **GNU Fortran**. О свойствах конвертера **g77** также получить информацию с помощью команд:

```
info g77
man g77
```

а также

```
g77 -version
```

и другими способами, описанными в главе 3. Исходные файлы **F77** обычно имеют суффиксы **.f** или **.for**; файлы, которые будут обрабатываться препроцессором **cpp** используют суффикс **.F** или **.fpp**.

Исходные тексты специального варианта **Фортран**а **Ratfor** содержатся в файлах с расширением ***.r** (хотя, компилятор Ratfor не поставляется как часть **g77**).

Следует обратить внимание, что если исходный текст, написанный на фортране, не полностью соответствует стандарту `ANSI FORTRAN 77`, то с компиляцией на **g77**, сборкой, а также с работоспособностью перекодированной программы могут быть проблемы.

5.1.3 Конвертер для фортран-90

Уже имеется конвертер для **Фортран-90** – **VAST/f90**. Этот конвертер перекодирует текст **Фортран-90** в текст **Фортран-77**. Сведения о конвертере можно найти на страницах

HSnamehttp://www.psrv.com/

5.2 C

Язык **C** имеется на всех аппаратных платформах и на всех операционных платформах. Разработан в 70-х. **UNIX** написан с использованием **C**. Многие воспринимают **C** как машинный язык. Это не так, хотя он находится ближе к машинным кодам, чем, например, **Фортран** или **Паскаль**. Таким образом, на **C** можно писать любые программы, включая программы обработки и анализа текста.

По языку **C** имеется обширная литература как в Интернет, так и в традиционном виде. В news группе `news:comp.answers` один или два раза в месяц рассылается **FAQ** по языку **C**. Можно обратить внимание также на архив **FAQ** *http://www.faqs.org*. Среди `html` страниц можно отметить *http://www.lysator.liu.se/c/index.html* или *http://www.strath.ac.uk/CC/Courses/CCourse/CCourse.html*. На русском языке описание языка **C** с примерами можно найти как во многих традиционных книгах, так и в Интернет: *http://cclib.nsu.ru/projects/gnudocs/texts/bogatir/book.html*.

5.3 C++

Язык **С++** является объектно-ориентированным языком. В основе синтаксиса лежит синтаксис языка **С** с включением объектно-ориентированных операций. В **С++** очень важен набор используемых библиотек. Различные библиотеки описываются в FAQ (см. например, *http://www.trumphurst.com/cpplibs1.html*). Полезно также обратить внимание на группу `news:comp.lang.c++`.

На **С++** можно писать программы любого назначения, в том числе и программы для анализа и преобразования текста.

5.4 Perl

Одним из важных языковых инструментов является язык **Perl** (PRACTICAL EXTRACTION AND REPORT LANGUAGE – язык практического извлечения информации и формирования отчетов). Это богатый язык, ориентированный в первую очередь на программистов, системных администраторов и, вообще, любых лиц, которым необходимо быстро извлечь какую-то полезную информацию из операционной или файловой систем и произвести небольшие вычисления. Удобен для анализа и преобразования текста.

Вот некоторые возможности:

- анализ подстрок;
- анализ текстов;
- печать в соответствии с форматами;
- etc.

Этот язык богаче многих мощных средств преобразования текстов в **UNIX**, например, **awk** (раздел 12) или **sed** (раздел 11). Для того, чтобы перекодировать в **perl** программы, составленные на языке **awk**, используется конвертер **a2p**. А конвертер **s2p** используется для перекодировки в **perl** программ, составленных с использованием **sed**.

perl можно использовать как обычную оболочку интерпретатор, так же как **tcsh** или **bash**. **perl** активно используется для составления программ **CGI** для `http` серверов. С использованием **perl** написана масса разнообразных приложений. **perl** свободно доступен во множестве мест, например: *http://www.gnu.org/software/perl/perl.html*. Может оказаться полезным **FAQ** *http://cpan.perl.org/doc/FAQs/*.

5.5 Tcl/Tk

Tcl/Tk - еще один свободно распространяемый интерпретатор, который часто используется для написания скриптов и организации диалога с экстенсивным использованием графических возможностей. **Tcl** обозначает TOOL COMMAND LANGUAGE (ИНСТРУМЕНТАЛЬНЫЙ КОМАНДНЫЙ ЯЗЫК). **Tk** обычно означает TOOL KIT (ИНСТРУМЕНТАЛЬНЫЙ НАБОР). Как правило, **Tcl/Tk** является частью поставки **Linux**, но может быть получен и отдельно, например, в *ftp://ftp.funet.fi/pub/languages/tcl.* Язык **Tcl/Tk** довольно развит и используется в сотнях приложений. Следует заметить, что **Tcl** и **Tk** часто хранятся отдельно, поскольку это вообще-то отдельные продукты. **Tk** представляет собой набор команд и описателей для создания и манипулирования с **X** виджетами. Интерпретатор, который интерпретирует строки **Tcl** и имеет средства **Tk** обычно носит имя **wish**. При вызове которого появляется строчное приглашение (обычно %) и графическое окно. Есть возможность вызвать лишь интерпретатор команд **Tcl**, который работает в строчном режиме. Его также можно использовать как обычную оболочку как, например, **bash**. Пример короткой тестовой программы на **Tcl** приведён ниже.

```
#!/usr/bin/tcl
puts stdout "";
puts stdout "This is test program for Tcl";
set x 4; set y 19;
set Result [expr sqrt(\$x * \$x + \$y * \$y)];
puts stdout Result=\$Result ;
```

При выполнении скрипт печатает пустую строку, затем

```
This is test program for Tcl
Result=19.4164878389
```

5.6 Python

В настоящее время **python** является одним из объектно–ориентированных интерпретаторов. Как правило, **python** поставляется в составе **Linux** или может быть свободно взят из *http://www.pythin.org*. **python** имеет средства для описания высокоуровневых структур данных. Имеется разработанный интерфейс для расширения **python** за счет библиотек написанных на **C** и

C++, а также других языков программирования. Графический интерфейс для **python** реализуется посредством пакета **Tk** из **Tcl/Tk**.

С использованием **python** можно анализировать и преобразовывать текст любого вида.

Во многих случаях выбор интерпретатора является просто волей случая. Если вы уже участвуете в крупном проекте, то, как правило, там уже утвердились те или иные соглашения (правила) по поводу используемых языков и даже на способы написания программ в уже выбранном языке.

5.7 Java

java является наиболее свежим и современным интерпретатором. Сейчас **java** – реальный претендент на место кроссплатформного средства для разработчиков, т.е. имеется большая уверенность, что если программная система написана на языке **java**, то она будет работать на всех вычислительных платформах. С другой стороны, **java** – высокоуровневый объектно-ориентированный язык программирования.

java – неплохая платформа для реализации алгоритмов анлиза и обработки текста.

5.8 Другие полезные языки

5.8.1 Ada

Язык **Ada** появился в конце 60-х. Это один из наиболее строго определённых языков программирования, который имеет весьма строго определённую систему компиляции. Благодаря хорошо проработанной стандартизации интерфейсов и учётом конкретных свойств вычислительного оборудования, язык **Ada** широко используется для программирования бортового вычислительного оборудования, а также различного вида контроллеров. Одной из реализаций системы программирования **Ada** является подсистема **GNAT**. Информация о подсистеме **GNAT** может быть получена на страницах *http://www.gnu.org/software/gnat/gnat.html*.

5.8.2 Pascal и конвертер p2c

Конвертер **p2c** представляет собой средство для трансляции (перекодировки) исходных текстов программ на языке **Pascal** в программы на языке **C**. Вводом может служить любой из следующих диалектов языка **Pascal**: **HP**

Pascal, **Turbo/UCSD Pascal**, **DEC VAX Pascal**, **Oregon Software Pascal/2**, **Macintosh Programmer's Workshop Pascal**, **Sun/Berkeley Pascal**. Поддерживается синтаксис языка **Modula-2**.

Большинство программ на **Pascal** транслируются в **C** и не требуют дальнейших усилий по модификации, чтобы сделать программу работоспособной. Хотя, в ряде случаев, могут печататься предупреждения, что в отдельных точках потребуется ручная коррекция исходного текста.

Pascal является языком общего назначения, поэтому подходит для разработки программ анализа и преобразования текста.

5.8.3 Prolog

Сокращение **Prolog** означает Programming in Logic (программирование в логике). **Prolog**, как предполагается, освобождает программиста от массы компьютерных деталей, чтобы сосредоточиться на логике решаемой задачи. Транслятор можно найти во многих местах, например, *ftp://swi.psy.uva.nl/pub/SWI-Prolog/* или *ftp://clement.info.umoncton.ca/BinProlog/UNCOMPRESSED/bin* Дополнительную информацию о языке **Prolog** можно найти в `http://aisun0.ai.uga.edu/ jae/ai-lang.html`.

Prolog используется в работах по искусственному интеллекту и в других, столь же нетривиальных областях применения.

5.8.4 Lisp

Lisp – это язык обработки списков. Часто используется в системах искусственного интеллекта. Реализация языка имеет массу вариантов, более подробную информацию о нем можно найти, например, в *http://www.elwood.com/alu/table/contents.htm* или *ftp://sunsite.unc.edu/pub/Linux/devel/lang/lisp/*. Один из интерфейсов к **Lisp** имеется в известном редакторе текста **emacs** (раздел 8.3).

Lisp удобен для написания весьма сложных программ анализа и преобразования текста.

Глава 6

Рабочие столы (desk top) в Linux

6.0.5 Введение

Фактически **UNIX** был необсуждаемым выбором профессионалов в области информационных технологий уже много лет. Что же касается стабильности, масштабируемости и открытости, то у **UNIX** (**Linux**) нет конкурентов. **UNIX** и **Linux** доминируют на рынке серверов и являются предпочтительной платформой как для научных приложений и так и для профессионалов в области компьютерных технологий.

Однако отсутствие удобного пользовательского интерфейса ограничивало применение **UNIX** (**Linux**) на домашних компьютерах и в офисах. В последнее время понимание факта, что операционная платформа ценна для рядового пользователя своими приложениями, дало толчок разработкам средств типа РАБОЧИХ СТОЛОВ.

Рабочий стол представляет собой визуальное изображение различных информационных компонентов, с которыми вы имеете дело в компьютере, поддержанное соответствующей программной интерфейсной системой типа **API** (Application Programer Interface). К обычным свойствам систем рабочего стола относятся: возможность конфигурирования визуального изображения рабочего стола, состав отображаемых объектов (файлы, каталоги, приложения, прочее), возможность использования удобного файлового менеджера и других систем упрощающих выполнение рутинных операций.

Обычными компонентами рабочих столов являются:

- менеджер файловой системы, который позволяет вам производить большую часть операций с файлами (копирование, перемещение, переименование, другие операции);
- управляющая панель, с помощью которой вы конфигурируете изображение иконок, их состав, а также соотвествие иконок реальным

программам;

- различные приложения, которые вы посчитали необходимым включить (или они включены по умолчанию) на ваш рабочий стол.

Какой рабочий стол в **Linux** является лучшим? Этот вопрос до сих пор (октябрь 1999) обсуждается в среде пользователей и профессионалов. Многое зависит от ваших целей и дополнительных условий: хотите ли вы использовать только лицензию **GPL** или вы согласны на другие виды лицензий; какой рабочий стол используют ваши ближайшие коллеги; наконец, ваши личные предпочтения.

6.1 Рабочий стол KDE

KDE означает THE K DESKTOP ENVIRONMENT, или К-среда рабочего стола.

KDE является прозрачной сетевой средой для рабочего стола (desktop) компьютерной системы под управлением **UNIX** (**Linux**). **KDE** стремится стать удобным пользовательским интерфейсом для **UNIX** (**Linux**)таким же как **MacOS** или **MS Windows**. Часть разработчиков и пользователей полагают, что **UNIX** (**Linux**) лучше других систем подходит к решению конкретных задач, стоящих перед пользователями.

Команда программистов, которая разработала **KDE**, разработала также приложение KOM/OpenParts, которое помогает быстро строить прототипы приложений. Технология **KOM/OpenParts** базируется на открытых индустриальных стандартах таких как **Object Request Broker CORBA 2.0**.

KDE является почти свободно распространяемой системой, которую можно найти в *http://www.kde.org/*. Термин ПОЧТИ означает в данном случае, что не все компоненты **KDE** являются полностью свободными продуктами. Так, подсистема **Qt** должна быть получена отдельно, а не в составе **KDE**.

Условия распространения **KDE** отличаются от условий **GNU**, хотя для некоммерческого использования **KDE** и его компоненты бесплатны для потребителя. Ранее, вопросы о правилах распространения возникал лишь по отношению к компоненту **Qt**. Но с декабря 1998 года **Qt** поставляется свободно. Видимо, для конкретных случаев в бизнесе следует проконсультироваться с законником, который имеет опыт в таких делах.

6.2 Рабочий стол GNOME

GNOME (GNU Netwoek Object Model Environment – Сетевая Объектная Среда GNU) есть дружественный к пользователю набор приложений и средств рабочего стола, которые используются совместно с системой X-Window. Цель **GNOME** практически та же, что и **KDE** и **CDE**, однако **GNOME** целиком базируется на программном обеспечении с полностью открытыми исходными текстами (Open Source software). **GNOME** является частью поставки многих вариантов **Linux** включая **RedHat**.

GNOME имеет свой сайт в Интернет: *http://www.gnome.org/*.

GNOME использует менеджер окон, который является часть вашей вашей установки **Linux**. Как известно менеджеров окон довольно много. **GNOME** имеет список менеджеров окон, с которыми он совместим. Список включает наиболее популярные менеджеры окон. **GNOME** весьма эффективно использует графические ресурсы, используя их не только для поддержки пользователя, но для конфигурирования самой системы **GNOME**. Почти всё конфигурирование выполняется с помощью КАППЛЕТОВ – CAPPLETS – небольших управляющих апплетов (графических программ), которые являются частью системы **GNOME**.

Система обеспечивает поддержку разработки с использованием различных языков программирования, автоматически реализуя совместимость различных приложений разработанных в **GNOME**. Взаимодействие **GNOME**-приложений друг с другом осуществляется на основе CORBA (Common Object Request Broker Architecture). Это даёт независимость от аппаратной платформы и, даже, версии UNIX/Linux.

Описание системы **GNOME** имеется как на английском языке, так и на многих других языках, включая русский. Описание на русском доступно в *http://www.gnome.org/users-guide/ru/gccedit.html*.

Глава 7

Список команд Linux

Список команд **Linux** весьма обширен, видимо несколько сотен или более. Многие команды выполняют схожие функции. Совсем не обязательно знать все имеющиеся команды или приложения. Достаточно создать (имеется в виду освоить) подходящий минимум, который обеспечит 90% ваших потребностей. Здесь мы рассмотрим несколько команд без которых нельзя продуктивно работать.

7.1 Вход в систему

Первоначальный вход в систему, иначе ЛОГИРОВАНИЕ, производится с помощью самой системы, которая вызывает программу **login**, которая вызывается из программ **init** или **getty**. Вы сами можете вызвать программу **login**, когда уже находитесь в системе

```
exec login user-name [var1=vvar1 var2=vvar2 ...]
```

после имени пользователя можно присвоить значения переменным окружения. Некоторые переменные не могут быть установлены. К ним относятся: SHELL, TERM, USER, HOME. Если вы делаете первоначальное логирования, то система выдаёт приглашение, которое хранится в файле `/etc/issue`.

После ввода пароля система выводит сообщения из файла `/etc/motd` на экран. Полезно помнить, что запретить вывод сообщений на экран можно разместив в домашнем каталоге $HOME файл нулевой длины с именем `.hushlogin`.

7.2 Выход из системы

Выход из системы производится если произошёл выход из оболочки. Из оболочки вы может выйти, например, выполнив команду **exit**. Выйти из оболочки в которую вы вошли по команде **login** вы может использовав команду **logout**. Во многих случаях выход из оболочки последует, если вы ввели комбинацию `Ctrl/d`. Для более подробной информации следует изучать руководства.

7.3 Типы файлов

Если взглянуть на оглавление любого каталога с помощью команды `ls -l`, то можно увидеть перечисление файлов в примерно следующем виде:

```
-rw-r--r--  1 shevel   users   2629 Aug 23 17:06 A2PS_Delegation.tex
```

Здесь самая левая часть `-rw-r--r--` представляет собой следующую информацию:

- тип файла (самый левый символ):

  $\boxed{\text{d}}$ - это каталог;

  $\boxed{\text{-}}$ - простой файл;

  $\boxed{\text{l}}$ - символический линк;

  $\boxed{\text{c}}$ - символьное устройство;

  $\boxed{\text{b}}$ - блоковое устройство;

  $\boxed{\text{p}}$ - именованный программный канал;

  Не обязатыельно помнить наизусть все эти обозначения. Можно воспользоваться командой **file**, которая сообщит вам тип файла, например,

  ```
  $ file /usr/lib/libjpeg.so.62
  /usr/lib/libjpeg.so.62: symbolic link to libjpeg.so.62.0.0
  $ file t
  t: fifo (named pipe)
  ```

- поле из девяти символов описывающее кому разрешён доступ к файлу: левые три символа описывают доступ для владельца файла, средние три символа информируют о доступе для пользователей, которые состоят в той же группе, что и владелец файла, а последние три символа описывают какое отношение к данному файлу имеют все остальные пользователи.

Для обозначения доступа используются следующие буквы:

`r` - разрешено чтение файла;

`w` - разрешена запись в файл;

`x` - разрешено исполнение, а если это каталог, то разрешен просмотр каталога.

Отметим кратко значения остальных полей. Число, которе следует сразу за описателем доступа, представляет собой количество символических линков к данному файлу. Далле следует имя владельца файла, название группы, к которой принадлежит файл, размер файла в байтах, время создания или модификации файла, наконец, имя файла.

7.4 Манипуляции с файлами

В **Linux** файловая система имеет иерархический характер. Дерево каталогов начинается с корневого каталога, который обозначается символом `/` (наклонённая вправо косая черта или прямой слеш). Любой каталог может содержать файл(ы) или/и каталог(и). Если имя файла начинается с обозначения корневого каталога, то говорят, что задано **абсолютное имя файла**. Например, `/etc/passwd`. Если имя файла не начинается с косой черты, то такое имя полагается **относительным именем файла**. Например, `passwd`. Если вам необходимо указать, что файл находится в текущем каталоге, то следует использовать обозначение `./file-name` (точка и слеш). Обозначение `../file-name` укажет, что файл с именем `file-name` находится как раз над текущим каталогом. Можно также использовать более сложные обозначения, например, файл `../../../file-name` находится на три уровня выше текущего каталога.

Каждый пользователь в **Linux** имеет свой каталог, который часто именуется основным или домашним. Домашний каталог для краткости обозначается знаком ~ (тильда – волнистая черта). Если используется нотация `ls ~`, то будет выведено содержание вашего домашнего каталога. А если вы укажете `ls ~ivanov`, то будет выведено оглавление домашнего каталога пользователя с именем `ivanov`.

Имеются несколько типов файлов: простые (обычные) файлы, которые содержат данные; специальные файлы, описывающие физические устройства ввода/вывода; специальные файлы, которые используются для организации очередей типа FIFO, или программных каналов.

Полезные сведение о свойствах файлов можно почерпнут на страницах руководства `info file`.

7.4.1 Изменение прав доступа к файлу, владельца и группы

Права доступа к файлу можно изменить с помощью команды **chmod** – change mode – сменить режим доступа. Например, команда

```
$ chmod o+r t
```

означает, что для всех пользователей устанавливется разрешение читать файл с именем `t`. Следует заметить, что остальные пользователи смогут реально прочесть данный файл, если только права доступа к оглавлению каталога, в котором находится данный файл, а также все вышележащие каталоги влоть до

```
/home/username
```

разрешают просматривать оглавление каталога. Гарантировать возможность чтения оглавления каталога можно так:

```
chmod o+x catalogname
```

Изменить владельца файла или каталога можно командой **chown** – change owner – сменить владельца.

Изменить группу, которой принадлежит файл или каталог можно командой **chgrp** – change group – сменить группу (или присвоить группу).

7.4.2 Создание файлов и каталогов

Мы не будем рассматривать создание файлов устройств, поскольку это есть тема для другой книги.

Специальный файл для организации именованной очереди типа **FIFO** (или именованного программного канала) создаётся посредством команды **mkfifo**. Например, `mkfifo TranSport`. По этой команде создаётся специальный файл с именем `TranSport`, который можно использовать в качестве среды передачи информации от одного процесса другому независимо выполняющемуся процессу. Например,

```
ls -l > TranSport &
cat TranSport
```

Здесь один процесс, который выполняет команду `ls -l` закачивает данные в очередь, а другой процесс, выполняющий `cat TranSport` выводит данные из очереди на экран.

Создание простых файлов может выполняться любой программой, которая выводит данные на диск. Это может быть редактор текстов, сборщик задач, готовая к исполнению пользовательская программа и т.д. Например, командой

```
cp /dev/null Temp
```

создаётся файл с именем `Temp` нулевой длины.

Создание каталога производится командой **mkdir**. Например,

`mkdir T`,

т.е. создаётся пустой каталог с именем `T`, куда можно сразу перейти командой `cd T` (change directory – сменить каталог).

7.4.3 Уплотнение файлов и каталогов и перенос каталогов

Часто оказывается необходимым уплотнить (сжать) редко используемые текстовые файлы, чтобы они занимали меньше места или для сокращемия времени передачи по сети. Для этого используется несколько программ. Стндартный комплект программ для уплотнения (сжатия) данных, который есть во всех **UNIX/Linux** системах называется `compress`/**uncompress**.

Во многих случаях при уплотеении текста лучшие результаты даёт комаплект программ **gzip/gunzip**. Например, типичная задача уплотнить файл типа **PostScript**: при уплотнении файла размером 1907497 программ **gzip** дала результурующий объём 580655 байтов, а программа **compress** получила обём 737319 байтов.

Обе программы, как правило, можно найти во всех вариантах **Linux**.

Наконец, перенос каталогов. Легко представить, что каталог со множеством подкаталогов, например, 7 или 10 уровней, часто непросто перенести на новое место, например, в другую файловую систему или на другую машину. Имеется несколько способов. Например, можно воспользоваться командой

```
rcp -r catalogname remotehost:newcatalog
```

Другой способ – перейти внутрь каталога с именем `catalogname`, выполнить программу **tar**, затем уплотнить полученный результат, а уже уплотнённый файл передать по сети. Например,

```
cd catalogname
tar zcvf My.tar
rcp My.tar.gz remotehost:.
```

На другой машине с именем *host* можно выполнить обратную операцию:

```
tar zxvf My.tar.gz
```

7.4.4 Удаление файлов и каталогов

Удаление файлов производится командой **rm**, например,

```
rm file-name1 file-name2 ...
```

т.е. одной командой можно удалить несколько файлов. Если вы хотите удалить не пустые каталоги, то это делается так:

```
rm -rf catalog-name1 catalog-name2 ...
```

т.е. вы можете удалить сразу несколько каталогов вместе с файлами и подкаталогами внутри.

Если требуется удалить файл с именем, например, `-C`, которое содержит специальные знаки, то следует использовать такую форму команды **rm**

```
rm - -C
```

Операция удаления и является наиболее опасной. В современных системах вы можете иметь доступ сразу к большому количеству данных, например, ко многим тысячам или десяткам тысяч файлов. Если вы удаляете одной командой много файлов, то в случае незначительной ошибки можете удалить свои труды и труды своих коллег за большое количество времени. Поскольку в `Linux` нет автоматически встроенной операции **undo** (т.е. восстановления удалённых файлов), то необходимо принимать специальные меры, чтобы обезопасить себя на случай своих ошибок при удалении файлов.

Одним из хороших методов является не использовать команду **rm**, а использовать вместо неё команду **mv** (move – переместить). Например,

```
mv Test OutGoing/.
```

вы перемещаете файл FSnameTest (или каталог) в каталог `OutGoing`, вместо удаления файла. Таким образом, если вы удалили нечто ошибочно, то есть возможность вернуть удалённый объект назад. Тем не менее, время от времени вам придётся чистить каталог `OutGoing` по соображениям занимаемого объёма, т.е. удаление будет происходить, но в отложенном режиме. В РАБОЧИХ СТОЛАХ, которые мы рассмотрим чуть позже в разделе 6.1, как правило, имеются средства для поддержания технологии отложенного удаления, пример которой приведён выше.

В разделе 14 мы будем рассматривать специальную систему, которая помогает поддерживать целостность ваших текстов в системе.

Глава 8

Редакторы текста

В **Linux** широко используются около десятка редакторов текста. Каждый редактор имеет какие-то ему свойственные особенности. В каждом отдельном случае может оказаться интересен тот или другой редактор, в зависимости от реального контекста использования, включая психологические предпочтения того, кто пользуется редактором.

Почти все редакторы имеют описания в форматах **man** и/или **info**. Чтобы вывести описание на печать проще всего воспользоваться программой **a2ps**, например,

```
man vi | a2ps -1 -catman
```

По этой команде на печать будет выведено описание редактора **vi** (подробнее смотрите раздел 13.

8.1 vi/vim

vi - один из старейших и наиболее мощных экранных редакторов (работает на терминалах типа **xterm**, **vt-100**). Обычно **vi** является частью поставки **Linux**.

По этому редактору текстов довольно часто рассылаются тексты с **FAQ** (Frequently Asked Questions, или ЧАсто задаваемые ВОпросы - **ЧАВО**) в news группе `comp.answers`. Архив news групп может быть найден в `http://www.faqs.org`.

Как сам **FAQ** по редактору **vi**, так и родственная информация может быть найдена в ряде мест

- *ftp://alf.uib.no/pub/vi*
- *ftp://ftp.uwp.edu/pub/vi*
- *ftp://ftp.uu.net/pub/text-processing/vi*

- *ftp://ftp.cc.monash.edu.au/pub/vi*
- *ftp://ftp.s.u-tokyo.ac.jp/misc/vi-archive*

Имеется более поздний улучшенный вариант редактора **vi** под именем **vim** (**V**i **IM**proved - улучшенный **vi**). Редактор **vi** целиком совместим по командам с **vim**. В то же время у **vim** имеется расширенный набор параметорв при вызове редактора и он имеет возможность вводить и редактировать Кириллицу.

Кроме того, **vim** имеет значительное число улучшений по сравнению с **vi**:

- многоуровневый процесс **undo**;
- использование нескольких окон и буферов редактирования;
- редактирование командной строки;
- механизм завершения имени файла, если вы напечатали только начало имени;
- on-line help (когда редактор уже вызван, следует ввести **:help**);
- и прочее.

8.2 sed

sed строчный редактор текста, его удобно вызывать внутри скриптов. Обычно **sed** является частью поставки **Linux**. Подробнее смотрите раздел 11.

8.3 emacs,xemacs

emacs относительно новый (по сравнению с **vi**) мощный экранный редактор текста с массой встроенных функций по изменению текста, созданию командных последовательностей редактирования текста и взаимодействия с операционной системой. **emacs** можно считать настраиваемой средой для программиста, в которой имеются развитые средства редактирования текста.

xemacs является модернизацией **emacs** главным образом в использовании графического интерфейса. Хорошим источником информации является *http://www.xemacs.org/*. Оба редактора обычно являются частью поставки **Linux**.

8.4 xedit

xedit - простой редактор текста с использованием возможностей **X/Window**. Подробнее можно посмотреть страницы `info xedit`. Имеется почти в каждой поставке **Linux**.

8.5 nedit

nedit - весьма развитый редактор текста для среды X/Window. Позволяет работать с использованием схемы клиент/сервер. Список рассылки по обсуждению вопросов по редактору {HSnamemailserv@fnal.gov. **nedit** фактически является свободно распространяемым продуктом с открытыми кодами. Он доступен по адресу: *ftp://ftp.fnal.gov/pub/nedit/*.

8.6 pico

pico является простым редактором текста с небольшим набором команд редактирования, которым легко пользоваться на терминалах типа `vt-100`. Этот редактор является частью клиентской подсистемы электронной почты **pine**. Исходные тексты находятся в *ftp://ftp.cac.washington.edu/mail/pine.tar.Z*.

8.7 joe

Имеется довольно мощный текстовый редактор. Фактически **joe** является некоторым сборным редактором, который включает в себя привлекательные модели работы с текстом заимствованные из различных редакторов: **pico**, **WordStar**, **emacs**.

Редактор **joe** имеется почти в каждом варианте **Linux**. Этот редактор можно найти в *ftp.std.com/src/editors/joe*.tar.Z*.

Глава 9

Оболочки

shell – оболочка, или командный интерпретатор представляет собой программу, которая анализирует и выполняет команды, вводимые с терминала. Командный интерпретатор не требует для себя каких-то специальных привилегий и рассматривается системой как обычная программа. Существует довольно много различных командных интерпретаторов. К типичным свойствам интерпретатора можно отнести интерпретацию командных последовательностей (scripts: скриптов), расширение имен файлов, которые содержат знак "*"(звёздочка), реализацию командных трубопроводов, повторное исполнение ранее выполненных команд, выполнение циклов и условных переходов, а также создание аббревиатур. Скрипт или командная последовательность представляет собой текстовый файл, который содержит команды в том виде, в котором они могли бы вводиться с терминала.

Очевидно, что скрипт удобно использовать, чтобы не повторять ввод одной и той же последовательности команд интерпретатора вручную. Достаточно ввести последовательность команд лишь однажды и после проверки (отладки) его можно использовать сколько угодно раз.

9.1 Команды

Простая команда – это последовательность слов, разделенных одним или несколькими пробелами. Первое слева слово определяет имя команды, которую следует выполнить. Команда может состоять из одного имени, например, **w**. За командой могут следовать параметры команды; как правило им предшествует знак - (минус). Параметры команды часто состоят из одной буквы. Например,

```
ls -l
```

или в случае нескольких параметров

```
ls -l -t
```

Если параметров несколько, то они могут быть объединены; так что предыдущий пример может быть переписан в виде

```
ls -lt
```

Довольно часто параметры состоят из одной буквы.

За параметрами могут следовать АРГУМЕНТЫ, например, имена файлов,

`ls -lt A*`.

Аргументы состоят из одного слова. Аргумент может относиться как к самой команде, так и к отдельному параметру. Например, как в команде

```
awk -f AwkProg FileName
```

Здесь команда **awk** обрабатывает файл с именем `FileName` под управлением программы с именем `AwkProg`.

Два минуса подряд означают конец параметров. Например, команда

```
rm - -abc
```

удалит файл с именем `-abc`. Если вы попробуете выдать команду

`rm -abc`,

то имя файла будет воспринято как параметр и вы получите сообщение об ошибке.

Наконец, еще одно значение знака - (минус). Если знак используется на месте аргумента, то вместо имени файла используется стандартный ввод (или вывод).

КОМАНДА – это ПРОСТАЯ КОМАНДА или КОМАНДА, за которой следует ПРОСТАЯ КОМАНДА.

ФИЛЬТР – это такая команда, которая читает данные из стандартного вводного файла, преобразует их каким-то образом и выводит результат в стандартный выводной файл.

ПРОГРАММНЫЙ КАНАЛ ИЛИ ТРУБОПРОВОД – это последовательность двух или более команд разделенных знаком `|` (вертикальная черта). Основная черта ПРОГРАММНОГО КАНАЛА состоит в том, что стандартный вывод каждой команды, стоящей слева от вертикальной черты, соединяется со стандартным вводом команды, стоящей справа от вертикальной черты. Как видно из определений, ПРОГРАММНЫЙ КАНАЛ составлен из ФИЛЬТРОВ.

СПИСОК – это один или более ПРОГРАММНЫХ КАНАЛОВ разделенных знаками ;, &, && или `||` и, возможно, ограниченными знаками ; и &. Из этих четырех знаков ; и & имеют равный приоритет, который ниже, чем приоритет у знаков && и `||`. Приоритет последних также одинаков. Точка с запятой (;) приводит к последовательному выполнению ПРОГРАММНОГО КАНАЛА, за которым она стоит; амперсанд (&) приводит к асинхронному выполнению ПРОГРАММНОГО КАНАЛА, за которым он стоит. Иными словами, оболочка-интерпретатор не будет ожидать завершения выполнения ПРОГРАММНОГО КАНАЛА, а перейдет немедленно к интерпретации следующего ПРОГРАММНОГО КАНАЛА. Символ && (||) приведет к тому, что следующий СПИСОК будет интерпретироваться только в том случае, если текущий ПРОГРАММНЫЙ КАНАЛ завершился с нулевым (ненулевым) кодом завершения. В качестве ограничителя команд вместо точки с запятой могут использоваться также один или более знаков перевода строки.

9.2 Командные последовательности – скрипты

Как уже было отмечено, TSnameскрипт – это файл, содержащий последовательность команд оболочки. Этот файл может быть приготовлен с помощью любых средств, как обычный текстовый файл. Например, его можно приготовить с помощью любого редактора текстов, или командой `cat`

```
cat > TestExecFile
echo "This is first exec file."
Ctrl/d
chmod +x TestExecFile
```

Замечание: `Ctrl/d` в предыдущем примере означает, что на клавиатуре терминала должны быть нажаты одновременно две клавиши: `Ctrl` и `d` (буква "d"). В **Linux** ввод такой комбинации воспринимается как КОНЕЦ ФАЙЛА при вводе текста или как КОНЕЦ СЕАНСА при вводе команд. КОНЕЦ СЕАНСА вызовет специальную процедуру выхода из системы. После выполнения такой процедуры работать с системой будет возможно лишь после выполнения процедуры входа в систему (команда **logon**).

Если следом попробовать запустить скрипт `./TestExecFile`, то на экране будет напечатана строка:

```
This is first exec file.
```

Скрипт можно сгенерировать с помощью другого скрипта. Например, нижеприведённый текст скрипта генерирует скелет (основу) нового скрипта:

```
#!/bin/bash
#set -x
#Call c_script script_name

SCRIPT_NAME=$1
if [ "" = "$1" ]; then
```

```
         echo
         echo "Usage: c_script scriptname"
         echo
         exit
fi

echo "#" >  $SCRIPT_NAME
echo "#!/bin/bash" >> $SCRIPT_NAME
echo  >> $SCRIPT_NAME
echo "#---------------------------------------------------------------+" >> $SCRIPT_NAME
echo "# |" >> $SCRIPT_NAME
echo "# Usage:" $SCRIPT_NAME >> $SCRIPT_NAME
echo "# |" >> $SCRIPT_NAME
echo "# The script is dedicated to: |" >> $SCRIPT_NAME
echo "# |" >> $SCRIPT_NAME
echo "# Creation date: " ‘date‘" |" >> $SCRIPT_NAME
echo "# The host where it was developed="‘hostname‘ " |" >> $SCRIPT_NAME
echo "# History of changes: |" >> $SCRIPT_NAME
echo "# |" >> $SCRIPT_NAME
echo "#---------------------------------------------------------------+" >> $SCRIPT_NAME
echo "# |" >> $SCRIPT_NAME
echo "#  Author: Andrei Chevel. email: Andrei.Chevel@pnpi.spb.ru |" >>$SCRIPT_NAME
echo "#---------------------------------------------------------------+" >> $SCRIPT_NAME
echo >> $SCRIPT_NAME
chmod +x $SCRIPT_NAME # To make the script executable
```

Легко видеть, что приведенный скрипт во время исполнения формирует основу нового скрипта и автоматически помещает в виде комментариев ряд информационных строк.

Обратим внимание на особенности текста любого скрипта.

Строка комментария помечается знаком `#` (решетка). Исключение составляет первая строка скрипта, если за `#` следует знак "!"(восклицательный знак). Так устанавливается абсолютное имя файла, который содержит программу, которая будет интерпретировать ваш скрипт.

Если не указан вид оболочки, то скрипт интерпретируется оболочкой, установленной по умолчанию на вашей машине. Если не установлено умолчание, то используется оболочка **sh**.

В начале скрипта полезно самому установить тип интерпретатора. При этом можно записать на месте интерпретатора любую программу. Например, если вы предпочитаете интерпретатор **bash**, то первую строку скрипта можно написать в виде:

`#!/bin/bash`

если, конечно, **bash** находится именно в каталоге `/bin`.

Для того, чтобы скрипт распознавался операционной системой как исполняемый файл следует выполнить команду:

```
chmod u+x TestExecFile
```

или просто

```
chmod +x TestExecFile
```

9.3 Оболочка sh

Это самая старая оболочка, которая быть может менее дружественна по отношению к пользователю, но она точно имеется на любой машине, на которой установлен любой вариант **UNIX** и **Linux**. Во многих случаях, если не указана оболочка, в которой следует выполнять командную последовательность, то она выполняется в оболочке **sh**.

9.3.1 Перенаправление вывода

До того, как команда выполнится, её ввод и/или вывод могут быть перенаправлены с использованием специальной нотации, которую интерпретирует оболочка. Перенаправление может быть также использовано, чтобы открыть или закрыть файлы для текущего окружения оболочки. Следующие перенаправляющие операторы могут предшествовать или появляться где угодно в простой команде или могут следовать за командой. Перенаправление обрабатывается оболочкой в том порядке, в котором оно появляется слева направо.

В следующих описателях, если номер файла отсутствует и первый символ оператора перенаправления является `<` (знак меньше), то перенаправление подразумевает стандартный ввод (номер файла $= 0$). Если первый символ оператора перенаправления есть `>` (знак больше), то перенаправление относится к стандартному выводу (номер файла $= 1$). Возможные варианты перенаправления приведены в таблице 9.1.

Таблица 9.1: ПЕРЕНАПРАВЛЕНИЕ ВВОДА/ВЫВОДА В SH

`<file_in`	использовать файл с именем `file_in` в качестве стандартного ввода.
`>file_out`	использовать файл с именем `file_out` в качестве стандартного вывода.
`>>file_out`	использовать файл с именем `file_out` в качестве стандартного вывода. Если файл существует, то вывод добавляется в конец файла. В противном случае файл создаётся.
`<<word`	после того как параметрическая и командная подстановка выполнена над словом `word`, оболочка-интерпретатор читает ввод до строки, которая содержит слово `word` или до конца файла. Однако, если слову `word` предшествует знак `-` (минус), то 1. лидирующие символы табуляции удаляются из слова `word` после подстановки, но до начала чтения ввода, 2. лидирующие знаки табулятора удаляются из читаемых строк до сравнения со словом `word`, 3. наконец, оболочка-интерпретатор читает до первой строки, которая содержит слово `word` или до конца файла. Если любой символ слова `word` закавычен, то никакой дополнительной обработки над введёнными строками не производится. Если нет закавыченных символов в слове `word`, то: 1. производится параметрическая и командная подстановка; 2. символы NewLine игнорируются; 3. знак \ (обратный слеш) должен использоваться, чтобы квотировать специальные символы , (запятая) $ (доллар) и **’** (одиночный апостроф).

Продолжение таблицы на следующей странице

Перенаправление ввода/вывода в sh (продолжение таблицы 9.1)	
`<&digit`	использовать файл связанный с номером `digit`, в качестве стандартного ввода. Подобным же образом для стандартного вывода используется нотация `>&digit`.
`<&-`	стандартный ввод закрывается. Подобная нотация используется для стандартного вывода `>&-`.

Например:

... `2>&1`

файл с номером 2 перенаправляется в файл с номером 1.

Если за командой следует знак `&` (амперсанд), то стандартным вводом для такой команды по умолчанию является пустой файл */dev/null*.

9.4 Сравнительные характеристики оболочек

Ниже приведена таблица, которая содержит информацию о сравнительных особенностях различных оболочек-интерпретаторов. Автор взял основные данные для этой таблицы из

`http://www.looking-glass.org/shell.html`

1-го октября 1998.

Таблица 9.2: Сравнение оболочек

Свойство	SH	CSH	KSH	BASH	TCSH	ZSH
Управление заданиями	есть	есть	есть	есть	есть	есть
Определение синонимов	есть	есть	есть	есть	есть	есть
Определение функций	есть(1)	нет	есть	есть	нет	есть
Чувствительность к переопределению ввода/вывода	да	нет	да	да	нет	да
Стек каталогов	нет	да	да	да	да	да
История команд	нет	да	да	да	да	да
Редактирование в командной строке	нет	нет	да	да	да	да
Редактирование в командной строке в стиле редактора Vi	нет	есть	есть	есть(3)	есть	есть
Продолжение таблицы на следующей странице						

Сравнение оболочек (продолжение таблицы 9.2)						
Редактирование в командной строке в стиле редактора Emacs	нет	нет	есть	есть	есть	есть
Переопределение режимов редактирования в командной строке	нет	есть	есть	есть	есть	есть
Возможность сделать автоматический logout после определенного числа минут неактивности	нет	есть	есть	есть	есть	есть
Возможность наблюдения за активностью отдельных пользователей или терминалов	нет	нет	нет	нет	есть	есть
Автоматическое завершение имён файлов	нет	есть(1)	есть	есть	есть	есть
Автоматическое завершение имён пользователей	нет	есть(2)	есть	есть	есть	есть
Автоматическое завершение имён хостов	нет	есть(2)	есть	есть	есть	есть
Завершение команд из файла истории	нет	нет	нет	есть	есть	есть
Полностью программируемое (конфигурируемое) завершение	нет	нет	нет	нет	есть	есть
Многозадачность	нет	нет	есть	нет	нет	есть
Встроенные арифметические вычисления	нет	есть	есть	есть	есть	есть
Периодическое выполнение команд (set tperiod)	нет	нет	нет	нет	есть	есть
Конфигурирование приглашения	нет	нет	есть	есть	есть	есть
Проверка написания команд	нет	нет	нет	нет	есть	есть
Подстановка процессов	нет	нет	нет	есть(2)	нет	есть
Основной (базовый) синтаксис команд	sh	csh	sh	sh	csh	sh

Продолжение таблицы на следующей странице

Сравнение оболочек (продолжение таблицы 9.2)						
Оболочка свободно распространяется	нет	нет	нет(5)	да	да	да
Имеется проверка почтового ящика	нет	нет	нет	нет	да	да
Возможность работы с большими списками параметров	есть	нет	есть	есть	есть	есть
Имеется не интерактивный инициализационный скрипт	нет	есть	есть(7)	есть(7)	есть	есть
Имеется инициализационный скрипт не только для оболочки, в которую логируется пользователь	нет	есть	есть(7)	есть	есть	есть
Имеется возможность обойти инициализационный пользовательский инициализационный скрипт	нет	да	нет	да	нет	да
Можно определить дополнительный инициализационный скрипт	нет	нет	да	да	нет	нет
Наличие списковых переменных	нет	есть	есть	нет	есть	есть
Полное управление сигналами прерывания	есть	нет	есть	есть	нет	есть
Возможность предупредить пользователя, что он пытается перезаписать существующий файл (noclobber)	нет	есть	есть	есть	есть	есть
Наличие локальных переменных	нет	нет	есть	есть	нет	есть

Замечания к таблице 9.2.

1. Этой возможности не было в исходной версии, но сейчас это почти стандарт.
2. Эта возможность довольно нова, но уже имеется во многих вариантах.
3. Эмуляция стиля редактора **vi** не может рассматриваться как полная.

4. Эта возможность нестандартная, однако имеются заплаты (patch) позволяют ее добавить.
5. версия с именем **pdksh** свободно доступна, но не обеспечивает полной функциональности версии **AT&T**.
6. Это может быть сделано посредством программируемого механизма завершения.
7. Только посредством определения файла через переменную окружения `$ENV`.

Здесь мы коснулись только стандартных оболочек. Однако в качестве оболочек можно использовать такие интерпретаторы как **perl** (смотрите раздел 5.4) и **python** (смотрите раздел 5.6).

9.5 bash

9.5.1 Специальные файлы

Оболочка **bash** использует ряд специальных файлов, которые кратко охарактеризованы в таблице 9.3.

Таблица 9.3: Специальные файлы bash

Имя файла	Назначение
`/etc/profile`	выполняется автоматически во время процедуры *login*, если эта оболочка (bash) указана в файле */etc/passwd*.
`$HOME/.bash_profile`	выполняется автоматически во время процедуры *login*.
`$HOME/.bashrc`	выполняется автоматически во время запуска оболочки.
`$HOME/.bash_logout`	выполняется автоматически во время процедуры *logout*.
`$HOME/.bash_history`	в этом файле содержатся команды, которые были выданы с терминала во время последней сессии.
Продолжение таблицы на следующей странице	

Специальные файлы bash (продолжение таблицы 9.3)	
`/etc/passwd`	из этого файла берутся имена домашних каталогов для аббревиатур типа ~*name*.

9.5.2 Метасимволы в именах файлов

В качестве параметров команд могут использоваться имена файлов. Часто необходимо указать некоторый класс имён, например, `*.c`, т.е. все файлы содержащие исходный текст на языке `C`. Для указания таких классов используются специальные знаки, которые в данном случае именуются как МЕТАСИМВОЛЫ. Основные метасимволы приведены в таблице 9.4.

Таблица 9.4: Метасимволы в именах файлов

Метасимвол	Значение
[*]	(звёздочка) любая строка нулевой или ненулевой длины;
[?]	(вопросительный знак) любой символ;
[abc...]	любой из символов, заключенных в скобки; между символами может быть знак **-** (минус), чтобы обозначить символьный интервал, например, [a-g] символы начиная с `a` до `g`;
[!abc...]	(восклицательный знак и *abc...*) любые символы, которые не эквивалентны тем, что помещены в квадратных скобках;
~name	(знак тильда и слово *name*) каталог пользователя с именем *name*;
~+	(знак тильда и знак плюс) текущий рабочий каталог;
~-	(знак тильда и знак минус) предыдущий рабочий каталог.

9.5.3 Редактирование в командной строке bash

Возможности редактирования в командной строке, которые предоставляет оболочка **bash**, довольно разнообразны и могут удовлетворить различные вкусы и предпочтения пользователей. Автор не уверен, что перечислил все возможные средства редактирования командной строки, которые реализует **bash**, а лишь те что являются, по мнению автора, наиболее используемыми. Кроме того, свойства редактирования могут зависеть также от версии оболочки. Автор пользовался оболочкой **GNU bash, version 1.14.7(1)**.

Возможные комбинации редактирования командной строки смотрите в таблице 9.5.

Таблица 9.5: Редактирование в командной строке

Комбинация	Что означает
CTRL-a	переместить курсор на самый левый символ строки;
CTRL-b	переместить курсор на один символ влево;
ESCc	символ, на который указывает курсор, заменить на тот же, но заглавный, а курсор переместить на первый пробел справа от текущего слова;
DEL	удалить символ, слева от курсора;
CTRL-d	удалить символ, на которой показывает курсор;
CTRL-e	переместить курсор на самый правый символ строки;
CTRL-f	переместить курсор на один символ вправо;
CTRL-k	удалить правую часть строки начиная с символа, на который указывает курсор;
CTRL-l	очистить экран и поместить текущую команду в верхней строке экрана;
ESCl	превратить символы начиная с символа, на который указывает курсор, до самого правого символа в данном слове в прописные буквы, а курсор установить на пробел справа от слова;
CTRL-m	то же что **RETURN**;
CTRL-o	то же что **RETURN**, затем отображается очередная команда из файла история;
CTRL-t	поменять местами два символа: символ, на который показывает курсор, и символ слева от курсора, затем, курсор переместить на один символ вправо;
ESCt	поменять местами два слова: слово, на которое указывает курсор и слово слева от первого;
CTRL-u	удалить левую часть строки включая символ, который находится слева от курсора;
ESC-u	сделать символы данного слова заглавными, начиная с символа, на который указывает курсор, до самого правого символа в слове, а курсор установить на пробел справа от слова;

Продолжение таблицы на следующей странице

Редактирование в командной строке (продолжение таблицы 9.5)	
CTRL-y	вернуть часть строки удалённую ранее с помощью **CTRL/u**: эта часть строки вставляется в текущую строку справа от курсора;
ESC.	вставить последнее слово из предыдущей команды перед символом, на который указывает курсор;
CTRL-[	то же, что **ESC**;
ESC_	то же, что **ESC.**

9.5.4 Специальное редактирование в bash

В оболочке bash имеются специальные комбинации управляющих символов, которые помогают сократить как время на печатание полных имён команд, файлов, переменных окружения, так и время на поиск нужных имён, которые могут оказаться совсем не короткими. Эти средства позволяют bash "догадаться"какое имя файла или какую команду вы имеете в виду и подставляет соответствующие имена в ответ на ввод описываемых ниже кодовых комбинаций. Описания этих команд приведены в таблице 9.6

Таблица 9.6: Специальное редактирование

Команда	Значение
TAB	оболочка пытается выполнить общий алгоритм подстановки текста в командной строке. Если вы попробуете ввести в пустой командной строке этот символ (**TAB**), то после первого ввода вы получите звуковой сигнал, а после второго получите примерно следующее сообщение: "There are 2179 possibilities. Do you really wish to see them all? (y or n)". Если вы ввели несколько символов команды, а после этого была нажата клавиша **TAB**, то вы получите значительно менее длинный список возможных команд. Если команда введена, то вы получите список имён файлов.
Продолжение таблицы на сдедующей странице	

Специальное редактирование (продолжение таблицы 9.6)	
ESC?	напечатать список возможных завершений текста;
ESC/	попытаться завершить имя файла;
CTRL-x/	напечатать список возможных завершений имени файла;
ESC~	попытаться завершить имя пользователя;
CTRL-x~	напечатать список возможных завершений имени пользователя;
ESC$	попытаться завершить имя переменной окружения;
CTRL-X$	список возможных завершений для имени переменной окружения.
ESC@	попытка возможного завершения для сетевого имени машины;
CTRL-X@	напечатать список возможных сетевых имен машин;
ESC!	попытка завершения имени команды;
CTRL-x!	список возможных завершений имени команды;
ESC TAB	попытка завершения имени команды из прежних команд, выданных в сессии.

9.5.5 Выполнение команд bash

Выполнение команд в оболочке **bash** производится последовательно, по мере ввода команд. Последовательные команды могут группироваться и можно сделать так, чтобы последовательность выполнения команд зависела от результатов выполнения. Как команды группируются показано в таблице 9.7.

Таблица 9.7: Выполнение команд bash

Команда	Значение
cmd	выполнение команды **cmd**;
cmd &	выполнение команды **cmd** в фоне;
cmd; cmd1	последовательное выполнение двух команд **cmd**, затем **cmd1**;
(cmd; cmd1)	запускается дополнительная bash-оболочка для выполнения двух команд как командной группы;
cmd | cmd1	трубопровод или программный канал: стандартный вывод команды **cmd** используется в качестве стандартного ввода в команде **cmd1**.

Продолжение таблицы на следующей странице

Выполнение команд bash (продолжение таблицы 9.7)	
cmd ˆcmd1ˆ	подстановка команд: результат выполнения команды **cmd1** используется в качестве параметра в команде **cmd**.
cmd $cmd1	командная подстановка: результат команды *cmd1* используется в качестве параметра к команде *cmd*.
cmd && cmd1	командный AND (И): команда **cmd1** выполняется только в том случае, если **cmd** выполнена успешно.
`cmd || cmd1`	командный OR (ИЛИ): команда **cmd1** выполняется только в том случае, если **cmd** выполнена НЕ успешно.
{ cmd; cmd1 }	выполнить последовательность команд в текущей оболочке. Следует заметить, что между фигурными скобками и командами должны быть пробелы, иначе оболочка даст информацию об ошибке.

9.5.6 Встроенные команды bash

Таблица 9.8: Встроенные команды **bash**

Встроенные команды bash	
Команда	Описание
Команда	Значение
#	Игнорировать текст, который следует за символом решетка. Используется для комментариев. Исключение составляет случай, когда в первой строке за знаком решетка следует восклицательный знак (!).
`#!`	используется для указания оболочки, которая будет интерпретировать данный скрипт. Например: #!/usr/local/bin/tcsh
:	пустая команда. Возвращает код завершения 0 (нуль). Иногда используется в качестве первой команды в скрипт, чтобы обозначить bash скрипт. Переменные окружения могут быть помещены после двоеточия, чтобы подставить их значения.
Продолжение на следующей странице	

Встроенные команды bash (продолжение)

Команда	Описание
. **. file [arguments]**	то же что команда **source**.
alias [*name*[=’*cmd*’]]	обозначить *name*, как синоним для *cmd*. Если *cmd* опущено, то вывести синоним для *name*; если *name* опущено, то напечатать все синонимы. Если за *cmd* следует пробел, то проверяется очередной аргумент на предмет другого синонима.
bg [*jobIDs*]	поместить данное задание в фоновый режим.
bind [*options*] **bind** [*options*] *keys:function*	напечатать или установить связь ключей и функций. Возможными значениями *options* могут быть следующие: SSname-m - keymap; SSname-l - напечатать все функции чтения строки; SSname-v - напечатать имена функций и связей; SSname-d - напечатать имена функций и связей, подходящие для повторного чтения; SSname-f PSnamefilename - произвести связывание в соответствии с файлом; SSname-q *function* - отобразить связывание, которое вызывает *function*.
break [*n*]	выйти из самого внутреннего цикла типа: **for**, **while**, **until** или, если указан параметр *n*, выйти из цикла, который находится выше текущего на PSnamen-1. Кроме этого, команда используется для выхода из **select**.
built-in *command [arguments]*	Выполнить команду, которая встроена в оболочку, а не другую программу с тем же именем. Полезно, когда скрипт имеет то же имя, что и встроенная команда.

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание
case **case** *string* **in** regex) commands ;; ... **esac**	Если *string* удовлетворяет регулярному выражению *regex*, то выполнить команды *commands*
cd [*dir*]	без аргументов переходит в домашний (основной) каталог пользователя. С аргументом - меняет текущий рабочий каталог на *dir*.
command *[options] command [arguments]*	Выполнить команду *command*, игнорируя функции определённые в оболочке с помощью конструкции **function** *name* {*commands*}. Команда может быть либо встроенной, либо находиться в каталоге **$PATH**. Код завершения устанавливается равным коду, возвращаемым командой *command* или 127 в случае, если *command* не найдена. Возможными значениями *options* может быть следующее: -p - произвести поиск команды *command* в каталогах по умолчанию используя значение $PATH. -v - только напечатать каталог, где находится команда не выполняя саму команду. -V только сообщить откуда оболочка берет данную команду: из каталога или из внутренней таблицы оболочки, не выполняя самой команды.

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание
continue [*n*]	Перейти к следующей итерации в циклах типа **for**, **while**, **until**. Если присутствует числовой параметр, то происходит переход к началу *n-1*-ого охватывающего цикла.
declare [*options*] [*name*[=*value*]] **typeset** [*options*] [*name*[=*value*]]	Объявляет переменные окружения и/или присваивает им значения и/или атрибуты. Если не дано никакого имени *name*, то печатаются имена и значения всех переменных окружения. Возможные значения *options*: `-f` - использовать только имена функций; `-r` - присвоить атрибут **только чтение**; `-x` - отметить имена для последующего экспорта; `-i` - присвоить переменной атрибут **арифметический**. Использование знака + (плюс) вместо знака **-** (минус) выключает присвоенные атрибуты. Когда используется в функции, то делает объявленные переменные локальными внутри функции. Код завершения есть 0, если не предпринята попытка выполнить нелегальные операции: попытка напечатать имя несуществующей функции, попытка сменить статус переменной объявленной с атрибутом **только чтение**.
dirs [*options*]	Напечатать имя каталога сохранённый ранее для операций **pushd/popd**, иными словами, стек имен каталогов. Возможными значениями *options* могут быть: `+n` - напечатать **n**-ное имя (начиная с нуля); `-n` - напечатать **n**-ное имя начиная с конца; `l` - напечатать всю информацию.

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание
echo [*options*][*string*]	вывести строку на стандартное устройство вывода. Вывод заканчивается символом `<NL>`. Если строки нет, то выводится только `<NL>`. Возможные значения *options*: **-n** - не выводить **New Line** в конце строки; **-e** - разрешить интерпретацию последовательностей *escape*. В этом случае полезно строку *string* помещать в кавычки ("). Список *escape* последовательностей: `\a` - звуковой сигнал; `\b` - возврат каретки на один символ назад (**BackSpace**); `\c` - запретить вывод **New Line** в конце строки (то же что параметр **-n**); `\f` - смена формы (**Form Feed**); `\n` - новая строка **New Line**; `\r` - возврат каретки **CR**; `\t` - горизонтальная табуляция; `\v` - вертикальная табуляция; `\\` - обратная косая черта; `\nnn` - символ **ASCII**, восьмеричный код которого равен *nnn*.
enable [*options*][*built-in-command*]	разрешить (*options* = **-a**) или запретить (*options* = **-n**) интерпретацию встроенной в оболочку команды *built-in command*. Без параметров - будет напечатан список внутренних команд оболочки разрешенных для интерпретации.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
	Если *options* = **-a** и параметр *built-in command* опущен, то будет напечатан весь список внутренних команд оболочки вместе с информацией о состоянии (*enabled* или *disabled*). Команда **enable** удобна для определения в скрипте своих собственных функций с теми же именами, которые имеют внутренние команды оболочки **bash**.
eval [*command args...*]	выполнить команду передав ей аргументы. Код завершения команды **eval** устанавливается равным коду завершения команды **command**. Если команда не найдена, то код завершения равен 127.
exec [[-] *command* [*arguments*]]	выполнить команду *command* на месте текущего процесса (вместо создания нового процесса как обычно происходит). Иными словами происходит замещение текущей оболочки. Если первый аргумент есть **-** (знак минус), то оболочка помещает этот знак в нулевой параметр, который передаётся команде, также как это делает процедура *login*.
Продолжение на следующей странице	

Встроенные команды bash (продолжение)

Команда	Описание

	Если файл не может быть выполнен по любым причинам, то неинтерактивный вариант оболочки завершается (т.е. если **exec** находится в скрипте). Но если существует переменная no_exit_on_failed_exec, то будет выдана диагностика, сформирован ненулевой код завершения, а выполнение скрипта будет продолжено. Если *command* опущена, то любые перенаправления ввода/вывода будут произведены в текущей оболочке, код возврата будет 0. Когда команда *command* будет завершена, то закроется соответствующее окно.
	Примеры.
	Заменить оболочкой *bash* текущую оболочку:
	`exec /bin/bash`
	Переназначить стандартный ввод на файл `input_file`
	`exec<input_file`

exit [*n*]	выйти из скрипта с кодом завершения *n*. Если параметр *n* опущен, то код завершения скрипта будет установлен равным коду завершения последней команды внутри скрипта.
export [*options*] [*Variable-Names*[=*value*]]	экспортировать переменные окружения, т.е. сделать их глобальными. Возможные значения *options*: **-** - последующую информацию воспринимать как переменные и их значения, а не как параметры; **-f** - рассматривать имя глобальной переменной как имя функции; **-n** - отменить экспорт конкретной переменной;

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание
	-p - напечатать список переменных экспортированных данной оболочкой.
fc [*options*][*first*][*last*] **fc -s** [*old=new*][*command*]	отобразить или отредактировать команды в сохранённом списке команд, выданных в данной сессии. Возможные значения *options*: **-e** - следующий параметр есть имя редактора для редактирования строки: по умолчанию имя редактора содержится в переменной окружения FCEDIT; если переменная не определена, то используется редактор **vi**; **-l** - показать сохранённый список команд; **-n** - не показывать номера команд в отображаемом списке; **-r** - показать сохранённый список команд в обратном порядке; **-s** *old=new* - заменить подстроку *old* на подстроку *new* в команде *command* и выполнить команду. Смотрите также примеры **??**.
fg [*jobIDs*]	перевести текущее задание или задание с номером *jobID* из режима фоновой задачи в режим задачи переднего плана.
for *x* [**in** *list*] **do** *commands* **done**	присвоить последовательно каждое слово из списка *list* переменной *x* и выполнить команды *commands* для каждого значения *x*. Если список *list* опущен, то предполагается использование списка позиционных параметров `$@`.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
function *name*(){*list*}	определяет функцию с именем *name*. Последовательность команд оболочки *list*, которая составляет тело функции, может использовать значения позиционных параметров функции также как и в обычном скрипте ($1, $2, etc.). Тело функции ограничивается: в начале - фигурной открывающей скобкой **{** и закрывающей фигурной скобкой **}** в конце.
getopts *optstring* *name* [*args*]	используется процедурами оболочки, чтобы разделить позиционные параметры и аргументы. Обычно применение - внутри скрипта для анализа параметров скрипта. Строка *optstring* содержит распознаваемые параметры, обозначаемые одним символом. Если за символом следует двоеточие, то данный параметр требует аргумент, который должен быть отделен от параметра одним пробелом. Например, если *optstring='ax:b'*, то возможно использование трёх параметров: **a**, **x** и **b**, причём параметр **x** требует аргумент. При каждом вызове **getopts** помещает значение параметра в переменную окружения с именем *name*, инициализируя переменную, если её нет, а индекс следующего аргумента будет помещен в переменную окружения $OPTIND. $OPTIND устанавливается в 1 каждый раз, когда вызывается оболочка или скрипт. Когда параметр требует аргумент, SSnamegetopts помещает этот аргумент в переменную окружения $OPTARG.
Продолжение на следующей странице	

Встроенные команды bash (продолжение)	
Команда	Описание
	Оболочка не переустанавливает переменную `$OPTIND` автоматически. Иными словами, если имеет место многократный вызов **getopts** в одном скрипте, то необходимо побеспокоиться об установке `$OPTIND` в начальное значение.
hash [-r] [*commands*]	произвести поиск команд, указанных в аргументе и запомнить оглавление, в котором находится каждая команда. Хеширование приводит к тому, что оболочка запоминает связь между именем команды, указанной в аргументе, и абсолютным именем каталога, в котором содержится данное имя. Таким образом, будущее использование команды уже не будет требовать никакого поиска по каталогам. Без аргументов, команда **hash** печатает на экране список команд, уже содержащихся во внутренней таблице оболочки. При печати указывается, сколько раз использовалась каждая команда из списка. Если указан параметр **-r**, то удаляется соответствующая запись из таблицы или все записи (если опущен параметр *commands*).
help [*string*]	печатает информацию о встроенных командах оболочки, имя которых начинается на *string*. Например, команда **help h** выдаст на экран следующее:

```
hash: hash [-r] [name ...]
help: help [pattern ...]
history: history [n] [ [-awrn] [filename]]
```

	При этом каждая команда сопровождается дополнительными иногда обширными пояснениями, которые здесь опущены.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
history [*lines*] [*options*]	печатает нумерованный список команд, выданных в текущей и/или прошлой сессий. Если указан параметр *lines*, то отображаются последние *lines* строк. Возможные значения *options*: • **-a** оболочка **bash** поддерживает файл с именем `.bash_history` в домашнем пользовательском каталоге. Данный параметр заставляет оболочку добавить в файл `.bash_history` список команд, выданных в текущей сессии до команды **history**. • **-n** добавить в список команд те, что еще не были прочитаны из файла `.bash_history`. • **-r** использовать файл `.bash_history` как список выданных команд вместо текущего рабочего списка. • **-w** записать в файл `.bash_history` текущий список выданных команд (замещая предыдущее содержание файла).
Продолжение на следующей странице	

Встроенные команды bash (продолжение)

Команда	Описание
if *test-cmd*	начинает условный оператор. Команда имеет следующие форматы. Простейший формат: **if** *test-cmd* **then** *command* **fi** Формат с оператором **else**: **if** *test-cmd* **then** *command1* **else** *command2* **fi** Формат с оператором **elif**: **if** *test-cmd* **then** *command1* **elif** *test-cmd2* **then** *command2* **else** *commmand3* **fi**

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание
jobs [*options*][*jobIDs*] **jobs -x** *command* [*args*]	печатает список всех заданий (выполняющихся или остановленных) или заданий, которые определяются параметром *jobIDs*. Возможные значения *options*: - **-l** перечисляет ID процессов в дополнение к обычной информации; - **-p** перечисляет только ID процессов руководителя группы; - **-n** перечисляет только задания с момента последнего уведомления; Если указан параметр **-x**, команда **jobs** замещает любой идентификатор задания, найденный в аргументах *command*, или *args*, соответствующим групповым идентификатором процесса и выполняет команду *command*, передавая ей аргументы *args*.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
kill [*options*] *IDs*	оборвать все процессы специфицированные в *ID*. Вы должны быть владельцем процесса или привилегированным пользователем. Возможные значения *options*: • **-signal** номер сигнала или имя сигнала (можно получить с помощью команды **kill -l**; • `-` означает, что дальнейшая информация является аргументами, а не параметрами; • **-l** перечислить имена сигналов; • -s *signal* определить сигнал, может быть имя сигнала.
let*expressions*	выполнить арифметические операции над целыми выражениями. Выражение *expressions* состоит из целых чисел, операторов и имен переменных окружения, которые не нуждаются в предшествующем знаке $. Выражения должны быть заключены в кавычки, если они содержат пробелы или другие специальные символы. Примеры: let k=k+1 let "k = k + 1" В обоих примерах переменная **k** увеличивается на 1.
logout	Выход из оболочки. Может быть использована, если планируется выход из оболочки, в которую производилось логирование. В противном случае следует использовать команду **exit**.

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание
popd [*options*]	управление стеком каталогов оболочки. По умолчанию, удаляется самый верхний каталог и производится переход в него. Возможные значения *options*: **+n** - удалить n-ый каталог из стека, начиная счет с вершины стека с 0. **-n** - удалить n-ый каталог из стека, начиная счет снизу стека с 0.
pushd *directory* **pushd** [*options*]	в первом варианте команда **pushd** добавляет каталог в вершину стека каталогов и делает его текущим каталогом. Во втором варианте команды **pushd**, если параметры опущены, то меняются местами два верхних каталога стека каталогов. Если задано +**n**, то **n**-ый каталог от вершины из стека каталогов становится самым верхним. Если задано **-n**, то верхним каталогом становится **n**-ый каталог от дна стека каталогов.
pwd	напечатать абсолютное имя текущего каталога. Если командой **set** установлен параметр **-p**, то команда **pwd** не будет печатать имя символической ссылки.
read [*options*] *var1* [*var2* ...]	читается одна строка из стандартного ввода. Каждое слово из введенной строки присваивается переменной *var1*, *var2*, ... Если переменные исчерпаны, то остаток строки присваивается последней переменной. Если переменная одна, то ей присваивается вся строка. Если переменных нет, то строка присваивается переменной окружения $REPLY. Возможные значения *options*: **-r** - игнорировать обратный слеш как символ продолжения строки.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
readonly [*options*] *var1* [*var2* ...]	предотвратить изменение значений специфицированных переменных (присвоить атрибут **только чтение**). Возможные значения *options*: • `-` рассматривать последующие данные как аргумент, а не параметр; • `-f [name(s)]` обозначает функцию; • `-p` напечатать все имена переменных с атрибутом **только чтение**.
return [*n*]	используется внутри определения функции. Выйти из функции с кодом завершения *n*. Если *n* опущено, то код завершения равен коду завершения команды, выполненной перед **return**.
select *name* [**in** *wordlist*;] **do** *commands* **done**	выбрать значение для имени *name* среди значений в *wordlist*. При выполнении команды на экране печатаются нумерованный список значений из *wordlist*. Когда оператор вводит номер значения, то выполняется последовательность команд *commands*, затем снова производится выбор нового значения, если только последовательность *commands* не содержит команд **break** или **return** или **exit**.
Продолжение на следующей странице	

Встроенные команды bash (продолжение)

Команда	Описание
set [*options*] [*arg1 arg2 ...*]	без параметров команда *set* печатает значения всех переменных окружения, известных в текущей оболочке. Параметры могут быть разрешены (-*options*) или запрещены (+*options*). Параметры могут быть установлены при вызове оболочки, т.е. при вызове оболочки **bash**. Аргументы присваиваются в естественном порядке: первый аргумент присваивается $1, второй - $2 и т.д. Возможные значения *options*: **-** (знак минус) выключить **-v** и **-x**; остальную информацию в строке рассматривать как аргументы; – используется как признак конца параметров; оставшуюся часть строки рассматривать как аргументы; **-a** отметить переменные, которые будут автоматически экспортированы после присваивания им значений; **-b** печатать состояние фонового задания в момент окончания, не ожидая очередного приглашения оболочки; **-e** немедленно завершить выполнение оболочки, если любая команда закончилась с ненулевым кодом завершения; **-f** запретить генерацию расширения имен файлов (установить режим noglobbing); **-h** определить расположение и запомнить функциональные команды сразу, как только определены функции;

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
	-k присваивание параметров переменным окружения будут происходить вне зависимости от их местоположения в командной строке; обычно аргументы должны предшествовать имени команды; **-l** когда команда **for** использует переменную окружения, которая уже используется в данном скрипте, то после выхода из цикла **for** переменная будет иметь то значение, которое она имела до входа в цикл **for**. В противном случае (если не установлено **set -l**) по выходе из цикла переменная, использованная в цикле **for**, будет иметь значение последней величины присвоенной в цикле; **-m** установить режим мониторирования. Разрешает управление заданиями, фоновые задания выполняются в отдельной группе процессов. Обычно этот режим устанавливается автоматически; **-n** оболочка читает команды, проверяет их, но не исполняет команды: удобно для проверки скриптов (команда, естественно, должна быть внутри скрипта); **-o** напечатать установленные режимы. Многие режимы могут быть установлены различными способами (не только командой **set**). Названия режимов: **monitor** то же, что **set -m** **noclobber** то же, что **set -C** **noexec** то же, что **set -n** **noglob** то же, что **set -f** **notify** то же, что **set -b** **nounset** то же, что **set -u**

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
	physical то же, что **set -P** **posix** то же, что УДОВЛЕТВОРЯЕТ СТАНДАРТУ POSIX **privileged** то же, что **set -p** **verbose** то же, что **set -v** **vi** то же, что ИСПОЛЬЗОВАТЬ СТИЛЬ РЕДАКТИРОВАНИЯ VI **xtrace** то же, что **set -x** **-p** начать как привилегированный пользователь: не обрабатывать `$HOME/.profile`; **-t** выйти как только будет выполнена одна команда; **-u** при подстановках рассматривать использование неинициализированных переменных (переменные, которым не присвоено никаких значений) как ошибку; **-v** показать каждую команду скрипта, когда она прочитывается; **-x** показать команды и аргументы в тот момент, когда они выполняются; **-C** то же самое, что **noclobber**; **-H** это умолчание; разрешить команды `!` и `!!`; **-P** напечатать абсолютное имя текущего каталога в ответ на команду **pwd**.
source *file* [*arguments*]	читать и выполнить строки из файла *file*. При этом файл *file* не обязан быть исполняемым, но должен находиться в группе каталогов, где производится поиск исполняемых модулей (переменная $PATH).

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
suspend [-f]	то же самое, что **CTRL-Z**. Часто используется, чтобы приостановить команду **su**. Параметр **-f** заставляет приостановить работу оболочку, даже если это оболочка, которая начала работать сразу после процесса **login**.
test *condition*	вычислить условие и, если условие верно, вернуть нулевой код завершения. В противном случае, вернуть ненулевой код завершения. Альтернативная форма определения условия заключается в использовании квадратных скобок вместо слова **test**. Выражения, которые могут использоваться для вычисления условий, перечислены ниже. **Проверка состояния файла(ов)** **-b** *file* - *file* существует и является специальным блоковым файлом для организации ввода/вывода на внешних устройствах, на которых обмен производится блоками; **-c** *file* - *file* существует и является специальным символьным файлом для организации ввода/вывода на внешних устройствах с сивольным обменом; **-g** *file* - *file* существует и имеет установленный бит **set-group-id**; **-k** *file* - *file* существует и имеет установленный бит **sticky**; **-p** *file* - *file* существует является именованным программным каналом (named pipe) с доступом FIFO;
Продолжение на следующей странице	

Встроенные команды bash (продолжение)

Команда	Описание
	-r *file* - *file* существует и разрешен для чтения; **-s** *file* - *file* существует и имеет размер в байтах больше нуля; **-t** *n* - открытый (командой open) описатель файла *n* связан с терминальным устройством; по умолчанию, $n=1$; **-u** *file* - *file* существует и имеет установленный бит **set-user-id**; **-w** *file* - файл с именем *file* существует и разрешена запись в файл; **-x** *file* - файл с именем *file* существует и является исполняемым файлом; **-G** *file* - *file* существует и его группа является эффективной группой процесса; **-L** *file* - файл с именем *file* существует и является символической ссылкой; **-O** *file* - файл с именем *file* существует и принадлежит вам; **-S** *file* - файл с именем *file* существует и является сокетом; *fA* **-ef** *fB* - файлы с именами *fA* и *fB* существуют и связаны между собой (другими словами, это один и тот же файл); *fA* **-nt** *fB* - файл с именем *fA* новее, чем файл с именем *fB*; *fA* **-ot** *fB* - файл с именем *fA* старее, чем файл с именем *fB*. **Сравнения строк** **-n** sA - строка sA имеет НЕ нулевую длину; **-z** sA - строка sA имеет нулевую длину; $sA=sB$ - строка sA равна строке sB; $sA!=sB$ - строка sA НЕ равна строке sB;

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
	string - строка *string* не пуста. **Сравнения целых чисел** *n1* **-eq** *n2* - *n1* равно *n2*; *n1* **-qe** *n2* - *n1* больше или равно *n2*; *n1* **-gt** *n2* - *n1* больше *n2*; *n1* **-le** *n2* - *n1* меньше или равно *n2*; *n1* **-lt** *n2* - *n1* меньше *n2*; *n1* **-ne** *n2* - *n1* НЕ равно *n2*. **Логические комбинации условий** !*condition* - верно, если условие *condition* ложно; *C1* **-a** *C2* - верно, если условия *C1* и *C2* верны одновременно (логический "И"); *C1* **-a** *C2* - верно, если хотя бы одно условий *C1* или *C2* верно (логический "ИЛИ"); **Замечание.** При построении логических выражений могут использоваться более, чем два условия.
times	напечатать время процессора, потребленное пользователем и системой в данной сессии.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
trap [-l] [*commands*] [*signals*]	выполнить команды *commands*, если получены любые сигналы *signals*. Общие сигналы включают сигналы с номерами 0, 1, 2 и 15. Несколько команд на месте *commands* должны быть объединены в группу и разделены точкой с запятой. Если *commands* представляет собой пустую строку, например, (**trap** *signals*), тогда сигналы *signals* будут игнорироваться данной оболочкой. Если команды *commands* полностью опущены, то обработка сигналов *signals* будет возвращена к стандартному значению. Если все параметры *commands* и *signals* опущены, то будут выведены текущие установки. Если указан параметр **-l**, то будет отпечатан полный список сигналов. **Какие бывают сигналы** **0** - выход из оболочки (обычно когда оболочка завершается); **1** - hangup (обычно logout); **2** - прерывание (обычно CTRL-c); **3** - Quit; **4** - Неверная машинная команда (illegal instruction); **5** - трассировка (trace trap); **6** - аварийное завершение (abort); **7** - не используется; **8** - исключительная ситуация при операциях с плавающей точкой (floating point exeption); **9** - завершение (termination);
Продолжение на следующей странице	

Встроенные команды bash (продолжение)

Команда	Описание
	10 - определяется пользователем; **11** - неправильная адресация памяти; **12** - определяется пользователем; **13** - запись в программный канал, который никто не читает; **14** - аварийный сигнал по timeout; **15** - завершение программы (обычно по команде kill); **16** - ошибка в стеке сопроцессора (coprocessor stack fault); **17** - завершение процесса-наследника (termination of child process); **18** - продолжение после остановки; **19** - остановить процесс (stop process); **20** - остановка вывода на терминал; **21** - фоновый процесс имеет ввод с терминала; **22** - фоновый процесс имеет вывод на терминал; **23** - ошибка ввода-вывода; **24** - временной лимит процессора превышен; **25** - лимит размера файла превышен; **26** **27** - Profile; **28** - изменение размера окна. **Примеры** **trap 2** - игнорировать сигнал 2 (прерывание); **trap 2** - восстановить стандартную реакцию на сигнал 2; **trap "rm -f tmp; exit"0 1 2** - удалить файл `tmp` в случае поступления сигналов 0, 1 или 2.

Продолжение на следующей странице

Встроенные команды bash (продолжение)

Команда	Описание

type [*options*] *commands*	напечатать тип ключевых слов: является ли ключевое слово именем программы (скрипта), или ключевым словом, используемым оболочкой, или встроенной командой оболочки, или это неопознанная комбинация символов.
	- - рассматривать все последующее как аргументы, а не как параметры;
	-a - вывести все варианты команд *commands*, а не только тот, который будет вызываться;
	-p - напечатать значения команд *commands*, которые находятся во внутреннем кэше оболочки;
	-t - определить состояние команд *commands*: являются ли они псевдонимами, ключевыми словами, встроенной функцией или файлом.
	Пример
	Здесь мы хотим узнать какой тип имеют команды **trap**, **while**, **LoadFarm** и **htpl**.
	Мы сделаем это командой:
	type trap while LoadFarm htpl
	В ответ, оболочка **bash** напечатала:
	`trap is a shell builtin`
	`while is a shell keyword`
	`LoadFarm is /home/shevel/bin/LoadFarm`
	`type: htpl: not found`
	Как видим, оболочка сообщила, что **trap** есть встроенная команда оболочки, **while** есть ключевое слово оболочки, а **LoadFarm** - это программа или скрипт, а htpl - не найдено.

typeset	смотрите описание команды **declare** на странице 83.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
ulimit [*options*] [*n*]	если опущено *n*, то напечатать одно или более ограничений на используемые ресурсы (если ресурс не имеет явных ограничений, то это не значит, что он может быть любым). Если определен параметр *n*, то присвоить соответствующему ограничению на ресурс значение *n*. Ресурсы могут быть жесткими (**-H**) и мягкими (**-S**). По умолчанию, **ulimit** устанавливает оба лимита, а печатает мягкий лимит. Параметры определяют какие лимиты используются: Значения параметров: - - далее следуют аргументы, а не параметры; **-a** - напечатать все текущие лимиты; **-H** - напечатать жесткие лимиты; **-S** - напечатать мягкие лимиты; **-c** - core файлы; **-d** - размер сегмента данных процесса; **-f** - размер файлов образованных оболочкой; **-m** - размер резидентного набора; **-n** - количество описателей файлов; **-P** - размер программного канала; **-s** - размер стека; **-t** - количество времени CPU в секундах; **-u** - количество процессов на пользователя; **-v** - виртуальная память используемая оболочкой.

Продолжение на следующей странице

Встроенные команды bash (продолжение)	
Команда	Описание
unset [-f] [-v] [*name...*]	Для каждого имени *name* удалить соответствующую переменную или функцию. Если задан параметр *-v*, то команда **unset** будет влиять только на переменные (а не функции). Если указан параметр **-f**, то **unset** будет действовать лишь на функции (а не переменные). Если параметры (флаги) не указаны, то **unset** сначала пытается воздействовать на переменные, а уже потом на функции. Некоторые переменные, например, $PATH или $IFS не могут быть удалены (см. атрибут **readonly**).
wait [*ID*]	запрашивается пауза в выполнении до тех пор, пока определённый процесс с номером *ID* или задание с номером *ID* не будут завершены. Если параметр опущен, то ожидается завершение выполнения всех фоновых заданий. Переменная окружения **$!** содержит номер фонового процесса, который был запущен последним. Если не включена возможность управления заданиями, то параметр *ID* может обозначать только номер процесса. Пример: **StartPar &** **wait $!** ожидать завершения фонового процесса, который выполняет скрипт **StartPar**.
while *condition*; **do** *commands*; **done**	Сделать все подстановки в последовательности команд *commands* и повторять выполнение этой последовательности до тех пор, пока условие *condition* не примет значение FALSE (неверно).

9.6 zsh

Оболочка **zsh** представляет собой командный интерпретатор. По своим свойствам она ближе всего к **Korn** оболочке **ksh**. С момента появления **zsh** ее совместимость с **ksh** значительно улучшилась. Улучшились свойства редактирования командной строки, возможности определения поведения оболочки, особенности, позволяющие пользователям, знакомым с языком **C** и оболочкой **csh**, чувствовать себя уверенней, а также часть полезных возможностей, взятых из **tcsh** (дополнительная "пользовательская"оболочка).

Трудно утверждать, что лишь **zsh** аккумулирует полезные свойства всех остальных оболочек, поскольку все они в той или иной степени наследуют полезные свойства более ранних оболочек. Тем не менее, можно обратить внимание на следующие позиции, по которым **zsh** имеет некоторое преимущества по сравнению с другими оболочками: **csh**, **bash**, **tcsh**.

- редактирование командной строки:
  - программируемое завершение: встроенная возможность использовать всю мощность свойства глобализации имен файлов (compctl–g),
  - редактирование команд, состоящих из нескольких строк, как единый буфер (даже файлы!),
  - редактирование переменных (vared),
  - стек команд,
  - печать текста прямо в буфер для последующего редактирования print -z),
  - выполнение не связанных (unbound) команд,
  - меню завершения,
  - расширение внутри строки переменных, команд из истории.
- глобализация имен файлов (globbing) – исключительно мощная:
  - рекурсивная глобализация (сравни с программой **find**),
  - задание параметров файла (размер, тип, прочее, снова сравни с программой **find**) ,
  - полные альтернативы и отрицания образцов имен файлов.

- Управление множественными перенаправлениями (проще, тем известная команда **tee**) .
- Большое количество возможностей подстройки.
- Расширение пути поиска программы (`=foo -> /usr/bin/foo`).
- Подстраиваемые сообщения (включая условные выражения).
- Именованные каталоги.
- Гибкая целочисленная арифметика.
- Манипулирование массивами (включая обратное индексирование).
- Коррекция неверно написанных слов.

9.7 tcsh, csh

tcsh - это улучшенный и продвинутый вариант **csh**, которая является одной из старых оболочек, включенной почти во все варианты **Unix**/**Linux**.

9.8 esh - простая оболочка

Есть оказываются разработчики, которые продолжают предлагать новые варианты оболочек. Среди относительно новых предложений - **esh**. К основным отличиям этой оболочки относится то, что синтаксис команд в значительной степени следует традициям **Lisp**. Это позволяет быть оболочке весьма малой по размеру занимаемой памяти, но иметь при этом больше возможностей языка программирования в сравнении с традиционными оболочками. Эта оболочка может быть найдена в *http://esh.netpedia.net/*.

Глава 10

Программы преобразования и анализа текста

10.1 Текст

Под текстом понимается последовательность символов (букв), которые могут перемежаться управляющими символами: символ табуляции `<TAB>`, символ перехода на новую строку `<LF>` и т.п. Обычно, текст разделяется на параграфы, фразы (предложения) и слова. Символом отделяющим одно слово от другого чаще является ПРОБЕЛ. Как правило, одна фраза отделяется от другой точкой. По умолчанию, абзацы отделяются один от другого пустой строкой.

Текст в файле представляет собой последовательность из одной или многих записей. Не обязательно, что разделение текста на слова, абзацы и фразы каким-то образом связаны с разбиением текстового файла на записи. Однако, на практике в качестве символов отделяющих одно слово от другого используется ПРОБЕЛ и символ конца записи `<LF>`.

Нетрудно догадаться, что в основе большинства видов человеческой деятельности с использованием компьютера лежит работа с различными видами текста: текстами руководств, описаний, книг, исходными текстами программ, конфигурационными файлами, и т.д.

Текст может быть введён или отредактирован с терминала любым текстовым редактором. Текст может быть записан в виде обычного файла на диск или передан по компьютерной сети. Текст может быть проанализирован или преобразован любой программой, предназначенной для преобразования или анализа текста.

10.2 Поиск по шаблону и регулярные выражения

Целый ряд программ обработки текста в **Linux** позволяют вам производить поиск, а в некоторых случаях и замену текста по шаблону, построенному с использованием специальных правил. К таким программам относятся редакторы текстов: **ex, vi, ed, sed**; язык **awk**, а также команды семейства **grep** и многие другие программы интерпретаторы. Текстовые шаблоны (patterns), которые часто упоминают как регулярные выражения (regular expressions), содержат обычные текстовые символы вместе со специальными символами (часто их называют МЕТАСИМВОЛЫ).

Толковый словарь **Collins Cobuild English Language Dictionary** даёт следующее толкование слова REGULAR: простое, обычное, повторяющееся без изменения, удовлетворяющее простым правилам. Представляется, что такое толкование исчерпывающим образом описывает смысл термина REGULAR EXPRESSIONS – РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ.

Данная глава содержит следующую информацию:

- Список метасимволов
- Описание метасимволов
- Имена файлов и шаблоны
- Примеры

10.2.1 Список метасимволов

Некоторые метасимволы могут иметь одно значение в одних программах и совершенно иное - в других. Те метасимволы, которые используются в конкретной программе помечены звездочкой в нижеприведенной таблице.

Таблица 10.1: Список метасимволов

Символ	**ed**	**ex**	**vi**	**sed**	**awk**	**grep**	**egrep**	**Смысл**
. (точка)	*	*	*	*	*	*	*	Удовлетворяет любому символу.
* (звездочка)	*	*	*	*	*	*	*	Удовлетворяет нулевому или большему числу появлений.

Продолжение таблицы на следующей странице

Список метасимволов (продолжение таблицы 10.1)								
`^`(крышка)	*	*	*	*	*	*	*	Должно быть в начале строки.
`$` (доллар)	*	*	*	*	*	*	*	Должно быть в конце строки.
`\` (левый слеш)	*	*	*	*	*	*	*	Далее идет escape.
`[ ]`	*	*	*	*	*	*	*	Один из набора.
`\(\)`	*	*		*				Запомнить шаблон для последующего использования.
`\{ \}`	*			*		*		Удовлетворить числу появлений.
`\<\>`		*	*					Должно быть на границе слова.
`+`					*		*	Одно или более появление.
`?`					*		*	Одно или нуль появлений.
`|`					*		*	Разделить два шаблона, чтобы показать, что требуется сравнить с тем или с другим шаблоном.
`( )`					*		*	Группировать выражения.

Заметим, что в **ed**, **ex** и **sed** во время выполнения операций редактирования (подстановки подстрок) вы определяете оба параметра: поисковый шаблон (слева) и заменяющая подстрока (справа). Метасимволы в приведенной таблице имеют значение лишь для шаблона поиска.

В **ed**, **ex** и **sed** могут использоваться метасимволы, которые имеют значение в заменяющей подстроке:

Таблица 10.2: Список метасимволов

Символ	**ex**	**sed**	**ed**	**Действие**
\ (левый слеш)	*	*	*	Далее следует символ ESCAPE.
`\n`	*	*	*	Использовать подстроку ранее запомненную в `\( \)`.
& (амперсанд)	*	*		Использовать предыдущий шаблон поиска.
~ (волнистая черточка)	*			Использовать предыдущую заменяющую подстроку.
\u или \U	*			Перевести все символы в верхний регистр.
\l или \L	*			Перевести все символы в нижний регистр.
\E	*			Отменить предыдущую команду \U или \L.
\e	*			Отменить предыдущую команду \u или \l.

10.2.2 Метасимволы

Следующие символы имеют специальное значение только в шаблонах поиска.

Таблица 10.3: Метасимволы в шаблонах поиска

Метасимвол	**Значение**
`.`	Удовлетворяет любому символу исключая `<NL>`.
`*`	Удовлетворяет любому числу (или нулю) символов, которые следуют за одиночным символом. Предшествующий одиночный символ может также быть регулярным выражением, например, если `.` (точка) обозначает любой символ, то выражение `.*` обозначает любое количество любых символов.

Продолжение таблицы на следующей странице

Метасимволы в шаблонах поиска (продолжение таблицы 10.3)	
`^`	Обозначает последующее регулярное выражение в начале строки.
`$`	Обозначает предшествующее регулярное выражение в конце строки.
`[ ]`	Обозначает любой символ из тех, что заключены в квадратные скобки. Знак переноса в скобках (минус) означает ряд последовательных символов. Знак `^` (крышка) в качестве первого символа в скобках играет роль отрицания: обозначает любые символы, которые *не* входят в список. Знак минус или правая (закрывающая) скобка в качестве первого символа рассматривается как члены списка.
`[^ ]`	Обозначает любые символы не входящие в список, ограниченный скобками.
`\{n,m\}`	Обозначает ряд появлений одиночного символа, который непосредственно предшествует этому выражению. Предшествующий символ тоже может быть регулярным выражением. Выражение `\{n\}` обозначает ровно n появлений, выражение `\{n,\}` означает как минимум n появлений, выражение `\{n,m\}` обозначает, что число появлений не меньше n и не больше m. При этом, n и m должны быть от 0 до 256.
`\`	Выключить специальное значение последующего символа.
`\( \)`	Сохранить подстроку, заключенную между `\(` и `\)` в специальном месте для последующего использования. До девяти подстрок может быть сохранено в одной строке. Сохраненные подстроки могут быть вызваны и использованы в последующих подстановках посредством ESCAPE-выражений от `\1` до `\9`.
Продолжение таблицы на следующей странице	

Метасимволы в шаблонах поиска (продолжение таблицы 10.3)	
\< \>	Обозначает символы в начале (\<) или в конце (\>) слова.
+	Обозначает одно или больше появлений предшествующего регулярного выражения.
?	Обозначает одно или нуль появлений предшествующего регулярного выражения.
|	Обозначает регулярное выражение определенное до или после.
( )	Группирование регулярных выражений.

Последующие символы имеют специальное значение только в заменяющих подстроках.

Таблица 10.4: **Метасимволы в шаблонах поиска**

Метасимвол	**Значение**
\	Выключить специальное значение последующего символа.
\n	Взять n-ную подстроку ранее сохранённую с помощью выражений \ (и \ ). Здесь n есть целое от 1 до 9. Самое левое выражение имеет номер 1.
&	Использовать поисковый шаблон в текущей заменяющей подстроки.
~	Использовать предыдущую заменяющую подстроку в текущей заменяющей подстроке.
\u	Преобразовать первый символ заменяющей подстроки в верхний регистр.
\U	Преобразовать всю заменяющую подстроку в верхний регистр.
\l \L	Преобразовать первый символ заменяющей подстроки в нижний регистр. Преобразовать всю заменяющую подстроку в нижний регистр.
Продолжение таблицы на следующей странице	

Метасимволы в шаблонах поиска (продолжение таблицы 10.4)

10.2.3 Примеры поиска

Таблица 10.5: **Метасимволы в примерах**

Шаблон	**Чему может соответствовать**
`big`	Последовательность символов `big`.
`^big`	Последовательность символов `big` в начале строки.
`big$`	Последовательность символов `big` в конце строки (в правой части).
`^big$`	Последовательность символов `big` должна быть единственной в строке.
`[Bb]ig`	Соответствует последовательностям `big` и `Big`.
`b[aeiou]g`	Соответствует любой из следующих последовательностей из трех букв: `bag`, `beg`, `big`, `bog`, `bug`.
`b[^aeiou]g`	Обозначает все слова или части слов из трех букв, которые между буквами `b` и `g` НЕ содержат букв, перечисленных в квадратных скобках: 'a', 'e', 'i', 'o', 'u'.
`^...$`	Обозначает любую полную строку, состоящую ровно из трех любых символов.
`b.g`	Обозначает все слова или части слов из трех букв, которые между буквами `b` и `g` могут содержать любой символ, например, `b2g`, `big`, `bug`, `b?g` и т.п.

Продолжение таблицы на следующей странице

Метасимволы в примерах (продолжение таблицы 10.5)	
`^\.`	Обозначает любую строку, которая начинается с точки.
`^[^.]`	Обозначает любую строку, которая начинается НЕ с точки. Заметим попутно, что если один и тот же текстовый файл просмотреть с использованием программы **egrep** с только что описанными шаблонами поиска `^\.`, а затем `^[^.]`, то мы обнаружим, что суммарное количество найденных строк может оказаться не равно общему числу строк в файле. Такое происходит из-за того, что часть строк файла могут содержать единственный символ `<NL>`.
`^$`	Обозначает пустую строку, которая состоит только из символа `<NL>`
`^\.[a-z][a-z]`	То же самое, но за точкой следуют две любые буквы в нижнем регистре.
`\.[a-z][a-z]\{2\}`	То же самое, но только для **grep** или **sed**.
`bog*`	Обозначает любое из слов или частей слова: `bog`, `bogs`, `bogus` и т.д.
`0\{3,\}`	Только для **sed** и **egrep**: три или более нулей подряд
`[0-9]\{2\}-[0-9]\{5\}-[0-9]\{1\}`	Только для **sed** и **egrep**: число которое имеет вид *nn-nnnnn-n*

10.2.4 Примеры поиска и замены

Нижеследующие примеры показывают как метасимволы могут использоваться в редакторе **sed**. Символ помеченный как ˝ обозначает в данном примере пробел. Символ табуляции обозначен как *tab*.

Таблица 10.6: **Метасимволы в примерах**

Команда	**Результат**
s/.*/(&)/	Заменить всю строку на нее же, но в скобках.
s/.*/mv & &.old/	Изменить список слов (одно слово на строку) в команде **mv**.
/$\hat{\$}$/d	Удалить пустые строки, т.е. строки содержащие один символ <NL>.
/$\hat{[}$˝*tab*]*$/d	Удалить пустые строки, а также строки содержащие пробелы или табуляторы.
s/˝˝*/˝/g	Превратить два и более пробелов в один пробел.
s/Yes/No/	Заменить слова в строке.

10.3 Введение

Работа с текстом: преобразование из одного вида в другой и анализ текста являются важными элементами любой работы. Здесь мы рассмотрим несколько общеупотребительных программ, которые помогают преобразовывать или анализировать текст.

Люди уже умеющие программировать или системные менеджеры взглянув на описания программ для преобразования текстов могли бы заметить, что алгоритмы преобразования достаточно просты. Многие из них смогли бы быстро набросать короткую программу на одном из популярных языков для реализации описанного алгоритма или вообще любого, который им более подходил бы, если бы им понадобилось преобразовывать тексты. На первый взгляд – это эффективный путь, т.е. написать свою программу преобразования текста, когда это потребуется.

Однако, если принять во внимание стоимость написания и отладки программы учитывая реальное время на такую работу, т.е. от момента осознания необходимости, до момента, когда программу можно будет уверенно использовать хотя бы небольшим коллективом пользователей, то во многих случаях правильнее будет использовать уже готовые средства преобразования текста, описываемые ниже. А если учесть возможные проблемы с переносом средств преобразования текста на другие машины или даже платформы, то очевидность использования уже готовых и многократно проверенных средств станет неоспоримой.

10.4 Программа cat

Использование: **cat** [*options*] [*file*]...

Программа **cat** читает каждый файл *file* или стандартный ввод и выводит вредёюные строки на устройство стандартного вывода. Если перечислено несколько файлов, то все они будут выведены на устройство стандартного вывода в той же последовательности, в которой были перечислены.

Таблица 10.7: ПАРАМЕТРЫ КОМАНДЫ **cat**

Параметры команды **cat**	
Параметры	Описание
-A -**show-all**	Эквивалентно комбинации параметров **-vET**.
-b -**number-nonblank**	Пронумеровать все не пустые строки начиная с 1.
-e	Эквивалентно комбинации параметров **-vE**.
-E -**show-ends**	Показать концы строк поместив знак $ сразу после конца каждой строки.
-n -**number**	Нумеровать все выводные строки начиная с 1.
-s -**squeeze-blank**	заменить множество подряд следующих пустых строк одной пустой строкой.
-t	Эквивалентно комбинации параметров **-vT**.
-T -**show-tabs**	Показать табуляторы отобразив их как символы `^I`.
-u	игнорируется (для совместимости с **UNIX**).
-v -**show-nonprinting**	Отобразить управляющие символы исключая `<LF>` и `<TAB>` с использованием нотации `^` (шляпка). Если в байте установлен самый левый бит, то символу будет предшествовать `M-`. Как читатель уже догадался, всем символам Кириллицы будет предшествовать последовательность `M-`.

10.5 Программа tac

Использование: **tac** [*option*]... [*file*]...

Программа **tac** копирует записи (по умолчанию - строки) каждого файла *file* или строки со стандартного устройства ввода на стандартное устройство вывода в обратном порядке. Иными словами, первая запись на вводе будет последней на выводе, а последняя запись на вводе будет первой на выводе. Записи разделяются специальными последовательностями (по умолчанию символ NL - New line). По умолчанию этот разделитель строк следует за концом записи.

Таблица 10.8: ПАРАМЕТРЫ КОМАНДЫ **tac**

Параметры команды **tac**	
Параметры	Описание
-b -**before**	Поместить разделитель записей в начале записи.
-r -**regex**	Обработать разделитель строк как регулярное выражение.
-s *separator* -**separator**=*separator*	Использовать значение *separator* как разделитель записей вместо символа NL (New Line) по умолчанию.

10.6 Программа nl

Использование: **nl** [*option*]... [*file*]...

Программа **nl** записывает строки файлов *file* или строки со стандартного устройства ввода на стандартное устройство вывода. При этом происходит нумерация всех или части строк на устройстве стандартного вывода. Во время работы программа разделяет ввод на логические страницы. По умолчанию номера строк начинаются с 1 на каждой логической странице. В то же время, если на вводе имеются несколько файлов, то **nl** рассматривает все файлы как единый поток ввода (единый документ).

Логическая страница состоит из трёх частей: ЗАГОЛОВОК, ТЕЛО СТРАНИЦЫ, ПОДСТРОЧНЫЕ ЗАМЕЧАНИЯ. Любая из перечисленных частей может быть пустой. Начало каждой из упомянутых частей страницы отмечается во вводном файле с помощью отдельной строки содержащей один из нижеследующих разделителей:

- `\:\:\:` – начало заголовка страницы;
- `\:\:` – начало тела страницы;
- `\:` – начало подстрочных замечаний.

Два символа \ и : могут быть замены посредством параметров, описанных ниже. Однако, образец и длина разделителей не могут быть изменены.

Сами строки с разделителями частей страницы при выводе заменяются пустыми строками. Любой вводимый текст, который следует до первого разделителя частей рассматривается как часть ТЕЛА СТРАНИЦЫ. Таким образом, **nl** рассматривает текст, который не содержит никаких описанных здесь разделителей, как одно тело одной логической страницы.

Программа **nl** воспринимает следующие параметры:

-b *style*

-**body-numbering**=*style* Выбрать стиль нумерации для строк в части ТЕЛО СТРАНИЦЫ для каждой логической страницы. Когда строка не нумеруется при выводе, то номер строки не увеличивается, однако в не нумерованных строках в начале строки помещается разделитель, который отделяет обычно номер от строки. Стили могут быть следующими:

- a - нумеровать все строки, включая заголовки и строки подстрочных замечаний (которые по умолчанию не нумеруются);
- t - нумеровать только не пустые строки (это умолчание для тела страницы);
- n - не нумеровать строки (это умолчание для заголовка и подстрочных замечаний);
- **p***regexp* - нумеровать только те строки, которые содержат подстроки удовлетворяющие регулярному выражению *regexp*.

-d *cd*

-**section-delimiter**=*cd* Установить разделитель частей равным символам *cd*]; по умолчанию используются символы `\:`. Если в параметре дан только символ *c*, то вторым символом остаётся :. Обратите внимание, что когда вы передаёте обратный слеш или другие специальные знаки из оболочки, то необходимо предусмотреть заключение в кавычки или использовать другие способы во избежание неверной интерпретации оболочкой данных специальных знаков.

-f *style*

-**footer-numbering**=*style* Аналогично параметру -*body-numbering*, но по отношению к подстрочным замечаниям.

-h *style*

-**header-numbering**=*style* Аналогично параметру -*body-numbering*, но по отношению к заголовку.

-i *number*

-**page-increment**=*number* Увеличивать номер строки на величину *number*. По умолчанию *number*=1.

-l *number*

-**join-blank-lines**=*number* Выбрать формат нумерации строк.

- **ln** – номера строк выровнены по левому краю, никаких ведущих нулей в написании номеров;
- **rn** – номера строк выровнены по правому краю, никаких ведущих нулей в написании номеров (это умолчание);
- **rz** – номера строк выровнены по правому краю, используются ведущие нули в написании номеров строк.

-p

-**no-renumber** Не сбрасывать номер строки с началом каждой логической страницы (по умолчанию происходит сброс номера строки).

-s *string*

-**starting-line-number**=*string* Разделить при выводе номер строки и саму строку последовательностью символов *string*] (по умолчанию используется символ `<TAB>`).

-v *number*

-**starting-line-number**=*number* Установить начальный номер нумерации на каждой логической странице равным *number*. По умолчанию *number*=1.

-**w** *number*

-**number-width**=*number* Использовать *number* символов для номеров строк. По умолчанию *number*=6.

10.7 Программа od

Использование:
od [*option*]... [*file*]...
od -C [*file*] [[+]*offset* [[+]*label*]]
Версия программы:

```
od --version
od (GNU textutils) 1.22
```

Программа **od** выводит на устройство стандартного вывода ясное представление каждого файла *file* (или устройство стандартного ввода, если использован знак - вместо имени файла).

Каждая строка вывода состоит из относительного адреса во вводном файле, за которым следуют группы данных из файла. По умолчанию **od** выводит адрес в восьмеричном виде, а каждая группа данных из файла представляет собой два байта вводного файла, выведенных как одно восьмеричное число.

Программа воспринимает следующие параметры.

-**A** *radix*

-**address-radix**=*radix* Выбрать в которой будет формироваться адрес во вводном файле. Значениями *radix* может быть следующие представления:

- d - десятичное;
- o - восьмеричное (умолчание);
- x - шестнадцатиричное;
- n - не печатаются адреса вообще.

-**j** *bytes*

-**skip-bytes**=*bytes* Пропустить *bytes* вводных байтов на вводе до начала вывода. Если *bytes* начинается с *0x* или `0X`, то число *bytes* интерпретируется как шестнадцатиричное. Если *bytes* начинается с `0`, то оно интерпретируется как восьмеричное. В остальных случаях оно рассматривается как десятичное число. Если *bytes* заканчивается буквой `b`, то *bytes* умножается на 512; буквой `k` – на 1024; буквой `m` – на 1048576.

-N *bytes*

-**read-bytes**=*bytes* Вывести *bytes* байтов максимум из вводного файла. Префиксы и суффиксы *bytes* интерпретируются также как с параметром **-j**.

-s*[n]*

-**strings***[=n]* Вместо обычного вывода, напечатать лишь `string constants:` (строковые константы), далее как минимум *n* последовательных символов (по умолчанию 3).

-t *type*

-**format**=*type* Выбрать формат в котором производится вывод данных. Строка *type* может содержать один или более символов, каждый из которых определяет отдельный вид вывода. Если строка состоит из более, чем одного символа, то каждая строка вводного файла будет выводиться столько раз, сколько заказано видов вывода и в том порядке, который был указан в строке *type*.

В строке *type* могут быть следующие символы:

- $\boxed{\text{a}}$ - выводятся именованные символы, т.е. все специальные знаки выводятся с помощью их имён, например, пробел - `<SP>`, перевод строки - `<NL>` и т.д.
- $\boxed{\text{c}}$ - выводятся символы ASCII (если есть изображаемый символ) или их коды;
- $\boxed{\text{d}}$ - десятичное со знаком;
- $\boxed{\text{f}}$ - представление в виде десятичных чисел в формате с плавающей точкой;
- $\boxed{\text{o}}$ - восьмеричное представление;
- $\boxed{\text{u}}$ - десятичное без знака;
- $\boxed{\text{x}}$ - шестнадцатиричное.

Исключая типы **a** и **c** вы можете определить число байтов, которые вы хотели бы использовать для интерпретации данных определённого типа. Число байтов определяется десятичным целым, которое следует за символом типа данных. Поскольку допускаемые значения целых должны соответствовать размерам типов данных для языка C (C - char, S - short, I - int, L - long; для чисел с плавающей точкой: F - float, D - double, L - long double), то разумнее их и употреблять. Например,

```
date | od -t cxC file_name
```

Будет выведено нечто похожее на нижеследующее:

```
0000000   F   r   i       J   u   l           2       1   7   :   0
        46 72 69 20 4a 75 6c 20 20 32 20 31 37 3a 30 35
0000020 :     5   0       M   S   D       1   9   9   9 \n
        3a 35 30 20 4d 53 44 20 31 39 39 39 0a
0000035
```

-v

-**output-duplicates** Включить вывод последовательных одинаковых строк. По умолчанию **od** выведет только первую строку, а вместо остальных выведет только звёздочку.

10.8 Поиск в файле символьных цепочек с помощью программ семейства grep

Семейство программ **grep** позволяет вам находить в одном или в группе файлов все строки, которые содержат цепочки символов удовлетворяющих регулярному выражению или шаблону (подробнее смотрите раздел 10.2). Само имя grep означает Global Regular Expresion Print - печать регулярных выражений общего вида. Программы этого семейства можно использовать как фильтры, т.е. извлекать подходящие под регулярное выражение строки из стандартного ввода и посылать их на устройство стандартного вывода.

Семейство состоит из трёх членов, которые различаются по скорости работы, а также по виду допустимых регулярных выражений.

- Программа **grep** производит поиск символьных цепочек соответствующих представленному шаблону. Имеется три следующих варианта **grep**, выбираемых соответствующими параметрами:

-G

-**basic-regexp** Интерпретирует шаблон как базовое регулярное выражение. Этот режим используется по умолчанию.

-E

-**extended-regexp** Интерпретирует шаблон как расширенное регулярное выражение.

-F

-**fixed-strings** Интерпретирует шаблон как список фиксированных символьных цепочек разделённых символами `<NL>`. Если любая цепочка из списка встречена в файле, то данная строка выводится на устройство стандартного вывода.

- Программа **egrep** весьма подобна, но не идентична программе **grep -E**.
- Программа **fgrep** есть то же, что **grep -F**.

Все программы семейства **grep** воспринимают следующие параметры.

-*num* Удовлетворяющие строки будут напечатаны с *num* предшествующими строками и *num* последующими. В этом случае **grep** не будет печатать никакую строку более, чем один раз.

-A *num*

-**after-context**=*num* Вывести на устройство стандартного вывода *num* строк после найденной.

-B *num*

-**before-context**=*num* Вывести на устройство стандартного вывода *num* строк перед найденной строкой.

-C

-**context** Эквивалентно значению `-2`.

-V

-**version** Вывести версию программы на устройство вывода диагностических сообщений.

-b

-**byte-offset** Вывести относительный адрес байта в файле для каждой выводимой строки.

-c

-**count** Запретить обычный вывод программы. Вместо него вывести число строк, которые удовлетворяют шаблону. Если этот параметр используется совместно с параметром **-v** (см. ниже), то будет выводится число строк не удовлетворяющих шаблону.

-e *pattern*

-**regexp**=*pattern* Использовать *pattern* в качестве шаблона. Эта запись удобна, когда шаблон начинается со знака - (минус).

-f *file*

-**file**=*file* Взять шаблон из файла с именем *file*. Эта возможность удобна при поиске с использованием символов Кириллицы.

-h

-**no-filename** Запретить указание имён файлов на выводе, если производится поиск по шаблону в нескольких файлах. По умолчанию будет указаны имена файлов, в которых найдены строки соответствующие шаблону.

-i

-**ignore-case** Игнорировать различие в регистрах как в шаблоне так и во вводных файлах. Обратим внимание, что эта возможность отлично работает для Латиницы, но не всегда для Кириллицы.

-L

-**files-without-match** Запретить обычный вывод. Вместо этого напечатать имя каждого последовательно проверяемого файла, в котором не найдено строк соответствующих шаблону. Операция будет остановлена как только будет найдена первая строка соответствующая шаблону или будет исчерпан список файлов.

-l

-**files-with-matches** Запретить обычный вывод. Вместо него выдать имя каждого файла, в котором имеются строки удовлетворяющие шаблону.

-n

-**line-number** Выдать номера строк в файле вместе с самими строками удовлетворяющими шаблону.

-q

-**quiet** Выполняться без вывода. Поиск строк соответствующих шаблону останавливается на первой же найденной строке. Если строка найдена, устанавливается код завершения 0, и 1 - в противном случае.

-s

-**silent** Запретить сообщения об ошибках если встретились несуществующие или нечитаемые файлы (например, нет доступа по чтению).

-v

-**revert-match** Инвертировать смысл соответствия шаблону, т.е. вывести строки, которые не соответствуют шаблону.

-w

-**word-regexp** Вывести лишь те соответствия шаблону, которые совпадают с границами слова. Слово может содержать букву, цифру и подчёркивание.

-x

-**line-regexp** Вывести лишь те соответствия, которые представляют собой целую строку.

-y удалённый синоним для параметра **-i**.

Программа **grep** понимает две различные версии синтаксиса регулярных выражений: BASIC - основной и EXTENDED - расширенный. В GNU **grep** нет различий при использовании обоих видов синтаксиса. Регулярные выражения рассматриваются в разделе 10.2.

10.8.1 Несколько примеров с использованием grep

Для начала полезно получить версию программы **grep**

```
grep -V
```

В ответ печатается:

```
grep (GNU grep) 2.1

Copyright (C) 1988, 92, 93, 94, 95, 96, 97 Free Software Foundation, Inc
This is free software; see the source for copying conditions. There is N
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

Найти в файле `Grep.tex` все строки, которые содержат последовательность **-i**

```
grep -e -i Grep.tex
```

В ответ печатается

```
\item [-i]
\item [-y] удал\"енный синоним для параметра \SSname{-i}.
```

Найти в файле все строки которые содержат в первой слева позиции обратный слеш

```
grep ^\\\\   Grep.tex
```

В ответ будут напечатаны все строки, которые содержат знак обратный слеш в первой слева позиции. А команда

```
grep -v ^\\\\   Grep.tex
```

выведет на экран все строки, которые не содержат знака обратный слеш в первой слева позиции. Ту же команду можно выполнить в другом варианте

```
cat > Pattern
^\\
Ctrl/D
grep -vf Pattern Grep.tex
```

В последнем примере шаблон для поиска помещён в файл с именем `Pattern`. При этом вы могли обратить внимание, что использовано разное количество обратных слешей. Дело связано с тем, что обратные слеши сначала были обработаны оболочкой **bash**, а затем – программой **grep**.

10.9 fmt – Форматировать текст

Программа **fmt** вводит текст, производит простое форматирование, которое в основном заключается в наиболее полном заполнении строк.

Использование программ:

fmt [*option*]... [*file*]...

Программа читает текст либо из файла, либо со стандартного устройства ввода, а выводит сформированный текст на стандартное устройство вывода. По умолчанию, пустые строки, пробелы между словами и пустые места в начале строк сохраняются и на выводе. Вводимые строки с пробелами в начале строк не объединяются. Знаки табуляции TAB расширяются обычным образом при вводе и используются затем при выводе.

Программа **fmt** предпочитает разбивать строки в конце предложения. Она пытается избежать разрыва строки после первого слова предложения и перед последним словом предложения. Конец предложения определяется как одно из двух наступивших условий:

- достигнут конец параграфа, т.е. встречена пустая строка или знак NL;
- слово закончилось одним из знаков: **.?!** - точка, вопросительный знак, восклицательный знак, за которыми следует два пробела или конец строки. Алгоритм разбиения параграфа на строки является вариантом алгоритма описанного в статье *Breaking Paragraphs Into Lines* (авторы: *Donald E. Knuth* и *Michael F. Plass*, журнал **Software–Practice and Experience**, 11 (1981), 1119-1184).

Программа воспринимает следующие параметры:

-c

-**crown-margin** Сохранить без изменения сдвиги первых двух строк параграфов и выровнять левый край всех последующих строк параграфа по левому краю второй строки параграфа.

-t

-**tagged-paragraph** Этим параметром устанавливается режим форматирования **tagged-paragraph**. Этот режим похож на режим **crown-margin** исключая то, что если сдвиг первой строки параграфов тот же самый, что сдвиг второй строки параграфа, то первая строка обрабатывается как однострочный параграф.

-s

-**split-only** Только разделять строки. Не объединять короткие строки, чтобы сформировать более длинные.

-u

-**uniform-spacing** Унифицировать пробелы. Уменьшить число пробелов между словами до одного пробела, а число пробелов между предложениями уменьшить до двух пробелов.

-width

-w *width*

-**width**=*width* Заполнить выводные строки до ширины *width* (умолчание 75).

-p *prefix*

-**prefix**=*prefix* Форматированию будут подвергнуты только строки, которым предшествует *prefix* (перед ним могут быть пробелы). Сам *prefix* и возможно предшествующие ему пробелы будут удалены из строки перед форматированием. После выполнения форматирования, при выводе результата в начало сформатированных строк будет помещён *prefix*.

Одно из полезных применений данного параметра – форматирование программных комментариев. Следует лишь иметь в виду, что если вы планируете отформатировать комментарии скриптов, то знак комментария полезно задать в кавычках, например,

```
fmt -p "#" script
```

10.10 pr – печать текста с разделением на страницы и колонки

Использование программы:

pr [*option*]... [*file*]...

По умолчанию на каждой странице печатается 5-и срочный заголовок: две пустые строки, строка содержащая дату, имя файла, число страниц, затем две пустые строки. Внизу страницы также выводится пять пустых строк. С параметром **-f** печатается только трёхстрочный заголовок: две начальные пустые строки опускаются, вывод пустых строк внизу страницы также опускается. По умолчанию, размер страницы в обоих случаях есть 66 строк. Текстовая строка заголовка занимает всю ширину страницы и имеет следующий вид:

```
yy-mm-dd HH:MM string Page nnnn
```

Строка центрирована. Знаки перевода формата на вводе (form feed) переход переход на следующую страницу.

Колонки имеют одинаковую ширину и отделяются друг от друга последовательностью символов (по умолчанию пробелами). Длина выводных строк ограничивается (по умолчанию 72 символа), если не использован параметр **-j**. По умолчанию вывод в одну колонку не ограничивается по длине строк; чтобы ввести в действие ограничение по длине строк можно использовать параметр **-w**.

Программа воспринимает следующие параметры:

+*first_page*[:*last_page*] Начать печатание начиная со страницы *first_page* и закончить на странице *last_page*. Имеютя в виду реальные страницы, а не их номера. Отсутствие параметра *last_page* означает, что на печать будет выводиться вся вводимая информация до конца вводного файла. Символ FF (Form Feed) смена формата) на вводе рассматривается как переход к новой странице. Несколько таких символов подряд вызовут выдачу на печать пустых страниц, что засчитывается при определении страницы *first_page*.

-*column* С каждым файлом `file`, который читается программой **pr** производится следующее.

- Производится определение ширины колонок исходя из величин *page_width* и *column* (все колонки одинаковой ширины).
- При считывании строк файла, если они оказались больше ширины колонки, строки усекаются по длине до величины до ширины колонки (правая часть строки отбрасывается) и помещаются в колонку до заполнения колонки (сверху вниз). Затем начинается новая колонка.
- Если страница заполнена, происходит переход к следующей странице. Если встречен конец вводного файла или символ смены формата, а страница не заполнена, то производится балансировка колонок, чтобы все колонки были заполнены примерно одинаково.

-a При многоколонном выводе заполнять колонки по очереди, т.е. заполнять вначале первую строку первой колонки, затем первую строки второй колонки и т.д. В остальном все также как при использовании параметра -{Snamecolumn.

-c При выводе печатать управляющие символы с использованием специальных обозначений, например, Ĉ. Неизображаемые символы печатать в восьмеричном виде (каждому символу предшествует обратный слеш). Нетрудно заметить, что таким образом можно выдать на печать файлы произвольного вида, а не только текстовые.

-d При выводе производить двойной перевод строк.

-e*[in-tabchar[in-tabwidth]]* Заменить знаки табуляции на соответствующее число пробелов при вводе. Возможный аргумент *in-tabchar* представляет собой символ табуляции (по умолчанию <TAB>). Возможный аргумент *in-tabwidth* представляет собой целое число, представляющее расстояние между соседними позициями табуляции (по умолчанию 8).

-f

-F Использовать символ `<FF>` (Form Feed - смена формата) вместо символа `<NL>` (New Line - новая строка) для разделения строк во вводном файле. При этом размер страницы в строках остётся неизменным (по умолчанию 66), но число строк текста на странице меняется с 56 до 63.

-h *header* Заменить имя файла в заголовке строкой *header*. Программа может обрезать строку *header*, если она окажется слишком длинной. С параметром **-h** `""` будет печататься пустой заголовок. При этом пробел между параметром **-h** и аргументом весьма важен.

-i*[out-tabchar[out-tabwidth]]* Заменить знаки табуляции на выводе соответствующим числом пробелов. Возможный аргумент *out-tabchar* обозначает символ табуляции (по умолчанию `<TAB>`). Возможный аргумент *out-tabwidth* означает расстояние между соседними колонками табуляции (по умолчанию 8).

-j Объединить строки полной длины. Не производится укорачивания строк даже если они выходят за пределы колонок; при этом не предпринимается никаких мер по выравниванию колонок. Может быть использовано совместно с параметрами *-column*, **-a**, **-m** или **-s**[*separator*].

-l *page_legth* Установить длину страницы в строках равной значению *page_legth* (по умолчанию 66 строк). Если величина *page_legth* меньше или равна 10 (меньше или равно 3 с использованием параметра **-f**), то заголовки и подстрочные заголовки будут опущены, а все символы `<FF>` на вводе будут игнорироваться (как с использованием параметра **-T**).

-m Объединить и напечатать все файлы представленные в виде аргументов программы **pr** параллельно, по одному файлу в каждой колонке. Если стока слишком длинна, чтобы поместиться в колонке, то она усекается.

Отсечённая часть отбрасывается. Могут использоваться совместно параметры **-j**, **-s**[*separator*].

-n*[number-separator[digits]]* В каждой колонке добавить в начале номер строки. С параметром **-m** будет лишь один номер на все колонки. Возможный параметр *number-separator* представляет собой символ, который помещается между номером строки и самой строкой (по умолчанию <TAB>). Возможный параметр *digits* определяет число цифр в номере строки (по умолчанию 5). Счёт начинается с первой введённой строки, а не с первой напечатанной строки (смотрите также параметр **-N**).

-N *line_number* Начать счёт строк с номера *line_number* для первой строки первой выведенной страницы.

-o *n* Сдвинуть каждую строку на *n* позиций вправо (умолчание нуль). Общая ширина вывода составит *n* плюс ширина установленная параметром **-w**.

-r Не печатать сообщений об ошибках, если аргумент *file* не может быть открыт.

-s*[separator]* Разделить колонки в выводном тексте строкой *separator*. Не должно быть пробела между параметром **-s** и значением аргумента.

-t Прекратить печатать заголовки и номера страниц и никак не печатать нижние заголовки страниц. Таким образом не формируется на печати структура страниц, однако символы `<FF>` остаются неизменными как они были на вводе. Параметр **-t** отменяет значение параметра **-h**, если он был использован.

-T Не печатать верхних и нижних заголовков страницы, а также удалить все символы `<FF>` из вводных файлов.

-v Печатать неизображаемые символы в виде восьмеричных кодов с обратным слешем.

-w *page_width* Установить ширину страницы в значение PSnamepage_width (по умолчанию 72). С использованием данного параметра и без него строки заголовка всегда усекаются до значения *page_width*, если длина строк заголовка страницы превышает *page_width*.

С использованием параметра **-w** строки текста на печати усекаются до значения *page_width*, если не использован параметр **-j**.

Без параметра **-w**, но с использованием *-column*, **-a** *-column* или **-m** строки усекаются до соответствующих (ширина строки или колонки) приемлемых величин.

Если не используются параметры управления колонками и не используется параметр **-w**, то это эквивалентно использованию параметров **-w** *72* **-j**.

10.11 fold – разделение длинных строк

Использование:

fold [*option*]... *file*]...

программа **fold** читает каждый файл `file` (или устройство стандартного ввода) и выводит введённую информацию на устройство стандартного вывода разбивая длинные строки, чтобы удовлетворить допустимой ширине выводной строки. По умолчанию **fold** разбивает строки, которые оказались длиннее 80 символов. Вводимая строка разбивается на столько строк, на сколько это необходимо.

Умалчиваемый алгоритм размера выводной колонки следующий:

- `<TAB>` (табулятор) означает более, чем одну колонку;
- `<BS>` (возврат на символ) означает, что счётчик размера выводной колонки уменьшается на 1;
- `<CR>` (возврат каретки) обнуляет счётчик колонки выводной строки.

Программа **fold** воспринимает следующие параметры.

-b

-**bytes** Считать байты, а не колонки, таким образом символы <TAB>, <BS>, <CR> рассматриваются как занимающие одну колонку, точно также как и остальные символы.

-s

-**spaces** Разбивать вводимые строки на границе слов, т.е. строка будет разбиваться после последнего пробела, но до максимального размера выводной строки. Если строка не содержит пробелов, то она разбивается в точке максимально допустимого размера выводной строки. По умолчанию, строка разбивается на куски не обращая внимание на границы слов.

-w *width*

-**width**=*width* Установить максимальную длину выводной строки в *width* (по умолчанию 80).

10.12 Примеры использования

Например, вам необходимо определить какие текстовые строки используются в программе **bash**. Такое легко сделать использовав последовательность команд:

```
pr -t -c -F /bin/bash | fold | less
```

После всех преобразований в среде команды **less** довольно легко найти любые символьные строки, которые вас могут интересовать.

10.13 head – вывести головную часть файлов

Использование:

head [*option*]... [*file*]...

head -*number* [*option*]... [*file*]...

программа выводит на устройство стандартного вывода головную часть каждого файла *file* (по умолчанию 10 строк). Если определён более, чем один файл, программа печатает однострочный заголовок состоящий из

```
==> FILE NAME <==
```

до вывода каждого файла с именем `file`.

Программа **head** воспринимает несколько параметров, которые могут быть представлены в двух форматах: старом или новом. В новом формате числовые значения аргументов представляются в виде:

-q -n 1

в то же время старый формат то же значение аргумента представляет так

-1q

-COUNTOPTIONS Этот параметр распознаётся программой только в том случае, если он определён раньше всех. *count* – это десятичное целое, за которым может следовать буква специфицирующая размер: **b** (блок – 512), **k** (1024) или **m** (1048576) как в параметре **-c** или **l** чтобы обозначить число строк или другие параметры: **c**, **q**, **v**.

Например, вы можете задать:

```
head -1b FileName
```

будет напечатан один блок (512 байтов) из файла с именем *FileName*. А если вы зададите

`head -1bl FileName`

то будет напечатана одна строка из файла с именем `FileName`. такая запись целиком эквивалентна записи

`head -1l FileName`

-c *bytes* Напечатать первые *bytes* байтов, вместо начальных строк. За десятичным целым может следовать буква **b** (умножить число на 512), **k** (умножить число на 1024), **m** (умножить число на 1048576).

-n *n*

-**lines**=*n* Вывести первые *n* строк.

-q

-**quiet**

-**silent** Не печатать заголовок с именем файла.

-v

-**verbose** Всегда печатать заголовок содержащий имя файла.

10.14 tail – вывести хвост файла

Здесь обсуждается программа **tail** версии `tail (GNU textutils) 1.22`.

Использование программы:

tail [*option*]... [*file*]...
tail -*number* [*option*]... [*file*]...
tail +*number* [*option*]... [*file*]...

программа **tail** выводит на устройство стандартного вывода последние (по умолчанию 10) строки каждого файла перечисленного в списке *file*. Если специфицирован более, чем один файл, то перед выводом каждого файла программа выводит однострочный заголовок:

```
==> FILE NAME <==
```

GNU **tail** может выводить любое количество данных. Программа не имеет параметра **-r** (вывод строк файла в обратном порядке), как другие версии данной программы. Вывод строк файла в обратном порядке производится программой **tac** 10.5.

Программа распознаёт несколько параметров.

-*count*

+*count* Этот параметр распознаётся, если он определён первым. Десятичное целое *count* означает размер. Если далее не следует никаких других букв, то это количество строк, которые будут выданы на печать с хвоста файла (если использован знак **-** (минус) или нет никакого знака. Если использован знак + (плюс), то на устройство стандартного вывода выводится информация из файла начиная со строки с номером *count* от начала файла.

За номером может следовать буква **l** (означает, что счёт идёт в строках) или буквы **b**, **k**, **m** с теми же значениями как в параметре **-c**. Могут следовать также другие параметры: **c**, **f**, **q**, **v**.

-c *bytes*

-**bytes**=*bytes* Вывести последние *bytes* байтов вместо последних строк. Добавление буквы **b** после десятичного целого *bytes* означает умножение на 512, **k** – на 1024, **m** – на 1048576.

-f

-**follow** Войти в бесконечный цикл, который пытается читать больше данных в конце файла. Предположительно файл постоянно растёт, поэтому вы будете видеть динамическое изменение содержимого файла. Параметр игнорируется если чтение производится с устройства стандартного ввода (в том числе из программного канала). Если определено несколько файлов, то **tail** выводит заголовки с именами файлов.

-n *n*

-**lines**=*n* Вывести последние *n* строк.

-q

-quiet

-**silent** Не выводить заголовки с именами файлов (по умолчанию заголовки файлов выводятся, если дано несколько файлов).

-v

-**verbose** Всегда печатать заголовки с именами файлов (по умолчанию заголовки печатаются если дано несколько файлов).

10.15 split – разделить файл на части одинакового размера

Использование:

split [*option*] [*input* [*prefix*]]

По умолчанию программа **split** помещает каждые очередную 1000 (тысячу) строк ввода в отдельный файл. Имена выводных файлов формируются на основе значения *prefix* (по умолчанию **x**), за которым следует группа буквенных комбинаций вида **aa**, **ab** и так далее. Таким образом, конкатенация всех выводных файлов составит исходный вводной файл. Если количество выводных файлов превышает число 676, то **split** добавляет количество букв в именах вновь создаваемых файлов. Этот факт индицируется дополнительной буквой **z**, которая следует за префиксом.

Программа воспринимает следующие параметры.

-*lines*

-l *lines*

-**lines**=*lines* Разбить вводной файл на части по *lines* строк и поместить результат в отдельные файлы.

-b *bytes*

-**bytes**=*bytes* Поместить в каждый создаваемый файл очередные *bytes* байтов. Добавление буквы **b** после *bytes* умножает число *bytes* на 512, **k** – на 1024, **m** – на 1048576.

-C *bytes*

-**line-bytes**=*bytes* Поместить в каждый создаваемый файл столько полных строк ввода сколько возможно без превышения *bytes* байтов в каждом создаваемом файле. Для строк, размер которых превышает *bytes* байтов, записывать очередные *bytes* байтов до исчерпания строки, затем действовать как обычно. Этот параметр может использоваться в таком же формате как и параметр **-c**.

-**verbose** Выдавать сообщения на устройство стандартного вывода диагностики перед открытием очередного выводного файла.

10.16 csplit – разделить файл на контекстно обусловленные части

Использование:

csplit [*option*]... *input pattern...*

input представляет собой имя файла или устройство стандартного ввода. Содержание выводных файлов определяется значениями аргументов *pattern* в соответствии с алгоритмом, который описывается ниже.

Если регулярное выражение *pattern* не соответствует никакой строке во вводном файле, то возникает состояние ошибки. После того как все выражения *pattern* найдены во вводном файле, остаток вводного файла просто копируется в последний выводной файл.

По умолчанию, программа **csplit** печатает число байтов записанных в каждый выводной файл.

Программа допускает следующие виды регулярных выражений *pattern*.

n Создать выводной файл содержащий строки вводного файла до строки *n* (не включая саму строку *n*). Оставшаяся часть файла будет записана во второй выводной файл. Например, последовательность команд

```
csplit crawler 10
wc *
```

даст в результате следующее:

```
$ csplit crawler 10
790
21371
$ wc *
    337    1441   22161 crawler
      9      81     790 xx00
    328    1360   21371 xx01
    674    2882   44322 total
```

Можно использовать два и более чисел:

```
csplit crawler 10 14 19 39
wc *
```

тогда получим следующее:

```
$ csplit crawler 10 14 19 39
790
948
374
667
19382
$ wc *
    337    1441   22161 crawler
      9      81     790 xx00
      4      69     948 xx01
      5      31     374 xx02
     20      63     667 xx03
    299    1197   19382 xx04
    674    2882   44322 total
```

При этом во второй файл (`xx01`) попадут строки с номера 10 по номер 13, в третий файл (`xx02`) – строки с номера 14 по номер 18, в четвёртый файл (`xx03`) – строки с номера 19 по номер 38, а остаток запишется в файл `xx04`.

/pattern/[*offset*] Создать выводной файл содержащий часть вводного файла от текущей строки до (не включая) строки вводного файла, которая удовлетворяет регулярному выражению *pattern*. Если использован аргумент *offset* (десятичное целое со знаком плюс или минус), то выводится часть вводного файла до строки удовлетворяющей регулярному выражению *pattern* + (плюс) или `-` (минус) *offset* и одна строка, после которой начинается очередная часть ввода.

Например, мы имеем файл с именем `t`, который содержит тестовую последовательность:

```
$ cat t
111111
222222
333333
444444
555555
666666
```

Тогда последовательность команд:

```
csplit t 2 /333/+1
```

```
wc *
```

даст

```
[shevel@pcfarm TEST]$ csplit t 2 /333/+1
7
14
22
[shevel@pcfarm TEST]$ wc *
      7       6      43 t
      1       1       7 xx00
      2       2      14 xx01
      4       3      22 xx02
     14      12      86 total
```

А содержимое файлов будет следующим:

```
[shevel@pcfarm TEST]$ cat xx00
111111
[shevel@pcfarm TEST]$ cat xx01
222222
333333
[shevel@pcfarm TEST]$ cat xx02
444444
555555
666666
```

Если слеши в выражениях заменить на знаки \% (процента) то никаких выводных файлов создаваться не будет. Таким образом, такое выражение позволяет пропустить определённую часть вводного файла. Например,

```
csplit t 2 %333%+1
```

В этом случае будет создано всего два выводных файла.

{*repeat-count*} Повторить предыдущее регулярное выражение *repeat-count* раз. Число *repeat-count* может быть десятичным положительным целым или звёздочкой. Звёздочка означает, что выражение повторяется до тех пор пока не закончится ввод.

Имена выводных файлов состоят из префикса (по умолчанию **xx**), за которым следует суффикс. По умолчанию суффикс состоит из двух цифр (от 00 до 99).

По умолчанию, если программа **csplit** встретила ошибку и завершается ненормально или получает сигнал терминирования, то она удаляет все файлы созданные до получения сигнала.

Программа воспринимает следующие параметры.

-f *prefix*

-**prefix**=*prefix* Использовать строку *prefix* в качестве префикса в именах создаваемых выводных файлов.

-b *suffix*

-**suffix**=*suffix* Использовать строку *suffix* как суффикс в именах выводных файлов. Когда этот параметр определён, то строка *suffix* должна включать точно один символ для определения типа преобразования (как в программе **printf**(3)). В стоке *suffix* могут быть также использованы флаги определения формата, количество знаков для числа, точность преобразования. Тип преобразования должен задавать перевод двоичного числа в читаемую форму, иными словами могут быть заданы типы: **d**, **i**, **u**, **o**, **x** и **X**. Если использован данный параметр, то параметр -**digits** игнорируется.

-n *digits*

-**digits**=*digits* Использовать суффикс размером *digits* цифр в именах выводных файлов. По умолчанию используется две цифры.

-k

-**keep-files** Не удалять выводные файлы, даже если встретилась ошибка и программа завершилась ненормально.

-z

-**elide-empty-files** Запретить генерацию выводных файлов нулевой длины.

-s

-q

-**silent**

-quiet Не печатать размеры выводных файлов. По умолчанию, печатаются размеры выводных файлов в байтах.

10.17 sort – сортировать текстовые файлы

Использование программы:

sort [*option*]... [*file*]...

Программа **sort** сравнивает, объединяет, сортирует вводной поток и выдаёт результат на устройство стандартного вывода. В качестве вводного потока может использоваться один или несколько файлов или устройство стандартного ввода. Программа имеет три режима работы: режим сортировки (умолчание), режим слияния, режим проверки отсортирован файл или нет. Переключение в режим слияния и проверки производится следующими параметрами.

-m Объединить данные файлы сортируя их как одну группу. Каждый вводной файл должен быть уже отсортирован индивидуально. Можно использовать вместо слияния режим сортировки, однако режим слияния там где его возможно использовать выполняется быстрее.

-c Проверить отсортированы ли уже данные файлы. Если файлы отсортированы, то устанавливается код завершения 0. Если файлы не отсортированы, то выдаётся сообщение об ошибке и устанавливается код завершения 1.

Несколько замечаний о характере сортировки.

Если определены ключевые поля, по которым производится сортировка, то программа сравнивает каждую пару полей в соседних строках в том порядке, в котором это определено в командной строке с учётом параметров упорядочивания, до того как будет встречено различие либо строки вводного потока будут исчерпаны.

Если определены любые глобальные параметры сортировки из списка **Mbdfinr**, но не определены ключевые поля, то программа будет сравнивать полные строки.

Программа GNU **sort** не имеет ограничений на длину вводной строки или на значения байтов, которые составляют вводную строку. Если возникла любая ошибка, программа завершается с кодом 2. Если установлена переменная окружения $TMPDIR, то **sort** будет использовать это значение как имя каталога, в котором будут располагаться временные файлы. По умолчанию используется имя каталога `/tmp`. Параметр **-T** `tmpdir`, в свою очередь, устанавливает новое значение имени каталога для временных файлов вне зависимости от значения переменной окружения $TMPDIR.

Нижеследующие параметры влияют на упорядочивание выводных строк. Они могут быть определены глобально или как часть определённого ключевого поля. Если никаких ключевых полей не используется, то глобальные параметры воздействуют на сравнение полных строк. В противном случае глобальные параметры наследуются ключевыми полями, которые не определяют никаких своих специальных параметров.

-b Игнорировать ведущие пробелы когда производится поиск ключей сортировки в каждой строке.

-d Сортировать по правилам телефонного справочника: игнорировать все символы, кроме букв, пробелов и цифр.

-f Рассматривать символы в разных регистрах как одинаковые, например, **G** и **g** будут рассматриваться как один символ.

-g Сортировать численно, но использовать **strtod**, чтобы достичь численных значений. Это позволяет сравнивать числа представленные в формате *nnnE+mm*. Это очень медленный режим работы, рекомендуется лишь если нет альтернативы. В дополнение числа выходящие за пределы точности двойного слова (8 байтов) рассматриваются как нулевые значения. Никакой диагностики о переполнениях не предусмотрено.

-i Игнорировать во время сортировки любые символы вне восьмеричных кодов ASCII 040–0176.

-M Исходные строки в который после любого количества пробелов следуют трёхбуквенные обозначения месяцев превращаются в верхний регистр и после производится сравнение. Основное правило сравнения таково:

JAN<FEB<MAR<...<DEC. Неверная аббревиатура меньше верной.

-n Использовать числовые сравнения. Числу в строке может предшествовать сколько угодно пробелов. Числу может предшествовать знак минус. Число может содержать десятичную точку. Число может отсутствовать (состоять из 0 цифр). Не распознаётся знак плюс и символ экспоненты. Чтобы их распознать используйте параметр **-g**.

-r Сортировать в обратном порядке.

Имеются и другие параметры.

-o *output-file* Произвести вывод не на устройство стандартного вывода, а в файл с именем `output-file`. Если имя файла `output-file` совпадает с именем вводного файла, то программа **sort** запишет результат сортировки во временный каталог, а потом скопирует результат во вводной файл. Иными словами вводной файл будет в данном случае изменён.

-t *separator* Использовать символ *separator* как разделитель полей во время поиска сортировочных ключей в каждой строке. По умолчанию поля разделяются пустой строкой между непробельными символами и пробелами. Например, если вводная строка содержит

```
miru mir
```

то программа **sort** разделит её на два поля `"miru"и "mir"`.

-u Для случаев по умолчанию или при использовании параметра **-m** выводит лишь первую из одинаковых строк. Если использован параметр **-c**, то проверяется, что нет одинаковых подряд идущих строк.

-k *pos1[,pos2]* Рекомендованная POSIX форма параметра определяющего сортировочное поле в строке. Поле состоит из символов заключённых между *pos1* и *pos2* или концом строки, если *pos2* опущено. Поля и символьные позиции нумеруются начиная с 1.

Позиция внутри строки определяется в форме F.C, где F - номер поля, а C - есть номер символа в поле (слева направо), с которого начинается сортировочный ключ (если *+pos*). Если имеет место *-pos*, то отсчёт производится с конца поля. Если C опущено, то подразумевается первый символ поля. Если определён параметр **-b**, то часть C отсчитывается от первого непробельного символа данного поля (если *+pos* или от первого непробельного символа следующего за предыдущим полем для *-pos*. Сортировочный ключ может иметь свои параметры сортировки **Mbdfinr**. В этом случае глобальные параметры сортировки не применяются к данному полю. Параметр **-b** может быть использован отдельно и независимо для случаев *+pos* и *-pos*. Если параметр **-b** наследуется из глобальных параметров, то он используется во всех случаях. Ключи могут включать несколько полей. Ниже смотрите примеры сортировки.

-z Обработать вводной поток как набор строк, каждая из которых заканчивается символом <NUL>, а не <LF>. Этот параметр может оказаться полезным в специальных случаях, например, вместе с `perl -0` или `find -print0` или `xargs -0`.

Отсортировать численно в обратном порядке (в сторону уменьшения значения чисел):

```
ps | sort -rn
```

вы получите список ваших процессов упорядоченный по номеру процесса, максимальный номер будет вверху.

Отсортировать в алфавитном порядке опустив при этом первые 10 полей.

```
ps ux | sort -k 11
```

вы получите упорядоченный по алфавиту по 11-ому полю (имя программы в RedHat 5.1) список.

Отсортировать по алфавиту файл `/etc/passwd` по второй букве поля комментариев, а с одинаковыми вторыми символами отсортировать по первой букве

```
sort -t : -k 5.2,5.2 -k 5.1,5.1 /etc/passwd
```

Заметим, что если написать не 5.2,5.2, а просто 5.2, то в качестве сортировочного ключа будет взято пятое поле начиная с символа 2 (напомним, нумерация начинается с 1).

Отсортировать по алфавиту файл `/etc/passwd` по полю пять игнорируя лидирующие пробелы, а поля с равными значениями сортировать по числовому значению в сторону уменьшения по полю 3

```
sort -t : -k 5b,5 -k 3,3rn /etc/passwd
```

Заметим, что запись **-5b,5** означает произвести сортировку по полю 5. Если написать просто **-5b**, то в качестве ключа сортировки будет использована часть строки начиная с поля 5 до конца строки. Наконец, чтобы игнорировать как лидирующие пробелы в поле, так и замыкающие пробелы поля, можно записать

```
sort -t : -k 5b,5b -k 3,3rn /etc/passwd
```

или использовать глобальный параметр **-b**

```
sort -t : -b -k 5,5 -k 3,3rn /etc/passwd
```

10.18 uniq – вывод только уникальных строк

Использование программы:

uniq [*option*] [*input* [*output*]]

По умолчанию, программа **uniq** читает вводной поток, который предположительно является отсортированным в каком-то смысле (по алфавиту или в соответствии с числовыми значениями). Если встречаются одинаковые строки, то программа удаляет повторения, оставляя лишь по одной строке из любого количества повторяющихся строк. Однако возможны варианты (смотрите ниже описание параметров программы **uniq**). В данном разделе обсуждается тот вариант программы который отвечает на команду

```
uniq -version
```

строкой

```
uniq (GNU textutils) 1.22.
```

Вводной поток должен быть отсортирован. Если не определён параметр *output*, то вывод производится на стандартное устройство вывода. Если параметр *output*, то вывод производится в файл с именем *output*.

Программа воспринимает следующие параметры.

-*n*

-f *n*

-**skip-fields**=*n* Пропустить *n* полей в каждой строке вводного файла до проверки на уникальность. Поле вводной записи является последовательность символов, которая не содержит внутри себя пробелов или знаков табуляции <TAB>. Одно поле отделяется от другого одним или более знаками пробела или табуляции <TAB>.

+*n*

-s *n*

-**skip-chars**=*n* Пропустить *n* символов до проверки на уникальность. Если используется пропуск сразу двух вещей: пропуск полей и пропуск символов, то вначале выполняется пропуск полей, а потом пропуск символов.

-c

-**count** Вывести число раз, которое каждая строка встретилась во вводном файле.

-i

-**ignore-case** Игнорировать регистр, в котором представлены символы во входном файле.

-d

-**repeated** Вывести только повторяющиеся строки.

-u

-**unique** Напечатать только уникальные строки (не повторяющиеся).

-w *n*

-**check-chars**=*n* Сравнивать только *n* символов в каждой строке (после пропуска полей и символов). По умолчанию, после пропуска полей и символов, если таковые имели место, сравнивается остаток строки целиком.

Рассмотрим несколько простых примеров. Пусть у нас имеется тестовый файл с именем `T`, который содержит нижеследующее:

```
12311 aabcd
45611 dc09
78911 wigs
21311 anka
32211 after
98722 gens
```

Тогда команда

```
uniq -c T
```

даст

```
    1 12311 aabcd
    1 45611 dc09
    1 78911 wigs
    1 21311 anka
    1 32211 after
    1 98722 gens
```

Что совершенно неудивительно, ведь входной файл не отсортирован. Но если применить команду

```
uniq +3 -w 2 -c T
```

то получим результат

```
    5 12311 aabcd
    1 98722 gens
```

т.е. программа найдёт повторяющиеся части строк. Мы можем использовать команду

```
uniq -1 -w 2 -c T
```

и увидим, что программа пропустила только первое поле, но не пробел после него

```
      6 12311 aabcd
```

но если учесть это явление

```
uniq -1 +1 -w 1 -c T
```

то получим, что ожидалось

```
      1 12311 aabcd
      1 45611 dc09
      1 78911 wigs
      2 21311 anka
      1 98722 gens
```

10.19 comm – сравнить два отсортированных файла

Использование программы:

comm [*option*] *file1 file2*

предполагается, что оба вводных файла *file1* и *file2* уже являются отсортированными файлами. Без параметров *option* программа *comm* производит вывод в три колонки. Самая левая колонка состоит из строк, которые содержатся только в файле с именем *file1*. Колонка 2 (средняя) состоит из строк, которые содержатся только в файле с именем *file2*. Третья колонка состоит из строк, которые являются общими для файлов с именами *file1* и *file2*. Колонки разделены знаками <TAB>.

Могут использоваться следующие значения параметров: **-1**, **-2**, **-3**, которые означают запрет печати соответствующей колонки (нумерация колонок - слева направо).

Один из примеров использования состоит в сравнении содержимого каталога или библиотеки. Например, вы хотите сравнить два похожих на первый взгляд каталога: `/lib` и `/usr/lib`

```
ls -1 /lib     > /tmp/L_lib
ls -1 /usr/lib > /tmp/l_usr_lib
comm /tmp/L_lib /tmp/l_usr_lib | less
```

У меня получился выводной файл размером 335 строк и я обнаружил только один файл, который находился в обоих каталогах. Фрагмент вывода показан ниже

```
...
libc.a
libc.so
        libc.so.5
        libc.so.5.4.38
        libc.so.6
                libc5-compat
libc_nonshared.a
        libcom_err.so.2
        libcom_err.so.2.0
...
```

В третьей колонке оказался единственный файл с именем `libc5-compat`. Естественно, что это ничего не говорит о содержании файла с данным именем в разных каталогах.

Аналогичным образом можно сравнивать содержимое двух библиотек

```
ar -t /usr/lib/libc.a | sort > Dc
ar -t /usr/lib/libm.a | sort > Dm
comm Dc Dm
```

Показываем лишь фрагмент вывода

```
        s_ceill.o
s_chown.o
                s_copysign.o
                s_copysignf.o
                s_copysignl.o
        s_cos.o
        s_cosf.o
        s_cosl.o
```

Как видно имеются часть программ с именами, которые встречаются в обеих библиотеках. Как и в предыдущем примере, мы ничего не можем сказать о самих программах.

10.20 cut – напечатать указанные части строк вводного файла

Использование программы:

cut *option ... file ...*

программа выводит на устройство стандартного вывода часть каждой строки из входного потока.

В нижеследующей таблице используются обозначения: *byte-list*, *character-list*, *field-list*, которые представляют собой последовательность чисел — одно число или несколько чисел через запятую, а также числовой интервал — два числа разделённых знаком минус. Нумерация байтов, символов и полей начинается с 1. Можно использовать открытые интервалы, например, -7 означает 1-7 от 1 до 7, 9- означает от 9 до конца строки.

Программа **cut** воспринимает следующие параметры:

-b *byte-list*

-**bytes**=*byte-list* Вывести только байты в позициях перечисленных в *byte-list*. Символы <TAB> и <BS> рассматриваются как 1 байт.

-c *character-list*

-**characters**=*character-list* Вывести только символы в позициях перечисленных в *character-list*. Практически то же, что в параметре **-b**, но в будущем интернационализация может изменить это положение (новые коды символов занимают могут занимать несколько байтов).

-f *fiels-list*

-**fields**=*fiels-list* Выводить только поля указанные в *fiels-list*. По умолчанию разделителем полей является символ <TAB>.

-d *delim*

-**delimiter**=*delim* Используется совместно с параметром **-f**, поля будут разделяться первым символом строки *delim*. По умолчанию разделитель полей есть `<TAB>`.

-n Не разделять многобайтные символы (пока не используется).

-s

-**only-delimited** Используется совместно с параметром **-f**, не печатать строки, которые не содержат символ разделителя полей.

10.21 expand/unexpand – преобразовать табуляторы в пробелы и обратно

Использование программ:

expand [*option ...*] [*file ...*]
unexpand [*option ...*] [*file ...*]

Программа **expand** заменяет знаки табулятора `<TAB>` в строках, где они встретились, на столько пробелов, на сколько необходимо. Программа **unexpand** производит обратное преобразование, т.е. пытается заменить встреченные пробелы знаками `<TAB>`.

Следует заметить, что такие преобразования могут оказаться весьма не безобидными: несмотря на то, что внешний вид текста практически не изменится, изменится его объём, а некоторые файлы (например, **make**-файлы) могут оказаться неработоспособными после **expand**.

Программа **expand** воспринимает следующие параметры.

-*tab1*,*tab2* **...**

-t *tab1*,*tab2* ...

-**tabs**=*tab1*,*tab2* **...** Обработать знаки табулятора `<TAB>` со значениями колонок табуляции *tab1*,*tab2* (нумерация колонок начинается с 0) и т.д. Кроме того, заменить все знаки табулятора за пределами представленного списка знаками пробела. Если колонки табуляции указаны значениями параметров **-t** или -**tabs**=, то их можно разделять как запятыми, так и пробелами.

-i

-**initial** Преобразовать в пробелы только начальные табуляторы в строках, т.е. табуляторы, которые непосредственно предшествуют символам не совпадающим с пробелом или табулятором.

Программа **unexpand** по умолчанию заменяет в символы табулятора только начальные пробелы в каждой вводной строке. Символы `<BS>` уменьшают счётчик колонок для помещения знака `<TAB>`. По умолчанию, знаки табуляции устанавливаются через каждые 8 колонок. Программа воспринимает следующие параметры.

-*tab1*,*tab2* **...**

-t *tab1*,*tab2* ...

-tabs=*tab1*,*tab2* **...** Установить знаки табулятора <TAB> со значениями колонок табуляции *tab1*,*tab2* (нумерация колонок начинается с 0) и т.д. Кроме того, оставить без изменений все знаки пробелов и табуляторов за пределами представленного списка. Если колонки табуляции указаны значениями параметров **-t** или **-tabs**=, то их можно разделять как запятыми, так и пробелами.

-a

-all Заменить где возможно все пробелы знаками табуляции, а не только начальные пробелы в каждой строке.

Итак, пример. Нам надо было сравнить два каталога, чтобы установить есть ли в них общие имена

```
ls -1 /lib     > /tmp/Lib
ls -1 /usr/lib > /tmp/Ulib
comm /tmp/Lib /tmp/Ulib | expand --tabs=10,50 | cut -b 51-70 | uniq -c
```

Выполнив приведённую выше последовательность мы увидим, что имеется лишь одно общее имя в обоих каталогах `libc5-compat`.

10.22 tr – перекодировка и уплотнение последовательностей символов

Использование программы:

tr [*option*]... *set1* [*set2*]

Программа **tr** копирует входной поток со стандартного устройства ввода на стандартное устройство вывода производя при этом одно из преобразований:

- перекодировка символов и возможно уплотнение повторяющихся символов на выводе;
- уплотнение повторяющихся символов (если встречается подряд два или более одинаковых символа, программа заменяет их одним символом);
- удаление символов;
- удаление символов и уплотнение повторяющихся символов.

Значением параметра *set1* (и, возможно, *set2*) являются строки из символов, которые подвергаются процессу обработки.

Программа воспринимает следующие значения поля *option*.

-d

-**delete** Удалить из входного потока те символы, которые содержатся в строке *set1*.

-s

-**squeeze-repeats** Когда используется только этот параметр, то программа заменяет одним символом те повторяющиеся символы, которые содержатся в строке *set1*. Если используется комбинация параметров -**delete** и -**squeeze-repeats**, то программа производит удаление тех символов из входного потока, которые содержатся в *set1*, затем производится выбрасывание тех повторяющихся символов, которые содержатся в *set2*.

-c

-**complement** Использовать для операции те символы, которые не входят в строку *set1*.

-t

-**truncate-set1** Усечь строку *set1* сделав её равной по размеру строке *set2* (усечение выполняется с правого края строки).

10.22.1 Наборы символов

Формат значений параметров *set1* и *set2* повторяет формат регулярных выражений (см. раздел 10.2), однако эти наборы не являются регулярными выражениями, а являются последовательностями символов. Большинство символов просто обозначают самих себя, но используются некоторые комбинации, которые означают специальные символы. Некоторые комбинации используются только в *set1* и *set2* как показано ниже. Каждая специальная комбинация начинается обратным слешем (знак \).

`a` Знак Control-G.

`b` Знак Control-H.

`f` Знак Control-L.

`n` Знак Control-J.

`r` Знак Control-M.

`t` Знак Control-I.

`v` Знак Control-K.

`000` Знак с восьмеричным кодовым представлением 000.

Знак обратный слеш (\).

Если за обратным слешем будет следовать какой-то символ не указанный в таблице, то возникнет ошибка.

Интервалы. Обозначение `m-k` включает символы `m` и `k` и все символы между ними. Первый символ должен быть ранее второго по алфавиту иначе возникнет ошибка. Другой пример, запись `0-9` означает то же, что и 0123456789. Заметим, что **tr** не поддерживает синтаксис, когда интервалы заключаются в квадратные скобки.

Повторяющиеся символы. Запись `[C*n]` в *set2* означает, что символ `C` должен быть повторен `n` раз. Так, запись `[y*4]` обозначает то же, что `yyyy`. Запись `[C*]` в *set2* означает, что следует столько раз повторить символ `C`, чтобы сделать длину *set2* равной длине *set1*. Если `n` начинается с 0, то это воспринимается как восьмеричное число, в противном случае – как десятичное.

Классы символов. Запись `[:class:]` означает все символы из заранее определённого класса `class`. Принадлежность классу не предполагает какого-либо упорядочивания символов в классе, исключая классы `upper` и `lower`, в которых символы упорядочены по возрастанию значений кодов.

Имена классов, которые понимает программа **tr** даны ниже.

alnum Буквы и цифры.

alpha Буквы.

blank Пробел (по горизонтали).

cntrl Управляющие символы.

digit Цифры.

graph Изображаемые символы (не включая пробел).

lower Буквы на нижнем регистре.

print Изображаемые символы (включая пробел).

punct Символы знаков пунктуации.

space Горизонтальный или вертикальный пробел.

upper Буквы верхнего регистра.

xdigit Шестнадцатеричные цифры.

Классы эквивалентности. Запись `[=C=]` обозначает все символы, которые эквивалентны C в каком-то смысле. Появление классов эквивалентности является относительно новым намерением поддержать не английские алфавиты. Однако, по всей видимости, не существует стандартного способа определить классы или их содержание. Поэтому GNU **tr** не имеет полной реализации данных классов. Пока **tr** поддерживает классы эквивалентности, в каждом из которых имеется одна буква. Так, `[=c=]` означает **c**, а `[=b=]` означает *b* и т.д.

Все рассуждения, которые относятся к способу изображения символов, верны только для знаков латинского алфавита.

10.22.2 Перекодировка (трансляция) символов

Программа **tr** выполняет перекодировку в том случае, когда присутствуют *set1* и *set2*, но не присутствует параметр -**delete** (-**d**). Программа читает водной поток символов и перекодирует только те символы входного потока, которые находятся в строке *set1*. Каждый символ входного потока, который присутствует в *set1*, перекодируется в соответствующий символ *set2* (первый в первый, второй во второй и т.п.). Если символ входного потока не представлен в строке *set1*, то он передаётся на устройство стандартного вывода без изменений. Если любой символ встречается в строке *set1* дважды, то берётся последнее значение. Например. нижеследующие две команды просто эквивалентны:

```
tr mmmm 1234
tr m 4
```

Приведём пример различных вариантов перекодировки прописных букв в строчные:

```
tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
tr a-z A-Z
tr '[:lower:]' '[:upper:]'
```

Все варианты эквивалентны. Напомним, что это верно для латинского алфавита.

Когда происходит перекодировка, то строки *set1* и *set2* обычно равны по длине. Если эти строки не равны по длине, то возникают неоднозначности, которые могут интерпретироваться в различных версиях по разному, т.е. о портабильности использования **tr** с параметрами *set1* и *set2* разной длины следует заботиться отдельно.

Конкретная программа GNU **tr** ведёт себя следующим образом:

- если *set1* короче *set2*, то остаток *set2* просто игнорируется;
- если *set1* длиннее *set2*, то по умолчанию **tr** "растягивает"*set2 до длины set1 путём дублирования последнего символа set2 столько раз сколько необходимо.*
- *если set1 длиннее set2 и использован параметр -truncate-set1 (*-**t***), то set1 будет усечена до длины set2. Этот параметр действует только для операции перекодировки.*

Рассмотрим примеры работы программы **tr**. Пусть файл с именем `T` содержит строку `\subsection{Простые примеры}`.

```
head T | tr -cs A-Za-z0-9 '\012'
head T | tr -tcs A-Za-z0-9 '\012'
```

В первом случае мы получим

```
subsection
```

а результатом второй строки будет

```
\subsection{Простые примеры}
```

В первой строке восьмеричная кодовая комбинация `<LF>` расширена до размеров `A-Za-z0-9`, все символы не совпадающие с `A-Za-z0-9` перекодированы в `<LF>` (в данном случае все специальные знаки и символы Кириллицы), из выводного потока удалены идущие подряд символы `<LF>`.

Во втором случае первый параметр *set1* усечен до одного символа (до длины *set2*), а первым символом является восьмеричный код `000`. Поскольку в вводном файле не было символа с кодом `000`, то мы и не видим изменений.

10.22.3 Уплотнение и удаление символов

Уплотнение символов подразумевает, что программа **tr** заменяет любые два и более одинаковых символа на один символ. Например, заменить все повторяющиеся переводы строк на один перевод строки:

```
tr -s '\n'
```

Удалить все символы с восьмеричным кодом `000`:

```
tr -d '\000'
```

Преобразовать все знаки кроме букв и цифр в знаки <LF> и ужать повторяющиеся знаки <LF>:

```
tr -cs '[a-zA-Z0-9]' '[\n*]'
```

10.22.4 Диагностические сообщения

В ряде случаев программа **tr** выдаёт диагностические сообщения, которые не оговорены стандартом **POSIX**. Чтобы избежать этих сообщений следует установить переменную окружения POSIXLY_CORRECT.

10.23 Объединить строки из разных файлов

Использование программы:

paste [*option*]... [*file*]...

Программа читает указанные файлы (один или больше) и выводит на устройство стандартного вывода строки текста, которые состоят по умолчанию из объединённых строк вводных файлов – первая выводная строка составляется из первых строк вводных файлов разделённых символами <TAB>, вторая строка – из вторых и т.п. Программа распознаёт несколько параметров.

-s

-serial Выводить последовательно строки одного файла, а лишь после второго и т.д. при этом все строки первого файла будут объединены (между строками будет вставлен знак табуляции <TAB>) в одну длинную выводную строку.

-d *delim-list*

-delimiters *delim-list* Последовательно использовать символы из *delim-list* вместо символа <TAB>, чтобы отделить объединяемые строки файлов. Если *delim-list* исчерпан, то он начинает использоваться сначала.

10.24 Объединить строки по общим полям

Использование программы:

join [*option*]... *file1 file2*

Файлы с именами *file1* и *file2* должны содержать уже отсортированные файлы в возрастающем порядке (не числовом) по объединяемым полям. Если не указан параметр **-t** они должны быть отсортированы игнорируя пробелы в начале объединяемого поля, как после программы **sort** с параметром **-b**. Если используется параметр -**ignore-case**, то строки должны быть отсортированы невзирая на регистр в объединяемом поле, как после программы **sort** с параметром **-f**.

По умолчанию: объединяемое поле является первым полем в каждой строке; поля на вводе разделены одним или более пробелами, ведущие пробелы в строках игнорируются; поля на выводе отделяются пробелом; каждая выводная строка состоит из объединяемого поля, за которым следует остаток строки из файла *file1*, а затем остаток строки из файла *file2*.

Программа **join** воспринимает следующие значения поля *option*.

-a *file-number* Вывести строку для каждой непарной строки в файле *file-1* или *file-2* в дополнение к обычному выводу.

-e *string* Заместить отсутствующие поля на вводе подстрокой *string*.

-i

-**ignore-case** Игнорировать различие регистров (т.е. **g** должно означать то же, что **G**). Вводные строки должны быть отсортированы тем же способом. Для такой сортировки можно использовать **sort -f**.

-1 *field*

-j1 *field* Объединить по полю *field* (положительное целое) файла *file1*.

-2 *field*

-j2 *field* Объединить по полю *field* (положительное целое) файла *file2*.

-j *field* Эквивалентно **-1** *field* и **-2** *field*.

-o *field-list*... Сконструировать каждую выводную строку в соответствии с форматом *field-list*. Каждый элемент *field-list* может состоять из одиночного символа **O** или имеет форму `M.N`, где `M` может быть 1 или 2 (номер файла), а `N` – это номер поля.

Поле, обозначенное как **O**, является полем объединения. В большинстве случает функциональность спецификации с помощью **O** может быть реализована с помощью явного описания типа `M.N`, которое соответствует полю объединения. Однако, если встречаются непарные строки во вводных файлах, то описание типа `M.N` не имеет возможности описать поле объединения.

Элементы в *field-list* разделены запятыми или пробелами. После параметра **-o** можно использовать несколько полей *field-list*; все они будут соединены вместе. Все выводные строки включая те, которые сгенерированы параметрами **-a** и **-v**, являются объектами описания в *field-list*.

-t *char* Использовать символ *char* в качестве символа разделителя полей на вводе и выводе.

-v *file-number* Вывести строку для каждой непарной вводной строки в файле *file-number* (1 или 2) вместо обычного вывода.

Глава 11

Потоковый редактор текста

11.1 Введение

sed – это не интерактивный потоковый редактор текста (sequential editor – последовательный редактор). Его основное назначение выполнять относительно простые операции преобразования текста, которые можно выполнить за один проход. Вводом для программы **sed** является файл или устройство стандартного ввода. Выводом является устройство стандартного вывода, т.е. **sed** – фильтр. Команды редактирования могут быть как в командной строке вызова **sed**, так и в файле, имя которого указано в командной строке. Из команд редактирования могут быть составлены программы.

В данной книге обсуждается версия **sed**, которая отвечает на команду `sed --version` следующее

```
GNU sed version 3.02

Copyright (C) 1998 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE,
to the extent permitted by law.
```

11.2 Вызов sed

Использование в командной строке:

`sed [OPTION]... {script-only-if-no-other-script} [input-file]...`

При вызове **sed** могут быть использованы следующие параметры:

-V

-version ␣

Вывести версию программы **sed**.

-h

-help ␣

Вывести краткие сведения о формате параметров **sed**.

-n

-quiet

-silent ␣

По умолчанию, **sed** выводит шаблон в конце цикла прохождения вводного файла. Данный параметр запрещает этот вывод, если в командах редактирование не использована команда **p**.

-e *script*

-expression=*script* ␣

Добавить команды редактирования в *script* к командам, которые будут выполняться во время обработки вводного потока.

-f *script-file*

-file=*script-file* ␣

Добавить команды, содержащиеся в файле *script-file*, к командам, которые будут выполняться во время обработки вводного потока.

Если в командной строке не встретилось параметров **-file**=, **-f**, **-e**, **-expression**=, тогда первый встреченный аргумент рассматривается как команда(ы) редактирования вводного потока.

Если после обработки параметров и команд редактирования в командной строке есть аргументы, то они рассматриваются как имена файлов, который следует отредактировать. Если имя файла представлено в виде знака **-** (минус) или оно отсутствует вовсе, то вводной поток ожидается со стандартного устройства ввода.

11.3 Команды редактирования sed

Программа редактирования, которую интерпретирует **sed**, состоит из одной или более команд редактирования, которые могут находиться как в командной строке, так и в файле, который указан в командной строке (параметры **-e**, **-f**, -**expression**, -**file**). В данном разделе мы будем иметь в виду одну программу или скрипт для редактирования, который является сцеплением всех отдельных команд и скриптов редактирования упомянутых в командной строке (неважно непосредственно в строке или в файле).

Каждая команда редактирования состоит из следующих частей:

- возможного адреса во вводном тексте или адресного интервала,
- за которым следует одно буквенное имя команды и, возможно, дополнительные коды детализирующие условия выполнения команды.

11.3.1 Адрес во вводном тексте

Адресация элементов текста может быть в одной из нижеследующих форм:

number ␣ Определение номера вводной строки *number* указывает только одну строку с номером *number*. Заметим, что **sed** постоянно считает номера строк начиная с первой введённой строки вне зависимости от количества вводных файлов.

first~step ␣

Это расширение **GNU** означает, что данному выражению удовлетворяют каждая *step*-ная строка, которая начинается с подстроки *first*. В частности, будут выбраны строки из вводного файла, если существует такое неотрицательное *n*, что номер текущей строки равен *first*+(*n***step*). Таким образом, чтобы выбрать все неётные строки, следует использовать выражение `1~2`. А если вы хотите выбрать каждую третью строку начиная со 2-ой строки, то следует использовать `2~3`. Если вы захотите указать каждую пятую строку начиная с 10-ой, то – `10~5`. Наконец, странное выражение `45~0` будет означать 45-ую строку.

$ ␣

Этот символ означает последнюю строку в последнем вводимом файле.

/regexp/ ␣

Этот адрес соответствует всем строкам вводного файла, содержание которых удовлетворяет регулярному выражению *regexp* (смотрите также 10.2). Если внутри регулярного выражения должна использоваться косая черта (`/`), то следует использовать комбинацию `\/`.

\%*regexp*% ␣

Такое выражение также соответствует всем строкам вводного файла, содержание которых удовлетворяет регулярному выражению *regexp*. Здесь на месте знака % может быть любой символ. Такая запись оказывается удобнее, если, например, вам требуется использовать много слешей внутри *regexp*.

/*regexp*/**I**

\%*regexp*%**I** ␣

Модификатор **I** (расширение **GNU**) используется, чтобы сделать поиск *regexp* независимым от регистра, в которых символы представлены во вводном потоке.

Если не указано никакого адреса, то все строки вводного потока подвергаются действию команд(ы) редактирования. Если имеется один адрес, то лишь те строки подвергается действию команд редактирования, которые удовлетворяют адресу.

Адресный интервал может быть обозначен как пара адресов, разделённых запятой. Адресный интервал включает все строки вводного файла включительно начиная со строки, соответствующей первому адресу, до строки, соответствующей второму адресу. Если второй адрес является регулярным выражением *regexp*, тогда проверка на соответствие начнётся со строки следующей за первым адресом. Если второй адрес есть целое n, которое меньше или равно первому адресу, то лишь одна строка будет считаться соответствующей адресному интервалу.

Добавление символа **!** (восклицательный знак) в конце адресной спецификации означает отрицание значения соответствия. Таким образом, если знак **!** следует после адресного интервала, то всё выражение будет означать все строки **не** попадающие в адресный интервал.

11.3.2 Буферы данных sed

sed использует два буфера для обработки вводного потока: основной буфер обработки (pattern buffer) и дополнительный буфер (hold buffer). В обычном режиме **sed** читает строки вводного потока и помещает из в основной буфер; там же производятся операции редактирования. Дополнительный буфер изначально пуст, но ряд команд позволяют перемещать данные из одного буфера в другой.

11.3.3 Часто используемые команды

␣

Комментарий; продолжается до конца строки (до символа `<NL>`). Предупреждение:

Если вам необходимо заботиться о переносе ваших скриптов на другие операционные платформы, то заметим, что в ряде реализаций **sed** интерпретация строк комментариев может отличаться от описанной здесь. В частности, может допускаться лишь одна строка комментариев, которая должна содержать знак # (решётка) только в первой слева позиции.

В первой строке скрипта **sed** два символа идущие подряд #**n** имеют специальное значение: будет включён режим **no-autoprint**.

s/*regexp*/*replacement*/*flags* ␣

sed будет искать части строк вводного потока, которые соответствуют регулярному выражению *regexp* и производить замены этих частей на значение *replacement* в соответствии со значением *flags*.

Значение *replacement* может содержать выражения вида `\n`, где `n` есть целое от 0 до 9. Такое выражение означает ту порцию соответствия, которая заключена в `n`-ные по порядку специальные скобки вида `\(` и `\)`. Кроме этого, внутри выражения *replacement* может содержаться символ `&`, который ссылается на полную подстроку соответствующую *regexp* в основном буфере. Если вы хотите ввести в *replacement* литеральные символы `&`, `\n` или `\`, то следует использовать перед ними знак `\` (обратный слеш). Пример:

```
$ echo "Жужжали" | sed -n 's/Жуж/Гуж<&>/p'
Гуж<Жуж>жали
```

Ещё пример:

```
$ echo "Жужжали Бабочки" | sed -n 's/\(жж\)\(али\)/Гуж<&>\2\1/p'
ЖуГуж<жжали>алижж Бабочки
```

Замечания.

Символ / (наклонная черта) может быть заменен на любой другой символ в пределах одной команды **s**.

Символ / (наклонная черта) может использоваться внутри выражения *regexp*, если ему предшествует знак \. Символ конца строки `<NL>` может использоваться внутри *regexp* с использованием последовательности из двух символов `\n` (обратный слеш и n).

За командой редактирования **s** могут следовать (или не следовать) флаги, к обсуждению которых мы переходим.

g ␣

Произвести замену во всех местах, а не только первой встреченной подстроки в строке.

p ␣

Если замена имела место, то вывести результат.

number ␣

Заменить только *number*-ное соответствие *regexp*.

w *file-name* ␣

Если подстановка имела место, то результат записать в файл с именем *file-name*.

I ␣

Проверять соответствие *regexp* вне зависимости от регистра (расширение **GNU**).

q ␣

Выйти из **sed** без обработки остальных команд редактирования или вводного потока.

d ␣

Очистить основной буфер и немедленно начать новый цикл обработки (чтение вводного потока, редактирование и т.д.).

p ␣

Вывести основной буфер (на устройство стандартного вывода). Эта команда используется обычно в сочетании с параметром **-n** в командной строке.

Замечание.

Некоторые реализации **sed**, такие как описываемая, будут печатать все строки вводного файла дважды, если режим **auto-print** не отключён и использована команда **p**. Другие варианты **sed** могут печатать каждую строку лишь однажды. Оба способа поведения **sed** не являются ошибкой.

n ␣

Если режим **auto-print** не выключен, вывести основной буфер на устройство стандартного вывода, затем ввести в основной буфер следующую строку из вводного потока. Если вводной поток завершён (больше нет записей на вводе), тогда **sed** завершает своё выполнение без обработки остальных команд в случае их наличия.

{*commands*} Группа команд редактирования *commands* может быть заключена в фигурные скобки.

11.3.4 Прочие команды sed

Здесь обсуждаются команды **sed**, которые тоже очень полезны при составлении скриптов и редактировании текстов внутри скриптов.

y/*source-chers*/*dest-chars*/ ␣

Заменить во водном потоке любой символ из *source-chers* на соответствующий ему символ из *dest-chars*. Другими словами, третий символ из *source-chers* заменить на третий символ из *dest-chars*. Строки *source-chers* и *dest-chars* **должны** содержать одинаковое число символов. Например, команда

```
echo жужжал | sed y/жул/gul/
```

даст в результате

```
guggal
```

a ␣

(Разрешается максимум один адрес.)

Строки следующие за этой командой будут помещены за обрабатываемой строкой. Все строки кроме последней должны заканчиваться знаком \ (обратный слеш). Например:

```
$ cat t
3 a\
Да вы переполнили \
все \
файловые системы

$ df | awk '{print $5, $6}' | sed -f t
Use% Mounted
```

```
12% /
33% /boot
Да вы переполнили
все
файловые системы
85% /data03
53% /data04
74% /data05
86% /usr
56% /var
99% /data02
```

Как видим, строки из файла `t` по команде **a** были введены после третьей строки в результирующем выводе.

i\\ ␣

(Разрешается использовать один адрес.)

Немедленно вывести строки текста, следующие за этой командой. Все строки кроме последней должны заканчиваться знаком \ (обратный слеш). Например:

```
$ cat t
1 i\
Да вы переполнили \
все файловые системы

$ df | awk '{print $5, $6}' | sed -f t
Да вы переполнили
всее файловые системы
Use% Mounted
12% /
33% /boot
85% /data03
53% /data04
74% /data05
86% /usr
56% /var
99% /data02
```

Как видим текст, следующий за командой **i**, введён **до** заданной строки. Если не задать адрес, то перед каждой строкой из вводного файла будут выводиться строки, следующие за командой **i**.

c ␣

(Разрешается использовать два адреса.)

Удалить из вводного потока строки соответствующие интервалу заданному адресами, а вместо них вывести строки, следующие после команды **c**. Все строки кроме последней должны заканчиваться знаком \ (обратный слеш). Например:

```
$ cat t
1,5 c\
Это будет новый заголовок

$df  | awk '{print $5, $6}' | sed -f t
Это будет новый заголовок
74% /data05
86% /usr
56% /var
99% /data02
```

Как видим, строки с 1 по 5 включительно были заменены на новую строку. Посмотрим, что будет, если не использовать адресный интервал:

```
$ cat t
c\
Это будет новый заголовок

$df  | awk '{print $5, $6}' | sed -f t
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
Это будет новый заголовок
```

Здесь адрес не был указан, следовательно **sed** заменил каждую строку из вводного потока на строку, следующую после команды **c**.

= ␣ Вывести номер текущей вводной строки с последующим символом `<NL>`. Например,

```
$ df  | awk '{print $6}' | sed =
1
Mounted
2
/
3
/boot
4
/data03
5
/data04
6
/data05
7
/usr
8
/var
9
/data02
```

l ␣

Вывести основной буфер в стандартной форме: не изображаемые символы выводятся в виде восьмеричных кодов, которым предшествует обратный слеш; длинные строки будут разбиты на более мелкие строки (места разбиения будут показаны обратными слешами); конец строки отмечается символом $ (знак доллар). Например,

```
$ echo "ЖужжалиБабочкиЖужжалиБабочки" | sed -n l
\366\325\326\326\301\314\311\342\301\302\317\336\313\311\366\325\326\
\326\301\314\311\342\301\302\317\336\313\311$
```

Как видим Кириллица распознаётся как не изображаемые символы.

r *filename* ␣

Прочесть файл с именем *filename* и поместить его в выводной поток в очередном цикле чтения вводного потока. Если файл с именем *filename* не может быть прочитан по любым причинам, то не выдаётся никакой диагностики, а файл рассматривается как файл с нулевой длиной.

w*filename* ␣

Записать основной буфер в файл с именем *filename* (мне удалось проверить эту команду, когда я убрал пробел между командой и именем файла).

D ␣

Удалить текст из основного буфера во первого конца строки. Если в основном буфере остался какой-то текст, то перейти к его обработке. В противном случае перейти к новому циклу, т.е. чтению очередной строки из вводного потока и т.д.

N ␣

Добавить `<NL>` в основной буфер, затем поместить туда очередную строку из вводного потока. Если вводной поток исчерпан, то завершить **sed** без обработки оставшихся команд.

P ␣

Вывести порцию основного буфера до первого символа `<NL>`.

h ␣

Заместить текущее содержание дополнительного буфера содержанием основного буфера.

H ␣

Добавить `<NL>` к содержимому дополнительного буфера. Затем добавить содержимое основного буфера к содержимому дополнительного.

g ␣

Заместить содержимое основного буфера содержимым дополнительного буфера.

G ␣

Добавить `<NL>` к содержимому основного буфера. Затем добавить содержимое дополнительного буфера к содержимому основного.

x ␣

Обменять содержимое основного буфера и содержимое дополнительного буфера.

11.3.5 Команды программирования, которые понимает sed

Использование команд программирования может потребоваться в специальных случаях.

: *label* ␣

(Не разрешается использовать поле адреса.)

Определить положение метки с именем *label*.

b *[label]* ␣

Безусловный переход к метке с именем *label*. Если имя метки опущено, то производится переход к следующему циклу обработки вводных строк.

t *[label]* ␣

Условный переход к метке с именем *label*. Переход выполняется в том случае успешного выполнения команды **s** (подстановки) с момента последнего ввода очередной строки текста или выполнялась команда условного перехода **t**. Если имя метки опущено, то производится переход к следующему циклу обработки вводного потока.

.

Пример простой несколько искусственной программы, которая заменяет первый встреченный в вводном потоке знак `/` (слеш) на букву `Щ` и завершает работу.

```
$ cat t
s/\//Щ/p
t Mylabel
b
: Mylabel
q

$ df | awk '{print $6}' | sed -n -f t
Щdev/sda6              497636    55293   416643  12% /
```

Здесь, в соответствии с командой **s**. заменяется только первый встреченный слеш. Далее производится условный переход к метке с именем `Mylabel` и завершается выполнение **sed**. До тех пор пока слеш не встретился, выполнение после команды `t Mylabel` продолжается и интерпретируется следующая команда `b`, т.е. снова вводится очередная строка вводного потока и над ней производится команда `s/\//Щ/p`.

Глава 12

Подсистема сканирования, анализа и обработки текстов – awk

12.1 Введение

Программа **awk** является интерпретатором специального языка **awk** для контекстного поиска и анализа текста. **awk** была написана в 70-х годах основателями операционной системы **Unix**. Позже она была переписана в свете проекта **GNU**. Вариант программы **awk** в проекте **GNU** естественно имеет имя **gawk**. Поскольку практически на всех установках Linux имя **awk** есть просто символический линк к программе **gawk**, поэтому мы будем именовать эту программу в данной книге – **awk**. Подробное описание программы следует искать на страницах руководств

```
man awk
info awk
```

или в книгах на эту тему, например, **The AWK Programming Language**, где авторами были A. V. Aho, P. J. Weinberger, B. W. Kernighan (Addison-Wesley, 1988, ISBN 0-201-07981-X), или книга Dale Doughberry **Sed and Awk** (ISBN: 0-937175-59-5, 1992, O"Railly). Полезная информация может быть почерпнута в *http://mx.nsu.ru/FAQ/F-computer-lang-awk/* и в других местах в Интернет. Здесь рассматривается версия **awk**, которая отвечает на команду

```
$ awk --version
```

```
GNU Awk 3.0.3
```

Основное назначение программы – интерпретация языка программирования **awk**, который благодаря простоте позволяет быстро создать программу прототип для анализа и преобразования текстовых файлов. Преобладающим сценарием или контекстом использования **awk** является следующая **awk**–программа:

шаблон `{действие}`
шаблон `{действие}`
и т.д. и т.п.

В соответствии с этим сценарием программа читает вводной файл (или устройство стандартного ввода) и анализирует каждую строку на предмет соответствия данному шаблону. Если соответствие имеет место, то производятся действия, описанные в фигурных скобках (быть может весьма нетривиальные). Могут встречаться частные случаи, например, когда **шаблон** отсутствует, т.е. действие применяется по отношению к каждой строке вводного файла. Может отсутствовать `{действие}`, тогда все строки соответствующие шаблону выводятся на стандартное устройство вывода. Программа **awk** есть фильтр.

12.2 Простые примеры использования awk

Например, необходимо напечатать третье поле записи (третью колонку, или третье слово записи, т.е. имя пользователя) следующего вводного текста:

```
ls -l | awk '{print($3)}'
```

Легко понять, что данный пример можно использовать для определения кому принадлежат файлы в текущем каталоге

```
ls -l | awk '{print($3)}' | sort | uniq -c
```

В результате будет напечатан список пользователей владельцев файлов в текущем каталоге и сколько файлов принадлежат каждому пользователю. А если требуется напечатать сначала девятую, а затем первую колонку (имя файла и его описатель), то можно сделать так:

```
ls -l | awk '{print($9,$1)}'
```

Для вставки дополнительных пробелов между выводными колонками используем

```
ls -l | awk '{print($9,"  ",$1)}'
```

или даже так

```
ls -l | awk '{print("Колонка 5=",$5," А это колонка 1 ",$1)}'
```

В предыдущих примерах предполагалось, что разделителем полей является пробел. Если используется другой символ, то это можно указать

```
awk -F: '{print($5,"  ",$1)}' FileName
```

Здесь двоеточие используется в качестве разделителя полей вводного файла. Наконец, если использовать оператор `print` без параметров, то будет напечатана полностью вводная строка

`awk '{print}' FileName`

Если вы захотите использовать готовый к интерпретации файл, содержащий программу на языке **awk**, то можно поступить двояко. Один из способов:

`awk -f awkprogram FileName` Здесь файл с именем `awkprogram` содержит программу на языке **awk**, а файл с именем `FileName` содержит анализируемый текст. Второй способ состоит в использовании нотации `#!/usr/bin/awk -f`, например,

`AwkProgram FileName`

где файл с именем `AwkProgram` содержит, к примеру, нижеследующее

```
#!/usr/bin/awk -f
{print($3)}
```

А файл с именем `FileName` содержит анализируемый текст.

12.3 Параметры в командной строке

Программа **GNU awk** воспринимает следующие параметры.

-F *fs*

-**field-separator** *fs* ␣

Использовать значение *fs* в качестве символа разделителя полей.

-v var=*val*

-**assign var**=*val* ␣

Переменной с именем var присвоить значение *var* до того, как начнётся выполнение программы **awk**. Значения таких переменных будут доступны уже в блоке **BEGIN**.

-f *program-file*

-**file** *program-file* ␣

Читать исходный текст программы на языке **awk** из файла с именем *program-file*. Можно использовать несколько параметров **-f** и, соответственно, несколько таких файлов. Таким образом, можно иметь библиотеку полезных **awk**–программ.

-W lint

-lint ␣

Обеспечить диагностику, если использованные конструкции языка **awk** не являются переносимыми в другие реализации **awk**.

-W re-interval

-re-interval ␣

Разрешить использование ШАБЛОННЫХ ИНТЕРВАЛОВ (смотрите раздел 12.8).

-W source *program-text*

-source *program-text* ␣

Рассматривать *program-text* как исходный текст на языке **awk**. Этот параметр можно использовать совместно с параметром **-f**. Такое совместное использование позволяет интенсивно использовать библиотеки **awk**–программ совместно с **awk**–операторами в командной строке. Пример:

```
ls -l | awk --source '{print}' --file library-file
```

здесь выполняется сначала программа **awk** из командной строки (в примере одна команда **print**), а затем – **awk**–программа из файла `library-file`.

Подробнее смотрите описание параметров в
`info gawk`

12.4 Переменные, записи, поля

12.4.1 Переменные

Переменные **awk** являются динамическими; они создаются в момент первого обращения к ним. Значениями переменных могут быть целые, плавающие числа или символьные строки, что определяется контекстом использования переменной. Имеют место одномерные массивы. Многомерные массивы могут быть смоделированы с помощью одномерных массивов. Несколько переменных имеющих специальное значение устанавливаются по ходу выполнения программы **awk**. Имена и значения таких переменных перечислены в разделе 12.5.

12.4.2 Записи

Обычно предполагается, что записи отделены одна от другой символами `<NL>`. В специальной переменной RS может быть установлен другой символ разделения записей. Значения, содержащиеся в RS, могут интерпретироваться по-разному. Если RS содержит одиночный символ, то это есть символ разделения записей. Если RS содержит более, чем один символ, то это регулярное выражение. Вводной текст, который удовлетворяет регулярному выражению является разделителем записей. Для более детального изучения свойств RS следует обратиться к `info awk` или `man awk`.

12.4.3 Поля

Программа **awk** читает вводной файл по записям. Каждую запись **awk** разбивает на поля или слова, используя символ разделения полей, который содержится в переменной FS. Если FS содержит один символ, то этот символ является разделителем полей. Если FS содержит нулевую строку, то каждый символ введённой записи становится отдельным полем. Если FS содержит более чем один символ, то эта строка рассматривается как регулярное выражение, которое используется для разбиения введённой записи.

12.5 Встроенные переменные awk

Программа **awk** позволяет использовать ряд встроенных переменных. Следует заметить, что речь не идёт о переменных окружения в текущей оболочке, а о переменных, которые могут быть использованы в программах на языке **awk**.

ARGC ␣

Число аргументов в командной строке. Это число не включает параметры программы **gawk** или текст программы на языке **gawk**.

ARGIND ␣

Содержит индекс в массиве ARGV имени текущего файла.

ARGV ␣

Массив аргументов командной строки. Индекс может быть от 0 до ARGC-1.

CONVFMT ␣

Содержит формат преобразования чисел, по умолчанию `%.6g`.

ENVIRON ␣

Массив содержащий значения текущего окружения. Это ассоциативный массив, индексами которого являются имена соответствующих переменных окружения. Например, значением **ENVIRON["HOME"]** может быть `/home/shevel`.

ERRNO ␣

Содержит код ошибки, если она возникла во время ввода.

FILENAME ␣

Содержит текущее имя вводного файла.

FNR ␣

Содержит текущий номер записи в текущем вводном файле.

FS ␣

Содержит символ разделителя полей.

IGNORECASE ␣

Если содержит не нулевое значение, то все поисковые операции **awk** производятся без учёта регистра, в котором введена конкретная буква. Следует заметить, что это верно в основном для набора символов **ISO 8859-1 Latin-1**.

NF ␣

Количество полей, на которые разбита текущая строка.

NR ␣

Количество записей обработанных до текущего момента.

OFMT ␣

Выводной формат для чисел. По умолчанию `%.6g`.

OFS ␣

Символ разделителя полей на выводе.

RS ␣

Символ разделителя записей на вводе.

RT ␣

Терминатор записи. **gawk** устанавливает RT равным последовательности входного потока, которая удовлетворяет символу или регулярному выражению определённому в RS.

RSTART ␣

Индекс первого символа, найденного функцией **match()** или 0, если нет соответствия.

RLENGTH ␣

Длина строки найденной с помощью функции **match()** или -1, если ничего не найдено.

SUBSEP ␣

Символ используемый для разделения нескольких индексов в элементах массива. по умолчанию значение SUBSEP равно \034.

12.6 Массивы

Как уже упоминалось ранее, массивы в языке **awk** являются одномерными. Массивы являются ассоциативными массивами, т.е. значениями индексов являются строками (последовательности символов). Индексы массива располагаются в квадратных скобках. Например, выражение

```
k="A"; l="B";: \
x[k, l] = "Привет ребята \n"
```

присвоит строку `Привет ребята \n` элементу массива `x`, который имеет индекс в виде строки `A\034B`, т.е. сцепленные символы `A` и `B` и значение переменной SUBSEP между ними.

Можно использовать следующие конструкции в условных операторах **awk**

```
if (Val in array) print array[Val]
```

т.е. если в массиве `array` имеется индекс `Val`, то вывести значение элемента массива соответствующего индексу `Val`. Если массив имеет индексы как показано выше [k, l], то следует использовать:

```
   if ((k, l) in array) print array[k, l]
```

12.7 Встроенные функции языка awk

Язык **awk** имеет набор встроенных функций для выполнения некоторых в известном смысле СТАНДАРТНЫХ операций.

12.7.1 Строковые функции

gensub(*r, s, h [, t]*) ⌣

предназначена для поиска в строке *t* последовательности, которая удовлетворяет регулярному выражению *r*. Если строка *h* содержит в качестве первого символа знак **g** или **G**, то если найдена последовательность удовлетворяющая *r*, то она заменяется на *s*. В противном случае, строка *h* должно быть числом, указывающим, которая по порядку последовательность удовлетворяющая выражению *r* должна быть заменена на *s*. Если строка *t* опущена, то вместо неё используется **$0**.

В пределах текста строки *s* может быть использованы комбинации вида `\n`, где `n` есть целое от 0 до 9 включительно. Такого рода комбинация применяется, чтобы указать текст в n-ных по порядку скобках. Комбинация `\0` означает полный текст соответствующий регулярному выражению *r*, как это делает символ `&`.

В отличие от **sub()** и **gsub()** здесь модифицированная строка возвращается как результат функции не изменяя при этом строку *t*.

gsub(*r, s [, t]*) ⌣

Каждую подстроку в строке *t* удовлетворяющую выражению *r* заменить на строку *s*. В качестве значения функции возвращается количество произведённых замен. Если строка *t* опущена, то используется **$0**. Знак `&` (амперсанд) в замещающем тексте заменяется на текст удовлетворяющий выражению *r*. Если вы хотите использовать литеральное значение амперсанда, то следует использовать `\&`. Перед интенсивным использованием знака `&` (амперсанд) полезно прочесть разделы описания языка **awk** касающиеся использования специальных комбинаций символов, начинающихся обратной косой чертой (\).

index(*s, t*) ⌣

Вернуть индекс (смещение от начала) строки *t* в строке *s*, если строка *t* содержится в строке *s*. В противном случае вернуть 0.

length(*[s]*) ␣

Вернуть длину строки *s* или длину **$0**, если *s* опущено.

match(*s, r*) ␣

Вернуть позицию, с которой начинается последовательность удовлетворяющая регулярному выражению *r* или 0, если такой последовательности нет, а также установить переменные RSTART и RLENGTH.

split(*s, a [, r]*) ␣

Разбить строку *s* на части, которые присвоить элементам массива *a*. Разбивание строки *s* произвести в соответствии с регулярным выражением *r*. Если *r* опущено, то в качестве разделителя используется значение переменной FS. Массив *a* очищается перед разбиванием строки *s*.

sprintf(*fmt, expr-list*) ␣

Вернуть сформированную строку для печати *expr-list* в соответствии с форматом *fmt*. Например

```
awk -v A=192 '{print(sprintf("%20d",A))}' t
```

где *t* – имя файла, `A` – имя переменной, которой присваивается значение 192, а затем печатается в соответствии с форматом `%20d`.

sub(*r, s [, t]*) ␣

Почти то же, что **gsub()**, но заменяется только первая подходящая подстрока.

substr(*s, i [, n]*) ␣

Вернуть подстроку, которая начинается в позиции *i* строки *s* с длиной максимум *n* символов. Если *n* опущено, то возвращается часть *s* начиная с позиции *i*.

tolower(*str*) ␣

Вернуть строку *str* с символами преобразованными в строчные буквы. Неизображаемые символы или числа не преобразуются.

toupper(*str*) ␣

Вернуть строку *str* с символами преобразованными в заглавные (прописные) буквы.

12.7.2 Функции времени

Для получения времени программа **awk** имеет две функции:

systime() ⌣

Возвращает текущее время в секундах начиная с 1 января 1970 года.

strftime(*[format [, timestamp]]***)** ⌣

Возвращает отформатированную отметку времени (`timestamp`) в соответствии с форматом `format`. Строка отметки времени должна иметь вид как после функции **systime()**. Если строка `timestamp` опущена, то используется текущее время. Если строка `format` опущена, то используется формат, который применяется в программе **date**. Для получения списка допустимых форматов можно обратиться к спецификации функции **strftime()** в ANSI C.

12.7.3 Арифметические функции

В **awk** имеются арифметические функции.

atan2(*y, x***)** ⌣

Возвращает значение arctan(y/x) в радианнах.

cos(*expr***)** ⌣

Возвращает значение cos(*expr*) в радианнах.

exp(*expr***)** ⌣

Возвращает значение экспоненциальной функции.

int(*expr***)** ⌣

Возвращает целое от *expr*.

log(*expr***)** ⌣

Возвращает значение натурального логарифма от *expr*.

rand() ⌣

Возвращает случайное значение в интервале [0-1].

sin(expr) ⌣

Возвращает значение **sin(***expr***)**.

sqrt(expr) ˽

Возвращает значение **sqrt(**_expr_**)** (квадратный корень из _expr_).

srand(_[expr]_**)** ˽

Устанавливает новое исходное значение для генератора случайных чисел. Возвращает предыдущее исходное значение. Если _expr_ опущено, то используется текущее время в секундах. Таким образом, если вы желаете, чтобы у вас при каждом новом запуске **awk**, генерировалась новая псевдослучайная последовательность в функции **rand()**, то вам полезно вызвать функцию **srand()** перед циклом обращений к функции **rand()**.

12.7.4 Опеределение новых функций

Наконец, в программах **awk** можно определять новые функции. Определение функций выполняется следующим образом:

```
function name(parameter list) { statements }
```

Внутри тела функции можно определить локальные переменные, которые должны быть описаны в поле параметров, но отделены от них более чем одним пробелом. Например,

```
function  MyFunction(i,j,   a,b,c)
# a, b и c являются локальными переменными.
{
...
}
{MyFunction(43,61)}
```

Заметим, что левая скобка в вызове функции должна следовать сразу за именем функции (без пробела).

12.8 Виды шаблонов используемых в awk

Программа **awk** допускает следующие виды шаблонов.

- **BEGIN**
- **END**
- **/регулярное выражение/**

- условное выражение
- **шаблон1 && шаблон2**
- **шаблон1 || шаблон2**
- **шаблон1 ? шаблон2 : шаблон3**
- **(шаблон)**
- **! (шаблон)**
- **шаблон1, шаблон2**

BEGIN и **END** являются специальными видами шаблонов, которые не проверяются на какое-то соответствие с записями входного потока. Если встречен шаблон **BEGIN**, то соответствующее ему действие будет выполнено лишь однажды до начала чтения вводного файла, т.е. так можно установить какие-то исходные значения. Если встречен шаблон **END**, то соответствующее ему действие тоже выполняется однажды, после достижения конца файла во входном потоке. Оба шаблона **BEGIN** и **END** должны иметь какие-то действия.

Регулярные выражения уже рассматривались в 10.2. **awk** понимает некоторые дополнительные специальные обозначения в регулярных выражениях. Для уточнения деталей следует обратиться к руководству.

Операции &&, || и ! являются соответственно логическим **И**, **ИЛИ** и **НЕТ** как в языке **C**. Оператор **?:** интерпретируется также как в **C**. Если имеется не пустой **шаблон1**, то для сравнения используется **шаблон2**, в противном случае используется **шаблон3**. Такое может случиться, если на месте **шаблона1** используется, к примеру, $9, но фактически девятое поле отсутствует в данной строке.

Можно использовать скобки для указания последовательности выполнения логических операций.

Если используется форма **шаблон1, шаблон2**, то это зовётся шаблонным интервалом. Шаблонному интервалу удовлетворяют последовательные строки вводного файла, которые начинаются строкой, удовлетворяющей **шаблону1**, и заканчиваются строкой, удовлетворяющей **шаблону2**.

12.9 Действия в awk

Действия должны быть заключены в фигурные скобки (`{}`). Ниже перечислены операторы, которые могут использоваться при описании действий.

(...) ␣

Группирование операторов.

$ ␣

Обозначает поле. $0 – вся строка, $1 – первое поле, $2 – второе поле и т.д.

++

- ␣

Увеличение и уменьшение как префиксное так и постфиксное.

^ ␣

(Шляпка). Возведение в степень. Может использоваться обозначение **.

+ - ! ␣

Унарный плюс, минус и логическое отрицание.

*** / % ␣**

Умножение, деление и модуль.

+ - ␣

Сложение и вычитание.

␣

(Пробел). Сцепление символьных строк.

< >

<= >=

= == ␣

Обычные условные операторы.

~ !~ ␣

Соответствие регулярному выражению и отрицание соответствия. Не рекомендуется использовать константы слева от знака `~` или `!~`

in ␣

Принадлежность массиву.

&& ␣

Логическое **И**.

`||` ␣

Логическое **ИЛИ**.

?: ␣

Условное выражение.

= += -=

`*= /= %= ^=` ␣

Варианты операторов присваивания.

12.9.1 Управляющие операторы

if (*условие*) *оператор1 [else оператор2]* ␣

Если выражение *условие* истинно, то выполнить оператор *оператор1*, в противном случае – оператор *оператор2*, если он имеется.

while (*условие*) *оператор* ␣

Выполнять *оператор* пока *условие* истинно.

do *оператор* **while** (*условие*) ␣

Выполнять *оператор* пока *(условие)* истинно.

for (*выражение1*; *выражение2*; *выражение3*) *оператор* ␣

1. Сначала выполнить *выражение1*.
2. Затем, если выражение *выражение2* истинно, то выполнить *оператор*.
3. Затем выполнить *выражение3*.
4. Перейти к пункту 2.

Например, `df | awk ’{ for (i = 1; i <= 2; i++);print $i }’` вывести первые две колонки вводного потока.

for (`var` **in** *array*) *оператор* ␣

Выполнить *оператор* для всех значений из массива *array*. Здесь `var` имя переменной цикла.

break ⌣

Выйти из цикла.

continue ⌣

Перейти к следующему значению переменной цикла.

delete array*[index]* ⌣

Удалить элемент массива.

delete array ⌣

Удалить весь массив.

exit *[выражение]* ⌣

Выйти из функции.

{операторы} ⌣

Составной оператор.

12.9.2 Операторы ввода/вывода

close(*file***)** ⌣

Закрыть файл или программный канал с именем *file*.

getline ⌣

Присвоить переменной $0 содержание следующей строки. Установить также переменные NF, NR и FNR.

getline <*file* ⌣

Присвоить переменной $0 содержание следующей записи вводного файла с именем *file*; установить переменную NF.

getline var ⌣

Присвоить переменной var содержание следующей записи вводного файла; установить переменные NR и FNR.

getline var <*file* ⌣

Присвоить переменной var содержание следующей записи из файла с именем *file*.

next ␣

Остановить обработку текущей записи и прочесть следующую запись. Обработку начать с первого шаблона.

nextfile ␣

Остановить обработку текущего вводного файла. Следующую запись прочесть из следующего вводного файла. Установить переменные FILENAME, ARGIND, а также установить FNR=1.

print ␣

Вывести текущую запись на устройство стандартного вывода.

print *expr-list* ␣

Вывести выражение *expr-list* на устройство стандартного вывода.

print *expr-list >file* ␣

Вывести выражение **expr-list** в файл с именем *file*.

printf *fmt*, *expr-list* ␣

Форматировать выражение *expr-list* в соответствии с форматом *fmt* и вывести на устройство стандартного вывода.

printf *fmt*, *expr-list >file* ␣

Форматировать выражение *expr-list* в соответствии с форматом *fmt* и вывести в файл с именем *file*.

system(*cmd-line*) ␣

Выполнить команду *cmd-line* и вернуть код завершения.

fflush(*[file]*) ␣

Вывести все буферы связанные с файлом с именем *file*. Если *file* опущен, то подразумевается стандартное устройство вывода. Если *file* есть нулевая строка, то будут выведены буферы всех открытых выводных файлов и программных каналов.

12.10 Дополнительные примеры

Вывести из файла `WgetWhere.tex` строки, содержащие слово `ftp` и слово `http` одновременно:

```
awk '/ftp/ && /http/ {print}' WgetWhere.tex
```

Просуммировать объём файлов в текущем каталоге:

```
ls -l | \
awk 'BEGIN{a=0} {if (index($1,"d") == 0) a=a+$5 } \
END{print a}'
```

Просуммировать объём файлов в текущем каталоге и вычислить средний размер файла:

```
ls -l | \
awk 'BEGIN{a=0;b=0} { if (index($1,"d") == 0) {a=a+$5;b=b+1} } \
END{printf "%s,%10d,%s,%10d,%s", \
"Объем=",a," Средний размер=",a/b,"\n"}'
```

12.11 Заключительные замечания по поводу awk

В заключение скажем, что поскольку программа **awk** является интерпретатором, то она интерпретирует программы на языке **awk** с примерно такой же скоростью, как, например, оболочка **bash** интерпретирует свои команды. Конечно, это оказывается медленнее, по сравнению с реализацией на **C**. Иными словами, если вы реализуете один и тот же алгоритм на языке **awk** и на **C**, то реализация на **C** будет выполняться в несколько раз быстрее. С другой стороны, составление программ для анализа и преобразования текста на **awk** оказывается во многих случаях проще и быстрее.

Для того, чтобы использовать преимущества обоих языков (**awk** и **C**), имеется несколько вариантов программных конвертеров с языка **awk** на язык **C**. Один из них, **awk2c**, доступен по адресу *ftp://sunsite.unc.edu/pub/Linux/utils/text/awk2c050.tgz.* О других конвертерах **awka**, **awkcc**, а также много полезного об **awk** можно узнать, например, в *http://ftp.umr.edu/pub/faqs/text/computer-lang/awk/faq.* Имеется также конвертер **a2p** (**awk**–программ в **perl**).

Глава 13

Подсистема печати текста a2ps

13.1 Введение

Программа **a2ps** позволяет готовить текстовые файлы (иначе, документы) для печати на принтере в формате **PostScript**: вывод производится либо на устройство печати, либо в файл.

Используемый по умолчанию план страниц довольно компактен и информативен: печатается две страницы документа на одной физической странице; каждая страница снабжена специальной рамкой; заголовки содержат имя файла, дату преобразования и другую информацию.

Поскольку программа имеет большое число параметров, которые позволяют производить весьма тонкую настройку вида печати в зависимости от конкретных особенностей вводимого текста, то **a2ps** следует рассматривать как гибкую настраиваемую подсистему преобразования произвольных текстов (программ на разных языках программирования, описаний с использованием национальных языков), с посимвольной кодировкой в формат **PostScript**. Конфигурационные возможности **a2ps** таковы, что её можно настроить, чтобы учесть конкретные особенности документов почти любого вида, а также предусмотреть использование любых других программ форматирования или проверки документов.

Программа **a2ps** постоянно дополняется и совершенствуется. При подготовке данного описания автор ориентировался на версию 4.12. Последнюю версию программы можно взять в `http://www.inf.enst.fr/~demaille/a2ps/`. Вместе с программой имеется довольно обширное описание, которое легко посмотреть посредством утилиты **info**. Полезно знать, что имеется отличная вспомогательная информация получаемая посредством команды

```
a2ps -help
```

Вопросы, комментарии могут быть направлены авторам: *Miguel San-*

tana (`Miguel.Santana@st.com`) и *Akim Demaille* (`demaille@inf.enst.fr`). Имеется список рассылки `a2ps-request@inf.enst.fr`, а также адрес, по которому следует сообщать об обнаруженных в программе ошибках `a2ps-bugs@inf.enst.fr`.

13.2 Описание ввода программы a2ps

Таблица 13.1: Параметры описания ввода программы **a2ps**

Параметр	**Описание**
-a[*page-range*] -**pages**[=*page-range*]	По умолчанию печатаются все страницы, однако имеется возможность выбрать страницы, которые должны быть напечатаны. Например, **-a1** означает, что следует напечатать одну первую страницу; **-a-5,7,13,43,179-** означает, что печатаются первые 5 страниц, затем страницы с номерами 7, 13 и 43, а далее страницы начиная с номера 179 до конца документа. Обсуждаемые страницы являются вводными страницами, а не выводными страницами, так что если у вас используется комбинация параметров **-2 -a1**, то вы получите наполовину заполненную выводную страницу. Заметим, что данный отбор страниц на вводе работает также и в режиме делегирования.
-c -**truncate-lines**=*boolean*	Усечь строки, которые оказались слишком длинны, чтобы быть помещёнными внутрь рамки. Максимальный размер строки зависит от используемого размера шрифта, формата символов, а также наличия или отсутствия нумерации строк.

Таблица 13.1: Параметры описания ввода программы **a2ps**

-i -**interpret**=*boolean*	Интерпретировать символы `<TAB>` и `<FF>`. Это означает, что `<FF>` (`^L`) приводит к переходу на следующую виртуальную выводную страницу, `<TAB>` приводит к переходу на следующую колонку табуляции.
-**end-of-line**=*type*	Определить какая последовательность символов воспринимается как конец строки. Значением *type* может быть: **n** или **unix** что эквивалентно `\n`, т.е. новая строка `<LF>`. **pc** или **rn**, что означает `\r\n` как в **MS DOS**, последовательно два символа `<CR><LF>`. **r** или **mac**, что эквивалентно `\r`. **nr**, т.е. эквивалентно `\n\r`. **any** или **auto**, что означает любой из предыдущих вариантов. Это значение параметра помогает избежать печати массы последовательных символов `^M`, которые появляются в файлах с **MS/DOS** - **MS/Windows**.
-X *key* -**encoding**=*key*	Входной поток имеет кодировку обозначаемую значением *key*. Типичные значения для *key*: `ASCII`, `latin1` и т.д. Полный список поддерживаемых кодировок и значений *key* можно получить посредством команды: `a2ps --list=encodings`
-**stdin**=*filename*	Вводному потоку с устройства стнадартного ввода дать имя файла *filename.*
-t *name* -**title**=*name*	Дать документу имя *name.* Это не имя выводного устройства, только имя документа.

Таблица 13.1: Параметры описания ввода программы **a2ps**

-prologue=*prologue*	Использовать пролог с именем *prologue* в качестве **PostScript** пролога для программы **a2ps**. Файл пролога должен иметь имя *prologue*.`pro` и находиться в библиотечном каталоге программы **a2ps**. Значениями *prologue* может быть следующее. **bold** - означает копию пролога для чёрно-белой печати, но все символы будут заглавными. **bw** - простой стиль для чёрно-белой печати с фонтами по умолчанию. **color** - будут использоваться цвета для выделения ключевых слов. **gray** - полутоновые символы будут использоваться для комментариев и меток. **gray2** - чёрные символы будут использоваться для комментариев и меток. **matrix** - план печати такой же как и для чёрно-белого пролога, однако поверх текста будут располагаться чередующиеся серые и белые горизонтальные полосы, которые удобны при просмотре больших таблиц.
-print-anyway=*boolen*	Если *boolen*=1, печатаь двоичные файлы. По умолчанию всякая печать прекращается, если встречен двоичный файл. Программа **a2ps** определяет тип файла по числу неизображаемых символов (если более 40% неизображаемых символов, то файл счиается двоичным). Если команда **file** отмечает, что файл имеет тип **data**, то файл также полагается двоичным и, как следствие, не печатается. Параметр **-print-anyway**=*boolen* позволяет преодолеть умолчание и файл будет напечатан.
-Z	

Таблица 13.1: Параметры описания ввода программы **a2ps**

-delegate=*boolen*	Разрешить или запретить процесс делегирования. Если делегирование разрешено **-delegate=1**), то программа **a2ps** не будет обрабатывать файлы, которые может оработать специализированная программа, а произведёт вызов подходящей специализированной программы делегируя ей полномочия по преобразованию вводного файла. Если делегирование запрещено (**-delegate=0**), то программа **a2ps** будет обрабарывать сама каждый вводной файл. Разницу в поведении программы легко понять, если вы попытаетесь напечатать с помощью программы **a2ps** файл в формате **PostScript**. Если процесс делегирования разрешён, то вы получите обычную печать файла. А если процесс делегирования запрещён, то вы получите символьное изображение внуреннего формата **PostScript** вашего файла, т.е. весьма длинную и многостраничную печать, которая окажется малоинформативной для вас, если вы не являетесь экспертом по языку **PostScript**. Известные программе **a2ps** специализированные утилиты можно узнать с помощью команды `a2ps --list=delegations`
-toc[=*format*]	Генерировать оглавление, где *format* является эскейпом, который обрабатывается как файл типа **PreScript** (смотрите раздел 13.5). Если *format* опущен, то не следует генерировать оглавление.

13.3 Простые примеры использования a2ps

Для того, чтобы оценить вид текста из файла `text.file` на печати, можно использовать следующую команду:

```
a2ps -P display text.file
```

Здесь **-P** означает, что далее идёт вид принтера; `display`, т.е. вывод будет произведён не на устройство печати, а сформированный файл печати будет передан на ввод вызванной программе `ghostview`.

Если вы захотите просто узнать сколько страниц займёт при печати ваш документ, но не печатать его, то это можно сделать так:

```
a2ps -P void text.file
```

Наконец, если требуется сохранить файл приготовленный для печати, то это можно селать так:

```
a2ps -P file text.file
```

В ответ программа напечатает следующее:

```
future76:shevel /usr/home/shevel> a2ps -P file Text.tex
[Text.tex (TeX): 2 pages on 1 sheet]
[Total: 2 pages on 1 sheet] saved into the file Text.ps'
```

То же самое можно получить использовав команду

```
a2ps Text.tex -o My.text.ps
```

В ответ программа напечатает:

```
future76:shevel /usr/home/shevel> a2ps Text.tex -o My.text.ps
[Text.tex (TeX): 2 pages on 1 sheet]
[Total: 2 pages on 1 sheet] saved into the file My.text.ps'
```

Возвращаясь к виду принтера, можно заметить, что имена принтеров, кроме уже перечисленных (`display`, `void` и `file`), означают имя принтера в операционной системе.

Если вы имеете достаточно узкий текст, например, текст программы, то можно разместить более двух страниц текста на одной физической странице

```
a2ps -3 Text.tex
```

Параметр **-3** (три) указывает, что на одной физической странице следует разместить три страницы текста.

Как вы уже обратили внимание программа **a2ps** многое делает по умолчанию. Чтобы узнать какие установлены умолчания можно использовать команду

```
a2ps -list=defaults
```

13.3.1 Делегирование

Если программа **a2ps** полагает, что определённое преобразование текста успешнее выполнит другая программа или подсистема, то **a2ps** производит обращение к этой другой подсистеме. Такой процесс в терминологии программы **a2ps** называется ДЕЛЕГИРОВАНИЕ.

Приведём примеры делегирования.

Пусть вам надо напечатать уже готовый файл в формате **PostScript**

`a2ps -4 -P display T.ps`

В данном случае запрошено вывести файл с именем `T.ps` на экран терминала по четыре страницы текста на одной физической странице. Поскольку исходный файл уже имеет формат **PostScript**, то программа **a2ps** сообщает об этом так

```
future76:shevel /usr/home/shevel> a2ps -4 -P display T.ps
[T.ps (ps, delegated to PsNup): 1 page on 1 sheet]
[Total: 4 pages on 1 sheet] sent to the printer Display'
```

В данном случае делегирование произведено к известному программному фильтру **PsNup** (преобразование файлов в формате **PostScript** таким образом, чтобы разместить на одной физической странице несколько страниц текста).

Процесс делегирования в программе **a2ps** можно описать в конфигурационном файле. Чтобы узнать какие сконфигурированы делегирования надо использовать команду

`a2ps -list=delegations`

которая в ответ может напечатать нижеследующее

```
future76:shevel /usr/home/shevel> a2ps --list=delegations
Applications configured for delegation
Delegation Groff', from roff to ps
        eval Grog -Tps $f | #{psselect} | #{psnup}
Delegation Gzip-a2ps', from compressed to ps
        gzip -cd $f | #{a2ps} --stdin=$N
Delegation Netscape', from html to ps
        netscape -noraise -remote 'openurl($f)'        -remote
'saveas(#f0,postscript)' &&     #{psselect} #f0 | #{psnup}
Delegation PsNup', from ps to ps
        fixps $f | #{psselect} | #{psnup}
Delegation Dvips', from dvi to ps
```

```
        #{dvips} $f -o #f0 && #{psnup} #f0
Delegation Pdf2ps', from pdf to ps
        pdf2ps $f #f0 && #{psselect} #f0 | #{psnup}
Delegation Texi2dvi', from texinfo to ps
        #{texi2dvi} $f && mv $N.dvi #f0 && #{dvips} -f #f0 | #{psnup}
```

Из вышеприведённого ответа программы, можно видеть, что зазипованный (текст, уплотнённый программой **gzip**) текст также можно смотреть с использованием **a2ps**

```
future76:shevel /usr/home/shevel> a2ps T.ps.gz -P display
[T.ps.gz (compressed, delegated to Gzip-a2ps): 3 pages on 2 sheets]
[Total: 4 pages on 2 sheets] sent to the printer Display'
```

Нетрудно видеть, что используя механизм делегирования **a2ps** можно любую программу вызывать из среды **a2ps**.

13.4 Стили печати с использованием a2ps

Для повышения выразительности печати подсистема **a2ps** даёт возможность определять и использовать различные стили печати. Нетрудно видеть, что для большей выразительности печати следует использовать особенности печатаемого текста. Представляется очевидным, что стили печати текста программы на языке **C** и на языке **fortran90** должны быть разными. То же самое можно сказать и о других видах текста: текстов скриптов на разных языках, текстов описаний или руководств и т.д. Программа **a2ps** имеет набор уже подготовленных стилей печати, к описанию которым мы приступим в данном разделе.

13.4.1 Параметры выразительной печати

Таблица 13.2: Параметры выразительной печати **a2ps**

Параметр	**Описание**

Таблица 13.2: Параметры выразительной печати **a2ps**

-**highlight-level**=*level*	определить степень выделения текста значением *level*. Возможные значения *level*: **none** - никакого выделения; **normal** - обычное выделение; **heavy** - более, чем обычное выделение. Более детальную информацию по поводу **heavy** можно получить посредством команды `a2ps -list=style-sheets`
-**g**	сокращённое обозначение для параметра `-highlight-level=heavy`.
-**E** *language* -**pretty-print**[=*language*]	без аргумента означает установку автоматического выбора стиля печати. В противном случае – установить стиль печати в значение *language*. Заметим, что если *language* равно **plain**, то это выключает выразительную печать. Имеющиеся стили печати можно получить посредством команды `a2ps -list=style-sheets` Часть из них рассматривается ниже. Если значение *language* имеет вид `KEY.ssh`, то программа не производит поиск стиля в библиотеке, а берёт указанное имя файла в качестве стиля печати.
-**strip-level**=*num*	в зависимости от значения *num* производится следующее: **0** - всё печатается; **1** - обычные комментарии не печатаются; **2** - серьёзные комментарии не печатаются; **3** - никакие комментарии не печатаются. Этот параметр удобен при печати программ **Java**, а также при использовании спецификаций, которые порождены графическими редакторами.

13.4.2 Готовые стили печати

Стили описываются в специальных конфигурационных файлах подсистемы **a2ps**. Имена файлов имеют вид *стиль*`.ssh`, например, `ada.ssh`. Конфигурационные файлы с описанием стилей находятся в подкаталоге `sheets` основного каталога **a2ps**.

Ниже перечислены стили имеющиеся по состоянию на октябрь 1999.

Таблица 13.3: Стили печати в программе **a2ps**

Обозначение стиля	**Назначение стиля печати**
`68000`	Предназначен для печати программ на ассемблере 68K. Предполагается, что такой стиль подходит и для печати программ на других видах ассемблеров.
`ada`	Стиль печати программ на языке Ada.
`sh`	Стиль печати текстов скриптов оболочек **sh** и **bash**.
`c gnuc`	Стили печати программ на языке **C**.
`csh tcsh`	Стили печати текстов скриптов оболочек **csh** и **tcsh**.
`cpp objc`	Стили печати языков **C**++ и **Objective C**.
`caml`	Стиль печати языка **ML**.
`claire`	Стиль печати языка **Claire**.
`clisp`	Стиль печати языка **Common Lisp**.
`coqv`	Стиль печати языка **Coq Vernacular**.
`dc_shell`	Стиль печати языка описания электронных схем **Design Compiler**.
`eiffel`	Стиль печати языка **Eiffel**.
`elisp`	Стиль печати языка **Emacs Lisp**.
`eps`	Стиль печати языка **Encapsulated PostScript**. Неверные операторы выделяются в тексте другим начертанием.
`tclx tk vtcl`	Стили печати языков **Extended Tcl**, **Tk**, **Visual Tcl**.

`fortran`
`for-fixed`
`for-free`
`for-free`
	Стили печати языка **Fortran**.

Таблица 13.3: Стили печати в программе **a2ps**

`for77-fixed`
`for77-free`
`for77kwds`
	Стили печати языка **Fortran 77**.

`for90-fixed`
`for90-free`
`for90kwds`
	Стили печати языка **Fortran 90**.

`java`	Стили печати языка **Java**.
`modula2` `modula3`	Стили печати текстов с использованием вариантов языка **Modula**.
`oberon`	Стиль печати языка **Oberon** (потомок **Pascal** и **Modula2**).
`pascal`	Стиль печати языка **Pascal**.
`perl`	Стиль печати языка **Perl**.
`python`	Стиль печати языка **Python**.
`rexx`	Стиль печати языка **REXX**.
`sather`	Стиль печати языка **Sather**.
`scheme`	Стиль печати языка **Scheme**.
`zsh`	Стиль печати языка (оболочки) **zsh**.

Имеются также стили печати для вывода файлов специального назначения.

Таблица 13.4: Стили печати различных файлов программой **a2ps**

Обозначение стиля	**Назначение стиля печати**
`card`	Этот стиль помогает подготовить выразительную карту с кратким описанием параметров любой программы в **Linux**. Пример такой карты можно получить с помощью команды `wget --help | a2ps -Ecard -1 --stdin=wget` Здесь **a2ps** имеет несколько параметров. **-E**`card` означает, что установлен стиль `card`. **-1** - одна страница документа на одной физической странице, наконец `-stdin=wget` означает, что вводному потоку следует присвоить имя **wget**. Это имя появится в заголовке напечатанных страниц.
`chlog`	Стиль описывает файлы, которые содержат список изменений (`ChangeLog`).
`gmake` `make`	Стили печати make-файлов.
`html` `vrml`	Стиль печати html-файлов и vrml-файлов.
`lace`	Эквивалент make-файлов для языка **Eiffel**.
`mail`	Стиль для печати электронных сообщений. Полезно использовать совместно с параметрами `-g` `-Email` `-strip`.
`initora`	Стиль печати инициализационного файла **Oracle** `init.ora`.
`ps`	Стиль печати файла в формате **PostScript**.
`pre`	Стиль печати **PreScript**. Это специальный стиль поддерживаемый подсистемой **a2ps**, который позволяет использовать во вводном потоке ряд операторов форматирования (см. 13.5).
`pretex`	Стиль печати **PreTeX**. Это специальный стиль поддерживаемый подсистемой **a2ps**, который позволяет использовать во вводном потоке ряд операторов форматирования подмножества операторов **LaTeX** (см. `info a2ps`).

Продолжение таблицы на следующей странице

Стили печати различных файлов (продолжение таблицы 13.4)	
`texscript`	Стиль печати **TextScript**. Это специальный стиль поддерживаемый подсистемой **a2ps**, который позволяет использовать во вводном потоке операторы форматирования как типа **PreTeX** так и **PreScript** (подробнее смотрите `info a2ps`).
`a2psrc`	Стиль печати инициализационных конфигурационных файлов программы **a2ps** `a2ps.cfg` или `.a2ps/a2psrc`.
`ssh`	Стиль печати конфигурационных файлов **a2ps**, описывающих стили печати, например, `a2psrc.ssh`, или `ada.ssh`.

13.5 PreScript

PreScript был разработан вместе с **a2ps**. Поскольку заглавные последовательности, специальные символы и прочее были реализованы в **a2ps**, то было бы неплохо иметь доступ к этим возможностям: таким механизмом является язык описания вводного потока данных, который получил имя **PreScript**. С помощью этого языка можно описать специальную обработку фонтов с использованием синтаксиса **ssh** (STYLE SHEETS IMPLEMENTATION - РЕАЛИЗАЦИЯ СТИЛЕВЫХ ЛИСТОВ).

К основным достоинствам **PreScript** можно отнести:

а) очевидную простоту и

б) доступность на любой аппаратной платформе.

13.5.1 Синтаксис

Каждая команда в языке **PreScript** начинается с обратного слеша (\\).

Если команда использует аргумент, то он обязательно заключается в фигурные скобки. Не допускается никаких пробелов между командой и аргументом.

Внутри команд **PreScript** не должно использоваться никаких других команд **PreScript**, т.е. суперпозиция команд запрещена. Например, следующая строка будет неверно интерпретироваться подсистемой **a2ps**

```
\Keyword{Problems using \keyword{recursive \copyright} calls}
```

Следует писать так

```
\Keyword{Problems using} \keyword{recursive} \copyright \Keyword{calls}
```

Комментарии начинаются знаком процент (`%`).

13.5.2 Команды PreScript

Таблица 13.5: Команды PreScript

Команда	**Назначение**
\keyword{*text*} **\Keyword**{*text*}	Выделить слегка/сильно текст *text*. Может использоваться лишь для нескольких расположенных рядом слов.
\comment{*text*} **\Comment**{*text*}	Тексту *text* придать специальное начертание. Текст *text* может быть удалён, если используется параметр **-strip**.
\label{*text*} **\Label**{*text*}	Текст *text* должен рассматриваться как определение или как важный пункт вводного документа.
\string{*text*}	Вывести текст *text* как выделенную строку, например, шрифтом **Times**.
\error{*text*}	Вывести текст *text* как сообщение об ошибке, т.е. с помощью отличающегося шрифта.
\symbol{*text*}	Текст *text* написан с использованием символьного шрифта **PostScript**. В целом эта возможность не совместима с **LaTeX**, однако рекомендуется использовать в тех случаях, где специальные ключевые слова обозначающие символы совпадают с **LaTeX**. Примерами таких команд могут быть: **\rightarrow** или **\Omega** которые произведут на печати стрелку вправо или греческую букву Ω соответственно. Полный список таких символов можно найти в файле с именем `symbols.ssh`, который находится в подкаталоге `sheets` основного каталога подсистемы **a2ps**.

Продолжение таблицы на следующей странице

Команды PreScript (продолжение таблицы 13.5)	
\header{*text*} **\footer**{*text*}	Использовать текст *text* как заголовок или подстрочное примечание на текущей странице. Используется текст *text* из последнего оператора, если их оказалось несколько.
\encoding{*key*}	Изменить динамически текущую кодировку входного потока. После этой команды текст будет печататься с использованием кодировки *key*.

13.5.3 Примеры использования PreScript

Пусть мы хотим напечатать список пользователей на сервере выделив часть выводимой информации:

```
cat /etc/passwd |  \
awk -F: \
’{print "\\Keyword{" $5 "} (" $1 ") \\rightarrow\\keyword{" $7 "}"}’\
| a2ps -Epre -P display -1
```

Обратите внимание на два обратных слеша, которые необходимо использовать чтобы учесть особенности интерпретации команды оболочкой **bash**. В то же время, если запрос для программы **awk** будет находиться в файле (тогда было бы написано `awk -F: -f input.awk ...`, то там не потребуется двух обратных слешей.

Итак, в примере указано следующее. Выделенным шрифтом печатается поле комментария, затем, в скобках, имя для логирования в систему (*login name*), далее стрелка вправо, затем выделенным шрифтом печатается имя оболочки, которую предпочитает пользователь. Здесь, **a2ps** имеет несколько параметров. Значением параметра **-E** является `pre`, т.е. определяет стиль печати **PreScript**. Значением параметра **-P** является `display`, т.е. вывод программы будет направлен во вьюер **gv**. Наконец, **-1** - означает, что одна страница документа должна размещаться на одной физической странице.

13.6 Инициализационные файлы a2ps

a2ps читает несколько файлов до того, как начинает интерпретировать параметры в командной строке. Файлы прочитываются в следующем порядке:

1. Системный инициализационный файл **a2ps**, который обычно располагается в `/usr/local/etc/a2ps.cfg` (или `/etc/a2ps.cfg`, если не установлена переменная окружения A2PS_CONFIG, которая указывает расположение инициализационного файла.
2. Пользовательский инициализационный файл `$HOME/.a2ps/a2psrc`.
3. Локальный инициализационный файл текущего каталога `./.a2psrc`.

В перечисленных файлах пустые строки и строки, которые начинаются знаком # (решётка) игнорируются. Все другие строки имеют единообразный формат:

TOPIC: *arguments*

где **TOPIC** есть один из параметров, которые вы устанавливаете, а *arguments* есть значения этого параметра. *arguments* может продолжаться на несколько строк. Если имеется продолжение, то строка должна завершиться обратным слешем.

Конфигурационные возможности **a2ps** насыщены множеством деталей, которые придают дополнительную гибкость в использовании. Для глубокого изучения возможностей инициализации следует использовать команду

`info a2ps`.

13.7 Кодировка входного потока для a2ps

Вводные текстовые файлы могут иметь весьма различную кодировку, которая отражает языковые особенности использования тех или иных букв. Как следствие, почти каждый национальный язык имеет особенности в начертании букв. Для вывода на печать этих особенностей **a2ps** имеет специальный механизм **encoding**, который позволяет не только указать подходящую кодировку, но и добавить новую, если имеется такая необходимость. В настоящей версии **a2ps** имеется несколько десятков кодировок включая `koi8r` (Кириллицу для операционной среды **Linux**).

Каким образом можно добавить новую кодировку следует познакомиться с помощью `info a2ps`.

13.8 Параметры a2ps

Чтобы прочесть и проанализировать параметры в командной строке, **a2ps** использует программу **GNU getopt** (смотрите страницы руководства `man getopt`, сравните с встроенной командой **bash getopts** 9.8). Использование программы **GNU getopt** предопределяет следующие особенности представления параметров в командной строке:

- параметры любого вида должны быть разделены пробелами;
- порядок файлов и параметров не имеет значения, например, `a2ps -1 -d document` означает то же что `a2ps -d document -1`;
- порядок параметров имеющих сходные по смыслу аргументы имеет занчение;
- параметры в краткой форме записи могут быть сгруппированы, например, можно записать `a2ps -4mg main.c` вместо `a2ps -4 -g -m main.c`;
- если нет двусмысленности, то длинные имена параметров могут быть сокращены, например, `-pro` будет верно понято как `-prologue`.
- двумя минусами (-) можно закончить перечисление параметров в командной строке; далее могут быть только имена файлов, что удобно, если имена файлов начинаются со знака минус.

Здесь мы будем обозначать словм *boolean* значение параметра, которое рассматривается как `истина`, если имеет значение **1** или **yes**. Разумеется, оно будет рассматриваться как `ложь`, если имеет значение **0** или **no**.

Если аргумент представлен в квадратных скобках, то он не является обязательным. Не обязательный аргумент должен записываться слитно с параметром, если последний представлен в краткой форме.

13.8.1 Параметры определения конкретной задачи для a2ps

Данная группа параметров определяет вид задачи для **a2ps**. При этом **a2ps** ничего не выводит на принтер, она лишь выдаёт информацию на экран и завершается.

Таблица 13.6: Параметры задания для a2ps

Параметр	*Назначение*
-V -**version**	Вывести версию программы.
-h -**help**	Вывести на экран краткое перечисление параметров **a2ps**.

Таблица 13.6: Параметры задания для a2ps

-copyright	Вывести на экран сведения о правах копирования для **a2ps**.
-guess	Действовать как программа `file`, т.е. в этом случае **a2ps** должна определить тип файла.
-list=*topic*	Отобразить какие умолчания имеются в **a2ps** относительно *topic*. Верными значениями *topic* могут быть следующие. **defaults** или **options** - вывести подробный отчёт об умолчаниях **a2ps**. **features** - вывести все значения, которые вы можете определить или изменить. **delegations** - вывести список имеющихся делегирований. **encodings** - вывести список имеющихся кодировок. **variables** - вывести детальный список переменных. **media** - вывести список известных типов свойств устройств печати. **prologues** - вывести список прологов для **PostScript**. **printers** - вывести список принтеров и устройств для вывода. **style-sheets** - вывести список известных программе стилей печати. **user-options** - вывести список пользовательских параметров.

13.9 Глобальные параметры a2ps

Таблица 13.7: Глобальные параметры a2ps

Параметр	*Назначение*
-q **-quiet** **-silent**	

Продолжение таблицы на следующей странице

Глобальные параметры a2ps (продолжение таблицы 13.7)	
	Программа должна выполняться без выдачи сообщений ("молча").
-v*level* -**verbose**=*level*	Определить уровень диагностики: *level*=0 - ничего не диагностировать; *level*=1 - **a2ps** будет сообщать только общее количество напечатанных страниц; *level*=2 (умолчание) - сообщается число отпечатанных страниц для каждого файла. *level*=**configuration** *level*=**options** - прочесть конфигурационные файлы и параметры; *level*=**files** - вывести имена всех вводных и выводных файлов **a2ps**, которые использовались в данном задании; *level*=**fonts** - перечислить использованные шрифты; *level*=**meta-sequences** - перечислить расширения макрокоманд; *level*=**parsers** - вывести прокол подстановки макроопределений программы **a2ps** и протокол анализа командной строки **a2ps**; *level*=**pathwalk** - вывести протокол поиска различных файлов (шрифтов, конфигурационных файлов, etc.); *level*=**ppd** - обработать стилевой лист **PostScript Printer Description** (этот файл содержит информацию о принтере); *level*=**sheets** - стилевые листы; *level*=**all** - выдать всю отладочную информацию о выполнении программы.

Продолжение таблицы на следующей странице

Глобальные параметры a2ps (продолжение таблицы 13.7)	
	Когда **a2ps** запускается, она проверяет переменную окружения A2PS_VERBOSITY. Если переменная установлена, то её установки определяют уровень детальности и характер сообщений программы **a2ps**, т.е. значения установленные в командной строке не будут оказывать влияния. Допустимыми значениями для A2PS_VERBOSITY являются теми же самыми, что и для `-verbose` в командной строке. То, что значения переменной A2PS_VERBOSITY оказываются более приоритетными для программы, позволяет проверять и отлаживать содержание конфигурационных файлов, которые программа прочитывает до анализа параметров в командной строке.
-=*shortcut* -**user-option**=*shortcut*	Использовать сокращения, которое определил пользователь. Такие сокращения могут перемежаться в командной строке с обычными параметрами программы **a2ps**. Имеются три встроенных пользовательских сокращения: **lp** - эмулировать простое устройство печати, т.е. удалить все элементы, повышающие выразительность печатаемого текста, например, выделение жирным шрифтом или курсивом и т.п. **mail** или **longmail** - предпочитаемые значения для печати электронной почты или сообщений **news**. **manual** - приготовить задание для печати файла на принтере с ручной подачей бумаги.
Продолжение таблицы на следующей странице	

Глобальные параметры a2ps (продолжение таблицы 13.7)	
-**debug**	Разрешить отладочные режимы, которые могут помочь понять почему принтер не печатает ваш файл.
-D *key*[=*value*] -**define:***key*[=*value*]	Без значения *value*, ключевое слово *key* будет деинициализировано (выполнена операция **unset**). Заметим, что **-Dtest=** присвоит переменной `test` пустое значения, а **-Dtest** деинициализирует переменную `test`. Напомним, что речь идёт о механизме внутренних переменных системы печати **a2ps**, а не о переменных окружения.

13.10 Общий стиль выводимых страниц

Рассматриваемые в этом разделе параметры программы **a2ps** касаются общего плана размещения текста на печатаемых страницах.

Таблица 13.8: Параметры описания плана вывода **a2ps**

Параметр	**Описание**
-M *medium* -**medium**=*medium*	Использовать формат страниц *medium*. Полный список форматов можно получить командой `a2ps -list=media` Типичными значениями *medium* являются **A3**, **A4**, **Letter** и т.п. Имеется специальное значение *medium* – **libpaper**, которое приводит к тому, что **a2ps** будет опрашивать библиотеку `libpaper` на предмет использования выводного формата. Однако, такое значение может быть использовано только если `libpaper` имелась во время конфигурирования программы **a2ps**.

Продолжение таблицы на следующей странице

Описание плана вывода a2ps (продолжение таблицы 13.8)

-r **-landscape**	Печатать как **ландшафт** (горизонтально).
-R **-portrait**	Печатать как портрет (вертикально).
-columns=*num*	Определить число колонок для размещения страниц текста на физической странице.
-rows=*num*	Определить число строк для размещения страниц текста на физической странице.
-major=*direction*	Определить в каком порядке будут располагаться станицы текста на физической странице: по рядам – значение *direction* равно **rows** или по колонкам – значение *direction* равно **columns**.
-1	1 x 1 вертикальное расположение (портрет), 80 символов в строке, т.е. замена нескольких параметров: `-columns=1 -rows=1 -portrait -chars-per-line=80 -major=rows`.
-2	2 x 1 ландшафт, 80 символов в строке, расположение по рядам, т.е. замена следующих параметров: `-columns=1 -rows=1 -portrait -chars-per-line=80 -major=rows`.
-3	3 x 1 ландшафт, 80 символов в строке, расположение по рядам.
-4	2 x 2 портрет, 80 символов в строке, расположение по рядам.
-5	5 x 1 ландшафт, 80 символов в строке, расположение по рядам.
-6	3 x 2 ландшафт, 80 символов в строке, расположение по рядам.
-7	7 x 1 ландшафт, 80 символов в строке, расположение по рядам.
-8	4 x 2 ландшафт, 80 символов в строке, расположение по рядам.
-9	3 x 3 портрет, 80 символов в строке, расположение по рядам.

Продолжение таблицы на следующей странице

Описание плана вывода a2ps (продолжение таблицы 13.8)	
-j -**borders**=*boolean*	Напечатать рамку вокруг каждой страницы, если *boolean* равно 1. Если *boolean* равно 0, то не печатать рамку.
-A *mode* -**file-align**=*mode*	Выровнять отдельные файлы при печати в соответствии со значением *mode*. Значениями *mode* может быть следующее: **virtual** - каждый файл начинается со следующей страницы текста; **rank** - каждый новый файл начинается на новой строке или колонке в зависимости от значения -*major*. **page** - каждый файл начинается на новой физической странице. **sheet** - каждый файл начинается на новом листе. *num* - (*num* есть целое) каждый новый файл начинается на странице с номером *num*+1. Это означает, среди прочего, что если потребуется пропустить несколько страниц, то они будут присутствовать в документе, но останутся пустыми.
-**margin**=[*num*]	Определить размер полей на странице (*num* единиц в смысле **PostScript** или 12 единиц, если не установлено значение *num*). Имеется в виду отступ на физическом листе слева для удобства подшивки в скоросшиватель.

13.11 Определение плана страницы документа

Эта группа параметров программы **a2ps** определяет план страницы документа.

Таблица 13.9: Параметры описания плана выводной страницы **a2ps**

Параметр	**Описание**
-**line-numbers**[=*number*]	Строки в выводном файле будут пронумерованы. Значение *number* задаёт интервал нумерации. По умолчанию интервал нумерации равен 1, т.е. будут пронумерованы все строки подряд (номер печатается слева от строки). Если использовано `-line-numbers=5`, то номера будут проставлены у каждой пятой строки, начиная с пятой строки, т.е. 5-ая, 10-ая, 15-ая и т.д.
-C	То же самое что `-line-numbers=5`.
-f *size*[*unit*] -**font-size**=*size*[*unit*]	Установить значение размера шрифта в значение *size*. Величина *size* является числом с плавающей точкой. Величина *unit* может иметь одно из следующих значений: **cm** - для обозначения сантиметров; **points** - для обозначения единиц **PostScript**; **in** - для обозначения дюймов.
-l *num* -**chars-per-line**=*num*	Установить размер шрифта таким, чтобы на странице документа помещалось бы *num* символов.
-L *num* -**lines-per-page**=*num*	Установить размер шрифта таким, чтобы на странице документа помещалось бы *num* строк. Это очень полезно, если ваш документ уже обработан какой-то программой формирования текста и имеет фиксированное число строк на странице. Минимально возможное число строк равно 40, а максимальное равно 160.

Продолжение таблицы на следующей странице

Описание плана страницы (продолжение таблицы 13.9)	
-m **-catman**	Полезно использовать в случае печати страниц описаний, например, `man genscript | a2ps -catman`
-T *num* **-tabsize**=*num*	Установить позиции табулятора в значение *num*. Это значение будет, естественно, игнорироваться, если одновременно использован параметр `-interpret=no`.
-non-printable-format=*format*	Определить как будут отмечаться при печати неизображаемые символы. Значением *format* может быть одно из следующих: **caret** - использовать обычное представление, употребляемое в **Unix/Linux:** `^A`, `M-^B` и т.д.; **space** - использовать пробел; **question-mark** - использовать знак вопроса; **octal** - использовать обозначения вида `\001`, `\123` и т.д. **hexa** - использовать обозначения типа `\x02`, `\xfd` и т.д. **emacs** - использовать обозначения типа C-h, M-C-a и т.п.

13.11.1 Параметры определения заголовков страниц

С помощью этих параметров вы можете описать, что вы хотели бы видеть вокруг ваших страниц.

Таблица 13.10: Параметры определения заголовков страниц

Параметр	Описание
-B **-no-header**	нет никаких заголовков страниц.

Таблица 13.10: Параметры определения заголовков страниц

-b*text* -**header**[=*text*]	установить заголовок страницы.
-**center-title**[=*text*] -**left-title**[=*text*] -**right-title***text*]	установить заголовок *text* соответственно в центре страницы документа, у левого края страницы документа, у правого края страницы документа.
-u*text* -**underlay**[=*text*]	Установить прозрачную надпись. Использовать текст *text* как фон (похожий на водяной знак), т.е. светло–серый текст *text* поместить по диагонали каждой страницы. **Замечание**. Если программа **a2ps** выполняет делегирование обработки текста другой программе, то этот параметр может не оказать никакого действия.
-**left-footer**[=*text*] -**footer**[=*text*] -**right-footer**[=*text*]	использовать текст *text* в качестве соответствующих нижних заголовков страницы документа: **left-footer** – бнизу слева, **footer** – внизу в центре, **right-footer** – внизу справа.

13.11.2 Параметры a2ps для описания вывода

Эти параметры позволяют вам описать, куда вы хотите направить ваш вывод из программы **a2ps**. Вывод может быть направлен лишь в одном направлении из нескольких возможных.

Таблица 13.11: Параметры определения заголовков страниц

Параметр	Описание
-o *file*	

Таблица 13.11: Параметры определения заголовков страниц

-**output**[=*file*]	Выводной файл должен быть сохранён в файле с именем *file*. Если имя выводного файла указано как `-`, то вывод следует направить на устройство стандартного вывода.
-**version-control**=*type*	Чтобы избежать случайной потери уже существующего файла, программа **a2ps** предлагает схемы сохранения файлов в том случае, если имя использованное в параметре `-output=`*file* совпадает с именем уже существующего файла. Тип резервной копии файла может быть указан с помощью переменной окружения VERSION_CONTROL. Если переменная не установлена и параметр не использован, то предполагается простое резервное копирование, эквивалентно значению `-output=existing`. Допустимыми значениями величины *type* могут быть следующие: **none** - не делать копирования, т.е. удалять существующий файл с таким же именем; **existing** - выполнять нумерованное копирование, если уже существуют резервные копии файла; **simple** - всегда выполнять простое копирование.
-**suffix**=*suffix*	Суффикс, который используется в в имени файла резервной копии, может быть определён значением переменной окружения SIMPLE_BACKUP_SUFFIX. Значение параметра более приоритетно по отношению к значению переменной. Если ничего не установлено (ни в параметре, ни в переменной), то используется символ ~.
-P *name*	

Таблица 13.11: Параметры определения заголовков страниц

-**printer**=*name*	Послать вывод на принтер с именем *name*. Обратим внимание на специальные значения параметра `-printer`, описанные в разделе 13.3. Смотрите также `a2ps -list=defaults`
-d	Направить вывод на стандартный принтер по умолчанию.

13.11.3 Параметры генерируемого PostScript

Следующие параметры определяют только изменения, которые могут быть внесены в генерируемый PostScript текст.

Таблица 13.12: Параметры генерируемого текста **PostScript**

Параметр	Описание
-**ppd**[=*key*]	Без аргументов установить автоматический выбор **PPD**, в противном случае использовать значение *key* в качестве **PPD**.
-n *num* -**copies**=*num*	Вывести *num* копий каждой страницы.
-s *duplex-mode* -**sides**=*duplex-mode*	Определить количество сторон листа, на которых производится печать. Допустимыми значениями являются: 1 или **simplex** - Одна страница на лист. 2 или **duplex** - Две страницы на лист, режим **DuplexNoTumble**.

Таблица 13.12: Параметры генерируемого текста **PostScript**

	tumble - две страницы на лист, режим **DuplexTumble**. Не обязательно предполагается режим **Duplex** на принтере. Используются возможности режима **Duplex** самой программы **a2ps**, т.е. изменение расположения полей на одной и другой стороне листа.
-S KEY[:*value*] -**setpagedevice**=**KEY**[:*value*]	Передать определение страничного устройства в генерируемый выводной файл **PostScript**. Если не дано никакого значения *value*, то ключевое слово **KEY** удаляется из выводного файла.
-**statusdict**=**KEY**[:*value*] -**statusdict**=**KEY**[::*value*]	Передать определение **statusdict** в генерируемый выводной файл **PostScript**. Если не указано никакое значение *value*, то ключевое слово **KEY** удаляется из определения в выводном файле. Например, если принтер позволяет выбирать лоток с бумагой, то команда `a2ps -statusdict=setpapertray:1 quicksort.c` задаёт печать файла `quicksort.c` с использованием бумаги из подающего лотка номер 1.
-k	

Таблица 13.12: Параметры генерируемого текста **PostScript**

-page-prefeed	разрешить предварительную подачи листа бумаги во время печати. Этот режим позволяет принтеру подавать бумагу одновременно с интерпретацией кода **PostScript** (вместо ожидания конца интерпретации кода). Этот режим может привести к заметному ускорению печати документа, состоящего из большого числа страниц (многие десятки или сотни страниц).
-K **-no-page-prefeed**	Запретить предварительную подачу бумаги одновремнно с интерпретацией кода **PostScript**.

13.12 Примеры использования a2ps

Классическим примером может быть использование программы **a2ps** для распечатки страниц описаний:

```
man awk | a2ps -stdin=awk -1 -Xkoi8
```

описание программы **awk** будет напечатано на принтере (около 30 страниц). Каждая страница будет иметь сверху заголовок **awk**, одна страница документа будет располагаться на одной физической странице. Дата и время будут печататься с использованием Кириллицы, поскольку на моей машине установлены соответствующие переменные окружения (по поводу локализации смотрите раздел 3.8).

Если вы захотите придать печати неповторимую индивидуальность, например, текст, напоминающий водяные знаки, то можно изменить команду так:

```
man awk | a2ps --stdin=awk -1 -Xkoi8 \
--underlay="Официальное Описание"
```

тогда на каждой странице появится текст в виде светло-серой надписи по диагонали страницы - `Официальное Описание`.

Если вам потребуется иметь на бумаге несколько кратких описаний параметров известных компиляторов, то это можно сделать так

```
gcc --help | a2ps --stdin=gcc -1 -Xkoi8 -Ecard
```

или для **Фортран-90**

```
vf90 2>&1 | a2ps --stdin=vf90 -1 -Xkoi8 -Ecard
```

Здесь пришлось выполнить перенаправление ввода-вывода с устройства 2 на устройство 1, поскольку имеющийся компилятор для **Фортран-90** выводит сообщения на устройство 2.

Таким образом, если вывести несколько полезных кратких описаний, то можно быстро, за несколько минут, приготовить себе подшивку из таких описаний на манер справочника. Аналогичным образом можно печатать тексты программ на любом языке. Поскольку по умолчанию **a2ps** пытается *догадаться* о стиле печати, то вы получите весьма выразительные страницы, которые также можно будет использовать как справочный материал или как часть вашего отчёта.

При отладке качества печати, когда вам быть может полезно вывести несколько вариантов для сравнения, очень полезно добавить параметр `-P display`. Тогда результат будет выведен на дисплей с помощью браузера **gv**, а не на принтер. Вы можете использовать параметр **-o**, чтобы использовать полученный файл позже, например,

```
man od | a2ps -stdin='Utility od' -1 -Xkoi8 -o od.ps
```

результат, т.е. описание утилиты **od** будет сохранён в файле `od.ps`.

Пусть вам надо напечатать список или таблицу. Тогда удобнее на листах разместить какие-то линии, чтобы было удобнее читать таблицы. Это легко сделать так

```
man bash | a2ps --stdin='bash' -1 -Xkoi8 \
--prologue=matrix -P display
```

Здесь результат выйдет на экран дисплея.

Пусть у вас имеется довольно объёмный документ в формате **PostScript**, например, 150 страниц. Вам хотелось бы напечатать этот документ компактнее, т.е. по несколько страниц документа на одной физической странице. Это легко сделать таким образом:

```
a2ps Book.ps -9 -Xkoi8 -P display
```

На одной физической странице будет помещено 9 страниц документа. Если вы выведете результат на принтере 1200 DPI, то все страницы будут выглядеть весьма мелко, но всё будет ясно видно, например, с соответствующими очками.

Замечание. Ряд интересных особенностей программы **a2ps** могут не срабатывать, когда происходит делегирование обработки файлов другим программам. Например, если вы попробуете использовать параметр `-prologue=matrix` для файлов вида ***.ps**, то не обнаружите никаких полос в выводном файле. Иными словами, в данном случае определяющими являются свойства той программы, которой производится делегирование (передача) обработки файла.

Глава 14

Система поддержки версий текстов – CVS

Аббревиатура **CVS** означает **Concurrent Versions System** (многопоточная система версий). Технически **CVS** представляет собой программную систему, которая помогает поддерживать много вариантов исходных текстов, над которыми работает одна или несколько групп разработчиков. Применяя эту систему, вы можете детально отслеживать историю изменений ваших исходных текстов, а также производить массу других операций по контролю за правильностью смены версий исходных текстов.

Система **CVS** позволяет на основе хранимой истории воссоздать в любой момент любую версию исходных текстов, которая только была в прошлом. Это верно как для всей совокупности исходных текстов, так и для отдельных файлов.

CVS предоставляет возможности надёжного доступа к исходным текстам с других компьютеров в Интернет. Разработчики на удалённых хостах могут выполнять те же операции над текстами в хранилище, что и локально. Поддерживается режим параллельных разработок, когда разные программисты имеют возможность относительно независимо корректировать одни и те же исходные тексты в одно и то же время.

Отслеживание истории изменения часто помогает найти ошибки и неточности в большой системе программ. Вы сможете легко вернуться к прежним версиям ваших программ, или просто перейти от одного варианта программ к слегка модифицированному варианту программ, подготовленному для других целей.

Исходные тексты могут иметь любой вид и назначение. Вы можете готовить книгу или описания программ или писать роман. Любой текст или его вариант может быть сохранён с помощью **CVS**.

Роль такой системы трудно переоценить, когда коллектив разработчиков постоянно меняется, но в то же время требуется сохранять и поддерживать в продолжающемся проекте полезные наработки, полученные разными людьми на разных этапах проекта, а также привнесённые извне.

Хранение многих вариантов может потребовать массу дисковой памяти и **CVS** предпринимает все меры для экономии места на диске. Так, при поддержке нескольких вариантов одного текста **CVS** хранит лишь базовый текст и изменения к нему.

Система **CVS** начиналась как набор скриптов, которые подготовил *Dick Grune* в 1986 году. В 1989 году работа в данном направлении была продолжена двумя людьми *Brian Berliner* и *Jeff Polk*.

Обычно **CVS** является частью стандартной поставки Linux. Система **CVS** свободно распространяется в Интернет. Можно отметить следующие серверы, на которых можно найти **CVS**:

http://www.cyclic.com/

`http://www.loria.fr/~molli/cvs-index.html`

Для публичного обсуждения вопросов, касающихся **CVS**, имеется news группа `news:comp.software.config-mgmt`.

14.1 Модель CVS

Перед дальнейшим обсуждением полезно отметить некоторые особенности модели работы системы **CVS**.

Все тексты хранятся в специальной системе каталогов **CVS**, которое называется ХРАНИЛИЩЕ, или РЕПОЗИТАРИЙ. Хранилище создаётся средствами системы **CVS**. К одному Хранилищу могут иметь доступ несколько разработчиков.

Система позволяет обращаться к ХРАНИЛИЩУ используя имя каталога в ХРАНИЛИЩЕ, имя отдельного файла или МОДУЛЯ. Модуль представляет собой группу файлов и/или каталогов, которые система **CVS** воспринимает как единый объект. Модуль определяется путём редактирования конфигурационного файла `modules`.

Тексты представляют собой, например, систему программ, которая разрабатывается и корректируется значительное время, может несколько лет. За время разработки и корректировки может выпускаться несколько версий исходных текстов, может меняться состав разработчиков. При смене разработчиков особенно важно документировать изменения, происходящие в исходных текстах программ.

Когда разработчик планирует корректировать тексты какой-то

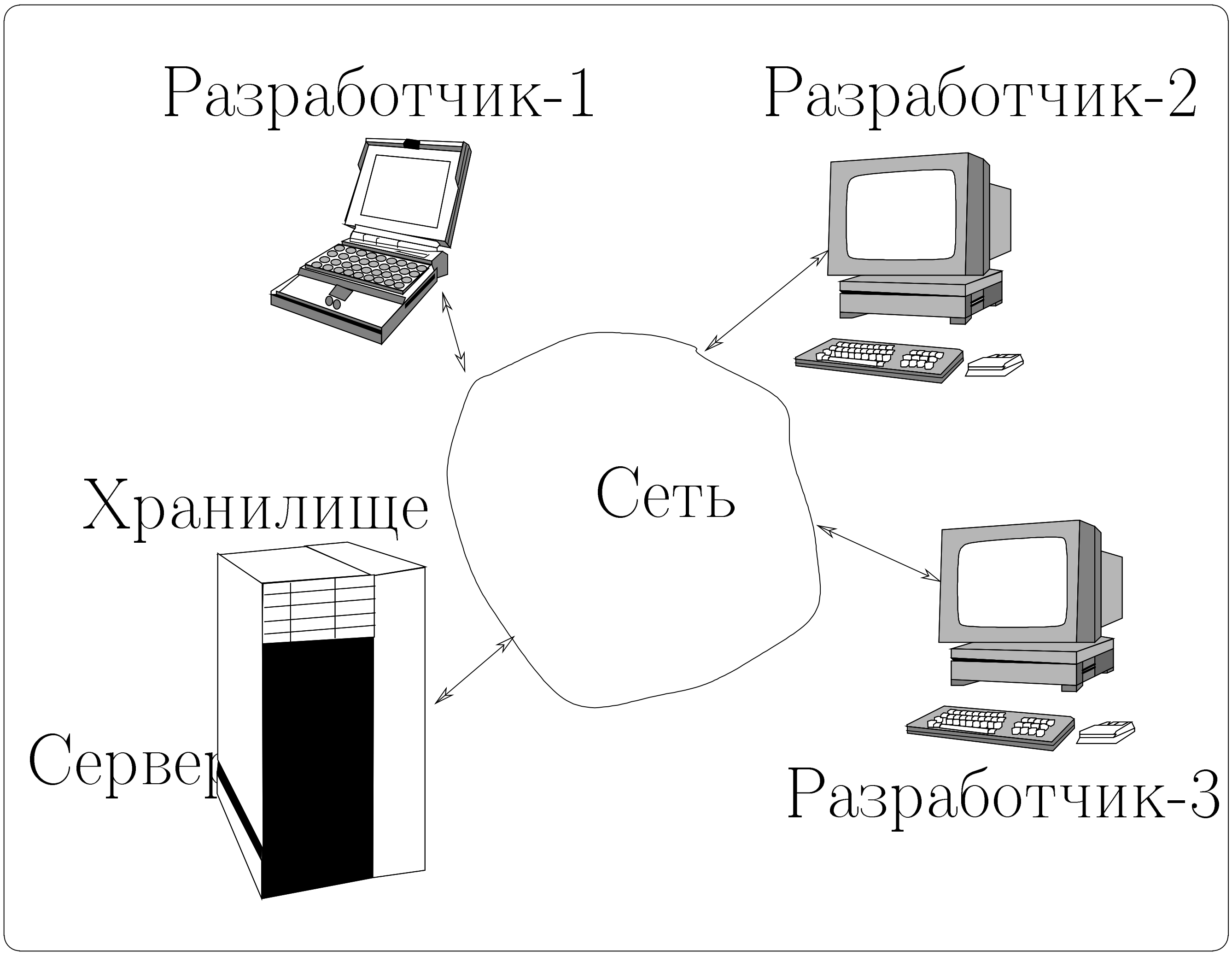

Рис. 14.1: Коллективная работа с использованием CVS.

программы, то он с помощью средств **CVS** копирует файлы из ХРАНИЛИЩА CVS в свой РАБОЧИЙ (текущий) каталог, создавая РАБОЧУЮ КОПИЮ исходных текстов. По завершении корректировок, обновлённые модули помещаются командами **CVS** в ХРАНИЛИЩЕ. При этом старая версия обновлённых текстов также остаётся в ХРАНИЛИЩЕ. Таким образом, несколько разработчиков могут работать с одной и той же программой (обычно с разными частями программы) независимо друг от друга.

ХРАНИЛИЩЕ может находиться как на той же машине, где работают разработчики, так и в любом месте в Интернет. Общая схема такой работы показана на рисунке 14.1.

CVS поддерживает возможность слежения за файлом (Watches), которое выражается в уведомлении разработчика, когда кто-то другой запросил копию этого файла, или получить список разработчиков, которые в данный момент работают с этим файлом.

Разработчик может пометить любой файл в ХРАНИЛИЩЕ как двоичный, чтобы запретить преобразования, которые имеют место при передаче текстовой информации.

Система позволяет также управлять всем процессом разработки. С использованием конфигурационных файлов COMMITINFO, Loginfo, других можно установить автоматический вызов скриптов, при выполнении определённых действий в ХРАНИЛИЩЕ. В скриптах могут быть как автоматическая рассылка уведомлений о конкретных операциях всем разработчикам, так и контроль правильности использования тех или иных параметров.

14.2 Простые примеры использования CVS

Чтобы понять как работает **CVS** мы пройдём через некоторые типовые шаги. Первое, что полезно понять – **CVS** хранит данные в специально организованном ХРАНИЛИЩЕ (РЕПОЗИТАРИИ).

Данное обсуждение предполагает, что ХРАНИЛИЩЕ существует и его имя задано в переменной окружения `$CVSROOT`.

Положим, что вы разрабатываете программу (имя программы `tc`) на языке **C**. Заметим, что, хотя программа и является небольшой, но все-таки состоит из нескольких модулей и одного файла `Makefile` для построения исполняемой программы.

14.2.1 Получение исходных текстов из хранилища

Первый шаг, который вы должны сделать, – это получить исходные тексты вашей программы из ХРАНИЛИЩА и поместить их РАБОЧУЮ КОПИЮ в вашем РАБОЧЕМ КАТАЛОГЕ. Это выполняется командой

```
cvs checkout tc
```

здесь:

cvs – основная программа системы;

checkout – параметр, команда системы **CVS**;

`tc` – как мы помним, имя вашей программы.

По этой команде будет создан новый каталог с именем `tc` и в него будут скопированы исходные тексты программы `tc` из хранилища **CVS**. После выполнения этой команды вы может перейти в каталог `tc`

```
cd tc
```

Если, далее, вы выполните команду `ls`, то увидите что-то похожее на нижеследующее:

```
$ ls
CVS    Makefile  backend.c   driver.c  frontend.c  parser.c
```

Здесь мы видим имена исходных модулей, которые составляют программу **tc** и каталог с именем `CVS`, который автоматически создан программой **cvs** во время выполнения команды **c**heckout. Каталог необходим самой системе **CVS**, поэтому не стоит его удалять или модифицировать.

Далее, вы можете редактировать ваши исходные тексты, отлаживать программу `tc` и, вообще, делать то, что вы обычно делаете во время разработки или отладки.

После отладки и правки исходного текста модуля `backend.c` вы решили поместить его обновлённый вариант в основное хранилище. Вы можете

сделать это посредством команды

```
cvs commit backend.c
```

при этом будет автоматически вызван редактор текстов для того, чтобы дать вам возможность ввести комментарий по поводу изменения исходного текста. Вы можете, например, написать: `Добавлена возможность обхода ошибок`. Далее, вы выходите из редактора, сохранив файл с вашим комментарием и **CVS** записывает все это (ваш новый вариант исходного текста и ваш комментарий) в ХРАНИЛИЩЕ CVS. Заметим, что поскольку вызывается редактор текста, то вводимый вами комментарий может быть весьма обширным. Например, он может состоять не из одной фразы, а из нескольких абзацев.

Имя редактора текста, который автоматически вызывает система, хранится в переменной окружения $CVSEDITOR. Если переменная $CVSEDITOR не установлена, то принимается переменная окружения $EDITOR. Если переменная $EDITOR тоже не установлена, то принимается умолчание имеющее место для вашей операционной системы.

Во время автоматического вызова редактора по команде **CVS commit** вы можете увидеть список файлов, которые возможно были изменены. Этот список базируется на времени модификации файла. Если время модификации рабочей копии в вашем рабочем каталоге отличается от времени модификации этого же файла в ХРАНИЛИЩЕ CVS, то файл будет на подозрении, что он модифицирован. Однако, во время операции **commit** система проверит, действительно ли файл был изменен или была модифицирована лишь дата модификации файла. Если была лишь изменена дата, то время модификации файла в ХРАНИЛИЩЕ CVS не будет изменено.

Если комментарий краток, вы можете избежать вызова редактора текста для ввода комментария:

```
  cvs commit -m "Добавлена возможность обхода ошибок" backend.c
```

14.2.2 Удаление рабочего каталога

По завершении работы с программой **tc** вы можете пожелать перейти к следующей работе. Тем самым вам может понадобиться удаление рабочего каталога (ведь все ваши изменения уже находятся в хранилище **CVS**!), чтобы не засорять ваш основной каталог.

Непосредственно перед удалением каталога полезно использовать команду **release** системы **CVS**, которая проверит состояние вашего рабочего каталога на предмет соответствия хранилищу. Ниже приводится последовательность: команда `cd` – вход из каталога `tc` в родительский каталог, выполнение команды **release** системы **CVS**, затем приведена диагностика времени выполнения команды.

```
$ cd ..
$ cvs release -d tc
  M driver.c
  ? tc
  You have [1] altered files in this repository.
  Are you sure you want to release (and delete) module ‘Tc’ n
  ** Release’ aborted by user choice.
```

Здесь:

`M driver.c`

означает, что данный файл был модифицирован после того, как был скопирован из ХРАНИЛИЩА CVS. Следующая строка

`? tc`

означает, что файл `tc` неизвестен системе **CVS**. Поскольку файл с именем `tc` – это готовая к исполнению программа, то её нет необходимости помещать в ХРАНИЛИЩЕ CVS. Следовательно данное сообщение можно игнорировать. Последняя строка приведённого примера означает, что операция удаления была отменена.

Таким образом, команда **release** проверяет каждый файл в каталоге и просит пользователя подтвердить удаление файла.

Раз мы забыли что были изменения в файле `driver.c`, то полезно установить что именно изменилось, например, командой

```
cd tc
cvs diff driver.c
```

Иными словами, мы снова перешли в каталог `tc`, удаление которого было отменено, и выполнили команду **diff** системы **CVS**. По этой команде система сравнивает рабочую копию модуля `driver.c` с той копией этого файла, которая содержится в ХРАНИЛИЩЕ системы. Вы обнаруживаете в чём разница и далее выполняете последовательность команд:

```
$ cvs commit -m "Добавлены новые циклы" driver.c
Checking in driver.c;
/usr/local/cvsroot/tc/driver.c,v  <--  driver.c
new revision: 1.2; previous revision: 1.1
done
$ cd ..
$ cvs release -d tc
? tc
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module 'Tc': y
```

После этого вы можете спокойно удалить ваш рабочий каталог.

14.3 Хранилище системы CVS

Хранилище **CVS** предназначено для хранения всех исходных текстов (файлов и каталогов), смена версий которых должна быть под контролем.

Обычно, вы никогда не имеете дело напрямую с файлами, содержащимися в хранилище. Предполагается, что вы с помощью средств **CVS** создаёте рабочую копию одного или группы файлов в вашем рабочем каталоге. Следовательно, напрямую вы имеете дело только с рабочей копией. Когда вы изменили рабочую копию, то она должна быть помещена снова в ХРАНИЛИЩЕ, что также выполняется средствами **CVS**. Заметим попутно, что ХРАНИЛИЩЕ и ваш РАБОЧИЙ КАТАЛОГ – разные вещи, лучше их не путать и держать в разных местах, например, в разных каталогах.

Поскольку ХРАНИЛИЩЕ может находиться как на той же машине, где работают разработчики, так и на другом компьютере, то имеется несколько методов доступа к ХРАНИЛИЩУ системы **CVS**: локальный и удалённый методы доступа. Если метод доступе никак не обозначен, то предполагается, что ХРАНИЛИЩЕ находится на том же компьютере. ХРАНИЛИЩЕ состоит из двух частей: одна часть – это собственно исходные тексты пользователя, другая – административная информация самой системы **CVS**.

14.3.1 Создание хранилища CVS

Выберите подходящий диск и каталог, учитывая возможный объём, который потребуется в будущем. Затем выполните команду создания хранилища **CVS**

```
cvs init -d ~/ProJ init
```

по этой команде в вашем главном каталоге будет создан подкаталог с именем `ProJ`. Если вы посмотрите внутрь каталога `ProJ`, то увидите созданный административный подкаталог системы **CVS** с именем `ProJ/CVSROOT`. если вы случайно укажете уже существующее хранилище в команде **init**, то команда не испортит существующего хранилища.

Параметр **-d** указывает, что имя каталога, в котором будет организовано хранилище находится в командной строке в качестве аргумента для данного параметра. Имя должно быть абсолютным именем. Имя каталога, в котором находится хранилище можно указать в переменной окружения $CVSROOT. Таким образом, предыдущую команду создания можно было написать

```
cvs init
```

если переменная $CVSROOT уже содержит абсолютное имя каталога, где планируется разместить ХРАНИЛИЩЕ.

14.4 Форма содержания данных в хранилище CVS

В ряде случаев полезно иметь некоторые представления об организации хранилища, чтобы понимать, например, какие права доступа надо устанавливать для файлов хранилища или как ограничивать доступ к файлам.

14.4.1 Какие файлы содержатся в хранилище

В целом, структура хранилища представляет собой иерархию (дерево) каталогов соответствующей структуре рабочего каталога. Предположим, что хранилище находится в каталоге `/usr/local/cvsroot`. На рисунке 14.2 приведено возможное дерево каталогов (показаны только каталоги):

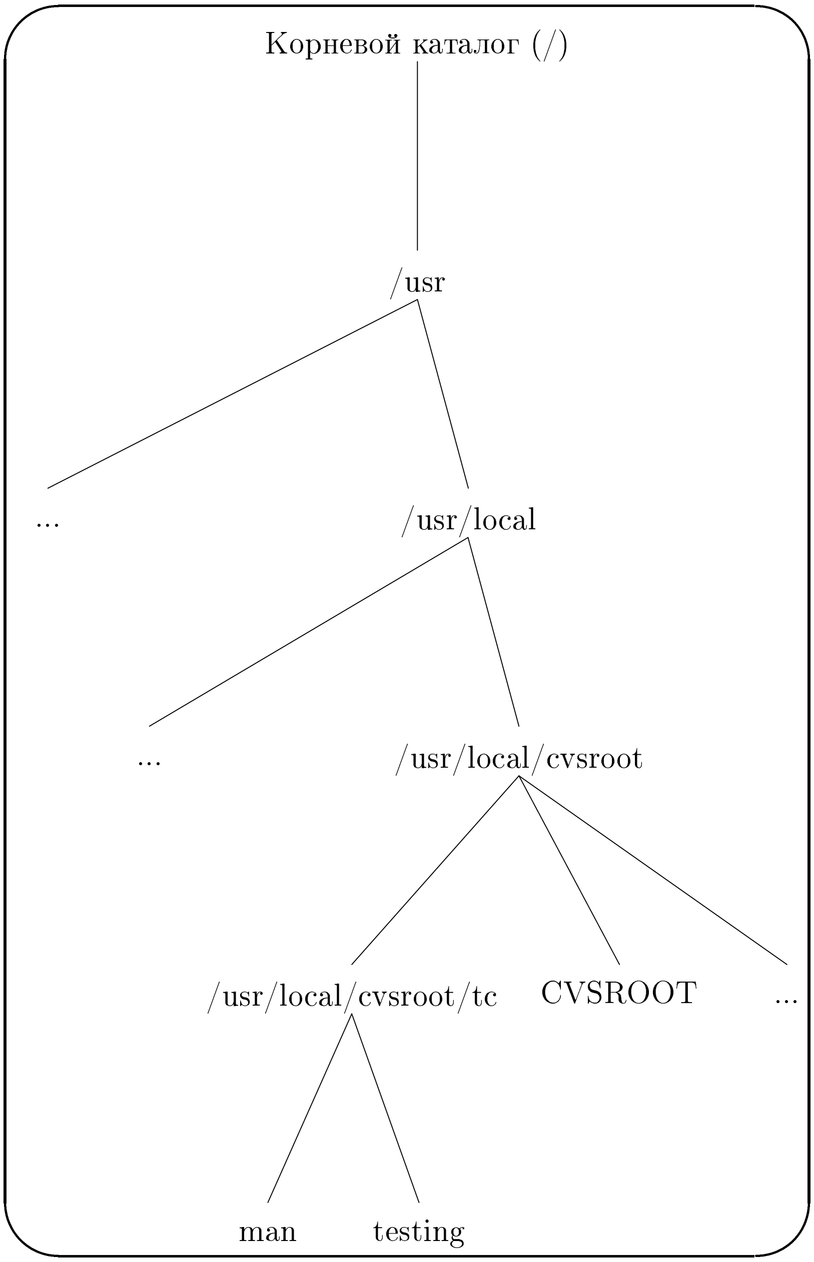

Рис. 14.2: Возможное дерево каталогов

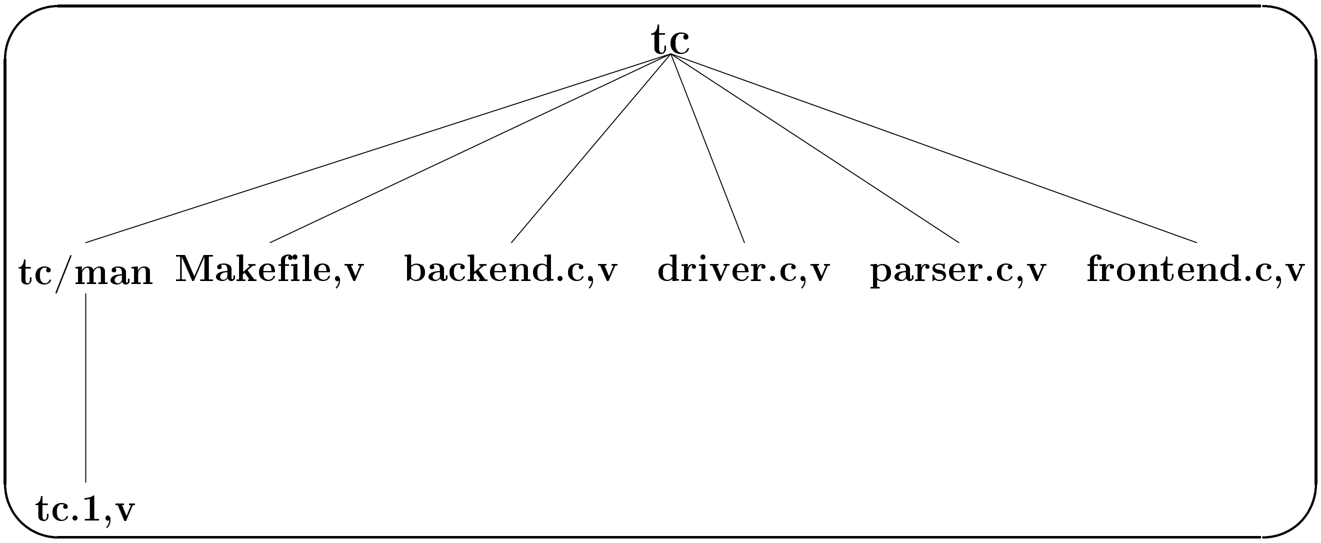

Рис. 14.3: Каталог tc

В каталогах имеются ИСТОРИЧЕСКИЕ ФАЙЛЫ, по одному на каждый файл, который находится под контролем **CVS**. Имя Исторического файла представляет собой имя файла под контролем CVS с окончанием `,v`. Таким образом, каталог `tc` выглядит примерно как показано на рисунке 14.3.

ИСТОРИЧЕСКИЕ ФАЙЛЫ содержат достаточно информации, чтобы на их основе воссоздать любую версию файла, протокол всех комментариев команды **commit** и имя пользователя, который выполнил команду **commit**. Исторические файлы ещё называют файлы **RCS**, поскольку **RCS** (система управления версиями) была первой системой, которая запоминала файлы в таком формате. Полное описание формата может быть найдено с помощью команды **man**, следует смотреть страницу **rcsfile(5)**. Этот формат файлов стал весьма общим — много различных систем а не только **CVS** или **RCS** могут как минимум импортировать исторические файлы в таком формате.

Тем не менее, файлы **CVS** несколько отличаются от файлов **RCS**. Здесь мы не станем рассматривать этот вопрос подробнее.

14.5 Удаленные хранилища

Рабочие копии программ вы можете хранить в рабочем каталоге на вашей личной машине, а само хранилище может располагаться на другой машине в другом конце офиса, в другом здании или на другом континенте. Использование удалённого хранилища в целом происходит таким же порядком как и использование локального хранилища. Исключение составляет лишь формат имени хранилища, который в удалённом случае имеет вид

:METHOD:USER@HOSTNAME:/path/to/repository

Детали формата зависят от метода доступа, с помощью которого вы обращаетесь к удалённому хранилищу.

14.5.1 Метод доступа rsh

Система **CVS** может использовать протокол **rsh** (имеется в виду **Remote SHell** - удалённая оболочка), чтобы выполнить необходимые операции в удалённом хранилище. Таким образом, на рабочей станции (сервере), где находится хранилище должен быть соответствующим образом заполнен файл с именем `.rhosts`.

Предположим, что вы зарегистрированы на вашей машине *local.pnpi.spb.ru* как пользователь с именем **mozart**. А ваш сервер находится на машине *rsgi01.rhic.bnl.gov*. При этом все тексты хранятся под именем пользователя **bach**. Тогда вам необходимо в файл *rsgi01.rhic.bnl.gov:*`~bach/.rhosts` добавить строку:

```
local.pnpi.spb.ru mozart
```

После этого на своей локальной машине *local.pnpi.spb.ru* проверьте, что всё работает нормально

```
rsh -l bach rsgi01.rhic.bnl.gov 'echo $PATH'
```

Вы должны увидеть содержимое переменной окружения `$PATH` на машине *rsgi01.rhic.bnl.gov*. Обратите внимание, что переменная должна указывать на имя каталога, в котором находится программный сервер **CVS**. Вы можете указать другим способом откуда вызывать сервер системы **CVS**. Для этого вам следует установить на локальной машине *local.pnpi.spb.ru* переменную окружения `$CVS_SERVER` с именем сервера (программы), который вы хотите использовать, например, `/usr/local/bin/cvs-2000`.

Если используется протокол **rsh**, то нет необходимости устанавливать другие системные файлы, чтобы запускать демон сервера системы **CVS**.

Имеется два метода доступа, которые вы можете указать в переменной окружения `$CVSROOT` при использовании протокола **rsh**. Использовав метод `:server:`, вы определяете внутренний **rsh** клиент, который поддерживается только некоторыми портами **CVS**. А метод `:ext:` определяет внешнюю программу **rsh**. По умолчанию используется программа **rsh**, однако вы можете задать имя программы в переменной окружения `$CVS_RSH`. Нетрудно догадаться, что если вы использовали другое имя, например, **ssh**, то вам следует использовать другие имена файлов, например, `.shosts` для **ssh**.

Продолжая наш прежний пример, предположим, что вы хотите получить доступ к файлу `foo` в хранилище `/usr/local/cvsroot/` на машине *rsgi01.rhic.bnl.gov*. Тогда, чтобы создать рабочую копию файла на вашей локальной машине *local.pnpi.spb.ru* вам следует написать:

```
cvs -d :ext:bach@rsgi01.rhic.bnl.gov:/usr/local/cvsroot checkout foo
```

При этом часть `bach@` может быть опущена, если имя пользователя на обеих машинах совпадает.

14.5.2 Прямое соединение с сервером

Клиент **CVS** может соединиться с сервером непосредственно с использованием одной из схем аутентификации. Такая возможность является крайне необходимой, если сервер находится за защитным сервером (firewall), а система **Kerberos** тоже недоступна.

Чтобы использовать такой метод следует произвести некоторые настройки на стороне клиента и на стороне сервера.

Установки на серверной стороне

На серверной стороне необходимо отредактировать системный конфигурационные файл **/etc/inetd.conf** таким образом, что программа **inetd** будет **знать**, что надо выполнить команду

cvs pserver

когда сервер обнаружит попытку соединения на соответствующий порт. По умолчанию номер порта равен 2401. Номер порта может быть другим, если ваш клиент был скомпилирован с установленной переменной $CVS_AUTH_PORT, которая указывала другой порт.

Если ваша **inetd** допускает в файле **/etc/inetd.conf** обычные номера портов, то следующих двух строк достаточно:

2401 stream tcp nowait root /usr/local/bin/cvs
cvs -b /usr/local/bin pserver

Параметр **-b** определяет каталог, в котором содержатся готовые к исполнению программы **RCS** на вашем сервере. Можно также использовать параметр **-T**, чтобы определить каталог для временных файлов.

Если ваша **inetd** предпочитает символические имена сервисов вместо простых номеров портов, то вы можете поместить в файл `/etc/services` строку

```
cvspserver      2401/tcp
```

Тогда вы сможете использовать символическое имя сервиса `cvspserver` в файле `/etc/inetd.conf` вместо номера порта.

Естественно, что после изменения файлов вам необходимо заставить **inetd** заново прочесть конфигурационные файлы или перезапустить программу **inetd**.

Поскольку клиент содержит и передаёт пароли также как обычный текст, то может быть использован отдельный парольный файл **CVS**. Этот файл находится в `$CVSROOT/CVSROOT/passwd`. Его формат такой же как `/etc/passwd` за исключением того, что он имеет всего два поля: имя пользователя и пароль. Например,

shevel:ULtgRLXo7NRxs
oreshkin:1sOp854gDF3DY

Пароль закодирован в соответствии со стандартом функции **crypt()**. Следовательно, вы можете скопировать поле пароля из файла `/etc/passwd`.

Когда проверяется пароль, то **CVS** сначала проверяет есть ли пользователь в файле `$CVSROOT/CVSROOT/passwd`. Если пользователь найден, то сверяется пароль. Если пользователя нет в этом файле или сам файл `$CVSROOT/CVSROOT/passwd` отсутствует, то сервер **CVS** пытается найти пользователя среди зарегистрированных пользователей сервера.

В то время когда **CVS** выполняет процедуру аутентификации, она работает при той же сумме привилегий, что и обычный пользователь с тем же именем.

Пока единственный способ записать пароль является копирование его из какого-то другого места. Быть может скоро появится команда `cvs passwd`.

Установки на стороне клиента

До того как соединиться с сервером, клиент должен ЛОГИРОВАТЬСЯ с помощью команды:

cvs login

Логирование заключается в проверке пароля на стороне сервера, а также в записывании пароля для более позднего взаимодействия с сервером **CVS**, иначе, для последующих транзакций. Команда **cvs login** должна получить имя пользователя, имя сервера и абсолютное имя каталога, где находится хранилище **CVS**. Она берёт эту информацию из переменной окружения $CVSROOT. Команда **cvs login** запросит интерактивно пароль пользователя. Например,

cvs -d :pserver:bach@rsgi01.rhic.bnl.gov:/usr/local/cvsroot login
CVS password:

Если пароль проверен успешно, то всё нормально, если нет - будет диагностика, что пароль неверен. Раз вы уже логированы, то можете выполнять и другие команды

cvs -d :pserver:bach@rsgi01.rhic.bnl.gov:/usr/local/cvsroot checkout foo

В обоих предыдущих командах используется метод **:pserver:**. Если он не был бы указан, то система **CVS** предполагала бы, что надо использовать **rsh**.

После того как файл из предыдущего примера **foo** переписан в ваш рабочий каталог, то при выполнении команд системы в этом каталоге нет необходимости каждый раз указывать в явном виде где расположено хранилище, поскольку оно было записано в вашем рабочем каталоге (подкаталог с именем CVS).

По умолчанию, пароль записывается в файл $HOME/.cvspass в специальном формате. Вы можете сменить умолчание посредством установки переменной окружения CVS_PASSFILE. Если вы планируете использовать данную переменную, то установите её новое значение ДО выполнения логирования на удалённый сервер.

Другая переменная окружения – CVS_PASSWORD сменит ВСЕ запомненные пароли. Если она установлена, то **CVS** будет использовать это значение для всех процедур аутентификации при соединении с любыми серверами.

Прямое соединение с использованием Kerberos

Основной недостаток использования протокола **rsh** состоит в том, что все данные должны пройти через дополнительные программы, что, естественно, замедляет передачу. Если вы используете систему **Kerberos**, вы можете соединятся и передавать файлы посредством прямого соединения TCP, поскольку аутентификацию выполняет **Kerberos**.

Чтобы реализовать такой способ требуется заново скомпилировать систему **CVS** с поддержкой **Kerberos**; когда конфигурируется **CVS**, она пытается определить имеется ли **Kerberos** или вы можете использовать флаг -**with-krb4**, чтобы явно указать использование **Kerberos** в конфигурации.

По умолчанию, передаваемые данные **не** шифруются. Возможность шифрования должна быть указана во время трансляции как клиентской части, так и серверной. Следует использовать параметр -**enable-encryption**, чтобы включить шифрование. Вы должны также использовать глобальный параметр **-x**, чтобы запросить шифрование.

Для использования **Kerberos** вам необходимо добавить в файл **inetd.conf** на серверной стороне строку

cvs kserver

Клиент использует по умолчанию порт номер 1999. Если вы хотите использовать другой порт его следует определить в переменной окружения `CVS_CLIENT_PORT` на стороне клиента.

Когда вы хотите использовать **CVS**, вам обычно надо получить разрешение (*ticket*), которое позволит вам логироваться на сервер. Таким образом, вы можете использовать команду

cvs -d :kserver:rsgi01.rhic.bnl.gov:/user/local/cvsroot checkout foo

Предыдущие версии **CVS** пытались бы использовать протокол **rsh** в случае не успеха обращения к **Kerberos**. Данная версия **CVS** не станет этого делать.

14.5.3 Несколько хранилищ

В некоторых ситуациях полезно иметь более, чем одно хранилище. Особенно это полезно, если вы имеете несколько групп разработчиков и/или несколько проектов, которые не имеют общих программ или частей программ. Если вы имеете несколько хранилищ, то, чтобы использовать нужное хранилище, вы должны задать его имя в переменной окружения `$CVSROOT` или указать в качестве аргумента в параметре **-d** при вызове программы `cvs`.

Большое преимущество использования нескольких хранилищ состоит в том, что хранилища могут находиться на разных серверах. Но, с другой стороны, вы не можете с помощью одной команды системы **CVS** обработать сразу несколько хранилищ (репозитариев).

Наконец, если вы планируете вести несколько проектов и планируете иметь несколько логически раздельных хранилищ, то технически проще завести одно хранилище с несколькими деревьями каталогов, каждое из которых соответствовало бы отдельному проекту.

14.6 Версии файлов и программных продуктов

Для большинства случаев применения CVS нет необходимости уделять много внимания номерам версий исходных текстов: CVS автоматически назначает номера 1.1, 1.2, 1.3 и т.д. Тем не менее, некоторые разработчики хотят знать больше о номерах версий.

Если кто-то захочет иметь список версий для более чем одного файла, то это проще сделать с использованием понятия **тег** (**tag**), который представляет собой символическое имя версии. Оно может быть присвоено числовому значению версии в каждом файле.

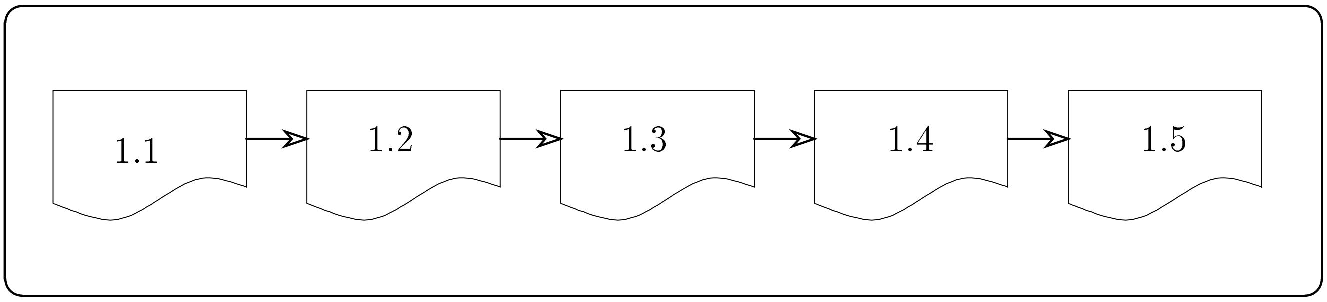

Рис. 14.4: Последовательные версии

14.6.1 Номера версий

Для каждой версии файла имеется уникальный номер версии (revision number). Номер версии выглядит как 1.1, 1.2, 1.3.2.2 или даже 1.3.2.2.4.5. Номер версии всегда имеет чётное число целых чисел, разделённых точками. По умолчанию версия 1.1 является первой (или исходной) версией файла. Каждая последующая версия добавляет единицу в самую правую позицию номера версии. Рисунок 14.4 показана простая структура последовательных версий.

Имеются номера версий, содержащих более, чем одну точку, например как 1.3.2.2. Такие номера версий представляют версии **ветвей**, которые обсуждаются в разделе 14.7.

14.6.2 Присваивание версий

Когда вы добавляете новый файл, вторая цифра будет всегда 1, а первое число будет равно наибольшему первому числу в версии любого файла в данном каталоге. Например, если текущий каталог содержит файлы, чьи наиболее свежие версии имеют номера: 1.7, 3.1 и 4.12, тогда вновь добавляемому файлу будет присвоен номер версии 4.1.

Обычно нет нужды специально беспокоиться о номерах версий, система **CVS** сама наблюдает за ними и модифицирует, когда необходимо. Проще их воспринимать как некоторые внутренние переменные **CVS**. Теги (символьные имена версий) дают более удобный механизм для того, чтобы отделить, например, версию 1 от версии 2 вашего продукта. Однако, если вы хотели бы установить версии файлов в виде числовом выражении, то вы можете использовать команду

```
cvs commit -r 3.0
```

Здесь параметр **-r** подразумевает параметр **-f** в том смысле, что все файлы будут отмечены в хранилище как обновлённые независимо от того, имело ли место реальное изменение файлов. При этом все файлы получат номер версии 3.0, если до выполнения данной команды не было файлов с версией больше, чем 3.0. Таким образом, если ваши файлы уже имеют версию 3.0, то вы не сможете присвоить им версию 1.3.

Если вы желаете поддерживать несколько параллельных версий вашего продукта (скажем, 1.x и 3.x), то следует использовать механизм **ветвей**.

14.6.3 Теги

Номера версий файлов живут своей собственной жизнью. Совсем необязательно, чтобы они как-то были связаны с версиями вашего продукта.

В зависимости от того, как вы используете **CVS**, версии отдельных файлов могут измениться несколько раз между двумя последовательными версиями вашего продукта. В качестве примера мы можем рассмотреть комплект исходных текстов системы RCS 5.6, которые имеют следующие номера версий:

`ci.c`	`5.21`
`co.c`	`5.9`
`ident.c`	`5.3`
`rcs.c`	`5.12`
`rcsbase.h`	`5.11`
`rcsdiff.c`	`5.10`
`rcsedit.c`	`5.11`
`rcsfcmp.c`	`5.9`
`rcsgen.c`	`5.10`
`rcslex.c`	`5.11`
`rcsmap.c`	`5.2`
`rcsutil.c`	`5.10`

Вы можете использовать команду **tag**, чтобы дать символическое имя (**тег**) определённой версии отдельного файла. Вы можете также использовать команду **status** с параметром **-v**, чтобы увидеть все теги, которые имеет данный файл, а также номер версии, которую обозначает данный тег.

Имена тегов должны начинаться с буквы в любом регистре и могут содержать любые буквы, цифры и только два специальных знака: **-** (минус) и _ (подчёркивание). Два тега с именами **BASE** и **HEAD** зарезервированы за системой **CVS**. При выполнении большого проекта полезно придерживаться определённых соглашений об используемых именах тегов. Например, если ваша система имеет имя TERCIYA, то имя тега TERCIYA-3_0 может представлять версию 3.0. Вы можете также использовать файл **taginfo**, чтобы описать соглашение об именах тегов.

Следующий пример показывает, как вы можете добавить тег к файлу. Команды добавления тега должны быть выданы внутри рабочего каталога, т.е. там, где находится рабочая копия вашего файла (модуля).

```
$ cvs tag rel-0-4 backend.c
T backend.c
$ cvs status -v backend.c
==================================================================
File: backend.c          Status: Up-to-date

    Version:             1.4     Tue Dec  1 14:39:01 1992
    RCS Version:         1.4     /u/cvsroot/yoyodyne/tc/backend.c,v
    Sticky Tag:          (none)
    Sticky Date:         (none)
    Sticky Options:      (none)

    Existing Tags:
       rel-0-4                      (revision: 1.4)
```

Здесь приведён нечасто встречающийся пример, когда тегом помечается единственный файл (модуль). Обычно необходимо пометить определённым тегом все файлы, которые входят в продукт или составляют вместе определённый программный модуль. Это выполняется в те моменты, которые важны для всего проекта, например, при выпуске очередной версии программного продукта.

```
$ cvs tag rel-1-0 .
cvs tag: Tagging .
T Makefile
T backend.c
T driver.c
T frontend.c
T parser.c
```

В примере тегом помечаются все модули рабочего каталога (заметим попутно, что и все подкаталоги, если они есть). Если впоследствии, вы захотите взять из хранилища лишь те файлы, которые относятся к версии продукта rel-1-0, то это выполняется командой

cvs checkout -r rel-1-0 tc

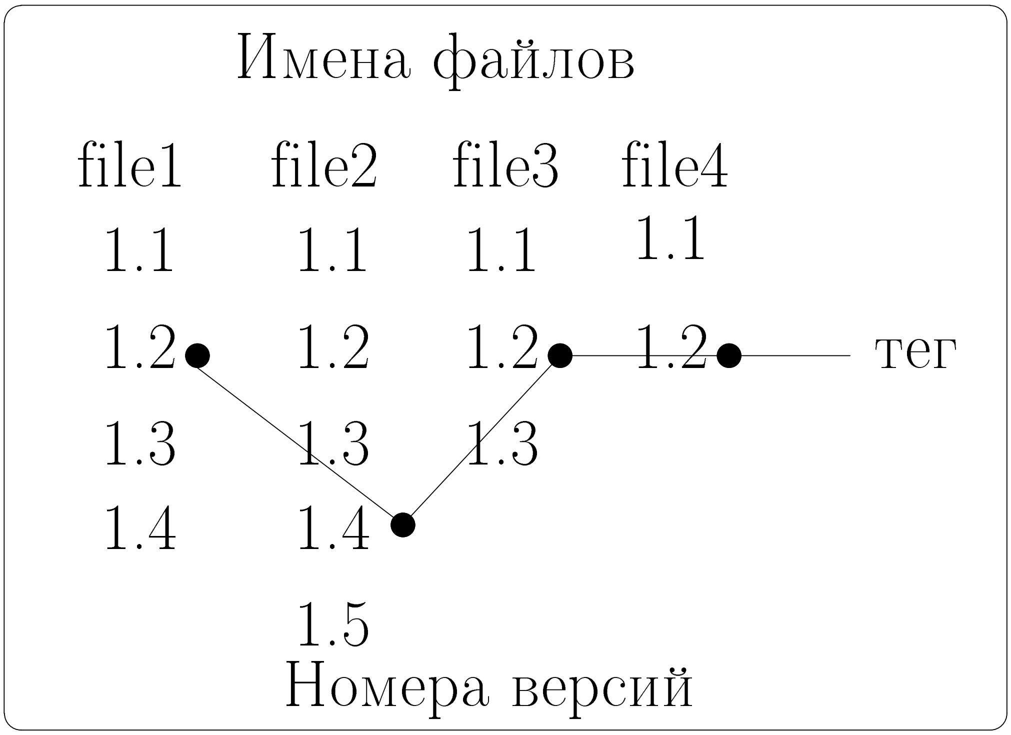

Рис. 14.5: Отметка тегом

что, естественно, чрезвычайно удобно.

Когда тегом отмечаются несколько файлов, то это можно изобразить графически двояким образом. Первый вариант представлен на рисунке 14.5.

Здесь видно, что одни файлы меняются со временем чаще, другие – реже. В какой-то момент времени мы отметили их тегом (звёздочки). Таким образом, мы сможем вызвать эти версии файлов в любой момент, когда нам это понадобится.

То же самое можно нарисовать иначе как показано на рисунке 14.6.

14.6.4 Липкий тег

Иногда рабочая копия комплекта исходных текстов имеет дополнительную информацию, например, такое может быть, когда вы работаете с ветвью или рабочая копия ограничена версиями определённой даты командами **checkout -D** или **update -D**. Поскольку такие данные сохраняются, т.е. воздействуют на выполнение последующих команд, то будем их называть ЛИПКИЕ.

В большинстве случаев ЛИПКОСТЬ представляет собой скрытое свойство **CVS**, поэтому вам не требуется его изучать. Однако, полезно знать о том, что липкие теги есть, хотя бы для избежания побочных эффектов.

Вы можете использовать команду **status**, чтобы увидеть существуют ли

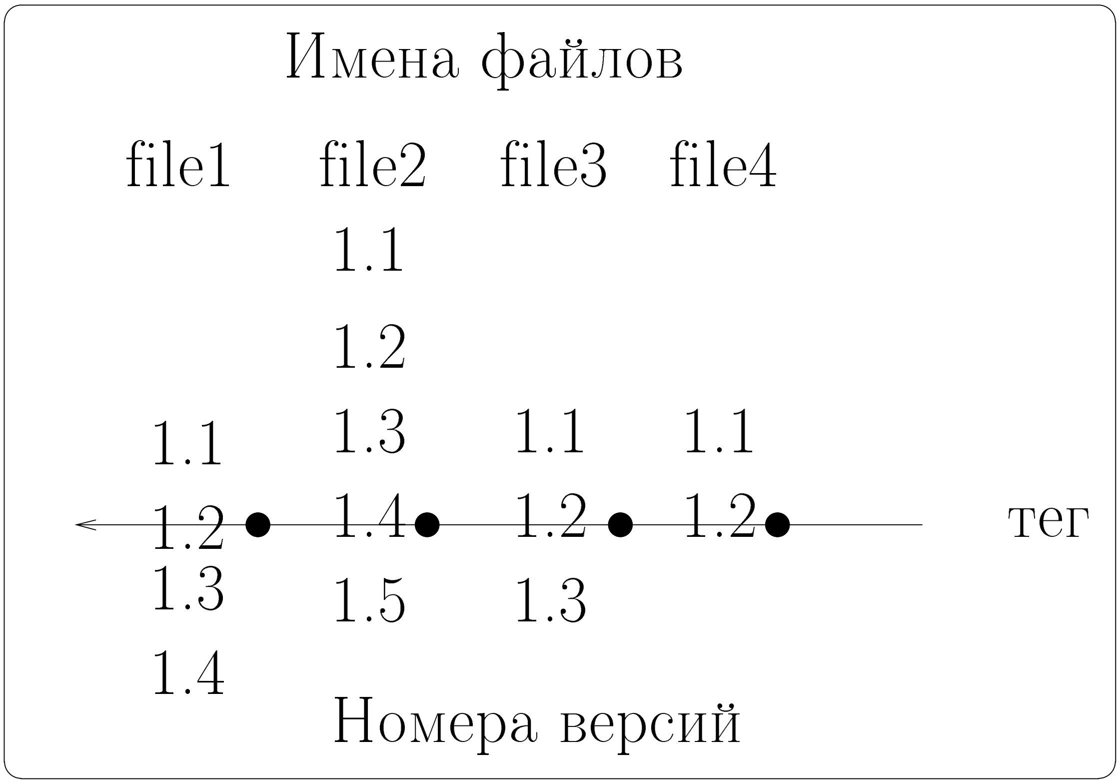

Рис. 14.6: Отметка тегом

липкие теги или установленные даты:

cvs status driver.c

```
===================================================================
File: driver.c          Status: Up-to-date

    Version:            1.7.2.1 Sat Dec  5 19:35:03 1992
    RCS Version:        1.7.2.1 /u/cvsroot/yoyodyne/tc/driver.c,v
    Sticky Tag:         rel-1-0-patches (branch: 1.7.2)
    Sticky Date:        (none)
    Sticky Options:     (none)
```

Липкие теги останутся в ваших рабочих файлах до того момента, пока вы не удалите их с помощью команды

```
cvs update -A
```

Параметр **-A** разыскивает версию файла в голове ствола и забывает все липкие теги, даты или параметры.

Наиболее общеупотребительное использование липких тегов состоит в идентификации того, какая ветвь является рабочей в данный момент. Однако, липкие теги используются и вне контекста ветвей. Например, вы хотите избежать обновления (команда **update**) вашего рабочего каталога, чтобы изолировать себя на время от дестабилизирущих частых изменений со стороны других разработчиков. Конечно, вы можете просто воздержаться от выполнения команды **update**. Однако вы хотели бы не обновлять лишь группу файлов из большого дерева каталогов. В этом случае может помочь липкий тег. Если вы выполните команду **cvs checkout** для определённой версии, например, 1.4, рабочая копия файлов станет липкой. Последующие команды **cvs update** не будут разыскивать последние версии файлов в хранилище для файлов, которые находятся в вашем рабочем каталоге до того момента пока вы не выполните **cvs update -A**, очистив липкий тег.

Таким же образом, использование параметра **-D** в командах **checkout** и **update** устанавливает ЛИПКУЮ ДАТУ, которая подобным же образом будет использоваться в последующих командах **CVS**.

Часто вы захотите найти старую версию файла без установки липкого тега. Способ состоит в том, чтобы использовать параметр **-p** в командах **checkout** или **update**, который направляет содержание файла на стандартное устройство вывода. Например, у вас есть файл **file1** версии 1.1 и вы его удалили из хранилища. Теперь вы хотите добавить файл к хранилищу снова с тем же содержимым, которое было ранее. Ниже приведён способ как это сделать

```
$ cvs update -p -r 1.1 file1 >file1
===================================================================
Checking out file1
RCS:  /tmp/cvs-sanity/cvsroot/first-dir/Attic/file1,v
VERS: 1.1
***************
$ cvs add file1
cvs add: re-adding file file1 (in place of dead revision 1.2)
cvs add: use 'cvs commit' to add this file permanently
$ cvs commit -m test
Checking in file1;
/tmp/cvs-sanity/cvsroot/first-dir/file1,v  <--  file1
new revision: 1.3; previous revision: 1.2
done
$
```

14.7 Ветвление и объединение

CVS позволяет вам выделить какие-то изменения проекта в отдельную линию разработки, которую называют **ветвь** (**branch**). Когда вы меняете файлы в одной из ветвей, то эти изменения не появятся в основном стволе или в других ветвях.

Позднее вы сможет переместить ваши изменения из одной ветви в другую или в основной ствол посредством **объединения** (**merging**). Объединение включает прежде всего выполнение команды **cvs update -j**, чтобы объединить изменения в рабочем каталоге. После этого вы можете утвердить (**commit**) эту версию, тем самым реально скопировав изменения в другую ветвь разработки.

14.7.1 Для чего удобны ветви?

Предположим, что версия вашего продукта **tc** 1.0 вышла и начала использоваться. Вы продолжаете разрабатывать продукт **tc**, очередная версия которого 1.1 выйдет в ближайшие несколько месяцев. Однако, в это время ваши заказчики или партнёры стали жаловаться на фатальные ошибки в версии 1.0. Вы начали искать и быстро обнаружили ошибку, которую немедленно следует устранить. Однако наиболее свежие версии файлов представляют весьма нестабильное состояние и станут стабильными лишь через месяц или более. Таким образом, вы не можете исправить ошибку в наиболее свежих версиях и предоставить их заказчикам.

В такой ситуации удобнее всего создать отдельную ветвь для всех файлов, которые составляют ваш продукт **tc** версии 1.0. Там вы сможете менять файлы, исправляя текущие ошибки в программах. Эти исправления никак не будут влиять на основной ствол разработки. Когда срочные исправления внесены и заказчики удовлетворены, вы можете выбрать стиль поведения: оставить ваши исправления в ветви или инкорпорировать их в основной ствол разработки.

14.7.2 Создание ветви

Вы можете создать ветвь командой **tag -b**; например, предполагая, что вы в рабочем каталоге

```
cvs tag -b rel-1-0-patches
```

Эта команда отделяет ветвь с именем **rel-1-0-patches**, которая базируется на текущих версиях файлов в рабочем каталоге.

Важно понять, что ветви создаются в хранилище, а не в рабочем каталоге. Создание ветви базирующейся на текущих версиях файлов, как показано в предыдущем примере, не означает, что рабочая копию комплекта файлов в рабочем каталоге немедленно начнёт рассматриваться как новая ветвь. Как это сделать мы узнаем чуть позже.

Новую ветвь можно создать, не используя рабочей копии, посредством команды **rtag**

```
cvs rtag -b -r rel-1-0 rel-1-0-patches tc
```

Здесь параметр **-r rel-1-0** говорит, что новая ветвь должна корениться (отрастать) в версии, которая соответствует тегу **rel-1-0**. Совсем не обязательно, что это будет наиболее свежая версия. Часто удобно выделить ветвь в старой версии продукта (той версии, где была исправлена ошибка).

Как и с командой **tag** параметр **-b** создаёт новую ветвь и с командой **rtag**. Заметим однако, что числовое значение номера версии, который удовлетворяет **rel-1-0** может быть различным для каждого файла.

Таким образом, полный эффект команды состоит в создании новой ветви с именем **rel-1-0-patches** в модуле **tc** и основывающейся на комплекте файлов, входящих в версию **rel-1-0** модуля (продукта) **tc**.

14.7.3 Доступ к ветвям

Вы можете получить копию ветви одним из двух способов: прочесть ветвь из хранилища с помощью команды **checkout** или переключить существующую рабочую копию на требуемую ветвь проекта.

Чтение конкретной ветви из хранилища с помощью команды **checkout** выполняется следующим образом:

```
cvs checkout -r rel-1-0-patches tc
```

Здесь за параметром **-r** следует имя тега с именем **rel-1-0-patches**, который описывает ветвь.

Переключить рабочий каталог, в котором уже имеется рабочая копия, на требуемую ветвь можно следующим образом:

```
cvs update -r rel-1-0-patches tc
```

Несущественно, если содержимое рабочего каталога соответствовало основному стволу или другой ветви. По команде **update -r** будет выполнено слияние (объединение) тех изменений, которые вы уже сделали в рабочем каталоге, и содержимое ветви в хранилище. Если при слиянии возникнут конфликты, то вы получите соответствующую диагностику.

Раз содержимое вашего рабочего каталога соответствует конкретной ветви, то это соответствие таким и останется, пока вы не скажете явно что-то другое. Это значит, что любые подтверждения (**commit**) изменений в файлах будут проявляться лишь в конкретной ветви. В то же время никаких изменений не будет происходить ни в других ветвях, ни в основном стволе.

Чтобы определить какая ветвь находится в данный момент в рабочем каталоге, можно использовать команду **status**:

```
cvs status -v driver.c backend.c
```

```
===================================================================
File: driver.c          Status: Up-to-date

    Version:            1.7     Sat Dec  5 18:25:54 1992
    RCS Version:        1.7     /u/cvsroot/yoyodyne/tc/driver.c,v
    Sticky Tag:         rel-1-0-patches (branch: 1.7.2)
    Sticky Date:        (none)
    Sticky Options:     (none)

    Existing Tags:
        rel-1-0-patches             (branch: 1.7.2)
        rel-1-0                     (revision: 1.7)

===================================================================
File: backend.c         Status: Up-to-date

    Version:            1.4     Tue Dec  1 14:39:01 1992
    RCS Version:        1.4     /u/cvsroot/yoyodyne/tc/backend.c,v
    Sticky Tag:         rel-1-0-patches (branch: 1.4.2)
    Sticky Date:        (none)
    Sticky Options:     (none)

    Existing Tags:
        rel-1-0-patches             (branch: 1.4.2)
        rel-1-0                     (revision: 1.4)
        rel-0-4                     (revision: 1.4)
```

В выводимой информации обратите внимание на ЛИПКИЙ тег (STICKY TAG). Он содержит информацию о конкретной ветви, если таковая есть.

Не стоит смущаться тем фактом, что номера версий ветви для каждого файла различны (1.7.2 и 1.4.2 соответственно). Тег ветви является тем же самым (**rel-1-0-patches**) и, следовательно, эти файлы находятся в одной ветви. Номера 1.7.2 и 1.4.2 отражают лишь конкретное состояние соответствующих файлов в момент создания ветви. По этим числам из вышеприведённого примера можно лишь заключить, что файл *driver.c* менялся чаще, чем файл *backend.c* до момента создания данной ветви.

14.7.4 Ветви и номера версий

Как уже отмечалось, обычно история последовательных версий продукта имеют линейную структуру, как показано на рисунке 14.4. Однако **CVS** не ограничивается лишь линейными представлениями процесса разработки. Основное дерево (ствол) версий может быть разбито на несколько независимых ветвей. Изменения, которые имеют место в конкретной ветви, могут быть позже перемещены как в основной ствол, так и в другую ветвь.

Каждая ветвь имеет номер ветви, который состоит из нечётного числа десятичных целых, разделённых точками. Номер ветви составляется из номера версии программного продукта с добавлением десятичного целого. Наличие номера ветви позволяет иметь несколько ветвей от одной версии продукта.

Все версии в ветви составлены из номера ветви, за которым следует порядковый номер версии. Рисунок 14.7 поясняет нумерацию ветвей на примерах.

Обычно вы не нуждаетесь в точном описании того, как образуются новые номера ветвей, поэтому кратко обрисуем как это работает в целом. Когда **CVS** образует ветвь, она берёт первое неиспользуемое чётное десятичное целое начиная с 2. Таким образом, если вы захотите образовать ветвь от версии 7.6, то номер ветви будет равен 7.6.2. Все ветви с номерами оканчивающимися на 0 (нуль) используются самой системой **CVS**.

14.7.5 Магические номера ветвей

Этот раздел описывает возможности **CVS** называемые МАГИЧЕСКИЕ ВЕТВИ. Они не требуются пользователю при обычной работе, однако они видны иногда в диагностике, поэтому полезно представлять для чего они служат.

Номера ветвей состоят из нечётного числа десятичных целых, разделённых точками. Однако это не полная правда. В целях повышения эффективности своей работы **CVS** иногда вставляет дополнительный ноль (0) во вторую справа позицию номера ветви (таким образом 1.2.4 становится 1.2.0.4, 8.9.10.11.12 становится 8.9.10.11.0.12 и так далее).

CVS выполняет всю работу связанную с магическими номерами внутри, но иногда они могут появляться снаружи:

- Магический номер появляется в выводе команды **cvs log**.
- Вы не можете определить символическое имя ветви в команде **cvs admin**.

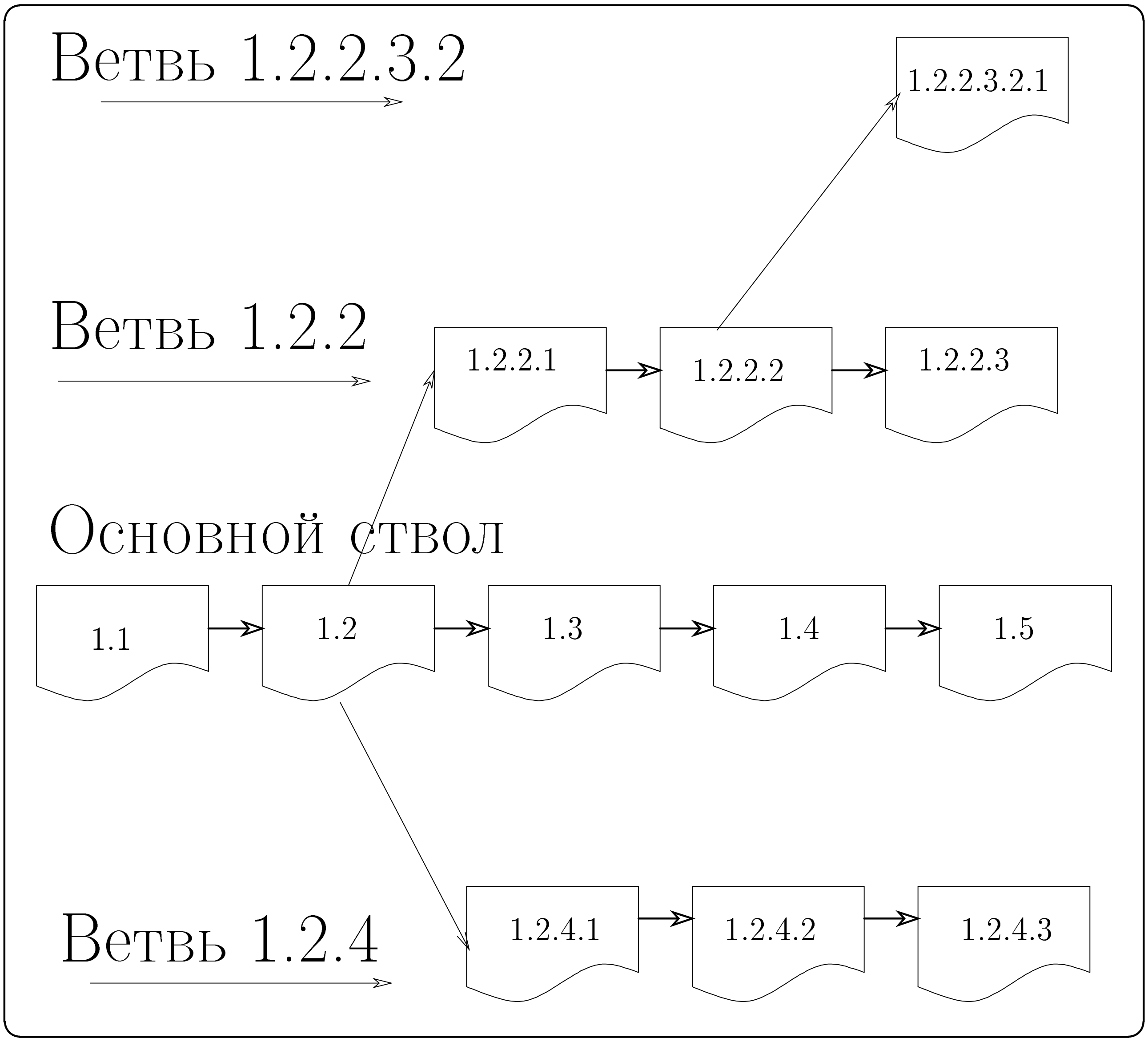

Рис. 14.7: Возможная структура ветвей

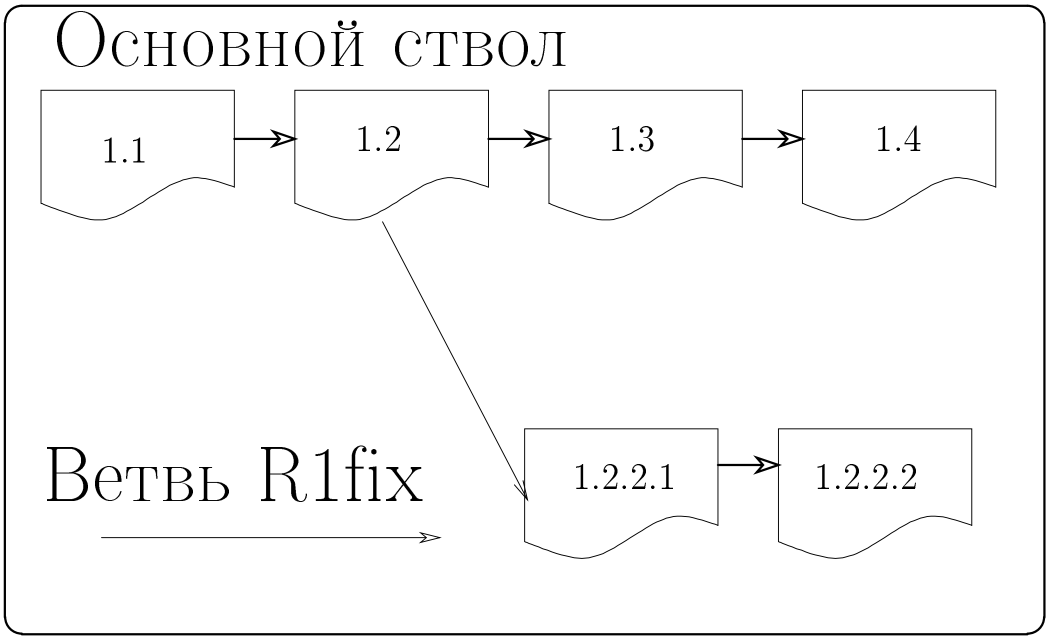

Рис. 14.8: Дерево версий

Вы можете использовать команду **admin**, чтобы переприсвоить символическое имя ветви, как его ожидает увидеть система **RCS**. Если имя **R4patches** присвоено ветви 1.4.2 (магический номер ветви 1.4.0.2) в файле **numbers.c**, то вы можете сделать:

```
cvs admin -NR4patches:1.4.2 numbers.c
```

Это будет работать, если только хотя бы одна версия была уже утверждена (**committed**) в ветви. Будьте внимательны, чтобы не назначить тег неверному номеру, поскольку на другой день не будет возможности увидеть чему назначен тег.

14.7.6 Слияние ветвей

Вы можете объединить изменения в отдельной ветви и изменения в вашем рабочем каталоге используя параметр **-j** *branch* в команде **update**. С одним параметром **-j** *branch* система **CVS** объединяет все изменения, сделанные в ветви от момента её создания и содержимое рабочего каталога. Очевидно, что **-j** означает объединение (**join**).

Рассмотрим дерево версий на рисунке 14.8.

Ветви 1.2.2 присвоено символическое имя (тег) **R1fix**. Следующий пример предполагает, что модуль **mod** содержит один файл **m.c**.

```
$ cvs checkout mod                 # Поместить в рабочий каталог
                                   # последнюю версию, 1.4
```

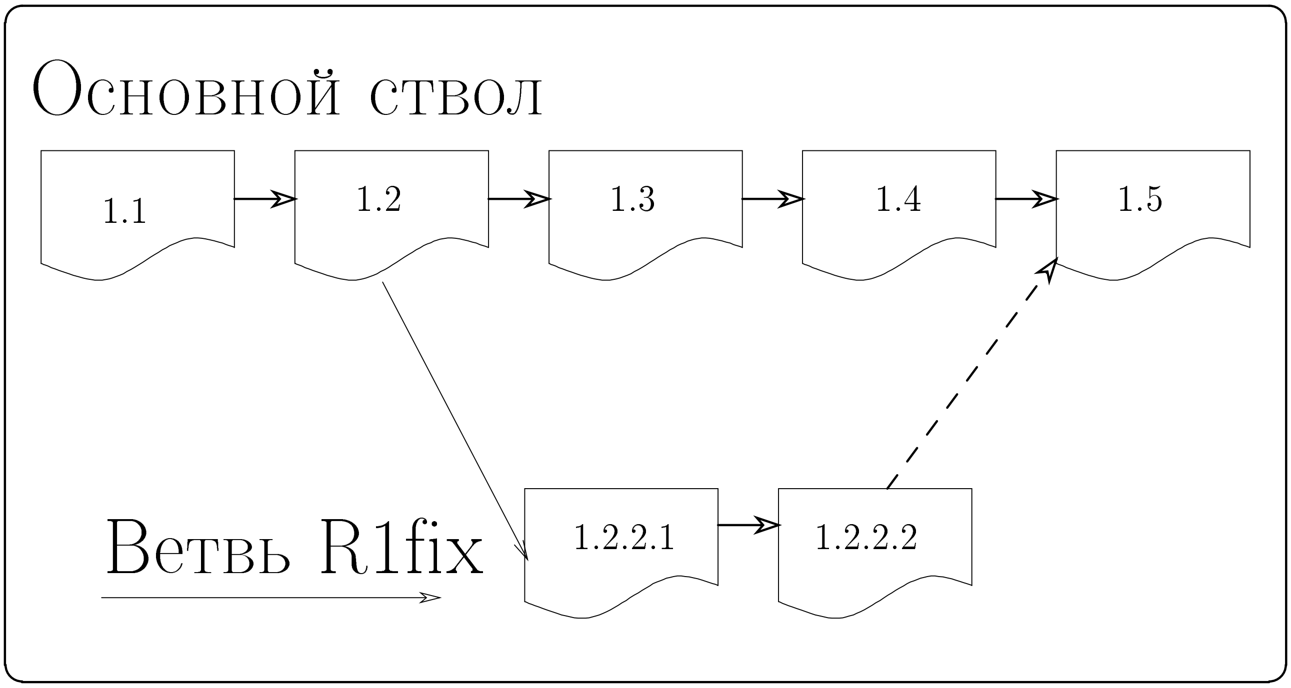

Рис. 14.9: Дерево версий (продолжение примера)

```
$ cvs update -j R1fix m.c        # Объединить все изменения сделанные
                                 # в ветви, т.е. изменения между версиям
                                 # 1.2 и 1.2.2.2, с изменениями, которые
                                 # вы сделали в рабочем каталоге

$ cvs commit -m "Included R1fix" # Создать версию 1.5.
```

Во время объединения разных версий могут возникнуть конфликты. Если такое случится, то конфликты должны быть разрешены до утверждения (**commit**) изменений.

14.7.7 Слияние из ветвей несколько раз

Продолжая наш пример, дерево версий теперь выглядит так, как показано на рисунке 14.9.

На рисунке 14.9 штрихпунктиром обозначены линии слияния ветви **R1fix** с основным стволом, как мы обсуждали в прошлом разделе.

Теперь, предположим, разработка ветви **R1fix** продолжается, как показано на рисунке 14.10.

Теперь вы захотите объединить эти новые изменения и основной ствол разработки. Если вы просто попытаетесь использовать команду

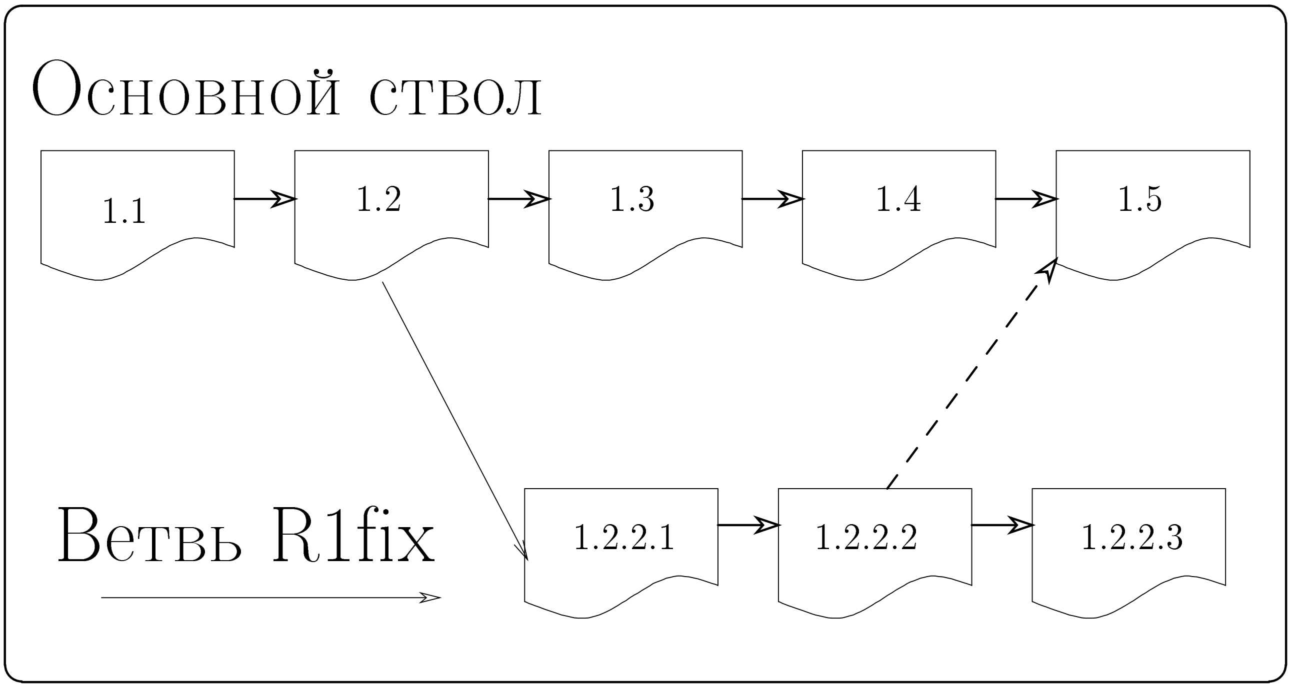

Рис. 14.10: Дерево версий (продолжение примера)

```
cvs update -j R1fix m.c
```

снова, то **CVS** вновь попробует объединить изменения, но это может также иметь нежелательные побочные эффекты, поскольку объединение уже выполнялось однажды ранее.

Чтобы избежать побочного эффекта, вам следует определить, что вы хотите объединить изменения в ветви, которая ещё не была объединена с основным стволом. Чтобы так сделать, вы определяете два параметра **-j**, тогда **CVS** объединит изменения из первой версии во вторую. Например, в данном простейшем случае могло бы быть так:

```
cvs update -j 1.2.2.2 -j R1fix m.c
```

объединить изменения в рабочем каталоге с ветвью 1.2.2.2, а затем с ветвью **R1fix** (т.е. 1.2.2.3). Проблема здесь лишь в том, что версию 1.2.2.2 вам следует указать вручную. Чуть лучшее решение могло бы быть применено с использованием даты, когда имело место последнее объединение:

```
cvs update -j R1fix:yesterday -j R1fix m.c
```

Еще чуть лучше, помечать тегом ветвь **R1fix** каждый раз, когда производится объединение в основном стволе, и позднее использовать этот тег в последующих объединениях:

```
cvs update -j merged\_from\_R1fix\_to\_trunk -j R1fix m.c
```

14.7.8 Объединение различий между двумя любыми версиями

С двумя параметрами **-j REVISION** команды **update** (и **checkout**) могут объединить отличия между любыми двумя версиями, записав результат в ваш рабочий файл. Так

```
cvs update -j 1.5 -j 1.3 backend.c
```

УДАЛИТ все изменения, сделанные между версиями 1.3 и 1.5. Обратите внимание на порядок версий в командной строке!

Если вы попытаетесь использовать эту возможность сразу на множестве файлов, то помните, что числовые значения версий могут весьма отличаться для различных файлов, которые вместе составляют модуль в смысле системы **CVS**. Почти всегда вы используете символические теги, а не номера версий, когда имеете дело сразу со многими файлами.

14.7.9 Объединение может добавить или удалить файлы

Если изменения, которые вы объединяете, влючают удаление одних файлов или добавление других, то команда **update -j** будет отражать такие изменения. Например

```
cvs update -A
touch a b c
cvs add a b c ; cvs ci -m "added" a b c
cvs tag -b branchtag
cvs update -r branchtag
touch d ; cvs add d
rm a ; cvs rm a
cvs ci -m "added d, removed a"
cvs update -A
cvs update -jbranchtag
```

После выполнения этих команд и после выполнения подтверждения (команда **commit**) файл **a** будет удалён, а файл **d** будет добавлен к основной ветви.

14.7.10 Каталог CVS в хранилище

Подкаталог с именем CVS в каждом каталоге хранилища содержит информацию об атрибутах файла. В основном, вся эта информация нужна для внутренней кухни системы CVS.

14.7.11 Форма хранения данных в рабочем каталоге

Раз мы обсуждаем некоторые детали работы хранилища CVS, то полезно кратко описать, что помещает система **CVS** в ваш рабочий каталог в подкаталог с именем `CVS`. В некоторых случаях может оказаться полезным заглянуть внутрь подкаталога `CVS` в вашем рабочем каталоге. Напомним, что такой подкаталог появится, когда вы выполните команду

cvs checkout ProJ

Здесь **ProJ** есть имя каталога в хранилище **CVS**. По этой команде в вашем рабочем каталоге будет создан подкаталог с именем **ProJ**, в который будет скопировано содержимое каталога **ProJ** из хранилища CVS. Когда вы перейдёте в каталог **ProJ**, то увидите подкаталог с именем **CVS**, который используется системой CVS.

Каталог **CVS** содержит несколько файлов, каждый из которых просто текстовый файл. Рассмотрим их чуть детальнее.

Таблица 14.1: **Файлы каталога CVS**

Имя файла	**Назначение**
`Root`	Этот файл содержит имя текущего каталога, в котором располагается хранилище CVS, то же что в переменной `$CVSROOT`.
`Repository`	Этот файл содержит имя каталога в хранилище **CVS**, которое соответствует данному рабочему каталогу. Имя может быть как абсолютным, так и относительным по отношения к содержимому файла `Root`.
`Entries`	Этот файл содержит списки файлов и каталогов в рабочем каталоге.
`Entries.Log`	Этот файл содержит практически то же самое, что и файл `Entries`. Он используется, чтобы обновление файла `Entries` не приводило к потере информации, даже если обновляющая программа завершилась аварийно.
`Entries.Backup`	Временный файл.
`Entries.Static`	Служебный файл **CVS**. Для **CVS** важно лишь существует этот файл или нет.

Продолжение таблицы на следующей странице

Файлы каталога CVS (продолжение таблицы 14.1)	
`Tag`	Этото файл содержит теги (описатели) или даты для подкаталогов.
`Checkin.prog` `Update.prog`	Эти файлы хранят программы, специфицированные параметрами **-i** и **-u** в файлах модулей соответственно.
`Notify`	Этот файл хранит сообщения, поке ещё не посланные на сервер, например, `edit` или `noedit`.
`Notify.tmp`	Вспомогательный файл для поддержания файла `Notify`.
`Base`	Если используются наблюдатели (watches), тогда команда **edit** запоминает исходную копию файла в каталоге `Base`. Это позволяет выполнить команду **unedit**, даже если нет возможности связаться с сервером.
`Baserev`	Этот файл перечисляет версии всех файлов в каталоге `Base`.
`Baserev.tmp`	Временный файл, служит для поддержания `Base`.
`Template`	Этот файл содержит заготовку, определённую посредством файла `rcsinfo`. Файл используется только на клиентской стороне.

14.8 Конфигурационный файл `modules`

Файл `modules` хранит ваши определения символических имён (модулей) для группы исходных текстов, находящихся в хранилище. Система **CVS** будет также использовать эти определения, если вы вызовете **CVS** для обновления самого файла **modules**.

Файл `modules` может содержать пустые строки и комментарии (строки начинающиеся знаком # – решётка) и строки с определением файлов, которые входят в объект (модуль). Длинные определения можно продолжать на другой строке, указав обратный слеш в конце той строки, которую хотите продолжить.

Имеется три типа объектов, которые могут быть описаны в файле `modules`:

- ALIAS MODULES – алиасные объекты **modules**;
- REGULAR MODULES – регулярные объекты **modules**;
- AMPERSAND MODULES – амперсандные объекты **modules**.

Различие между ними состоит в способе отображения файлов хранилища в файлы рабочего каталога. Удобнее всего рассмотреть это на примерах. Во всех нижеследующих примерах мы полагаем, что хранилище содержит каталог `first-dir`, который содержит два файла: `file1` и `file2`, а также подкаталог `sdir`. Каталог `first-dir/sdir` содержит файл `sfile`.

14.8.1 Простейший вид объекта: алиасный модуль

Алиасный объект - это простейший вид объекта в хранилище. Общий вид этого описания (строки) приведён ниже.

MNAME -a *aliases ...*]

Это простейший способ определения модуля с именем MNAME. Параметр **-a** означает, что **CVS** будет воспринимать MNAME, если встретит в качестве аргумента команды, как список имён файлов *aliases ...*. Список *aliases ...* может содержать другие имена модулей или пути.

Когда вы используете имя пути в качестве элемента *aliases ...*, то команда **checkout** создаёт промежуточный каталог в вашем рабочем каталоге, как будто вы использовали имя пути явно в командной строке **checkout**.

Например, если файл `modules` содержит строку:

```
amodule -a first-dir
```

тогда следующие две команды просто эквивалентны по своим результатам:

```
cvs checkout amodule
cvs checkout first-dir
```

и они произведут вывод похожий на тот, что приведён ниже:

```
cvs checkout: Updating first-dir
U first-dir/file1
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
```

14.8.2 Регулярные модули

Общий вид строки файла `modules` описывающей регулярные модули представлен ниже

MNAME **[options]** *dir* [*files ...*]

В простейшем случае эта форма записи редуцируется до

MNAME *dir*

Эта строка определяет все файлы в каталоге *dir* как модуль с именем MNAME. Имя *dir* является относительным к $CVSROOT. Во время выполнения команды **checkout** одиночный каталог с именем MNAME будет создан как рабочий каталог. Не будет использовано никаких промежуточных каталогов, даже если *dir* включает несколько промежуточных каталожных уровней.

Например, если модуль определён как

`regmodule first-dir`

то `regmodule` будет содержать файлы из каталога `first-dir`. Тогда выполнение показанной ниже команды даст следующее

```
cvs checkout regmodule
cvs checkout: Updating regmodule
U regmodule/file1
U regmodule/file2
cvs checkout: Updating regmodule/sdir
U regmodule/sdir/sfile
```

Явно описывая файлы в определении модуля после *dir*, вы можете выбрать отдельные файлы из каталога *dir*. Например,

`regfiles first-dir/sdir sfile`

С этим определением получение модуля с именем `regfiles` создаст одиночный рабочий каталог с именем `regfiles`, содержащий перечисленные файлы, которые будут взяты из нижележащего каталога хранилища **CVS**.

```
cvs checkout regfiles
U regfiles/sfile
```

14.8.3 Амперсандные модули

Модульное определение может ссылаться на другие модули включением строки `&module` в определение:

MNAME **[options]** *&module ...*

Тогда получение модуля создаёт подкаталог для каждого такого модуля в каталоге, содержащем модуль. Например, если модули содержат

```
ampermod &first-dir
```

тогда команда **checkout** создаст каталог с именем `ampermod`, который содержит подкаталог с именем `first-dir`, который, в свою очередь, содержит все каталоги и файлы живущие там. Например, команда

```
$ cvs checkout ampermod
```

создаст следующие файлы:

```
ampermod/first-dir/file1
ampermod/first-dir/file2
ampermod/first-dir/sdir/sfile
```

Имеется небольшая огрех в диагностике: сообщения, которые выводит **CVS** не содержат слова **ampermod**, т.е. положение создаваемых файлов указывается не совсем корректно.

```
$ cvs co ampermod
cvs checkout: Updating first-dir
U first-dir/file1
U first-dir/file2
cvs checkout: Updating first-dir/sdir
U first-dir/sdir/sfile
$
```

В будущем это, видимо, будет изменено.

14.8.4 Исключение каталогов

Алиасный модуль может исключать отдельные каталоги из других модулей посредством указания восклицательного знака перед именем каждого исключаемого каталога.

Например, если файл `modules` содержит

```
exmodule -a !first-dir/sdir first-dir ,
```

тогда создание рабочих копий по команде **checkout** для модуля `exmodule` будет создавать рабочие копии всех файлов и каталогов из каталога `first-dir` исключая файлы из каталога `first-dir/sdir`.

14.8.5 Модульные параметры

Регулярные модули и амперсандные модули могут содержать параметры, которые обеспечивают дополнительную информацию, касающуюся модуля.

Таблица 14.2: **Параметры модулей**

Параметр	**Назначение**
-d *name*	Дать рабочему каталог имя *name*, а не использовать для этого имя модуля.
-e *prog*	Определить программу *prog*, которая будет вызвана с параметром имя модуля в момент экспорта модуля.
-i *prog*	Определить программу *prog*, которая будет вызвана в момент выполнения команды **commit**. Единственным параметром будет абсолютное имя каталога в хранилище. Файлы `commitinfo`, `loginfo`, `verifymsg` обеспечивают другие способы для вызова какой-то программы в момент выполнения **commit**.
-o *prog*	Определить программу *prog*, которая будет вызвана в момент выполнения команды **checkout**. Единственный параметр – имя модуля.
-s *status*	Присвоить статус модулю. Когда модульный файл печатается по команде **cvs checkout -s**, модули отсортированы в соответствии с основными состояниями модуля и в соответствии с именами модулей. Этот параметр не имеет никакого другого значения. Можно использовать для того, чтобы перечислить лиц, ответственных за модуль.
-t *prog*	Определить программу *prog*, которая будет вызвана в момент выполнения команды **rtag**. Программа *prog* выполняется с двумя аргументами: имя модуля и символическое имя, ТЕГ (TAG), которое определено командой **rtag**. Программа *prog* не вызывается, когда исполняется команда **tag**. Во многих случаях файл `taginfo` даёт лучшее решение.
-u *prog*	Определить программу *prog*, которая будет вызвана в момент выполнения команды **cvs update**, начиная с верхнего каталога модуля, для которого выполняется **checkout**. Программа *prog* выполняется с единственным аргументом – абсолютное имя для модуля в хранилище.

Продолжение таблицы на следующей странице

Параметры модулей (продолжение таблицы 14.2)

14.9 Файл установки фильтров (`cvswrappers`)

Фильтры позволяют вам сделать автоматически специальное преобразование файлов при внесении их в хранилище **CVS**, а также при копировании файлов из хранилища.

Файл с именем `cvswrappers` определяет скрипт, который будет выполняться над файлом, когда его имя удовлетворяет регулярному выражению. Более точно, существует два скрипта. Один – тот, который выполняется над файлом/каталогом перед тем, как файл помещается в хранилище (параметр **-t**), а другой – тот, который вызывается, когда файл копируется из хранилища (параметр **-f**). Параметры **-t** и -f не действуют, если используется **CVS** типа клиент/сервер.

Скрипт `cvswrappers` имеет параметр **-m**, чтобы определить методологию слияния файлов, если только они не двоичные, когда файлы обновляются. Метод *MERGE* означает обычное поведение **C**VS, т.е. будет предпринята попытка слияния. Метод *COPY* означает, что **cvs update** не будет пытаться слить файлы, как она не делает СЛИЯНИЯ для двоичных файлов.

Предупреждение: не стоит пробовать этот параметр для версий **CVS** ранее, чем 1.9.

Параметр **-m** в файле `cvswrappers` воздействует на поведение **CVS** лишь когда производится слияние по команде **update**. Нет никакой разницы в каком виде файлы хранятся. Базовый формат файла `cvswrappers` следующий:

wildcard [option value][option value]... ,

где параметрами могут быть:

-f фильтрация при выдаче из **CVS**;

-t фильтрация при записи в **CVS**;

-m метод обновления (update);

-k подстановка ключевых слов,

а значениями параметров являются строки, ограниченные одиночными апострофами, например:

***.nib -f ’unwrap %s’ -t ’wrap %s %s’ -m ’COPY’**

***.c -t ’indent %s %s’**

WILDCARD означает в примере шаблон, описывающий имена файлов. Вышеприведённый пример файла `cvswrappers` устанавливает, что все файлы и каталоги в хранилище, имена которых заканчиваются на .nib, должны фильтроваться программой `wrap` непосредственно перед помещением в хранилище. Такие файлы должны быть подвергнуты фильтрации программой `unwrap` после выдачи из хранилища. В примере устанавливается также, что будет использован метод *COPY* во время выполнения команды **update**, т.е. не будут предприняты попытки слияния.

В последней строке примера устанавливается, что для файлов, имена которых оканчиваются на `.c` будет предпринята фильтрация программой `indent` перед помещением в хранилище.

Фильтр **-t** вызывается с двумя аргументами: первый – имя файла/каталога, который должен фильтроваться, второй – путь, куда должен быть помещён результат фильтрации.

Фильтр **-f** вызывается с одним аргументом, именем файла/каталога, который подлежит фильтрации. Конечный результат фильтрации будет находиться в пользовательском каталоге.

Заметим, что параметры **-t** и **-f** не вполне эффективно справляются с частью операций **CVS**: установление факта модификации файлов. Например, **CVS** может ошибочно воспринимать каталог немодифицированным, даже если какой-то файл внутри модифицирован, но вы сможете заставить **CVS** выполнить операцию **commit**, использовав параметр **-f** в команде **cvs commit**.

Другой пример. Следующая команда импортирует каталог, обрабатывая файлы, имена которых заканчиваются на `.exe` как двоичные:

```
cvs import -I ! -W "*.exe -k 'b'" first-dir vendortag reltag
```

14.10 Конфигурационные файлы для поддержки команды commit

Параметр **-i** в файле `modules` может быть использован, чтобы запустить определённую программу в момент обновления хранилища по команде **commit**. Конфигурационные файлы, рассматриваемые в данном разделе, обеспечивают более гибкие способы для запуска программ, когда что-то обновляется в хранилище по команде **commit**.

Имеется три вида программ, которые могут быть запущены по команде **commit**. Эти программы определяются в конфигурационных файлах в хранилище в каталоге `$CVSROOT/CVSROOT` и описываются ниже.

Нижеследующая таблица описывает имена конфигурационных файлов и назначение соответствующих программ.

`commitinfo` - в этом файле может быть указана программа, которая ответственна за проверку того, что команда **commit** разрешена. Если программа завершится с ненулевым кодом завершения, то операция **commit** будет отвергнута.

`verifymsg` - здесь может быть указана специальная программа, которая используется, чтобы обработать сообщение для протокола и, возможно, проверить, что сообщение содержит необходимые поля. Это наиболее интересно в комбинации с файлом `rcsinfo`, который может содержать шаблон для сообщения в протокол.

`editinfo` - в файле может быть определена программа, которая вызывается, чтобы отредактировать сообщение для протокола.

`loginfo` - может быть указана специальная программа, которая вызывается, когда **commit** завершена. Она воспринимает сообщение для протокола и некоторую дополнительную информацию, что позволяет записать все это в файл, послать сообщение конкретным лицам или сделать что угодно.

Конфигурационные файлы `commitinfo`, `loginfo`, `rcsinfo`, `verifymsg`, а также другие имеют общий формат. Каждая строка содержит следующее:

- Регулярное выражение. Синтаксис регулярного выражения тот же, что используется в **GNU** Emacs.
- Разделитель полей – один или более пробелов и/или знаков табуляции (TAB).
- Шаблон имени файла или команды.

Пустые строки игнорируются. Строки начинающиеся с символа # (решётка) рассматриваются как комментарий. Длинные строки НЕ могут быть разбиты на две строки.

Используется первая часть регулярного выражения, которая соответствует имени текущего каталога в хранилище. Остаток строки используется как имя файла или как командная строка соответственно.

14.10.1 Файл `commitinfo`

Файл `commitinfo` определяет программы, которые будут вызваны, когда система **CVS** готовится выполнить операцию **commit**. Эти программы используются для проверки, что добавляемые, удаляемые или модифицируемые файлы действительно готовы для планируемых операций. Такая проверка могла бы заключаться в том, что внесённые изменения соответствуют стандартам, принятым в вашей организации.

Каждая строка файла `commitinfo` состоит из регулярного выражения и шаблона команды. Шаблон состоит из имени программы (скрипта) и любого количества параметров. Полный путь к текущему хранилищу добавляется к шаблону, за которым следуют имена любых файлов, над которыми выполняется операцию **commit**.

Будет использована первая строка файла `commitinfo`, в которой встретилось регулярное выражение, соответствующее имени каталога в хранилище. Если команда возвращает ненулевой код завершения, то операция прекращается.

Если имя хранилища не соответствует никакой строке файла `commitinfo`, то будет выполнена команда из строки DEFAULT, если таковая есть.

Все появления имени ALL в регулярных выражениях в файле `commitinfo` будут использованы после соответствующего регулярного выражения или после DEFAULT.

Замечание. Когда **CVS** использует удалённое хранилище, то будет использован файл `commitinfo` на серверной стороне, а не на стороне клиента.

14.10.2 Файл `verifymsg`

Раз вы ввели сообщение для протокола, то вы можете обработать данное сообщение, чтобы выделить специфическое содержание, например, номер ошибки. Используйте файл `verifymsg`, чтобы определить имя программы (скрипта), которая будет вызываться для анализа сообщения.

Файл `verifymsg` наиболее полезен вместе с файлом `rcsinfo`, который может задавать шаблон (формат) сообщения.

Каждая строка в файле `verifymsg` состоит из регулярного выражения и шаблона команды. Шаблон должен включать имя программы (скрипта) и может включать любое число аргументов. Полный путь к текущему файлу, содержащему шаблон сообщения, добавляется к шаблону.

Следует отметить, что ключевое слово ALL не поддерживается.

Если найдено два регулярных выражения соответствующих каталогу, то используется первое из найденных.

Если имя хранилища не соответствует никакой строке файла `verifymsg`, то будет выполнена команда из строки DEFAULT, если таковая есть.

Если код завершения ненулевой, то выполнение команды **commit** прекращается.

Заметим, что проверочная программа (скрипт) не может изменить сообщение для протокола, а может лишь принять или отвергнуть его. Ниже следует пример использования файла `verifymsg` совместно с файлом `rcsinfo`, а также с шаблоном сообщения и проверочным скриптом.

Мы начнём с шаблона сообщения. Здесь мы хотели бы установить стандарт сообщения, чтобы в первой строке был номер ошибки (BugID). Остаток сообщения может содержать произвольный текст.

Предположим, шаблон сообщения находится в файле `/usr/cvssupport/welcome.template`.

```
cat /usr/cvssupport/welcome.template
BugId:
```

При этом скрипт `/usr/cvssupport/bugid.verify` используется для обработки сообщения:

```
#!/bin/sh
#
#       Имя файла bugid.verify
#
#  Проверить, что сообщение для протокола содержит номер
#  ошибки в правильном формате в первой строке сообщения.
#
if head -1 < $1 | grep '^BugId:[ ]*[0-9][0-9]*$' > /dev/null;
  then
     exit 0
  else
     echo "No BugId found."
     exit 1
fi
```

Файл `verifymsg` содержит строку:

```
^welcome     /usr/cvssupport/bugid.verify
```

Файл `rcsinfo` содержит строку:

```
^welcome     /usr/cvssupport/welcome.template
```

14.10.3 Файл `loginfo`

Файл `loginfo` используется для определения, куда должна посылаться протокольная информация о **cvs commit**. Первая часть строки является регулярным выражением, сличаемое с именем каталога, в котором производятся изменения (относительно $CVSROOT). Если соответствие найдено, тогда остаток строки есть программа (скрипт) типа фильтр, которая должна читать протокольное сообщение на устройстве стандартного ввода.

Если имя хранилища не соответствует никакой строке файла `loginfo`, то будет выполнена команда из строки DEFAULT, если таковая есть.

Все появления имени ALL появляющиеся как регулярные выражения используются в дополнение к ранее найденным соответствиям или к DEFAULT.

Принимается во внимание первое найденное регулярное выражение.

Пользователь может определить форматную строку как часть фильтра. Строка состоит из знака % (процент), за которым может следовать:

- пробел;
- одиночный символ формата;
- или группа символов формата, которая окружена фигурными скобками.

Символами формата могут быть:

s имя файла;

V старый номер версии (до внесения в хранилища, до **commit**);

v новый номер версии (после внесения в хранилище, после **commit**).

Все другие символы, которые появляются в форматной строке, замещаются пустой строкой.

Например, некоторые правильные форматные строки:

% , %s , %s , %{sVv} .

Выводом будут строки токенов, разделённые пробелами. Первым токеном является имя хранилища. Остаток токенов будет разделённый запятыми список компонентов, запрошенный форматной строкой. Например, если `/u/src/master` есть хранилище, а `%{sVv}` есть форматная строка и два файла (Makefile и ChangeLog) были изменены, то вывод мог бы быть таким:

```
/u/src/master ChangeLog,1.1,1.2 Makefile,1.3,1.4
```

Если в вышеприведённом примере форматная строка будет иметь вид `%{}`, то на выводе будет лишь имя хранилища.

Замечание. Когда **CVS** обращается к удалённому хранилище, то принимается во внимание файл **loginfo** на удалённом сервере, на котором располагается хранилище, а не на клиентской стороне.

Рассмотрим пример. Приведённый ниже файл `loginfo` вместе с маленьким скриптом добавляет все сообщения в протокол `$CVSROOT/CVSROOT/commitlog` к файлу `/usr/adm/cvsroot-log`. Любые изменения в программе `prog1` сообщаются электронной почтой пользователю shevel.

Содержимое файла `loginfo`:

```
ALL   /usr/local/bin/cvs-log $CVSROOT/CVSROOT/commitlog $USER
^CVSROOT  /usr/local/bin/cvs-log /usr/adm/cvsroot-log
^prog1    Mail -s %s shevel
```

Содержание скрипта /**usr**/**local**/**bin**/**cvs-log**:

```
#!/bin/sh
(echo "-----------------------------------------------------";
 echo -n $2"  ";
 date;
 echo;
 cat) >> $1
```

14.10.4 Поддержка свежей рабочей копии

Часто оказывается очень желательным поддерживать дерево рабочих каталогов так, чтобы оно содержало наиболее свежие версии файлов в хранилище. Например, часть разработчиков могла бы пожелать использовать наиболее свежие исходники без необходимости постоянно следить за этим самому. Другой пример. Допустим вы обслуживаете WWW сервер и содержите страницы в хранилище **CVS**. Тогда было бы весьма удобно, чтобы любое изменение в хранилище сразу становилось доступно вашему WWW серверу.

Хороший способ выполнять всё то, о чём мы говорили, – это использовать в файле `loginfo` команду **cvs update**. Использовать команду **cvs update** прямо как записано будет затруднительно из-за того, что **CVS** блокирует доступ к хранилищу на время изменения файлов хранилища. Поэтому команда **cvs update** должна выполняться в фоновом режиме. Ниже следует пример, в котором всё должно быть записано в одной строке:

```
^cyclic-pages   (date; cat; (sleep 2; cd /www/docs/CSD;
cvs -q update -d) &) >> $CVSROOT/CVSROOT/updatelog 2>&1
```

Эта последовательность выполнится, когда будут изменяться файлы, имена которых начинаются с **cyclic-pages**, т.е. будет обновлено дерево `/www/docs/CSD` (выполнена команда **update**).

14.10.5 Файл `rcsinfo`

Файл `rcsinfo` может быть использован, чтобы определить форму для редактирования, когда формируется сообщение, которое будет записано в протокол операции **commit**. Файл `rcsinfo` строится в соответствии с синтаксисом, похожим на тот, что используется в файлах `verifymsg`, `commitinfo` и `loginfo`. Однако, в противоположность другим файлам, вторая часть строки после регулярного выражения является не командой, а абсолютным именем файла, в котором хранится шаблон сообщения.

Если имена в хранилище не соответствуют никакому регулярному выражению в этом файле, то используется строка с именем DEFAULT, если она имеется.

Все появления имени ALL как регулярного выражения используются в дополнение к другим строкам.

Шаблон сообщения будет использоваться как сообщение по умолчанию. Если вы укажете **cvs commit -m** *message* или **cvs commit -f** *file*, то сообщения, которые вы указали в команде, заменят ваш шаблон.

Когда **CVS** получает доступ к удалённому хранилищу, содержание файла `rcsinfo` во время первоначального формирования рабочего каталога (выполняется **checkout**) будет определять шаблон, который там не изменится. Если вы редактируете файл `rcsinfo` или его шаблоны, то вам возможно понадобится выполнить команду **checkout** в новом рабочем каталоге.

14.11 Протоколирование операций CVS

Вы можете определить виды протоколирования и описать дополнительные действия, которые должны будут выполняться при различных командах **CVS**. Такие действия выполняются путём интерпретации скриптов в то или иное время. Скрипты могут добавлять какую-то стандартную информацию к протоколу, например, имя программиста, другую полезную информацию, посылать уведомления некоторому кругу лиц, а также производить другие полезные действия. Для того, чтобы протоколировать команду **commit** можно использовать файл **loginfo** и другие (смотрите раздел 14.10). Чтобы протоколировать выполнение команд **commit**, **checkout**, **export**, **tag**, необходимо использовать параметры **-i**, **-o**, **-e**, **-t** в файле **modules** соответственно. Более гибким способом уведомления нескольких лиц об изменениях в хранилище является вызов команды **cvs watch add**. Эта команда полезна, даже если вы не используете команду **cvs watch on**.

Файл **taginfo** определяет программу, которая вызывается, когда кто-то выполняет команду **cvs tag** или **cvs rtag**. Файл **taginfo** имеет стандартную форму для административных файлов такого вида, где каждая строка представляет собой регулярное выражение и команду, которая выполнится, если регулярное выражение сработало. Аргументы передаются команде в следующем порядке:

TAGNAME, **OPERATION**, **REPOSITORY** и все остальное передаётся парами: **FILENAME REVISION**. Ненулевой код завершения программы прекращает выполнение соответствующей команды, т.е. тег не будет установлен. Поле **OPERATION** может иметь следующие значения:

add для команды **tag**;

mov для команды **tag -F**;

del для команды **tag -d**.

Ниже приведён пример файла **taginfo**
ALL /usr/local/cvsroot/CVSROOT/loggit
где файл `/usr/local/cvsroot/CVSROOT/loggit` содержит следующее:

```
#!/bin/sh
echo "$@" >>/home/kingdon/cvsroot/CVSROOT/taglog
```

14.12 Кто редактирует файл

Для многих групп разработчиков использование **CVS** вместе с умолчаниями вполне приемлемо. Рад других групп предпочитают знать кто и какие файлы редактирует. Здесь описываются средства **CVS**, которые позволяют разработчикам автоматически информировать друг друга о предпринимаемых действиях, что позволяет редактировать и отлаживать один и тот же файл двум и более лицам.

Максимальная польза для разработчиков состоит в использовании команд **cvs edit**, чтобы сделать файл доступным по чтению/записи на время его модификации, и cvs release, чтобы удалить рабочий каталог, который более не используется. Однако, **CVS** не имеет средств, чтобы заставить всех поступать именно таким образом.

14.12.1 Включение/выключение режима слежения

Чтобы включить режим слежения за определёнными файлами используется команда:

cvs watch on [‘-lR’] FILES ...

Эта команда определяет, что разработчики должны использовать команду **cvs edit ДО** начала редактирования файлов **FILES**. При этом **CVS** будет создавать рабочую копию файлов **FILES** с правами доступа ТОЛЬКО ЧТЕНИЕ, чтобы напомнить разработчикам, что **ДО** редактирования следует использовать команду **cvs edit**.

Если **FILES** содержит имя каталога, **CVS** будет использовать режим слежения для всех файлов этого каталога в хранилище. Режим слежения распространяется также на все вновь добавленные файлы. Содержимое каталогов обрабатывается рекурсивно, если явно не указано обратное параметром **-l**. Параметр **-R** снова включает рекурсивный режим обработки каталогов, если он был выключен, например, в конфигурационном файле **.cvsrc** в домашнем каталоге пользователя.

Если **FILES** опущено, то подразумевается текущий каталог.

Выключение режима слежения производится командой:

cvs watch off [‘-lR’] FILES ...

Не производить никаких уведомлений о работе с файлами **FILES**, а рабочие копии создавать с правами доступа ЧТЕНИЕ/ЗАПИСЬ.

Значения **FILES** и других параметров имеют тот же смысл, что и в команде **cvs watch on**.

14.12.2 Способы уведомления о действиях CVS

Вы можете сообщить **CVS**, что вы хотели бы иметь уведомление о действиях, предпринятых над конкретным файлом. Это можно реализовать без использование команды **cvs watch on** для данного файла. Хотя в общем случае, вы будете использовать команду **cvs watch on**.

Формат команды:

cvs watch add [‘-a’ ACTION] [‘-lR’] FILES ... Назначение:

добавить текущее имя пользователя к списку лиц, получающих уведомление о действиях над файлами **FILES**.

Параметр **-a** определяет действие о котором вы будете получать уведомление. Значением **ACTION** может быть следующее:

edit Другой пользователь использовал команду **cvs edit** по отношению к указанному файлу.

unedit Другой пользователь применил команду **cvs unedit** к указанному файлу.

commit Другой пользователь изменил указанный файл в хранилище (применил команду **commit**).

all Другой пользователь применил к указанному файлу одну из перечисленных выше команд.

none Ничего из вышеприведённого, т.е. не надо уведомлять. (Полезно вместе с командой **cvs edit**).

Параметр **-a** может появиться в одной команде более, чем один раз или не появиться совсем. Если параметр опущен, то подразумевается значение **all**.

Значение **FILES** и другие параметры имеют тот же смысл, что и в команде **cvs watch**.

Наконец, по команде

cvs watch remove [‘-a’ ACTION] [‘-lR’] FILES ...

удаляются запросы на уведомление, которые были созданы командой

cvs watch add .... Аргументы – те же самые. Если присутствует параметр **-a**, то удаляются только те действия, которые определены значением параметра.

Когда возникает условие для уведомления, то **CVS** вызывает файл **notify**. Этот файл редактируется также как и любые другие административные файлы. Файл следует обычным соглашениям для административных конфигурационных файлов: он состоит из строк, где каждая содержит регулярное выражение, за которым следует команда. Команда должна содержать одиночное вхождение **%s**, которое будет заменено именем пользователя, которого надо уведомить. Остаток информации касающейся уведомления будет передан команде на стандартное устройство ввода. Ниже следует пример строки файла `notify`:

```
ALL mail %s -s "CVS notification"
```

Эта строка означает, что пользователи будут уведомлены посредством электронной почты.

Заметим, что если вы сконфигурировали точно как написано, то пользователи получат сообщения на серверной машине. Для того, чтобы указать произвольный адрес (не обязательно на серверной машине) **CVS** предлагает механизм ассоциативных адресов для рассылки уведомлений. Для того, чтобы его использовать следует создать файл **users** в каталоге **CVSROOT**. Каждая строка файла **users** имеет вид:

USER:VALUE

Здесь **USER** – имя пользователя на данной машине, а **VALUE** – предпочтительный электронный адрес. Если имеется файл **users**, то **CVS** будет использовать значение **VALUE** в файле **notify** для рассылки уведомлений.

CVS не уведомляет вас о ваших собственных изменениях.

14.12.3 Как редактировать наблюдаемый файл

Поскольку рабочая копия наблюдаемого файла во время выполнения команды **checkout** создаётся с правами доступа ТОЛЬКО ЧТЕНИЕ, то вы не можете немедленно начать редактировать такой файл. Чтобы сделать наблюдаемый файл доступным для редактирования и, одновременно, информировать об этом других разработчиков, вам следует использовать команду **cvs edit**.

Команда выглядит так:

cvs edit [OPTIONS] FILES ...

Приготовиться к редактированию файлов **FILES**. Это включает смену прав доступа к файлам **FILES** с ТОЛЬКО ЧТЕНИЕ на ЧТЕНИЕ/ЗАПИСЬ и уведомляет пользователей, которые запросили уведомление о редактировании любого файла из **FILES**.

cvs edit воспринимает те же параметры, что и команда **cvs watch add**, а также организует для пользователя, выдавшего команду **cvs edit**, временное слежение за файлами **FILES**. Временное слежение за файлами **FILES** будет прекращено, после выполнения команд **cvs unedit** или **cvs commit** по отношению к файлам **FILES**. Если пользователь не хочет получать никаких уведомлений, то он должен использовать параметр **-a none**.

FILES и другие параметры имеют тот же смысл, что и в команде **cvs watch**.

Предупреждение. Если параметр **PreservePermissions** включён в хранилище, то **CVS** не будет изменять права доступа ни к каким файлам из **FILES**.

Обычно, когда вы закончили редактирование группы файлов, вы производите операцию **cvs commit**, которая вносит изменения в хранилище и возвращает права доступа наблюдаемых файлов в состояние ТОЛЬКО

ЧТЕНИЕ. Однако, если вы передумали и решили не делать никаких изменений или решили аннулировать уже сделанные изменения в рабочей копии, то вам следует воспользоваться командой **cvs unedit**. Команда выглядит так:

cvs unedit [‘-lR’] FILES ...

Она аннулирует любые изменения в рабочей копии файлов **FILES**, если таковые имели место. **CVS** устанавливает права доступа в состояние ТОЛЬКО ЧТЕНИЕ для тех файлов из **FILES**, для которых были запросы на уведомление с использованием команды **cvs watch on** со стороны других пользователей. Кроме того, **CVS** уведомляет пользователей, которые ранее запрашивали уведомление об операциях **cvs unedit** для каких-то файлов из **FILES**.

FILES и другие параметры в команде имеют тот же смысл, что и в команде **cvs watch**.

Если слежение не было включено, то команда **unedit** может не выполниться. Тогда единственный способ вернуться к версии, которая имеется в хранилище, состоит в удалении файлов, а затем в выполнении команды **cvs checkout** для этих файлов. Значение такой последовательности действий не совсем точно совпадает с командой **unedit**, поскольку будут внесены также изменения, которые возможно имели место с момента предыдущей команды **checkout** или **update**.

Когда используется схема **CVS** клиент/сервер, то вы можете использовать **cvs edit** или **cvs unedit** даже если **CVS** не смогла в данный момент успешно связаться с сервером. Уведомления будут посланы сразу после очередной успешной командой **CVS**.

14.12.4 Кто наблюдает и редактирует

Команда

cvs watchers [‘-lR’] FILES ...

перечисляет всех пользователей, которые следят за изменениями в файлах **FILES**. Отчёт будет содержать всех наблюдателей, их электронные адреса, имена файлов, за которыми они следят.

Команда

cvs editors [‘-lR’] FILES ...

перечисляет пользователей, которые работают в данный момент над файлами из **FILES**. Отчёт включает адреса каждого пользователя, время, когда пользователь начал работать над файлом, имя хоста, где пользователь работает, а также имя рабочего каталога, в котором пользователь работает с данным файлом.

FILES и другие параметры имеют тот же смысл, что и в команде **cvs watch**.

14.13 Создание файлов проекта в хранилище

Поскольку переименование файлов и каталогов является неудобной операцией, то прежде чем начать новый проект полезно установить некоторые соглашения об именах и об организации файлов в целом. Переименование не является полностью невозможным делом, однако, если изменяются имена десятков или сотен файлов и каталогов, то такая операция увеличивает вероятность внесения ошибок. Может случиться так, что ошибки в именах обнаружатся (если такое произойдёт) спустя значительное время после выполнения переименований.

Что делать для создания файлов зависит от вашего текущего состояния.

14.13.1 Существующие файлы

Когда вы решили начать использование **CVS**, вы, возможно, уже ведёте какие-то проекты и имеете группы файлов, которые вы хотели бы поместить в хранилище **CVS**. В таком случае удобнее всего использовать команду

cvs import.

Предполагается, что установлена переменная окружения **$CVSROOT**, указывающая на местоположение хранилища **CVS**. Пусть ваши файлы находятся, например, в каталоге **WorkDIR**, то вам удобнее сначала перейти в **WorkDIR**

cd WorkDIR

Предположим далее, что вам удобнее разместить ваш проект в каталоге хранилища под названием **PHENIX**. Тогда

cvs import -m "Imported sources"PHENIX MyProgs start

По этой команде содержимое каталога **WorkDIR** будет переписано в хранилище **$CVSROOT/PHENIX**. Несмотря на то, что вы указали параметр **-m** CVS вызовет редактор текста для ввода комментария. Строка **MyProgs** является специальным тегом, который должен быть здесь, а **start** – это освобождающий тег.

Далее можно проверить, что все сработало так как ожидалось.

```
cd ..
mv WorkDIR WorkDIR\_Initial
cvs checkout PHENIX
```

```
ls -R PHENIX
rm -r WorkDIR\_Initial
```

Удаление исходного каталога является неплохой идеей, поскольку это будет гарантировать, что вы случайно не отредактируете файлы из каталога **WorkDIR** в обход системы **CVS**.

Команда **checkout** системы **CVS** может иметь в качестве аргумента как имя каталога в хранилище, так и имя отдельного модуля, но любое имя является относительным к `$CVSROOT`

Если некоторые файлы, которые вы хотели бы импортировать являются двоичными, то, возможно, вы захотите использовать особенность **wrappers**, чтобы установить какие файлы являются двоичными, а какие текстовыми.

14.14 Исходные тексты из разных компаний

Если вы модифицируете программу, учитывая особенности применения в вашей организации, то вы захотите, чтобы в следующих версиях данной программы ваши изменения также были включены. Система **CVS** также может помочь вам в этом.

В терминологии, которая используется в **CVS**, поставщик программ называется ВЕНДОР (vendor). Немодифицированная программа, полученная от вендора, помещается в хранилище в отдельную ветвь, которая именуется ВЕНДОРНОЙ ВЕТВЬЮ (vendor branch). **CVS** резервирует для этой цели номер ветви 1.1.1.

Когда вы модифицируете исходники и производите операцию **commit**, то ваша версию попадёт в основной ствол (main trunk). Когда вендор приготовил новую версию программы, вы производите операцию **commit** в вендорной ветви и копируете модификации в основной ствол.

Для создания и обновления вендорной ветви используется команда **import**. Когда вы импортируете новый файл, вендорная ветвь становится ГЛАВНОЙ ВЕРСИЕЙ (head revision). Иными словами, любой, кто выполняет операцию **checkout**, получит именно это версию, если специально не указано ничего иного. Когда изменения в рабочем каталоге вносятся в хранилище операцией **commit**, то они помещаются в основной ствол, который и становится ГЛАВНОЙ ВЕРСИЕЙ (head revision).

14.14.1 Первоначальный импорт

Для первоначального помещения исходников в хранилище следует использовать команду **import**. Когда вы используете **import** для поддержки ваших изменений в поставляемых со стороны исходниках, то тег **vendor tag** и теги **release tags** очень полезны. Тег **vendor tag** является символическим именем ветви. Обычно эта ветвь имеет номер 1.1.1, если вы не использовали параметр **-b BRANCH**. Теги **release tags** являются символическими именами отдельных версий, например, SF1005.

Заметим, что команда **import** не изменяет содержимого каталога, откуда импортируются исходники. В частности, эта команда не устанавливает этот каталог в качестве рабочего каталога системы **CVS**. Если вы хотите работать с исходниками, то их следует импортировать, затем выполнить команду **checkout** в рабочий каталог.

Положим, вы имеете исходники программы с именем **wusage** в каталоге *wusage-0.1* и вы хотели бы сделать изменения в исходниках для ваших персональных нужд так, чтобы использовать эти изменения и в будущих версиях **wusage**. Вы начинаете с помещения ваших исходников вхранилище:

```
$ cd wusage-0.1
$ cvs import -m "Import of WUSAGE v. 0.1" wusage WDIST WUSAGE_0_1
```

Вендорный тег имеет имя **WDIST**, а единственный тег версии имеет имя **WUSAGE_0_1**.

14.14.2 Изменение модуля (продукта) командой import

Когда новая версия исходников получена от поставщика, вы помещаете её в хранилище той же самой командой **import**, которую вы использовали при создании хранилища. Единственное отличие состоит в различных тегах:

```
$ tar xfz wusage-0.2.tar.gz
$ cd wusage-0.2
$ cvs import -m "Import of WUSAGE v. 0.2" wusage WDIST WUSAGE_0_2
```

Для файлов, которые не были модифицированы локально, вновь создаваемая версия становится главной версией (head revision). Если вы сделали какие-то изменения в конкретном файле, то команда **import** предупредит вас, что вы должны объединить изменения в основном стволе и попросит вас выполнить **checkout -j**.

```
$ cvs checkout -jWDIST:yesterday -jWDIST wusage
```

Вышеприведённая команда создаст рабочую копию наиболее свежей версии **wusage** и объединит в рабочей копии изменения, сделанные в вендорной ветви со вчерашнего дня. Если возникнут конфликты, то они разрешаются обычным образом. После этого модифицированные файлы могут быть помещены в хранилище командой **commit**.

Использование даты, как описано выше, предполагает, что вы импортируете не более одной версии программного продукта в день. Если же вы импортируете более одной версии программного продукта в день, то вам следует использовать команду:

```
$ cvs checkout -jWUSAGE_0_1 -jWUSAGE_0_2 wusage
```

14.14.3 Возврат к версии вендора

Вы можете проигнорировать все локальные изменения и получить наиболее свежую версию, которую поставил вам вендор. Это выполняется переходом к вендорной ветви для всех файлов. Например, если вы имеете копию исходников в локальном рабочем каталоге `work.d/wusage` и хотите вернуться к наиболее свежей версии, которая была вам поставлена для всех файлов в данном каталоге, то вы должны сделать

```
$ cd work.d/wusage
$ cvs admin -bWDIFF .
```

Не должно быть пробелов между **-b** и значением в последней строке примера.

14.14.4 Как управлять подстановкой ключевых слов во время импорта

Исходники, которые вы импортируете могут содержать ключевые слова. К примеру, поставщик может использовать систему **CVS** или другую систему поддержки версий, которая использует подобные ключи. Если вы просто импортируете файлы в режиме, который используется по умолчанию, то значения ключей будут присвоены такие, которые определяются вашим вариантом системы **CVS**. Однако, может оказаться более удобным поддерживать подстановку ключевых слов, которые обеспечивает поставщик, поскольку такая подстановка обеспечивает вас информацией об источнике, откуда получены исходники.

Чтобы обеспечить подстановку ключевых слов, установленную поставщиком, следует использовать параметр **-ko** в команде **import**, когда вы импортируете файл в первый раз. Этот параметр выключит подстановку ключевых слов для всего файла. Если вы захотите более селективно выбирать режим подстановки, то вам следует подумать, как использовать параметр **-k** в командах **update** и **admin**.

14.14.5 Несколько вендорных ветвей

Все примеры до сих пор предполагали, что имеется лишь одна вендорная ветвь, из которой вы получаете исходники. В ряде случаев вы можете получить исходники из нескольких мест. Предположим, вы связаны с проектом, который ведут несколько групп разработчиков, которые модифицируют программы. Имеется несколько путей, как этим можно управлять, но один из самых эффективных - это сложить все варианты в хранилище **CVS**.

Для управления ситуацией, когда имеется более, чем один поставщик, полезно использовать параметр **-b** в команде **cvs import**. По умолчанию **-b 1.1.1**. Для простоты, предположим, что у нас две группы разработчиков: красные и голубые. Они посылают вам свои исходники. Когда вы хотите использовать исходники красных, вы используете тей **RED**. Если вы хотите использовать исходники голубых, то следует использовать тег вендора **BLUE**. Команды получения текстов могли бы быть такими:

```
$ cvs import dir RED RED_1-0
$ cvs import -b 1.1.3 dir BLUE BLUE_1-5
```

Заметим, что если вендорный тег не соответствует значению параметра **-b**, **CVS** не распознает такой случай. Например,

```
$ cvs import -b 1.1.3 dir RED RED_1-0
```

Будьте внимательны: этот вид несоответствия гарантированно приведёт к конфузиям.

14.15 Как ваша система построения программ взаимодействует с CVS

Как упоминалось во введении, **CVS** не обеспечивает средств для построения готовых программ из исходных текстов (таких как **make**). В этом разделе описывается каким образом ваша система построения программ могла бы взаимодействовать с **CVS**.

Один общий вопрос, как их система построения могла бы соответствовать наиболее свежим версиям исходников. Ответ с использованием **CVS** имеет две стороны. Во-первых, поскольку **CVS** сама по себе обеспечивает рекурсивную обработку каталогов, то нет нужды как-то специально заботиться о соответствующих изменениях файла `Makefile` (или какого-то другого конфигурационного файла, который используется в вашей системе построения программ из исходных текстов), чтобы сделать его соответствующим наиболее свежей версии исходников. Вместо этого, просто используйте две команды: **cvs -q update** и затем **make** для вызова вашей системы построения. Во-вторых, вам нет нужды иметь наиболее свежие копии от остальных разработчиков, пока не завершена ваша работа.

Предлагается следующий подход. Сначала выполнить команду **update**, чтобы обновить файлы в рабочем каталоге. Затем внести изменения, которые вы планируете добавить. Отладить и проверить их и лишь после выполнить операцию **commit**, т.е. внести ваши изменения в хранилище. Может оказаться, что вам следует снова выполнить **update**, а затем - **commit**.

Производя постоянно только что описанную последовательность с каждой вашей разработкой, вы можете быть уверены, что в ваш рабочий каталог в целом соответствует наиболее свежим версиям исходников.

Следующий вопрос, который является весьма общим для многих разработок, состоит в том, как составить список версий всех файлов, которые должны использоваться для построения конечного продукта (готовой к исполнению программной системы). Наилучший способ реализовать этот вид функциональности состоит в использовании тега, т.е. команды (**tag**), чтобы записать какие версии каких модулей будут включены в данном варианте конечного продукта.

Использование **CVS** прямо в соответствии с основным назначением поддержки множества версий является сравнительно нетрудным. В этом случае каждый разработчик будет иметь копию дерева исходников в своём рабочем каталоге. Это дерево и будет источником для построения конечной программы или системы. Если исходное дерево невелико или разработчики рассеяны географически, то это лучший способ выполнения совместных работ. Если дерево очень велико, т.е. проект велик, то стандартный способ - разбить проект на независимые части. Тогда каждая группа разработчиков будет заниматься своей частью, а не полным набором (полным деревом) исходных файлов.

Другой подход состоит в том, что каждый разработчик имеет у себя относительно небольшой фрагмент кодов, который он отлаживает или модифицирует, а остальная часть проекта содержится в централизованном хранилище. Многие люди приходят с некоторыми такими системами, используя особенности символических линков (ссылок) или особенности **VPATH**, которая имеется во многих версиях утилиты **make**. Одно из таких средств полезных при построении программ можно найти в *ftp://ftp.cs.colorado.edu/pub/distribs/odin*.

14.16 Специальные файлы

В обычных обстоятельствах **CVS** работает только с регулярными файлами. Предполагается, что каждый файл проекта является постоянным (persistent): он может быть открыт, прочитан, закрыт и т.д. **CVS** игнорирует права доступа к файлу и права собственности; эти моменты по предположению разрешаются во время установки. Таким образом, невозможно внести в хранилище устройство (файл устройства). Если файл устройства не может быть открыт, то **CVS** не станет с ним работать. Обычно файлы теряют признаки прав доступа и прав собственности после помещения в хранилище.

Если в хранилище установлена конфигурационная переменная PreservePermissions, то **CVS** сохранит некоторые характеристики файла:

- принадлежность данной группе и данному пользователю;
- права доступа;
- главный и дополнительный номера устройств;
- символические линки (ссылки);
- постоянную линковую структуру.

Если установлена переменная PreservePermissions, то это сильно влияет на поведение **CVS**. Часть операций **CVS** будет возможна только для пользователя с именем **root**.

Часть команд **CVS** не могут быть выполнены успешно, например, **cvs status**, т.к. команда не распознаёт постоянную линковую структуру.

Более серьёзные последствия могут быть когда **CVS** полагает файл с изменёнными правами доступа изменённым файлом. В этом случае команда **update** может заменить все или часть прав доступа в вашем рабочем каталоге.

Изменение постоянных линков в каталоге **CVS** – весьма деликатная операция.

Наконец, особенность PreservePermissions не работает в схеме клиент/сервер. Ну и постоянные линки между каталогами не поддерживаются тоже. Иными словами, постоянные линки должны быть в одном каталоге.

14.17 Игнорирование файлов посредством файлов `cvsignore`

Имеются несколько имён файлов, которые часто встречаются в вашем рабочем каталоге, но вы не планируете их запоминать в хранилище и хотели бы, чтобы **CVS** не обращала на них внимание. Такими файлами могут быть, например, файлы типа *.ps, *.bak, core, прочие. Список шаблонов имён файлов которые **CVS** игнорирует по умолчанию приведён ниже:

```
RCS     SCCS    CVS     CVS.adm
RCSLOG  cvslog.*
tags    TAGS
.make.state     .nse_depinfo
*~      #*      .#*     ,*      _$*     *$
*.old   *.bak   *.BAK   *.orig  *.rej   .del-*
*.a     *.olb   *.o     *.obj   *.so    *.exe
*.Z     *.elc   *.ln
core
```

Могут существовать также дополнительные списки шаблонов имён файлов, которые необходимо игнорировать.

- Список для хранилища в файле `$CVSROOT/CVSROOT/cvsignore` добавляется к вышеприведённому списку, если такие файлы существуют.

- Список для пользователя в файле `.cvsignore`, который располагается в главном домашнем каталоге, также добавляются к вышеприведённому списку, если такие файлы существуют.

- Любые шаблоны, которые хранятся в переменной окружения $CVSIGNORE, также добавляются к вышеприведённому списку, если такие файлы существуют.

- Любые шаблоны, добавленные параметром **-I**.

- Любой подкаталог в вашем рабочем каталоге может содержать файл `.cvsignore`. Таким образом, любые шаблоны из этого файла будут добавлены с ранее сформированным спискам шаблонов имён файлов. Однако, этот файл действует лишь на тот подкаталог, в котором он хранится.

В любом из перечисленных 5 мест вы можете поставить восклицательный знак, который очистит весь список шаблонов имён файлов, которые **CVS** должна игнорировать. Таким образом вы имеете возможность сохранять даже те файлы, которые **CVS** игнорирует по умолчанию.

Определяя **-I !** для команды **cvs import** означает, что вы будете импортировать всё подряд без разбора, что оказывается необходимым, если нет каких-то экстраординарных файлов. Перед выполнением такой команды полезно удалить все файлы **.cvsignore** из импортируемых каталогов во избежание недоразумений.

Заметим, что простой синтаксис для игнорирования файлов не позволяет указывать имена файлов, содержащих пробелы. Пробелы здесь являются разделителями. Также нет способов указать комментарии.

14.18 Простой пример разрешения конфликта при объединении версий

Предположим, что версия 1.4 файла **driver.c** содержит следующее:

```
#include <stdio.h>

void main()
{
    parse();
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? 0 : 1);
}
```

А версия 1.6 того же файла содержит:

```
#include <stdio.h>

int main(int argc,
         char **argv)
{
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: No args expected.\n");
        exit(1);
```

```
        }
        if (nerr == 0)
            gencode();
        else
            fprintf(stderr, "No code generated.\n");
        exit(!!nerr);
    }
```

В то же время вы работаете с рабочей копией, которая основывается на версии 1.4. Такая ситуация легко может возникнуть, если два человека независимо модернизируют один файл. Итак, ваша рабочая копия содержит следующее:

```
#include <stdlib.h>
#include <stdio.h>

void main()
{
    init_scanner();
    parse();
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
}
```

Теперь вам надо выполнить команду **cvs update**. При выполнении команды вы получаете диагностику:

```
$ cvs update driver.c
RCS file: /usr/local/cvsroot/yoyodyne/tc/driver.c,v
retrieving revision 1.4
retrieving revision 1.6
Merging differences between 1.4 and 1.6 into driver.c
rcsmerge warning: overlaps during merge
cvs update: conflicts found in driver.c
C driver.c
```

Таким образом, **CVS** информирует вас, что возникли конфликты при обновлении. Ваш исходный немодифицированный рабочий файл будет сохранён под именем `.#driver.c.1.4`. А новая версия файла `driver.c` теперь содержит следующее:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,
         char **argv)
{
    init_scanner();
    parse();
    if (argc != 1)
    {
        fprintf(stderr, "tc: No args expected.\n");
        exit(1);
    }
    if (nerr == 0)
        gencode();
    else
        fprintf(stderr, "No code generated.\n");
<<<<<<< driver.c
    exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
=======
    exit(!!nerr);
>>>>>>> 1.6
}
```

Заметим, что все НЕперекрывающиеся модификации выполнены в новом варианте рабочей копии файла **driver.c**. В то же время перекрывaщиеся части ясно показаны маркерами <<<<<<<, ======= и >>>>>>>. Вы можете разрешить конфликт простым редактированием, удалив ошибочные строки и маркеры.

Предположим, вы получили следующий файл:

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc,
         char **argv)
{
```

```
        init_scanner();
        parse();
        if (argc != 1)
        {
            fprintf(stderr, "tc: No args expected.\n");
            exit(1);
        }
        if (nerr == 0)
            gencode();
        else
            fprintf(stderr, "No code generated.\n");
        exit(nerr == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
    }
```

Теперь вы можете выполнить операцию **commit** и получите версию 1.7:

```
$ cvs commit -m "Initialize scanner. Use symbolic exit values." \
driver.c
Checking in driver.c;
/usr/local/cvsroot/yoyodyne/tc/driver.c,v  <--  driver.c
new revision: 1.7; previous revision: 1.6
done
```

CVS не будет ничего менять в хранилище (не выполнит команду **commit**), если в файле остались неразрешённые конфликты. Сейчас чтобы разрешить конфликт вам необходимо сменить дату модификации файла. Если в файле остались маркеры, то **CVS** предупредит вас об этом, но создаст новую версию в хранилище.

В компоненте **pcl-cvs** (GNU emacs интерфейс для CVS) имеется специальный пакет, который может помочь разрешать конфликты. Подробности следует смотреть в описании **pcl-cvs**, например, *http://www.loria.fr/~molli/cvs/pcl-cvs/*.

14.19 Список команд CVS

Команды **CVS** выглядят таким образом:

cvs *[global-options]* **COMMAND** [COMMAND-OPTIONS] **[***command-args***]**

Далее следует список команд системы **CVS**.

add [options [files]] Добавить в хранилище новый файл/каталог.

-k *kflag* Установить подстановку ключевых слов.

-m *msg* Установить описание файла (комментарий).

admin [options [files ...]] Администрирование протокольными файлами в хранилище.

-b[*rev***]** Установить ветвь по умолчанию.

-c*string* Установить начало комментария.

-k*subst* Определить подстановку ключевых слов.

-l[*rev***]** Ограничить доступ версии *rev* или последней версии.

-m*rev:msg* Заменить комментарий в хранилище.

-o*range* Удалить версии из хранилища.

-q Выполняться с минимумом сообщений.

-s*state[:rev]*Установить состояние файла.

-t Установить ввод описания (комментария) со стандартного ввода.

-t*file* Установить ввод описания (комментария) из файла *file*.

-t-*string* Заменить описание (комментарий) на строку *string*.

-u[*rev***]** Открыть доступ к версии *rev* или к последней версии.

annotate [options [files ...]] Показать изменённые строки из последней версии файлов.

-D *date* Показать наиболее свежую версию, но не позднее даты *date*.

-f Использовать версию из основного ствола разработки, если указанный тег или дата не найдены.

-l Выполнять локально, без подкаталогов.

-R Выполнять рекурсивно, с подкаталогами.

-r *tag* Показать изменённые строки из версии *tag*.

checkout [options MODULES...] Получить копию исходных текстов.

-A Сбросить любые липкие теги, даты, параметры.

-c Вывести базу данных с именами модулей.

-D *date* Получить копию исходных текстов до даты *date* (липкий параметр).

-d *dir* Получить копию исходных текстов в каталог *dir*.

-f Использовать версии из основного ствола, если указанные теги или даты не найдены.

-j *rev* Объединить изменения.

-k *kflag* Использовать строку *kflag* для постановки ключевых слов.

-l Выполнять локально; без подкаталогов.

-N Не сокращать путь к модулю, если используется **-d**.

-n Не выполнять программу модуля, если она есть.

-P Пропустить пустые каталоги.

-p Вывести копию исходных текстов на стандартное устройство вывода (избегая липкости).

-R Выполнять рекурсивно; включая подкаталоги (умолчание).

-r *tag* Получить копию исходных текстов версии *tag* (липкий параметр).

-s То же что **-c**, но включая состояние модуля.

commit [options [files ...]] Проверить есть ли изменения в рабочем каталоге и перенести изменения файлов из рабочего каталога (если они имели место) в хранилище.

-F *file* Прочесть комментарий на изменение из файла *file*.

-f Отметить файл проверенным; запретить рекурсию.

-l Выполнять локально; без подкаталогов.

-m *msg* Использовать строку *msg* в качестве комментария, который будет записан в журнал (log).

-n Не выполнять программу модуля, если таковая есть.

-R Выполнять рекурсивно; с подкаталогами (умолчание).

-r *rev* Выполнить **commit** для версии *rev*.

diff [options [files ...]] Показать различия между версиями файлов. В дополнение к нижеописанным параметрам имеется ряд дополнительных параметров, которые используются для описания вывода программы.

-D *date1* Получить diff между версией с определённой датой и версией в рабочем каталоге.

-D *date2* Получить diff между *date1/rev1* и *date2*.

-l Выполнять локально; без подкаталогов.

-N Включить diff для добавленных и удалённых файлов.

-R Выполнять рекурсивно; с подкаталогами (умолчание).

-r *rev1* Получить diff между версией *rev1* и рабочим каталогом.

-r *rev2* Получить diff между версией *date1/rev1* и *rev2*.

edit [options [files ...]] Приготовиться редактировать НАБЛЮДАЕМЫЙ (watched) файл.

-a *actions* Определить действия для временного наблюдения, где действия могут быть такими: EDIT, UNEDIT, COMMIT, ALL, NONE.

-l Выполнять локально; без подкаталогов.

-R Выполнять рекурсивно; с подкаталогами (умолчание).

editors [options [files ...]] Показать кто редактирует наблюдаемый файл.

-l Выполнять локально; без подкаталогов.

-R Выполнять рекурсивно; с подкаталогами (умолчание).

export [options MODULES...] Экспортировать файлы из системы **CVS**.

-D *date* Экспортировать версию, датированную *date*.

-d *dir* Экспортировать версию в каталог *dir*.

-f Использовать версии из основного ствола, если указанные теги или даты не найдены.

-k *kflag* Установить подстановку ключевых слов.

-l Выполнять локально; без подкаталогов.

-N Не сокращать путь к модулю, если используется **-d**.

-n Не выполнять программу модуля, если она есть.

-P Пропустить пустые каталоги.

-p Вывести копию исходных текстов на стандартное устройство вывода (избегая липкости).

-R Выполнять рекурсивно; включая подкаталоги (умолчание).

-r *tag* Получить копию исходных текстов версии *tag* (липкий параметр).

history [options [files ...]] Показать историю доступов к хранилищу.

-a Все пользователи.

-b *str* Назад к записи содержащей строку *str* в поле MODULE/FILE/REPOS.

-c Отчёт о модифицированных (committed) файлах.

-D *date* Начиная с даты *date*.

-e Отчёт о записях всех типов.

-l Отчёт о наиболее поздних изменениях.

-m *module* Отчёт о модуле *module*.

-n *module* В модуле *module*.

-o Отчёт о прочитанных (checkout) модулях.

-r *rev* Начиная с версии *rev*.

-T Выдать отчёт о всех тегах.

-t *tag* Начиная с момента, когда тег *tag* попал в файл истории (от любого пользователя).

-u *user* Для пользователя *user*. Может использоваться несколько раз в командной строке.

-w Рабочий каталог должен соответствовать.

-x *types* Отчёт о типах *types*, может быть одним из следующих: TOEFWUCGMAR (одна буква – один тип).

-z *zone* Вывод для временной зоны *zone*.

import [options repository vendor-tag release-tags...] Внести файлы в хранилище **CVS** используя ветвь поставщика.

-b *bra* Внести ветвь *bra* в ветвь поставщика.

-d Использовать время модификации файлов в качестве времени импорта.

-k *kflag* Установить режим подстановки ключевых слов по умолчанию.

-m *msg* Использовать *msg* как запись в журнале (log).

-I *ign* Файлы, которые следует игнорировать (! для сброса).

-W *spec* Фильтры (wrappers).

init Создать хранилище, если оно не существует.

log Напечатать историю файлов в хранилище.

-b Только список версий в ветви по умолчанию.

-d *dates* Определить даты (*D1*<*D2* для интервала и *D* для свежайшего до даты *D*).

-h Напечатать только заголовки.

-l Локально; без подкаталогов.

-N Не перечислять теги.

-R Печать только имена файлов **RCS**.

-r*revs* Печатать только версии *revs.*

-s *states* Только версии с определённым состоянием.

-t Только заголовок и описательный текст (комментарий).

-w*logins* Только список версий внесённых определёнными пользователями.

login Запрос на аутентификацию пользователя.

logout Удалить запомненный пароль для аутентифицирующего сервера.

rdiff [options MODULES...] Показать различия между версиями.

-c Контекстное различие (умолчание).

-D *date* Выбрать версии, базирующиеся на дате *date.*

-l Локально; без подкаталогов.

-R Рекурсивно; с подкаталогами (умолчание).

-r *rev* Выбрать версии, базирующиеся на версии *rev.*

-s Краткий формат - одна строка на файл.

-t Различие между двумя последними версиями.

-u Выводной формат **unidiff**.

-V *vers* Использовать **RCS** версию *vers* для подстановки ключевых слов (устаревшее).

release [options DIRECTORY] Отметить для системы **CVS**, что каталог **DIRECTORY** более не используется.

-d Удалить данный каталог.

remove [options [files ...]] Удалить вход (файл, каталог) в хранилище.

-f Вычеркнуть файл до его удаления.

-l Локально; без подкаталогов.

-R Рекурсивно; с подкаталогами (умолчание).

rtag [options tag modules ...] Добавить символический тег к модулю.

-a Очистить тег из удалённых файлов, которые иначе не смогли бы быть помечены тегом.

-b *tag* Создать ветвь с именем *tag*.

-D *date* Пометить версию с датой *date*.

-d Удалить данный тег.

-F Переместить тег, если он уже существует.

-f Использовать основной ствол, если указанные тег или дата не найдены.

-l Локально; без подкаталогов.

-n Не выполнять программу тега.

-R Рекурсивно; с подкаталогами (умолчание).

-r *tag* Пометить существующий тег *tag*.

status [options files ...] Отобразить информацию о состоянии в рабочем каталоге.

-l Локально; без подкаталогов.

-R Рекурсивно; с подкаталогами (умолчание).

-v Включить информацию о тегах в каждом файле.

tag [options tag [files ...]] Добавить символический тег к файлам, скопированным (checkouted) из хранилища в рабочий каталог.

-b Создать ветвь с именем *tag*.

-D *date* Пометить тегом версию с датой *date*.

-d Удалить данный тег.

-F Переместить тег, если он уже существует.

-f Использовать основной ствол, если указанные тег или дата не найдены.

-l Локально; без подкаталогов.

-n Невыполнять программу тега.

-R Рекурсивно; с подкаталогами (умолчание).

-r *tag* Пометить тегом существующий тег *tag*.

unedit [options [files ...]] Отменить действие команды **edit** (**undo**).

-a *actions* Определить действия для временного наблюдения, где действиями могут быть: EDIT, UNEDIT, COMMIT, ALL, NONE.

-l Локально; без подкаталогов.

-R Рекурсивно; с подкаталогами (умолчание).

update [options [files ...]] Привести рабочее дерево (в рабочем каталоге) в соответствии с хранилищем.

-A Сбросить любые липкие теги/даты/параметры.

-D *date* Получить из хранилища наиболее свежие версии файлов, но не позднее даты *date*.

-d Создать каталоги.

-f Использовать основной ствол, если указанные тег или дата не найдены.

-I *ign* Игнорировать файлы из списка *ign* (восклицательный знак '!' означает отмену игнорирования).

-j *rev* Объединить версии.

-k *kflag* Установить подстановку ключевых слов.

-l Локально; без подкаталогов.

-P Опустить пустые каталоги.

-p Выдать файлы на стандартное устройство вывода (избежать липкости).

-R Рекурсивно; с подкаталогами (умолчание).

-r *tag* Использовать при выводе из хранилища версию *tag* (липкий параметр).

-W *spec* Установить фильтры (wrappers).

watch [on|off|add|remove [options] [files ...]] **on/off**: включить/выключить доступ ТОЛЬКО ЧТЕНИЕ файлов, которые скопированы (checkouted) из хранилища.

-a *actions* Определить действия для временно наблюдаемых файлов. Действия могут быть EDIT, UNEDIT, COMMIT, ALL, NONE.

-l Локально; без подкаталогов.

-R Рекурсивно; с подкаталогами (умолчание).

watchers [options [files ...]] Показать, кто наблюдает за файлами (т.е. выдал ранее команду **watch**).

-l Локально; без подкаталогов.

-R Рекурсивно; с подкаталогами (умолчание).

14.20 Описание команд CVS

Общий формат команд **CVS** имеет следующий вид:

cvs [cvs_options] cvs_command [command_options] [command_args]

здесь

cvs Имя программы **CVS**.

cvs_options Некоторые параметры программы **CVS**, которые воздействуют на все команды программы **CVS**.

cvs_command Одна из команд программы **CVS**. Некоторые из этих команд имеют превдонимы, которые могут быть использованы вместо основного имени команды. Рассматриваются только две ситуации, когда может быть опущено поле команды **cvs_command**:

(1) **cvs -H** – показать некоторые вспомогательные возможности системы **CVS** и

(2) **cvs -v** – показать версию программы `cvs`.

command_options Параметры, которые являются специфичными для этой команды.

command_args Аргументы команды.

Можно, к сожалению, перепутать **cvs_options** и **command_options**. Например, **-l** как параметр программы **CVS** воздействует лишь на некоторые команды. Когда **-l** используется на месте **command_options** он имеет совершенно другое значение и воспринимается большим числом команд.

14.20.1 Коды завершения CVS

CVS может устанавливать код завершения. Если все нормально, то **CVS** устанавливает нулевой код завершения. Исключение составляет `cvs diff`. В данном случае нулевой код завершения будет установлен, если не найдено различий в сравниваемых файлах. В противном случае будет установлен ненулевой код завершения.

14.20.2 Инициализационный файл CVS: `.cvsrc`

Часть значений поля **command_options** могут быть установлены постоянно в инициализационном файле `.cvsrc`.

Формат файла `.cvsrc` очень прост. Файл просматривается программой `cvs`, чтобы найти строки, которые начинаются с имени текущей команды **CVS** (поля **cvs_command**). Если такая строка найдена, то остаток строки рассматривается как последовательность параметров данной команды, которые используются ДО любых параметров использованных в командной строке.

Если команда имеет два имени (основное имя и синоним), то основное имя (а не синоним) будет использоваться при сканировании файла **.cvsrc**, независимо от того, какое имя (синоним или основное) использовалось в командной строке. Предположим, что содержимое файла **.cvsrc** таково, как приведено ниже

```
log -N
diff -u
update -P
co -P
```

Если вы введёте команду `cvs checkout foo`, то **CVS** добавит параметр **-P**, поскольку **co** есть синоним **checkout**.

Если вы используете имя **cvs**, то, тем самым, сможете установить глобальные параметры **CVS**. Например, строка

`cvs -z6` в файле `.cvsrc` будет означать, что **CVS** будет использовать уровень компрессии (сжатия данных при передаче по сети) 6.

14.21 Глобальные параметры CVS

Здесь перечислены имеющиеся параметры (поле **cvs_options**, которое находится слева от поля **cvs_command**).

-**allow-root**=*rootdir* Определить допустимое значение для имени корневого каталога *rootdir*.

-a Аутентифицировать все обмены между клиентом и сервером **CVS**. Воздействует лишь на клиентскую часть. Такая аутентификация препятствует некоторым видам атак хакеров. Включение аутентификации не подразумевает включение шифрования информации, передаваемой между клиентом и сервером **CVS**.

-b *bindir* Используется для совместимости с прежними версиями **CVS**. В текущей версии **CVS** ничего не делается.

-T *tempdir* Определяет каталог, где будут находиться временные файлы. Каталог должен задаваться абсолютным именем. Параметр заменяет значение, которое было определено при трансляции, а также то, которое было определено в переменной окружения $TMPDIR.

-d *cvs_root_directory* Использовать *cvs_root_directory* как корневой каталог, вместо определённого в переменной окружения $CVSROOT.

-e *editor* Использовать редактор текста *editor* для ввода комментариев во время операции **commit** вместо другого редактора текста.

-f Не читать конфигурационный файл `.cvsrc`.

-H | -**help** Вывести информацию об использовании команды **cvs_command**.

-l Не записывать поле **cvs_command** в историю команд, но выполнить её.

-n Не изменять никаких файлов. Параметр полезен, когда вы хотите попробовать незнакомую команду или не уверены в правильности использования команды. Ничего не будет изменено, но диагностика будет похожа на ту, что выдаётся при реальном выполнении команды.

-Q Выполняться без диагностики. Сообщения будут выданы лишь в случае очень серьёзных проблем.

-q Выполнение команд не должно сопровождаться обширными сообщениями.

-r Создать новые рабочие файлы с доступом ТОЛЬКО ЧТЕНИЕ. Такой же эффект как от установленной переменной окружения $CVSREAD. По умолчанию все рабочие файлы умеют доступ ЧТЕНИЕ И ЗАПИСЬ, если не установлена возможность **watches**.

-s *variable=value* Установить перемнную окружения пользователя.

-t Трассировать выполнение программы; отображать шаги выполнения программы **cvs**. Может оказаться полезной совместно с параметром **-n**, чтобы понять, как выполняется конкретная команда **CVS**.

-v Отобразить версию программы **cvs**.

-x Зашифровать все обмены данными между клиентом и сервером. Оказывает влияние лишь на клиента. По умолчанию возможность шифровки не обеспечивается. Она должна быть разрешена специальной конфигурационной переменной *-enable-encryption*, когда вы строите программу `cvs`.

-z *gzip-level* Установить уровень сжатия (компрессии) данных при передаче по сети. Имеет смысл лишь на стороне клиента.

14.22 Общие параметры команд CVS

Этот раздел описывает параметры команд, которые являются общими для многих команд **CVS**. Эти параметры всегда находятся справа от команд **CVS**. Не все команды **CVS** поддерживают все общие параметры. Заметным исключением является команда **history**, которая интерпретирует параметры иначе.

-D DATE Использовать наиболее свежую версию, но не позднее, чем **DATE**. Здесь аргумент **DATE** означает одиночный аргумент имеет смысл ДАТЫ создания или ообновления файла или просто ДАТЫ в прошлом.

Эта спецификация является *липкой*, когда вы используете её для создания частной рабочей копии исходного текста. Таким образом, когда вы получаете рабочий файл с использованием параметра **-D**, то **CVS** записывает дату, которую вы определили, так что дальнейшие изменения в том же каталоге будут использовать ту же самую дату.

Параметр **-D** используется в командах **checkout**, **diff**, **export**, **history**, **rdiff**, **rtag**, **update**, хотя в команде **history** этот параметр интерпретируется чуть иначе, чем в остальных.

CVS понимает довольно широкий набор форматов, в которых может быть представлена дата. Используется стандартное представление в соответствии с ISO 8601 (International Standards Organization) и в соответствии со стандартами, используемыми в e–mail (RFC-822).

Даты в соответствии с ISO 8601 могут выглядеть различно:

1999-04-06 означает 6 апреля 1999 года

1999-04-06 10:44 означает 6 апреля 1999 года 10 часов 44 минуты

При этом время интерпретируется в соответствии с локальной временной зоной, если не указано специально что-то иное.

В дополнение к стандартам принятым в e–mail (имеется в виду Internet e–mail) **CVS** распознаёт некоторые сокращения, например:

```
10 Apr 1999 10:05
10 Apr
```

Описанные два формата дат являются рекомендованными, однако воспринимаются и другие форматы, которые часто привязаны к локальным национальным правилам и не являются хорошо документированными.

Один из таких форматов **MONTH/DAY/YEAR**.

Не забудьте использовать кавычки в командной строке, когда будете использовать параметр **-D**, например:

cvs rdiff -D "4 days agos BOOK/CVS_Struct.tex

-f Когда вы определяете частную дату или тег для команд **CVS**, то файлы не соответствующие данной дате или тегу обычно игнорируются. Если же указан параметр **-f** и не найдены файлы соотвествующие заданной дате, то будет произведён поиск самой свежей версии данного файла.

-f может использоваться в командах **annotate**, **checkout**, **export**, **rdiff**, **rtag**, **update**.

Команды **commit** и **remove** также могут использовать параметр **-f**, но он интерпретируется чуть иначе.

-k KFLAG Изменить стандартную подстановку ключевых слов на значение **KFLAG**. Ваша спецификация **KFLAG** является ЛИПКОЙ, когда вы используете параметр, чтобы создать вашу частную рабочую копию исходного текста. Таким образом, когда вы используете этот параметр вместе с командами **update** или **checkout**, то **CVS** связывает заданный вами **KFLAG** с вашим файлом и продолжает использовать это значение параметра **-k** с последующими командами до тех пор, пока вы не определите явно что-то другое.

Параметр **-k** может использоваться с командами **add**, **checkout**, **diff**, **import**, **update**.

-l Локально; работать только в текущем каталоге, без подкаталогов.

Замечание. Это не тот же параметр, что **cvs -l**, который используется СЛЕВА от команды **CVS**.

Параметр **-l** используется с командами **annotate**, **checkout**, **commit**, **diff**, **edit**, **editors**, **export**, **log**, **rdiff**, **remove**, **rtag**, **status**, **tag**, **unedit**, **update**, **watch**, **watchers**.

-m MESSAGE Использовать **MESSAGE** в качестве информации, которая будет записана в журнал (или протокол – log) вместо вызова редактора текста.

Используется в командах **add**, **commit**, **import**.

-n Не выполнять никакие программы для команд **checkout**, **commit** и **tag**. Вызов таких программ записываются в базе данных (файле) **modules**. Параметр **-n** предотвращает вызов этих программ.

Замечание. Это не тот же параметр, который может использоваться слева от команды **CVS -n**.

Параметр может использоваться с командами **checkout**, **commit**, **export**, **rtag**.

-P Опустить пустые каталоги.

-p Направить найденные в хранилище файлы на стандартное устройство вывода, а не записывать их в текущий каталог. Параметр может быть использован вместе с командами **checkout**, **update**.

-R Обрабатывать каталоги рекурсивно, начиная с текущего каталога. Такой стиль обработки установлен по умолчанию.

Этот параметр может использоваться с командами **annotate**, **checkout**, **commit**, **diff**, **edit**, **editors**, **export**, **rdiff**, **remove**, **rtag**, **status**, **tag**, **unedit**, **update**, **watch**, **watchers**.

-r TAG Использовать версию, указанную в **TAG** вместо главной (наиболее свежей) версии исходных текстов по умолчанию. В качестве **TAG** могут использоваться как произвольные теги, установленные командами **tag**

и **rtag**, так и два стандартных тега: **HEAD** - тег, обозначает наиболее свежую версию в хранилище и **BASE** - тег, обозначающий версию в хранилище, которая была переписана последней в ваш рабочий каталог с помощью команды **checkout**.

Спецификация параметра **-r** является ЛИПКОЙ, когда вы используете этот параметр вместе с **checkout** или **update**, чтобы образовать рабочую копию исходного текста в рабочем каталоге. **CVS** помнит эти теги и будет использовать указанные значения в дальнейших командах, если вы не укажете явно обратное. Тег может быть как символическим, так и числовым значением.

Использование глобального параметра **-q** вместе с параметром **-r** часто оказывается полезным, т.к. предотвращает предупредительные сообщения, когда файл **RCS** не содержит указанный тег.

Замечание.Обсуждаемый здесь параметр **-r** нельзя путать с глобальным параметром **-r**.

Параметр **-r** может быть использован вместе с командами **checkout**, **commit**, **diff**, **history**, **export**, **rdiff**, **rtag**, **update**.

-W Определить имена файлов, которые должны подвергнуться фильтрации. Этот параметр можно использовать несколько раз в одной командной строке. Спецификация может содержать шаблон имени файлов в таком же виде как, используется в файле `.cvswrappers`.

Параметр может использоваться с командами **import**, **update**.

14.23 admin – административный интерфейс для rcs

- Требуется: **хранилище, рабочий каталог**.
- Изменяет: **хранилище**.
- Синоним: **rcs**.

Это есть административный интерфейс для системы **RCS**, которая документирована в `rcs(1)`. Команда **admin** просто передаёт все параметры и аргументы системе **RCS** без всякого изменения или проверки. Эта команда выполняется рекурсивно вниз по всей иерархии каталогов, начиная с текущего, поэтому следует использовать её весьма осмотрительно.

В UNIX, если существует группа **cvsadmin**, то только члены этой группы могут выполнять команду **cvs admin**. Такая группа должна существовать на сервере и на машине, на которой выполняется **CVS** в режиме одиночного исполнения (не клиент/сервер). Чтобы запретить использование **cvs admin** для всех пользователей, создайте группу **cvsadmin**, которая не содержит ни одного пользователя.

14.23.1 Параметры команды admin

Часть из перечисленных параметров сохранились лишь по историческим соображениям и, возможно, будет исключена в будущем.

-A*oldfile* Добавить в конец списка доступа **RCS** строку *oldfile*. Может не работать с **CVS**.

-a*logins* Добавить разделённый запятыми список *logins* к списку доступа файла **RCS**. Может не работать с **CVS**.

-b[*rev* **]** Установить умолчание для имени ветви *rev*. В **CVS** вы обычно не манипулируете умалчиваемыми именами ветвей; липкие теги являются более мощным средством для этих целей.

Не должно быть пробелов между **-b** и *rev*.

-c*string* Установить начало комментария в *string*. Поскольку **CVS** не использует теперь таких параметров, то вы можете не беспокоиться об этом.

-e[*logins* **]** Удалить имена, содержащиеся в списке разделённом запятыми *logins*, из списка доступа в файле **RCS**. Если *logins* опущено, то удаляется весь список. Может не работать с **CVS**.

-I Выполнять интерактивно, даже если стандартным вводом является не терминал. Видимо, этот параметр скоро не будет поддерживаться.

-i Бесполезно использовать с **CVS**.

-k*subst* Установить умалчиваемое значение для подстановки ключевых слов в *subst*. Употребление параметра **-k** в командах **update**, **export** и **checkout** будет замещать это умолчание.

-l[*rev* **]** Закрыть версию с номером *rev*. Если дано имя ветви, то закрывается самая свежая версия ветви. Если *rev* опущено, то закрывается наиболее свежая версия ветви по умолчанию. Не должно быть пробелов между **-l** и *rev*.

-L Установить строгое ограничение доступа. Строгое ограничение доступа означает, что владелец файла **RCS** не имеет никаких преимуществ при записи файла. Для использования с **CVS** должно быть включено строгое ограничение доступа.

-m*rev:msg* Заменить текст комментария в версии *rev* текстом в строке *msg*.

-n*name[:[rev]]* Ассоциировать символическое имя *name* с ветвью или версией *rev*. Обычно, лучше использовать **tag** или **rtag**. Если двоеточие и *rev* отсутствуют, то удалить символическое имя. Если *name* уже связано с другим номером версии, то выдать сообщение об ошибке. Если *rev* есть символическое имя, то оно расширяется до связывания. Строка *rev*, содержащая номер ветви, за которым следует точка, обозначает наиболее свежую версию данной ветви. Двоеточие с пустым значением *rev* подразумевает наиболее свежую версию в ветви по умолчанию, обычно главный ствол.

Например, **cvs admin -n***name:* связывает (ассоциирует) *name* с текущими наиболее свежими версиями всех файлов **RCS**, что контрастирует с

cvs admin -n*name***:$**, которая связывает *name* с номерами версий, извлечёнными из ключевых слов, соответствующих рабочих файлов.

-o*range* Удалить версии данные в *range*.

Заметим, что команда довольно опасна. Перед выполнением вы должны хорошо понимать, что вы делаете.

range может быть в различных форматах.

rev1::rev2 Выбросить все версии между *rev1* и *rev2*, таким образом, что **CVS** запомнит лишь различие между *rev1* и *rev2*, удалив все промежуточные шаги. Например, если вы выполнили **-o1.3::1.6**, то вы сможет позже получить версию 1.3 или 1.6, но не сможете получить версию 1.4 или 1.5 или разницу между 1.4 и 1.5. Другой пример, поясняющий работу команды: **-o 1.3::1.4**, не будет иметь никаких последствий, т.к. нет никаких промежуточных версий между 1.3 и 1.4.

::rev Выбросить все версии между началом ветви, которая содержит *rev* и самой *rev*. Например, **-o ::1.3.2.6** удаляет версии 1.3.2.1 и т.д., но оставит 1.3 и 1.3.2.6.

rev:: Удалить все версии между *rev* и концом ветви, содержащей *rev*.

rev Удалить версию *rev*.

rev1:rev2 Удалить версии от *rev1* до *rev2* включительно.

:rev Удалить версии от начала ветви, содержащей *rev* до *rev* включительно.

rev: Удалить все версии, начиная с версии *rev* (включительно), до конца ветви, содержащей *rev*.

-q Выполняться без диагностики.

-s*state[:rev]* Полезно вместе с **CVS**. Установить состояние версии *rev* в значение *state*. Если *rev* есть номер ветви, то предполагается наиболее свежая версия в ветви. Если *rev* опущено, то предполагается наиболее свежая версия в ветви по умолчанию. Любой идентификатор может быть использован как значение *state*. Полезные значения состояний могут быть следующими:

- **Exp** - экспериментальное (experimental);
- **Stab** - стабильное (stable);
- **Rel** - для выпуска (release);

По умолчанию, всем новым модулям состояние *state* устанавливается как **Exp**. Состояние видно в выводе команды **log**, а также в подстановках ключевых слов **Log** и **$State$**.

Заметим, что **CVS** использует состояние **dead** для своих внутренних целей; чтобы перевести файл в состояние **dead** или из состояния **dead** следует использовать команды **remove** и **add** соответственно, а не **admin**.

-t[*file* **]** Полезно с **CVS**. Записать текст комментария из файла *file* в файл **RCS**, удаляя при этом существующий текст. Имя файла *file* не может начинаться со знака минус. Не может быть пробелов между **-t** и *file*. Текстовый комментарий будет виден в выводе команды **cvs log**.

Если *file* опущен, текст будет введён с устройства стандартного ввода. В этом случае текст заканчивается знаком КОНЦА ТЕКСТА или знаком ТОЧКА. Это не работает в схеме клиент/сервер.

-t-*string* Похоже на **-tFILE**. Записать строку *string* на место комментария в файл **RCS**, удалив при этом существующий текст. Не допускается пробелов между **-t** и аргументом.

-U Установить нестрогое ограничение доступа. Нестрогое ограничение доступа означает, что владелец файла не закрывает доступ к версии для помещения её в хранилище. Однако, для использования с **CVS** должно быть установлено строгое ограничение доступа.

-u[*rev*] Смотрите использование параметра **-l** для обсуждения использования параметра **-u** с **CVS**. Снять ограничения по доступу к версии номер *rev*. Если дано имя ветви, то снять ограничения по доступу к наиболее свежей версии в ветви. Если *rev* опущено, то удалить последние по времени ограничения по доступу, которые были установлены данным пользователем. Обычно снять ограничения по доступу может лишь тот, кто установил их. Если кто-то дугой пытается снять ограничения, то это нарушит защиту и приведёт к автоматической посылке сообщения тому, кто наложил ограничения по доступу. Не должно быть пробелов между **-u** и его аргументом.

-VN Устаревший параметр. Вызовет ошибку при исполнении.

-xSUFFIXES В предыдущих версиях **CVS** этот параметр описывался как метод формирования имён файлов **RCS**. Однако, сейчас **CVS** требует, чтобы имена файлов **RCS** всегда заканчивались последовательностью **,v** (запятая и строчная латинская буква **v**). Следовательно, этот параметр не сможет принести много пользы в системе **CVS**.

14.24 checkout – получение исходных текстов из хранилища для редактирования

- Формат использования:

  cvs checkout [options] modules ...

- Требуется: **хранилище**.

- Изменяется: **рабочий каталог**.

- Синонимы: **co**, **get**.

Создать рабочий каталог, содержащий копии исходных файлов, которые описаны как **modules**. Вы должны выполнить команду **checkout** до использования большинства других команд **CVS**, поскольку они предполагают доступ к рабочему каталогу с рабочими копиями файлов.

Аргумент **modules** представляет собой одно из двух:

- символические имена, обозначающие некоторое множество каталогов и/или файлов;

- пути к каталогам или файлам в хранилище (относительно $CVSROOT).

Символические имена определены в конфигурационном файле `modules`.

В зависимости от модулей, которые вы специфицировали, **checkout** может рекурсивно создавать каталожную иерархию в вашем рабочем каталоге и заполнять её файлами из хранилища. После этого вы можете редактировать исходные тексты (файлы) в любое время, несмотря на то, что другие файлы могут в это же время редактироваться другими разработчиками. Вы можете обновлять файлы в рабочем каталоге, чтобы включить те изменения, которые были внесены другими разработчиками в файлах, содержащимися в хранилище. Наконец, вы можете все ваши изменения записать в хранилище исходных текстов (файлов).

Обратим внимание, что **checkout** используется для создания дерева каталогов. Верхний каталог создаётся в том рабочем каталоге, в котором вы вызвали **cvs**. Обычно создаваемый каталог имеет то же имя, что и модуль, который вы хотите отредактировать.

Файлы, которые создаёт команда **checkout**, имеют по умолчанию права доступа ЧТЕНИЕ–ЗАПИСЬ, если не указан глобальный параметр **-r** программы **cvs**, или, если не установлена переменная окружения $CVSREAD, или, если не установлена возможность **watch** по отношению к данному файлу.

Вторичное выполнение команды **checkout** в том же каталоге, в котором уже выполнялась ранее предыдущая команда **checkout**, вполне допустима и имеет то же значение, что и команда **update** с параметром **-d**. Иными словами, любые изменения в хранилище, которые произошли с момента предыдущей команды **checkout**, будут отображены в вашем рабочем каталоге.

14.24.1 Параметры команды checkout

Ниже приведены стандартные параметры, которые поддерживаются командой **checkout**.

-D *date* Использовать наиболее свежую версию, но не позднее, чем дата *date*. Этот параметр предполагает параметр **-P**.

-f Полезно использовать вместе с параметорм **-D** или с параметорм **-r**. Если не найдено подходящих версий, то будет произведён поиск наиболее свежих версий.

-k *kflag* Обработать ключевые слова **RCS** в соответствии со значением строки *kflag*. Подробнее смотрите описание **co(1)**. Будущие обновления этого файла в этом рабочем каталоге будут использовать то же самое значение *kflag*, т.е. этот параметр ЛИПКИЙ.

-l Выполнять только в текущем рабочем каталоге, без подкаталогов.

-n Не выполнять никакой программы, определённой в файле `modules`.

-P Пропустить пустые каталоги.

-p Направить файлы не в рабочий каталог, а на устройство стандартного вывода. Например, по команде

cvs checkout -p `BOOK/CVS_Checkout.tex`

файл `CVS_Checkout.tex` из каталога `BOOK` в хранилище будет выведен на устройство вывода.

-R Обработать дерево каталогов рекурсивно, вместе с подкаталогами (этот режим включён по умолчанию).

-r *tag* Использовать версию помеченную тегом с именем *tag*. Этот параметр ЛИПКИЙ, предполагает параметр **-P**.

В дополнение вы можете использовать ряд специальных параметров команды **checkout**.

-A Сбросить (reset) значения всех ЛИПКИХ тегов, т.е. даты, значения параметра **-k**.

-c Копировать отсортированные определения модулей, записанных в конфигурационном файле `modules` на устройство стандартного вывода, не создавая и не изменяя никаких файлов в рабочем каталоге. Например,

```
cvs checkout -c
```

-d *dir* Создать рабочие копии файлов в новом каталоге с именем *dir*, а не с именем модуля. В общем использование данного параметра эквивалентно последовательности команд

```
mkdir dir
cd dir
cvs checkout ...  без параметра -d
```

Имеется важное исключение. Очень удобно во время выполнения операции **checkout** над отдельным объектом, чтобы он появлялся на выходе без пустых промежуточных каталогов. Только в этом случае **CVS** пытается "укоротить"имена файлов (пути относительно $CVS-ROOT), чтобы пропустить пустые каталоги.

Например, модуль с именем `trans` содержит файл с именем `base.c` и команда

```
cvs checkout -d dir trans
```

создаст каталог `dir` и поместит внутрь этого каталога модуль с именем `base.c`. Подобным же образом, если модуль `barn`, который имеет подкаталог с именем `room`, который, в свою очередь содержит файл `mouse.c`, то команда

```
cvs checkout -d dir barn/room
```

создаст каталог *dir* и поместит в него файл `mouse.c`.

Использование параметра **-N** отменяет такое поведение. Таким образом прежние примеры будут выглядеть иначе. Так, команда

cvs checkout -N -d `dir trans`

создаст каталог `dir`, а в нём создаст каталог с именем `trans`, в котором и поместит файл `base.c`. А команда

```
cvs checkout -N -d dir barn/room
```

создаст дерево каталогов `dir/barn/room`, в котором поместит файл `mouse.c`.

-j *tag* С двумя параметрами **-j** команда объединит версии помеченные первым и вторым тегами и поместит результат в рабочий каталог. Если использовался один параметр **-j**, то будет произведено объединение предыдущей версии с версией, определённой параметром **-j**.

В дополнение, каждое значение параметра **-j** может содержать дату, которая при использовании в ветвях может ограничить множество версий внутри ветви лишь одной с заданной датой модификации. В этом случаеt формат использования параметра **-j** таков:

-j*tag***:***date*

Более подробное обсуждение ветвей смотрите в разделе 14.7.

-N Параметр имеет смысл лишь при совместном использовании с параметром **-d**. **CVS** предотвращает укорачивание имени файла в рабочем каталоге при выполнении команды. Смотрите описание параметра **-d**.

-s Похоже на параметр **-c**, но включает состояние всех модулей и сортирует их по статусной строке.

14.25 diff – показать отличия между версиями

- Формат использования:

  diff [-lR] [format_options] [[-r *rev1* | **-D** *date1*] [**-r** *rev2* | **-D** *date2*]] [*files...*]

- Необходимы: **рабочий каталог, хранилище**.

- Изменения: **ничего не меняется**.

Команда **diff** используется, чтобы увидеть отличия в различных версиях одного файла. По умолчанию сравниваются файл в рабочем каталоге с версиями этого файла в хранилище и показываются все отличия. Если дано имя файла, то сравнивается файл с этим именем, а если дано имя каталога, то сравниваются все файлы каталога. Выработка кода завершения для команды **diff** отличается от других команд системы **CVS**. Эта команда возвращает значение **0** (нуль), если нет никаких различий между файлами, и ненулевое значение, если встретились различия или ошибки.

14.25.1 Параметры команды diff

Команда **diff** поддерживает следующий стандартный набор параметров.

-D *date* Использовать наиболее свежую версию до даты *date*. Смотрите параметр **-r**, чтобы яснее понять влияние на сравнение.

-k *kflag* Обрабатывать ключевые слова в соответствии со значением *kflag*.

-l Выполнять только в текущем каталоге (без подкаталогов).

-R Обрабатывать файлы рекурсивно, т.е. текущий каталог вместе с подкаталогами. Это значение параметра принимается по умолчанию.

-r *tag* Сравнить с версией *tag*. Могут присутствовать: один параметр **-r**, два параметра **-r** или ни одного. Если не указан параметр **-r**, то будет производиться сравнение рабочего файла с той версией в хранилище, на которой базируется файл в рабочем каталоге. Если присутствует два параметра **-r**, то будут сравниваться две версии в хранилище (содержание рабочей копии файла в рабочем каталоге не будет иметь влияние).

Один параметр **-r** или оба могут быть заменены параметрами **-D**, если это требуется.

Следующие параметры определяют формат вывода. Они имеют то же значение, что и в утилите GNU **diff**. В связи с этим, мы только перечислим их.

-0 -1 -2 -3 -4 -5 -6 -7 -8 -9

-binary

-brief

-changed-group-format=ARG

-c [-C NLINES]
[-e -ed]

-t -expand-tabs

-f -forward-ed

-horizon-lines=ARG

-ifdef=ARG

-w -ignore-all-space

-B -ignore-blank-lines

-i -ignore-case

-I REGEXP [-ignore-matching-lines=REGEXP]

-h

-b -ignore-space-change

-T -initial-tab

-L LABEL -label=LABEL

-left-column

-d -minimal

-N -new-file

-new-line-format=ARG

-old-line-format=ARG

-paginate

-n -rcs

-s -report-identical-files

-p

-show-c-function

-y -side-by-side

-F REGEXP -show-function-line=REGEXP

-H -speed-large-files

-suppress-common-lines

-a -text

-unchanged-group-format=ARG

-u [-U NLINES]
[-unified[=LINES]

-V ARG

-W COLUMNS [-width=COLUMNS]

14.25.2 Примеры использования команды diff

Следующая строка производит сравнение между версиями 1.1 и 1.2 файла CVS_Diff.tex Параметр **-u** означает **Unidiff**. Параметр **-kk** означает, что никакие подстановки ключевых слов не производятся, так что не будет никакой разницы между файлами, которая порождена лишь различной подстановкой ключевых слов.

```
cvs diff -kk -u -r 1.1 -r 1.2 CVS_Diff.tex
```

Предположим, что экспериментальная ветвь **EXPR1** базируется на наборе файлов, которые описываются тегом с именем **RELEASE_1_0**. Чтобы увидеть, что происходит с этой ветвью, можно воспользоваться командой

```
cvs diff -r RELEASE_1_0 -r EXPR1
```

Здесь будут сравниваться файлы из ветви **EXPR1** и комплект файлов с тегом **RELEASE_1_0**.

Подобная команда может использоваться, чтобы получить изменения в версиях

```
cvs diff -c -r RELEASE_1_0 -r RELEASE_1_1 > diffs
```

Если вы ведёте журнал изменений, то нижеследующая команда позволит вам перед выполнением команды **commit** подготовить записи в ваш журнал. Будут напечатаны все изменения, которые вы ещё не подтвердили в хранилище.

```
cvs diff -u | less
```

14.26 export – экспорт исходных текстов из хранилища

- Использование:

  ```
  export [-flNnR] [-r rev | -D date] [-k subst] [-d dir] module...
  ```

- Требует: **хранилище**.
- Изменения: **текущий каталог**.

Эта команда представляет собой особый вариант команды **checkout**. Команда **export** выдаёт исходный текст модуля из хранилища, но не создаёт административных каталогов **CVS**, как это делает команда **checkout**. Например, если вы предполагаете передать кому-то исходный текст модуля, то, возможно, будет удобно воспользоваться командой **export**. Эта команда требует, чтобы вы указали дату или тег (**-D** или **-r**), так что вы сможете произвести необходимые действия над текстом, который вы передаёте (you can count on reproducing).

Обычно полезно использовать параметр **-kv** с командой **export**. Это приведёт к подстановке всех ключевых слов в файле. Следовательно, никакая информация о модуле не будет утеряна. Однако, вам следует знать, что двоичные файлы не могут быть корректно экспортированы таким образом. Также следует помнить, что после использования параметра **-kv** нельзя выполнять команду **ident**, которая является частью пакета **RCS** (смотрите описание ident(1)) и ищет ключевые строки. Если вы желаете использовать позже команду **ident**, вы не должны использовать параметр **-kv**.

14.26.1 Параметры команды export

Команда **export** поддерживает следующие стандартные параметры.

-D *date* Использовать наиболее свежую версию до даты *date*.

-f Если не найдено указанной версии, использовать последнюю версию (наиболее свежую).

-l Локально; выполнять (экспортировать) только в локальном рабочем каталоге (без подкаталогов).

-n Не выполнять никаких программ, которые вызывает **checkout**.

-R Экспортировать каталоги рекурсивно, т.е. включая подкаталоги.

-r *tag* Использовать версию *tag*.

В дополнение поддерживаются следующие параметры, которые являются общими с командой **checkout**.

-d *dir* Создать каталог с именем *dir* для рабочих файлов вместо использования имени модуля.

-k *subst* Установить режим подстановки ключевых слов.

-N Полезно использовать лишь вместе с параметром **-d** *dir*. Смотрите также описание команды **checkout**.

14.27 history – показать историю изменения состояния хранилища

- Использование:

  history [-report] [-flags] [-options args] [files...]

- Требуется: файл **$CVSROOT/CVSROOT/history**

- Изменяется: **ничего не изменяется**.

CVS может хранить историю в файле, который содержит каждое использование команд: **checkout**, **commit**, **rtag**, **update**, **release**. Вы можете использовать команду **history**, чтобы отобразить содержание файла `$CVSROOT/CVSROOT/history` в различных форматах.

Предупреждение. Команда **history** интерпретирует параметры **-f**, **-l**, **-n**, **-p** иначе, чем принято в системе **CVS**.

14.27.1 Параметры команды history

Ниже приведён список параметров для управления какая информация будет напечатана.

-c Вывести информацию, когда выполнялась команда **commit** (когда изменялось содержимое хранилища).

-e Вывести все типы записей из файла `$CVSROOT/CVSROOT/history`.

-m *module* Вывести сведения об отдельном модуле. Можно использовать этот параметр несколько раз в командной строке.

-o Вывести информацию о том, когда выполнялась команда **checkout**.

-T Вывести информацию о всех тегах.

-x *type* Выдать из файла `$CVSROOT/CVSROOT/history` записи типа *type*. Типы записей указываются одной буквой. В одной строке можно указать более одного типа записей, т.е. использовать комбинацию букв.

Часть команд имеют лишь один тип записи.

F – команда **release**.

O – команда **checkout**.

E – команда **export**.

T – команда **rtag**.

Команда **update** может давать четыре типа записи в файл истории.

C – было необходимо слияние изменений, однако обнаружены коллизии, которые могут быть устранены только вручную.

G – Было успешно выполнено необходимое слияние изменений.

U – Рабочий файл был скопирован из хранилища.

W – Рабочая копия файла была удалена во время выполнения команды **update**, поскольку её нет в хранилище.

Команда **commit** может порождать три типа записей в файле истории.

A – вывести данные всех пользователей (по умолчанию выводятся данные лишь того пользователя, который выполняет команду **history**).

M – файл был модифицирован.

R – файл был удалён.

Параметры, которым предшествует знак **-** (минус), ограничивают или расширяют объём вывода без дополнительных аргументов для этих параметров.

-a Показать данные для всех пользователей (по умолчанию показываются данные лишь того пользователя, который выполняет команду **history**).

-l Показать только последнюю модификацию.

-w Показать только записи модификации, которые были выполнены из того же рабочего каталога, где выполняется команда **history**.

Параметры, которым предшествует знак **-** (минус) и за которыми следует аргумент, приводят к тому, что вывод строится в соответствии со значением аргумента.

-b *str* Показать записи, которые содержат строку *str* в имени модуля, имени файла или в имени пути к хранилищу.

-D *date* Показать записи, начиная с даты *date*. Это чуть-чуть отличается от обычного использования параметра **-D** в **CVS**, когда оно обозначает наиболее свежую версию до даты *date*.

-p *repository* Показать данные для отдельного хранилища. Можно использовать несколько параметров **-p** в одной командной строке.

-r *rev* Показать записи, ссылающиеся на версии, начиная с версии или тега с именем *rev*, который появляется в отдельных файлах **RCS**. Поиск каждого файла производится в соответствии с версией или тегом *rev*.

-t *tag* Показать записи, начиная с момента, когда тег с именем *tag* был добавлен к файлу истории. Это отличается от параметра **-r** (выше), в котором читается лишь файл истории, а не файлов **RCS**, что значительно быстрее.

-u *name* Показать записи для пользователя с именем *name*.

14.28 import – импортирование текстов в систему CVS

- Использование:

  import [-options] repository vendortag releasetag...

- Требуется: **хранилище, каталог с исходными текстами**.
- Изменения: bf хранилище.

Команда **import** предназначена для внесения исходных текстов в хранилище системы **CVS** из внешнего источника. Вы можете использовать эту команду как для первоначального создания хранилища, так и для полного обновления в модуле из внешнего источника.

Аргумент **repository** даёт имя каталога внутри корневого каталога хранилища системы **CVS**. Если каталог не существует, то он будет создан.

Когда команда **import** используется для обновления исходных текстов в хранилище, которые были модифицированы прошлым вызовом команды **import**, вы будет проинформированы о любых файлах, которые конфликтуют в двух ветвях разработки. Следует использовать **checkout -j**, чтобы объединить модификации.

Если существует конфигурационный файл $CVSROOT/CVSROOT/CVSWRAPPERS, то любые импортируемые файлы, чьи имена удовлетворяют спецификациям в упомянутом конфигурационном файле, будут рассматриваться как пакеты. К ним будет применена специальная фильтрация до выполнения реального импортирования.

Импортированный внешний комплект исходных текстов сохраняется в хранилище в ветви первого уровня, по умолчанию: 1.1.1. Обновления остаются в той же ветви. Например, файлы из импортированной коллекции исходных текстов будут иметь версию 1.1.1.1, тогда как файлы из только что импортированной и раз обновлённой версии будут иметь номер версии 1.1.1.2, и т.д.

В команде **import** требуются как минимум три аргумента.

repository требуется, чтобы идентифицировать данный комплект исходных текстов.

vendortag является тегом для обозначения полной ветви, например, 1.1.1.

releasetag используется, чтобы обозначить файлы, которые создаются каждый раз при выполнении команды **import**.

Заметим, что команда **import** не изменяет каталог, в котором она выполняется. В частности, она не устанавливает текущий каталог в качестве рабочего каталога **CVS**. Если вы желаете работать с исходными текстами в контексте **CVS**, вам следует импортировать исходные тексты, затем выполнить команду **checkout** в другом каталоге.

14.28.1 Параметры команды import

Это стандартные параметры, которые поддерживаются командой **import**.

-m *message* Записать строку *message* в файл протокола, вместо вызова редактора текста.

Имеются также дополнительные параметры, которые воспринимаются командой **import**.

-b *branch* Смотрите раздел 14.14.5, где рассматривается несколько ветвей поставщика (vendor branches).

-k *subst* Требуется режим подстановки ключевых слов. Этот режим будет установлен для всех файлов, созданных во время импортирования, но не будет действовать для тех файлов, которые уже имеются в хранилище. Смотрите "Режимы подстановки ключевых слов"в разделе 14.36.

-I *name* Определить имена файлов, которые будут игнорироваться во время импортирования. Вы можете использовать этот параметр несколько раз. Чтобы избежать игнорирования файлов вообще, используйте **-I !**.

Стока *name* может содержать то же, что и строка для игнорирования имён в файле .CVSIGNORE. Смотрите раздел 14.17.

-W *spec* Определить имена файлов, которые должны быть отфильтрованы во время выполнения команды **import**. Этот можно использовать несколько раз.

Строка *spec* может быть образцом имени файла такого же типа как и в файле .CVSWRAPPERS. Смотрите раздел 14.9.

14.28.2 Вывод команды import

Команда **import** информирует вас о выполняемых действиях выводом диагностических строк о состоянии каждого файла (по одной строке на файл). Каждому имени файла предшествует буква, показывающая статус файла.

U FILE Файл уже существует в хранилище и не модифицирован локально. Создана новая версия файла (если это было необходимо).

N FILE Новый файл, который был добавлен в хранилище.

C FILE Файл уже существует в хранилище, но был модифицирован в локальном рабочем каталоге. Сначала следует выполнить операцию слияния изменений с содержимым хранилища.

I FILE Файл игнорируется, смотрите раздел 14.17.

L FILE Файл является символическим линком; команда **import** игнорирует символические линки. Люди периодически предлагают сменить поведение команды, но это не совсем очевидное улучшение. В то же время различные параметры в файле MODULES позволяют воссоздать символические линки в командах: **checkout**, **update** и т.д. Смотрите раздел 14.8.

14.28.3 Примеры использования команды import

Читайте раздел 14.14.

14.29 log – выдать протокольную информацию для файлов

- Использование:

  cvs log [options] [files...]

- Требуется: **хранилище, рабочий каталог**.

- Изменения: **ничего не изменяется**.

Команда отображает информацию о файлах в хранилище и в рабочем каталоге (точнее выводит информацию на устройство стандартного вывода). Для выполнения команда **log** вызывала ранее утилиту системы **RCS**. Сейчас это не соответствует действительности (по меньшей мере для версии **CVS 1.10**), но стиль вывода и способ использования параметров чуть отличаются от обычного в **CVS**.

Вывод включает в себя расположения файла **RCS**, заголовок версии, все символические имена (теги) и другую информацию.

Предупреждение.

Команда **log** использует параметр **-R** необычным способом, который отличается от обычного внутри **CVS**.

14.29.1 Параметры команды log

По умолчанию **log** печатает всю имеющуюся информацию о файлах. Параметры используются для того, чтобы ограничить объём вывода.

-b Выдать информацию о версиях в подразумеваемой ветви, обычно последняя по времени ветвь в основном стволе разработки.

-d *dates* Выдать информацию о версиях файлов вместе с датами и временами ввода в хранилище. Даты могут быть разделены точкой с запятой, если их несколько. Даты могут быть комбинированы, как показано ниже:

d1<*d2*

d1>*d2* Выбрать версии, которые были помещены в хранилище между датами *d1* и *d2*.

<*d*

d> Выбрать все версии, которые были внесены в хранилище с датой *d* или раньше.

>*d*

d< Выбрать все версии, которые были внесены в хранилище с датой *d* или позже.

d Выбрать одну версию, датированную *d* или ранее.

За знаками > (больше) и < (меньше) могут следовать знак = (равно), чтобы обозначить замкнутый интервал, а не разомкнутый.

Заметим, что разделителем является знак '**;**' (точка с запятой).

-h Напечатать только имя файла **RCS**, имя файла в рабочем каталоге, заголовок, ветвь по умолчанию, список доступа, защиту, символические имена и суффикс.

-l Локально; выполнять только в локальном каталоге (без подкаталогов). Умолчание - выполнять рекурсивно, с подкаталогами.

-N Не выводить список тегов для файла. Этот параметр очень полезен, если в вашей группе разработчиков используется много тегов, например, пять страниц.

-R Вывести только имя файла RCS.

-r*revisions* Вывести информацию о версиях. Нижеследующая таблица показывает допустимые форматы строки *revisions*.

rev1:rev2 Версии от *rev1* до *rev2*, которые должны быть в одной ветви.

:rev Версии начиная с ветви по умолчанию до *rev* включительно.

rev: Версии начиная с *rev* до конца ветви, содержащей *rev*.

branch Аргумент означает имя ветви, т.е. имеются в виду все версии из этой ветви.

branch1:branch2 Ряд ветвей, подразумеваются все версии из этого ряда ветвей.

branch. Наиболее свежая версия в ветви *branch* (в параметре точка на конце).

Параметр **-r** без аргументов означает наиболее свежую версию в ветви по умолчанию. Обычно это основной ствол. Между параметром **-r** и аргументом не должно быть пробелов.

-s *states* Выдать информацию о версиях, чьи атрибуты состояния удовлетворяют одному из состояний данных в списке состояний, разделённых запятыми *states*.

-t Выдать то же самое, что и **-h** плюс поле комментариев.

-w*logins* Выдать информацию о версиях, которые были внесены в хранилище пользователями, имена которых перечислены в списке имён, раздлённых запятыми, в строке *logins*. Если *logins* опущена, то подразумевается имя пользователя, который выполняет команду **log**. Между параметром **w** и его аргументом не должно быть пробелов.

Команда **log** выдаёт пересечение (intersection) версий, выбранных параметрами **-d**, **-s**, **-w** с объединением версий, отобранных посредством параметров **-b** и **-r**.

14.30 rdiff – различия между версиями в формате PATCH

- Использование:

  rdiff [-flags] [-V vn] [-r t | -D d [-r t2 | -D d2]] modules...

- Требуется: **хранилище**.
- Изменения: **ничего не меняется**.
- Синоним: **patch**.

Команда **rdiff** строит входной файл для программы PATCH, которую разработал **Larry Wall**, чтобы преобразовать старую версию исходного текста в последнюю версию. Команда **rdiff** является одной из нескольких команд **CVS**, которые работают прямо с хранилищем, не требуя выполнения перед ними команды **checkout**. Вывод команды **rdiff** направляется на стандартное устройство вывода.

Используя стандартные параметры **-r** и **-D**, вы можете определить любую комбинацию одной или двух версий или дат модификации. Если указана лишь одна версия или дата, то результирующий файл будет отражать различия между указанной версией и текущей главной версией в файле **RCS**.

Заметим, что если выбранная программная версия содержится более, чем в одном каталоге, тогда может оказаться необходимым определить параметр **-p** для программы PATCH, когда будет происходить обновление старых версий. Таким образом, PATCH сможет найти файлы, которые находятся в других каталогах.

14.30.1 Параметры rdiff

Нижеследующие стандартные параметры поддерживаются командой **rdiff**.

-D *date* Использовать наиболее свежую версию, но не позже даты *date*.

-f Если не найдено соответствующей версии, то взять последнюю версию, вместо игнорирования файла.

-l Локально; не обрабатывать подкаталоги, работать только в локальном каталоге.

-R Обрабатывать все подкаталоги.

-r *tag* Использовать версию *tag*.

В дополнение к вышеприведённым, имеется несколько дополнительных команд.

-c использовать контекстный формат **diff**. Это есть умолчание.

-s Создать сводный отчёт об изменениях вместо файла для программы **patch**. Сводный отчёт включает информацию о файлах, которые были изменены или добавлены в промежутке между двумя версиями. Отчёт посылается на стандартное устройство вывода. Этот параметр удобен, чтобы определить, какие файлы менялись в промежутке между двумя датами.

-t Различия двух последовательных последних версий посылаются на стандартный вывод. Это одна из удобных возможностей определить, как менялся конкретный файл.

-u Использовать формат **undiff** для контекстного построения различий. Этот параметр недоступен для вашей программы **diff**, если она не поддерживает формат **unidiff**. Помните, что старые версии программы PATCH не понимают формат **unidiff**.

14.30.2 Примеры использования rdiff

Предположим, что вас спросили о последних изменениях файла `CVS_Struct.tex` в модуле **BOOK**. Проще всего ответить так:

cvs rdiff -t BOOK/CVS_Struct.tex

В ответ вы получите что-то вроде нижеследующего:

```
*** BOOK/CVS_Struct.tex:1.10    Thu Apr  1 16:24:04 1999
--- BOOK/CVS_Struct.tex Fri Apr  2 19:00:58 1999
***************
*** 69,74 ****
--- 69,75 ----
  \input{CVS_Export.tex}                % Export
  \input{CVS_History.tex}               % History
  \input{CVS_import.tex}                % Import
+ \input{CVS_Log.tex}           % Log
  \input{CVS_commit.tex}                % Commit
  \input{CVS_Update.tex}                % Update
  \input{CVS_Rtag.tex}          % Rtag
```

Здесь знаком + (плюс) помечена строка, где произошли изменения последний раз. Видно также, что последний раз файл `BOOK/CVS_Struct.tex` изменялся 2-ого апреля 1999, а предыдущее изменение имело место 1-ого апреля 1999.

Предположим, вы образовали версию 1.3 и создали ветвь с именем **R13fix** для коррекции возможных ошибок. При этом ветвь **R131** соответствует версии 1.3.1, которую вы сделали некоторое время назад. Теперь, вы хотите увидеть, сколько было сделано работы по изменениям исходных текстов. Это можно выполнить с помощью команды:

```
cvs rdiff -s -r R131 -r R13fix module-name
```

14.31 commit – команда записи изменеий в хранилище

- Формат команды:

  commit [-lnRf] [-m 'log_message' | -F file] [-r revision] [files...]

- Требуется: **рабочий каталог, хранилище**.

- Изменяется: **хранилище**.

- синоним: **ci**.

Команда **commit** используется, чтобы внести изменения, произведённые в рабочем каталоге, в основное хранилище.

Если вы не определили конкретный файл или группу файлов, то система **CVS** будет проверять все файлы в рабочем каталоге, выясняя изменились ли они. **commit** проверяет файлы весьма тщательно и делает изменения в хранилище лишь в случае реального изменения содержимого файла, а не просто даты создания. По умолчанию, или явным указанием параметра **-R** будут проверяться все подкаталоги рабочего каталога. Вы можете ограничить область действия команды **commit** лишь текущим каталогом, использовав параметр **-l**.

Если содержимое файла не изменилось, то **commit** сообщает об этом и ничего не изменяет. Если **commit** обнаруживает, что необходимо выполнить команду **update**, то вам будет сообщено об этом, но не будет никакого автоматического вызова **update**. Предполагается, что вы лучше знаете, когда следует выполнять команду **update**.

Когда все нормально завершилось, вызывается редактор текста, чтобы вы могли ввести какие-то комментарии к данной операции **commit**. Комментарии будут переданы одной или больше протокольных программ, которые запишут эти комментарии в файл RCS внутри хранилища. Эти комментарии могут быть найдены позже с помощью команды **log**. Короткий комментарий может быть определён в командной строке с помощью параметра **-m** *message*, тогда редактор не будет вызываться. Если вы хотите записать относительно длинный комментарий и, одновременно, избежать вызова редактора текста, можно использовать параметр **-F** *file*. Комментарий будет взят из файла с именем *file*.

14.31.1 Параметры commit

Здесь описаны стандартные параметры, которые поддерживаются командой **commit**.

-l Ограничить область действия текущим каталогом (без подкаталогов).

-n Не выполнять никаких программ, которые определены в файле `modules`.

-R Установить область действия команды **commit** ТЕКУЩИЙ РАБОЧИЙ КАТАЛОГ + ВСЕ ПОДКАТАЛОГИ РАБОЧЕГО КАТАЛОГА.

-r *revision* Выполнить **commit** для версии *revision*. Значением *revision* должно быть одно из двух: ВЕТВЬ или версия в основном стволе, которая находится выше, чем любой существующий номер версии. Выполнить **commit** для отдельной версии в ветви невозможно.

Кроме вышеперечисленных, команда **commit** поддерживает следующие параметры.

-F *file* Взять текст комментария из файла с именем *file* вместо вызова редактора текста.

-f Заставляет систему **CVS** выполнить команду **commit**, образовав ори этом новую версию исходных текстов, даже если вы не производили никаких изменений в файлах. Если текущая версия текстов есть 1.4, то следующие две команды эквивалентны:

```
cvs commit -f FILE
cvs commit -r 1.5 FILE
```

Параметр **-f** выключает рекурсию, т.е. предполагает **-l**. Чтобы заставить **commit** обработать все файлы во всех подкаталогах, вам следует использовать два параметра **-f -R**.

-m *message* Использовать строку *message* в качестве комментария вместо вызова редактора текста.

14.31.2 Примеры использования commit

Вы можете выполнить **commit** для версии в ветви (содержит четное число разделительных точек) с использованием параметра **-r**. Чтобы создать версию ветви используйте параметр **-b** в командах **tag** или **rtag**. После этого вы сможете, использовав команды **update** или **checkout**, привести в соответствие состояние вашего рабочего каталога и вновь образованной версии в ветви, не разрушив ваши исходные тексты в основном стволе разработки. Например, вам необходимо создать заплату (исправление) вашего продукта версии 1.2, хотя вовсю идёт разработка варианта 2.0. Сделать это можно следующим образом.

```
cvs rtag -b -r FCS1_2 FCS1_2_Patch product_module
cvs checkout -r FCS1_2_Patch product_module
cd product_module
[[ hack away ]]
cvs commit
```

Это действует автоматически поскольку параметр **-r** является липким.

Создание ветви после редактирования

Скажем вы работали над экспериментальным программным обеспечением, которое базировалось на какой-то версии, которую вы взяли в рабочий каталог (выполнили **checkout**) на прошлой неделе. Если другие разработчики тоже работают с теми же версиями исходных текстов, то чтобы не портить основной поток разработки, вы могли бы записать ваши варианты экспериментальные в отдельную новую ветвь. Другие разработчики, смогут использовать ваши экспериментальные наработки и, одновременно, целиком использовать мощь **CVS** в отношении разрешения конфликтов. Сценарий мог бы быть таким.

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs update -r EXPR1
$ cvs commit
```

Команда **update** сделает параметр **-r EXPR1** липким для всех файлов. Заметим, что ваши изменения в файлах никогда не будут удалены командой **update**. Команда **commit** будет автоматически работать с верной ветвью, поскольку параметр **-r** – липкий. Вы также можете делать так:

```
[[ hacked sources are present ]]
$ cvs tag -b EXPR1
$ cvs commit -r EXPR1
```

но тогда, только изменённые вами файлы станут липкими. Если вы уберёте изменения и выполните **commit** без указания параметра **-r EXPR1**, то некоторые файлы неожиданно для вас могут попасть в основной ствол.

Другие разработчики могут использовать ваши экспериментальные наработки, если выполнят команду:

```
cvs checkout -r EXPR1 whatever_module
```

14.32 update – синхронизировать рабочий каталог и хранилище

- Формат вызова:

  cvs update [-AdflPpR] [-d] [-r tag|-D] files ...

- Требуется: **хранилище, рабочий каталог.**

- Изменения: **рабочий каталог.**

После того как вы выполнили команду **checkout**, чтобы создать рабочий каталог, который содержит вашу персональную копию исходных текстов из общего хранилища, другие разработчики продолжают работать над содержимым хранилища. Естественно, при этом меняются исходные тексты в общем хранилище, в том числе и те тексты, которые вы используете в своей работе. Это означает, что время от времени, когда вам это удобно, необходимо приводить содержимое вашего рабочего каталога (включая содержание файлов) в соответствие с состоянием общего хранилища.

Эта операция производится с помощью команды **update**.

14.32.1 Параметры команды update

Здесь приведены стандартные параметры команды **update**.

-D *date* Использовать наиболее свежую версию, но не позже даты *date*. Этот параметр ЛИПКИЙ и подразумевает **-P**.

-f Полезно только вместе с параметрами **-D** *date* или **-r** *tag*. Если не найдено подходящих версий, то производится поиск наиболее свежих версий.

-k *kflag* Обрабатывать ключевые слова **RCS** в соответствии с *kflag*. Смотрите описание **co(1)**. Этот параметр ЛИПКИЙ: будущие изменения этого файла в этом рабочем каталоге будут использовать то же значение *kflag*. Команда **status** показывает установленные значения ЛИПКИХ (sticky) тегов. Например,

```
cvs status CVS_cvscat.tex
```

в ответ будет напечатано что-то похожее на следующее.

```
===================================================================
File: CVS_cvscat.tex    Status: Up-to-date

   Working revision:    1.1     Wed Feb 24 18:09:45 1999
```

```
Repository revision: 1.1     /home/shevel/WorkCVS/BOOK/CVS_cvscat.tex,v
Sticky Tag:          MyCVS (revision: 1.1)
Sticky Date:         (none)
Sticky Options:      (none)
```

-l Локально; выполнять только в текущем рабочем каталоге (без подкаталогов).

-P Удалить пустые каталоги.

-p Вывести файлы на стандартное устройство вывода, например, направить в следующий фильтр.

-R Выполнять рекурсивно. Это есть умолчание.

-r *tag* Найти версию *tag*. Этот параметр связан; предполагает использование **-P**.

Далее следуют специальные параметры, которые могут использоваться с командой **update**.

-A Сбросить (reset) ЛИПКИЕ значения: *tag*, *date* или значения установленные параметром **-k**.

-d Создать любые каталоги, которые имеются в общем хранилище, но отсутствуют в вашем рабочем каталоге. Обычно, **update** воздействует только на те каталоги и файлы, которые уже находятся в вашем рабочем каталоге.

Такая возможность полезна для приведения вашей собственной копии рабочего материала (программ, описаний, прочего) в соответствие с общим хранилищем. Если вы намеренно избегаете копирования определённых каталогов из общего хранилища в ваш рабочий каталог посредством использования имени модуля или точным перечислением имён файлов и каталогов в командной строке команды **checkout**, то использование параметра **-d** в команде **update** приведёт к копированию всех каталогов. Может оказаться, что копирование всех каталогов и файлов из хранилища совсем не входит в ваши планы.

-I *name* Игнорировать файлы, чьё имя удовлетворяет строке *name* (в вашем рабочем каталоге) во время выполнения команды **update**. Вы можете использовать параметр **-I** несколько раз в одной команде, чтобы указать несколько файлов, которые вы хотели бы пропустить. Чтобы отменить игнорирование всех файлов, следует использовать **-I !**.

-W*spec* Определить имена файлов, которые (имена) должны быть отфильтрованы во время выполнения команды **update**. Этот параметр может применяться несколько раз в одной команде.

-j*revision* Если параметр **-j** используется однократно, то следует выполнить слияние предыдущей версии с версией, которая описывается параметром **-j**. Результат поместить в рабочий каталог. Термин ПРЕДЫДУЩАЯ ВЕРСИЯ означает общего предка версии, которая находится в рабочем каталоге и версии, которая указана параметром **-j**.

Два параметра **-j** означают, что следует выполнить слияние изменений в версии, описываемой первым параметром **-j**, с изменениями в версии, описываемой вторым параметром **-j**, а результат записать в рабочий каталог.

В дополнение, каждый параметр **-j** может специфицировать дату, которая при использовании с ВЕТВЯМИ может ограничить выбранные версии определёнными датами. Дату можно задать в следующем виде:

-j*symbolic_tag:date*

Как видно, дата (*date*) отделяется от тега (*symbolic_tag*) двоеточием.

14.32.2 Описание диагностики команды update

Команды **update** и **checkout** информируют вас о выполняемых действиях путём вывода сообщений (по одной строке на обработанный файл). В качестве первого символа в строке используются несколько букв для обозначения состояния файла:

U FILE - Файл в вашем рабочем каталоге приведён в соответствии с состоянием файла в общем хранилище. Это делается для всех файлов, которые уже имеются в вашем рабочем каталоге, но не в хранилище. Даже если вы не изменяли какой-то файл в вашем рабочем каталоге, но он был изменён в хранилище, то он будет также приведён в соответсвие с состоянием в хранилище.

A FILE - Файл был добавлен к вашей рабочей копии исходных текстов. Он будет записан в хранилище когда вы выполните команду **commit**.

R FILE - Файл был удалён из вашего рабочего каталога (из вашей рабочей копии исходных текстов) и будет удалён из хранилища, когда вы выполните команду **commit**.

M FILE - Файл был модифицирован в вашем рабочем каталоге. Обозначение **M** может означать одно из двух состояний, в котором находится файл, с которым вы работаете:

- не было никаких модификаций этого файла в хранилище, а имеют место только те изменения, что вы сами внесли в вашу копию файла в вашем рабочем каталоге;
- имели место как изменения в хранилище так и изменения в вашем рабочем каталоге, однако эти обе версии изменений были объединены без конфликтов в вашем рабочем каталоге.

CVS будет выводить ряд сообщений, если она выполняет слияние вашей работы и делает резервную копию (backup) вашего рабочего файла, сохраняя тот вид, который он имел до выполнения команды **update**.

C FILE - Встречен конфликт (перекрывающиеся изменения) во время выполнения слияния ваших изменений в вашем рабочем каталоге в файле с именем **FILE** с изменениями, которые имели место в хранилище. Файл с именем **FILE** в вашем рабочем каталоге представляет собой вывод команды **rcsmerge(1)** и имеется в двух версиях. Немодифицированная копия файла с именем **FILE** также находится в вашем рабочем каталоге и имеет имя `.#FILE.REVISION`, где **REVISION** есть версия в стандарте **RCS**, с которой вы начали модификации. Разрешить конфликт можно так, как объяснено в разделе 14.18.

? FILE - Файл с именем **FILE** в вашем рабочем каталоге не соответствует ничему в общем хранилище и не находится в списке файлов, которые следует игнорировать в параметре **-I**. Иными словами, система **CVS** ничего о нём не знает.

Заметим, что не выводится никаких сообщений по поводу ложных каталогов, которые **CVS** просто игнорирует.

14.32.3 Примеры использования команды update

Нижеследующая строка является неплохим примером использования команды **update**, чтобы посмотреть сразу все файлы в текущем рабочем каталоге без какого бы то ни было изменения.

```
cvs -n -q update
```

Это может служить неплохим средством, чтобы понять, что происходит с проектом.

14.33 rtag – добавить символический тег

- Формат использования:

  cvs rtag [-falnR] [-b] [-d] [-r tag|-Ddate] symbolic_tag modules...;
- Требования: **хранилище**.
- Изменения: **в хранилище**.
- Синоним: **rfreeze**.

Вы можете использовать эту команду, чтобы присвоить символические теги (описатели, ярлыки, имена) явно определённым версиям в хранилище. Команда **rtag** работает непосредственно с содержимым хранилища (не требует предварительного выполнения команды **checkout**). Для присваивания имён версиям файлов в вашем рабочем каталоге следует использовать команду **tag**, а не **rtag**. При этом команду **tag** можно выполнять лишь после команды **checkout**.

Если вы попытаетесь использовать имя тега, которое уже используется, **CVS** будет диагностировать такую попытку, но ничего не изменит. Если же вы хотите переместить имя тега (назвать этим же именем другую группу файлов), то чтобы изменения имели место надо использовать параметр **-F**.

14.33.1 Параметры команды rtag

Ниже приведён стандартный набор параметров, который поддерживается командой **rtag**.

-D *date* - Присвоить тег наиболее свежей версии, но не позже чем дата *date*.

-f - Полезно только с параметрами **-D** *date* или **-r** *tag*. Если не найдено подходящих версий, то используется наиболее свежая версия.

-F - Переприсвоить существующий тег с тем же именем другой версии.

-l - Локально; выполнять лишь в текущем рабочем каталоге (без подкаталогов).

-n - Не запускать никакую теговскую программу, которая была определена параметром **-t** в файле **modules**.

-R Обновить каталоги рекурсивно. Это есть умолчание.

-r *tag* Присвоить символический тег только тем файлам, которые содержат тег с именем *tag*. Этот параметр может быть использован, чтобы переименовать тег. Иными словами, присвоить символическое имя (тег) лишь тем именам файлов, которые были отмечены старым тегом *tag*, а затем удалить старый тег.

В дополнение к общим параметрам имеются специфические параметры, которые понимает команда **rtag**.

-a Этот параметр используется для того, чтобы **rtag** смотрел в файл `Attic` для нахождения удалённых файлов, которые содержат определённый тег. Тег удаляется из этих файлов, так что это позволяет продолжать использовать символический тег по мере продолжения разработки, т.е. файлы удаляются из будущих поставок (distribution).

-b Обозначить данным тегом новую ветвь в дереве версий.

-d Удалить тег.

В общем, теги (часто символические имена различных вариантов поставок) не должны удаляться. Тем не менее, **-d** используется, чтобы удалить целиком старую версию или ошибочно помеченную версию.

14.34 tag – добавить тег в рабочем каталоге

- Формат использования:

  tag [-lR] [-b] [-c] [-d] symbolic_tag [files...]

- Требуется: рабочий каталог, хранилище;

- Изменения: **в хранилище**;

- Синоним: **freeze**.

Следует использовать эту команду, чтобы присвоить символические теги (имена, признаки) в хранилище наиболее свежим версиям ваших рабочих исходных текстов. Теги немедленно присваиваются файлам хранилища, также как в случае команды **rtag**, однако версии определяются опосредованно, по записям **CVS** в вашей рабочей истории файлов (обычно, в рабочем каталоге файл **CVS/Entries**), а не применяются явно.

Одно из использований тегов – записать снимок текущих исходных файлов, когда наступит дата замораживания программ, т.е. дата, когда программы конкретного релиза разработчики перестают менять. Поскольку ошибки фиксируются после даты заморозки, то лишь изменённые исходные тексты, которые будут частью нового релиза надо будет пометить заново (ре-тегировать).

Символическими тегами обозначают, какие версии каких файлов были использованы в программной дистрибуции (distribution). Команды **checkout** и **update** позволяют вам извлечь точную копию помеченного тегом релиза в любое время в будущем, несмотря на то, что файлы могли быть модифицированы, добавлены или удалены с того момента, когда релиз был помечен тегом.

Кроме того, данная команда может быть использована, чтобы удалить символический тег или чтобы создать новую ветвь дерева версий.

Если вы попытаетесь использовать имя тега, который уже существует, то **CVS** выдаст диагностику, но ничего не изменит. Использованием параметра **-F** можно преодолеть данное правило.

14.34.1 Параметры команды tag

Ниже приведены стандартные параметры, которые поддерживаются командой **tag**.

-F Присвоить существующий тег с тем же самым именем другой версии.

-l Локально; выполнять только в текущем каталоге.

-R Обрабатывать подкаталоги рекурсивно. Это есть умолчание.

Имеется несколько дополнительных параметров.

-b Создать тег ветви дерева версий. Это позволяет выполнять параллельную полностью изолированную разработку. Такая возможность очень удобна для создания заплат (patch) к ранее выпущенным релизам программного обеспечения.

-c Проверить, что все файлы, которые будут помечены тегом, являются немодифицированными. Это может оказаться полезным, чтобы гарантировать, что вы можете реконструировать текущее содержание файлов.

-d Удалить тег. Если вы используете **cvs tag -d symbolic_tag**, то символический тег сименем **symbolic_tag** будет удалён.

> **Предупреждение:** будьте очень внимательны; удаление тега вызывает удаление без возврата части информации, которая позже может оказаться весьма важной.

14.35 release – освободить рабочий каталог

- Формат использования:

  cvs release [-d] directories...

- Что требуется: **рабочий каталог**.

- Где производятся изменения: **рабочий каталог, журнал истории**.

Команда **release** сообщает системе **CVS**, что вы намерены освободить рабочий каталог. Если рабочий каталог освобождён, то его можно удалить без потери информации. Команда **release** не выполняет никаких реальных удалений файлов или других данных, а только проверяет состояние вашего рабочего каталога. Иными словами, **CVS** по этой команде сообщит вам может ли ваш рабочий каталог быть освобождённым без потери информации. Эта команда означает нормальное окончание цикла редактирования и изменения содержимого рабочего каталога. Иначе, команда информирует систему **CVS**, что цикл работ, начатый командой

```
cvs checkout ...
```

завершён. Рекомендуется всегда выполнять команду **release** перед тем, как удалять рабочий каталог или надолго прерывать работу с ним. Очевидно, что если каталог не удалялся, то после перерыва в работе полезно выполнить команду **update**.

Команда **release** проверяет есть ли в рабочем каталоге изменённые файлы, которые не сохранены в хранилище. Предполагается, что команда **release** выполняется точно над вашим рабочим каталогом, т.е. выше по дереву каталогов.

14.35.1 Параметры команды release

Команда **release** поддерживает один параметр **-d**. Использование параметра означает: удалить рабочую копию того файла, который успешно проверен (т.е. копия в хранилище совпадает с рабочей копией). Если параметр **-d** не использован, то все файлы остаются в рабочем каталоге без изменений.

Предупреждение. Команда **cvs release** удаляет каталоги и файлы рекурсивно. Это имеет тот побочный эффект, что если вы создали новый каталог внутри вашего рабочего каталога, но не добавили его к хранилищу посредством команды **add**, то он будет удалён без диагностики, даже если каталог не пуст.

14.35.2 Вывод release

До того как команда **release** освободит ваши исходные тексты, она будет выводить сообщения для каждого имени файла, который изменён по сравнению с тем же файлом в хранилище.

Предупреждение. Любые новые каталоги созданные внутри рабочего каталога, но не добавленные к хранилищу командой **add**, будут просто игнорироваться. А если был установлен параметр **-d**, то они будут удалены без всякой диагностики, даже если они содержали файлы.

U FILE | P FILE - Существует более свежая версия этого файла в хранилище, а вы не модифицировали вашу локальную копию этого файла.

C FILE - Имя файла **FILE** конфликтное.

A FILE - Файл с именем **FILE** был добавлен в вашем рабочем каталоге, но не сохранён в хранилище с помощью команды **commit**. Если теперь файл будет удалён, то он будет потерян.

R FILE - Файл с именем **FILE** удалён из вашего рабочего каталога, но не удалён из хранилища, так как удаление не было подтверждено командой **commit**.

M FILE - Файл с именем **FILE** модифицирован в вашем рабочем каталоге.

? FILE - Файл с именем **FILE** находится в рабочем каталоге, но система **CVS** ничего о нём не знает. Если вы удалите ваш рабочий каталог, то этот файл будет потерян.

14.35.3 Примеры

Освободить модуль и удалить ваш рабочий каталог.

```
cd ..  # Вы должны стать точно над вашим рабочим каталогом
cvs release -d tc # В ответ вы получите такие сообщения
You have [0] altered files in this repository.
Are you sure you want to release (and delete) module ‘tc’: y
```

Ответив *y*, вы освободите файлы рабочего каталога и он будет удалён.

14.36 Подстановка ключевых слов

Пока вы редактируете исходные тексты внутри вашей рабочей копии модуля вы в любой момент сможете определить состояние файлов посредством команд **cvs status** и **cvs log**. Однако, вскоре после того, как вы экспортируете файлы из вашего разработческого окружения и настроек, вы обнаружите, что стало труднее определить с какой версией модуля вы работаете.

Чтобы как-то помочь идентифицировать исходные тексты, **CVS** позволяет использовать механизм, который называется ПОСТАНОВКА КЛЮЧЕВЫХ СЛОВ. Строки вида **$KEYWORD$** и **$KEYWORD:...$** внутри файла будут заменены строками вида **$KEYWORD:VALUE$**, когда вы получаете новую версию файла.

14.36.1 Список ключевых слов

Ниже приведён список ключевых слов.

$Author$ Имя пользователя (имя для login), кто ввёл данную версию в хранилище.

$Date$ Дата и время, когда данная версия была внесена в хранилище.

$Header$ Стандартный заголовок, который содержит полное имя файла **RCS**, номер версии, дату, имя пользователя, состояние, кто закрыл файл (если он закрыт). Обычно файлы не будут закрыты, если вы используете **CVS**.

Id То же самое, что **$Header$** исключая то, что имя файла будет неполным.

$Name$ Имя тега, который использовался, чтобы взять этот файл (т.е. для выполнения команды **checkout**).

$Locker$ Имя пользователя (имя login), который закрыл файл.

Log Комментарий, который использовался во время выполнения команды **commit**, которому предшествует стандартный заголовок. Существующие комментарии не замещаются. Вместо этого, новый комментарий вставляется после строки

```
% $L o g: CVS_Key.tex,v $}
```

Обратите внимание, в строке слово **Log** написано с разрядкой, чтобы **CVS** не пыталась подставить значение в данном месте. Каждая новая строка начинается с одной и той же последовательности, которой предшествует ключевое слово `%{\bf \$Log}`.

Например, если файл содержит

```
%  $Log: Chap_14.tex,v $
%  Revision 1.12  1999/11/28 22:45:16  shevel
%  corr
%
%  Revision 1.11  1999/11/14 10:49:05  shevel
%  corr
%
%  Revision 1.10  1999/11/11 17:16:28  shevel
%  Updated with adding the longtables.
%
%  Revision 1.9  1999/11/09 14:01:08  shevel
%  corrections
%
%  Revision 1.8  1999/11/05 15:48:19  shevel
%  corr
%
%  Revision 1.7  1999/11/04 16:24:29  shevel
%  corr
%
%  Revision 1.6  1999/11/03 16:04:40  shevel
%  Added figures and changed minor .
%
%  Revision 1.5  1999/11/03 14:13:52  shevel
%  all pict
%
%  Revision 1.4  1999/11/03 11:22:30  shevel
%  Pict 07 i 08
%
%  Revision 1.3  1999/11/03 07:37:18  shevel
%  corr
%
%  Revision 1.2  1999/11/01 20:28:54  shevel
%  Corr
```

```
%
% Revision 1.1  1999/10/21 12:31:40  shevel
% New
%
% Revision 1.10  1999/05/18 09:29:41  shevel
% Minor
%
% Revision 1.9  1999/05/18 09:07:21  shevel
% Minor
```

Дополнительные строки, которые добавляются, когда подставляется значение, **$Log** будут начинаться со знака процента. В противоположность прежним версиям **CVS** и **RCS** "начало комментария"из файла **RCS** не используется. Ключевое слово **$Log** очень удобно, чтобы аккумулировать протокол изменений исходного текста. Однако, в некоторых случаях это может оказаться непростым.

$RCSfile$ Имя файла **RCS** без имени пути (т.е. не абсолютное имя).

$Revision$ Номер версии присвоенный данному варианту.

$Source$ Абсолютное имя файла **RCS**.

$State$ Состояние, присвоенное данной версии. Состояние может быть присвоено командой

cvs admin -s.

14.36.2 Использование ключевых слов

Чтобы включить ключевое слово, вы просто включаете соответствующий текст. Например, вставляете строку **Id** в ваш файл и выполняете команду **commit**. Система **CVS** автоматически выполнит соответствующую подстановку как часть выполнения **commit**.

Обычно, включив строку **Id** в исходные файлы, вы получите в результате то, что подставленное значение будет попадать в сгенерированные файлы (готовые к исполнению двоичные коды). Например, если вам необходимо поддерживать исходные тексты компьютерных программ, то вы могли бы использовать какую-то переменную, которая содержала бы эту строку.

Команда **ident**, которая является частью системы **RCS**, можете быть использована, чтобы извлечь ключевые слова и их значение из файла. Это безусловно удобно для текстовых файлов, но ещё важнее для двоичных файлов.

```
$ ident samp.c
samp.c:
     $Id: Chap_14.tex,v 1.12 1999/11/28 22:45:16 shevel Exp $
$ gcc samp.c
$ ident a.out
a.out:
     $Id: Chap_14.tex,v 1.12 1999/11/28 22:45:16 shevel Exp $
```

Существует другая система поддержки версий **SCCS**, которая имеет команду **what**. Команда **what** очень похожа на команду **ident** и используется для тех же целей. Во многих местах нет системы **RCS**, но есть система **SCCS**. Поскольку **what** ищет символы `@(#)`, то легко использовать ключевые слова, которые распознаются обеими системами. Просто используйте ключевые слова **RCS** вместе с "магическими"фразами **SCCS**, как, например, показано ниже:

static char *id="(#) $Id: ab.c,v 1.5 1993/10/19 14:57:32 ceder Exp $";

14.36.3 Обход подстановок

Механизм подстановки ключевых слов не всегда удобен. Так, если вы желаете, чтобы в тексте появилась строка литерально совпадающая с **$A u th o r$**, без того чтобы **CVS** заменила её на что-то вроде `$Author: shevel $`. К сожалению нет возможности селективно отключить часть подстановок. Вы можете отключить только все подстановки сразу, используя параметр **-ko**.

Во многих случаях вы можете избежать использования ключевого слова в исходном тексте, даже если оно должно появиться в финальном варианте текста. Например, исходный вид ключевого слова **$Author$** в данном тексте таков `\$Author\$`.

14.36.4 Режимы подстановки

Для каждого файла имеется умалчиваемый режим подстановки. То же самое верно и для копии файла в рабочем каталоге. Первое устанавливается посредством параметра **-k** в командах **cvs add** и **cvs admin**. Последняя устанавливается посредством параметров **-k** или **-A** в командах **cvs checkout** или **cvs update**. Команда **cvs diff** также имеет команду **-k**.

Имеются следующие режимы подстановки:

-kkv Генерировать подстановку с использованием форм по умолчанию, т.е. `$Revision: 1.12 $` для ключевого слова **Revision**.

-kkvl Почти то же, что **-kkv**, но будет добавляться имя того, что закрыл файл, если файл закрыт. Обычно такой режим не используется с **CVS**.

-kk Генерировать только имена ключевых слов, опуская их значения. Например, для ключевого слова **Revision** будет сгенерировано **$Revision$** вместо `$Revision: 1.12 $`. Эта возможность полезна, когда необходимо сравнить две версии одного файла, поскольку это позволит исключить влияние различия в значениях ключевых слов.

-ko Генерировать старую ключевую строку, представленную в рабочем файле непосредственно перед помещением файла в хранилище. Например, для ключевого слова **Revision** будет сгенерировано **$Revision: 1.1 $** вместо `$Revision: 1.12 $`, т.к. будет использовано значеное при котором файл был помещён в хранилище.

-kb То же что **-ko**, но для некоторых операционных систем (не типа Unix/Linux) может отличаться от **-ko**.

-kv Генерировать только значения ключевых слов. Например, для ключевого слова **Revision** генерируется строка **1.9** вместо **$Revision: 1.9$**.

Следует помнить, что этот параметр не может быть корректно использован при экспорте двоичных файлов.

14.36.5 Проблемы с ключевым словом Log

Ключевое слово **Log** является до некоторой степени противоречивым. Информация об истории изменения вашего файла всегда доступна, даже если вы не используете ключегого слова **Log** - достаточно использовать команду **cvs log**.

Более серьёзная проблема с данным ключевым словом состоит в том, что **CVS** не слишком успешно может управляться со значениями этого ключевого слова, например, когда вы пытаетесь объединить боковую ветвь и основной ствол разработки. Частым результатом могут оказаться сообщения о конфликтах.

Чтобы избежать проблем, лучше воздержаться от использования ключевого слова **Log**, елси это не является абсолютно необходимым.

14.37 Переменные окружения, которые использует система CVS

Ниже приведён полный список переменных окружения, которые оказывают влияние на фумкционирование системы **CVS**.

CVSIGNORE - список разделённых пробелами шаблонов имён файлов, которые **CVS** должна игнорировать.

CVSWRAPPERS - список разделённых пробелами шаблонов имён файлов, которые **CVS** должна рассматривать в качестве фильтров (wrappers).

CVSREAD - если эта переменная установлена, то команды **checkout** и **update** попробуют записать файлы в ваш рабочий каталоге с правом доступа только чтение. Если переменная **CVSREAD** не установлена, то, по умолчанию, файлы будут записаны с правами доступа чтение и запись.

CVSUMASK Права доступа по умолчанию (маска прав доступа).

CVSROOT эта переменная должна содержать абсолютное имя каталога, в котором находится хранилище **CVS**. Если переменная не установлена, то вы должны во всех командах указывать это имя как значение параметра **-d**. Если использован параметр **-d**, то значение аргумента используется в качестве абсолютного имени каталога, где находится хранилище, даже если установлено значение переменной CVSROOT.

EDITOR, CVSEDITOR определяют программу редактора, которая вызывается во время работы команды **commit**. Переменная CVSEDITOR главнее, чем переменная EDITOR.

PATH - Если переменная RCSBIN не установлена, то используется переменная PATH.

HOME, HOMEPATH, HOMEDRIVE - Эти переменные используется, чтобы найти каталог, в котором находится файл `.cvsrc`. В UNIX **CVS** использует только HOME. В MS/Windows вам, возможно, потребуется установить HOMEPATH и HOMEDRIVE.

CVS_RSH - Эта переменная определяет имя программы, которая используется для соединения с удалённых сервером, если использован метод доступа **:ext:**.

CVS_SERVER используется в режиме клиент-сервер, когда требуется получить доступ к удалённому хранилищу с использованием **rsh**. Значение по умолчанию есть `cvs`.

CVS_PASSFILE используется в режиме клиент-сервер во время доступа с использованием **cvs login server**. Значение по умолчанию есть `$HOME/.cvspass`.

CVS_CLIENT_PORT используется в режиме клиент-сервер, когда доступ производится с использованием системы аутентификации **Kerberos**.

CVS_RCMD_PORT используется в режиме клиент-сервер. Если установлено, то используется как номер порта в демоне **RCMD** на стороне сервера.

CVS_CLIENT_LOG используется для отладки только в режиме клиент-сервер. Если установлено, то все что посылается на сервер записывается в файл `$CVS_CLIENT_LOG.in`, а все что посылается клиенту записывается в файл `$CVS_CLIENT_LOG.out`.

sf CVS_SERVER_SLEEP используется для отладки режима клиент-сервер. Если установлено, то воспринимается как число секунд, на которые должен быть задержан запуск процессов на серверной стороне, чтобы вы успели запустить

отладчик.

CVS_IGNORE_REMOTE_ROOT для чего это?

TMPDIR, TMP, TEMP каталог, в котором **CVS** располагает временные файлы.

14.38 Проблемы при использовании CVS

Если при использовании системы **CVS** возникли проблемы любого вида, то вам полезно ознакомиться с этим разделом.

14.38.1 Сообщения об ошибках

Здесь приведены не все сообщения об ошибках, а лишь часть из них, которая является наиболее часто встречается или потенциально сложные для интерпретации. Сообщения упорядочены по алфавиту текста сообщения, а не его идентификатора. Например,

cvs COMMAND: authorization ...

здесь текст сообщения начинается со слова **authorization**.

cvs COMMAND: authorization failed: server HOST rejected access Это общий ответ на попытку соединения к серверу **pserver**, который решил не детализировать причину отказа в обслуживании. Полезно проверить **username** и **password**, а также то, что имя сервера в **CVSROOT** допустимо в **-allow-root** в файле **inetd.conf**.

FILE:LINE: Assertion TEXT failed Точный формат этого сообщения может весьма отличаться от системы к системе. Сообщение говорит о том, что в **CVS** встречена ошибка. Как обойти такую неприятность описано в разделе **Ошибки CVS**.

cannot change permissions on temporary directory

Operation not permitted Такое сообщение встречалось на старых версиях Linux RedHat 3.0.3 и 4.1. Неясно что за причина сообщения.

cannot open CVS/Entries for reading:

No such file or directory Означает какую-то внутреннюю ошибку **CVS**.

cvs [init aborted]:

cannot open CVS/Root:

No such file or directory Такое сообщение довольно бессодержательно, поскольку всё выполняется нормально. Оно не должно появляться в версии **CVS** 1.9 и старше.

cvs [checkout aborted]:

cannot rename file FILE to CVS/„FILE:

Invalid argument Такое сообщение может появляться в **CVS** 1.9 в системе **Solaris** 2.5. Причина сообщения неясна.

cvs [COMMAND aborted]:

cannot start server via rcmd К сожалению, это весьма общая диагностика, когда вы запускаете клиент **CVS**, который встречает проблемы при попытке соединиться с сервером. Если вы получили такую диагностику во время локальной работы, то, быть может, вы забыли специфицировать **:local:**, как описано в разделе **Хранилище**.

ci: FILE,v: bad diff output line:

Binary files - and /tmp/T2a22651 differ **CVS** 1.9 и старше будут выдавать такую диагностику, когда в хранилище вносится (CHECK IN) двоичный файл, если **RCS** установлена некорректно.

cvs checkout: could not check out FILE **CVS** 1.9, это может означать, что программа **co** (часть системы **RCS**) завершилась ненормально. Если этому сообщению не предшествует никакое другое сообщение об ошибке, то скорее всего вы имеете дело с ошибкой **CVS**.

cvs [login aborted]:

could not find out home directory Это означает, что не установлены переменные окружения **HOMEDRIVE**, **HOMEPATH** или **HOME**.

cvs update: could not merge revision REV of FILE:

No such file or directory **CVS** 1.9 и старше будет выводить это сообщение, если не найдена программа **rcsmerge**. Установите переменную окружения **PATH** в правильное значение. Можете также установить последнюю версию **CVS**, которая не требует внешней программы **rcsmerge**.

cvs update: could not patch FILE; will refetch Это означает, что по каким-то причинам клиент был неспособен использовать **patch**, посланный сервером. Сообщение почти ничего не означает, поскольку лишь замедлится часть операций, но не изменит содержания того, что делает **CVS**.

dying gasps from SERVER unexpected При появлении такого сообщения просто повторите операцию.

end of file from server

(consult above messages if any) Наиболее общая причина данного сообщения – это завершение программы **rsh** с ошибкой. Возможно, что имеют место неточности в конфигурировании клиента или сервера с использованием протокола **rsh**.

cvs commit: Executing 'mkmodules' Это означает, что ваше хранилище было установлено до версии **CVS** 1.8. В версиях **CVS** 1.8 или старше этому сообщению будет предшествовать другое сообщение

cvs commit: Rebuilding administrative file database

Если вы увидели оба сообщения, то база данных перестраивается дважды. Это не является необходимым, но и не наносит вреда. Тем не менее, если вы хотели бы избежать этого, то удалите **-i mkmodules** везде, где встречается в вашем файле **modules**.

missing author Обычно это может иметь место если вы создали **RCS** файл, когда ваше имя пользователя **username** не было установлено. Решением может быть установить верное значение **username** и создать файл заново.

***PANIC* administration files missing** Обычно это означает, что имеется каталог с именем **CVS**, но он не содержит административных файлов, которые **CVS** туда помещает. Если вы создали каталог с таким именем случайно, то переименуйте его. Если нет, то возможно вы встретили ошибку **CVS**.

rcs error: Unknown option: -x,v/ Это сообщение будет следовать за сообщением **RCS**. Это означает, что вы имеете очень старую версию **RCS**. **CVS** использует **RCS** версии 5 или выше.

cvs [server aborted]:

received broken pipe signal Похоже, что такая диагностика имеет место уже после выполнения запрошенных операций. Таким образом, вы можете её игнорировать.

Too many arguments! Может выдаваться при установке. Выдаёт программа **log.pl**. Если вы в ней не нуждаетесь, то можете закомментировать её. Проверьте также административный файл **loginfo**.

cvs commit: Up-to-date check failed for FILE Это означает, что кто-то другой изменил файл, изменения в котором вы пытаетесь подтвердить (**commit**). Это означает, что сначала вам следует выполнить операцию **update**, а потом повторить **commit**. Если во время **update** возникли конфликты, вам следует устранить их вручную.

14.39 Разделение доступа к файлам в системе CVS

Для получения представления как работают блокирующие средства в CVS полезно посмотреть раздел **Замечания по одновременному использованию**. В данном разделе мы ориентируемся на лиц, которые разрабатывают программные средства, имеющие доступ к хранилищу **CVS** предотвараща одновременно доступ других программ к хранилищу.

Любой файл в хранилище с именем начинающимся на **#cvs.rfl** является блокиратором чтения. Любой файл в хранилище с именем начинающимся на **#cvs.wfl** является блокиратором записи. Каталог с именем начинающимся на **#cvs.lock** служит главным блокиратором. Иными словами, необходимо иметь этот каталог прежде, чем создавать любые блокираторы.

Например, надо создать блокиратор по чтению. Сначала мы создаём каталог **#cvs.lock**. Если такой файл уже существует, то следует подождать немного, а потом снова попытаться создать этот же каталог. После того как каталог **#cvs.lock** создан, пробуем создать файл, чьё имя начинается на **#cvs.rfl**, за которым следует, например, имя хоста или номер процесса. После создания файла можно удалить главный блокиратор (каталог **#cvs.lock**), чтобы дать возможность другим процессам тоже создавать блокираторы. Затем вы читаете хранилище. Когда чтение закончено, то файл, который вы создали ранее (начинающийся на **#cvs.rfl**), тоже должен быть удалён, чтобы разрешить чтение другим процессам или пользователям.

Чтобы организовать блокиратор по записи, вам необходимо сначала создать главный блокиратор как при чтении. Когда главный блокиратор создан, то надо проверить есть ли блокиратор чтения (файл, имя которого начинается на **#cvs.rfl**). Если такие файлы есть, то следует удалить главный блокиратор и подождать какое-то время, а затем попробовать всё снова.

Если файлов типа **#cvs.rfl** нет, то создать файл, начинающийся комбинацией **#cvs.wfl**, за которой следует информация по вашему выбору (как в случае с блокиратором чтения). Оставить каталог (не удалять) главного блокиратора (**#cvs.lock**. Записывайте данные в хранилище. Когда операция записи закончится, то удалите сначала файл **#cvs.wfl**, а затем каталог `#cvs.lock`.

Заметим, что каждый блокиратор (на чтение или на запись) блокирует соответствующую операцию лишь с одним подкаталогом хранилища, не включая наследников или родителей. Если вам потребуется блокировать все дерево каталогов, то необходимо будет блокировать все подкаталоги. Естественно, после завершения операции, все подкаталоги следует разблокировать. Если при блокировании всего дерева каталогов оказалось, что какие-то подкаталоги блокированы кем-то ещё, то вам следует всё уже заблокированное разблокировать, а затем, после некоторого ожидания, снова попытаться блокировать всё дерево. Это следует делать во избежание взаимного блокирования (мёртвого блокирования или клинча).

CVS предполагает блокирование по записи отдельных файлов типа `*,v`.

Глава 15

Программа перекачки файлов из Интернет (Wget)

15.1 Введение

GNU Wget – свободно распространяемая программа, которая позволяет копировать файлы из Интернет на локальный хост с использованием протоколов *HTTP* и *FTP*, как наиболее распространённых протоколов, используемых в Интернет. Программа имеет много полезных свойств, среди которых можно отметить:

- **Wget** является неинтерактивной программой, иными словами она может работать в фоновом режиме даже тогда, когда пользователь не логирован в системе. Таким образом, вы можете запустить **Wget**, например, на ночь, а утром посмотреть результаты работы программы.

- **Wget** способна копировать файлы с удалённого сервера рекурсивно, следуя структуре *HTML* документов и деревьев каталогов *FTP*, создавая локальную копию иерархии каталогов имеющейся на удалённом сервере. Это свойство может быть использовано, чтобы зеркально скопировать архивы и домашние страницы, или перемещаться со страницы на страницу для поиска каких-либо данных, моделируя поведение поискового робота. Программа **Wget** *понимает* соглашения для файлов `/robots.txt`.

- Расширения имён файлов и рекурсивное копирование каталогов имеет место, когда используются протоколы *FTP*. **Wget** может воспринимать информацию о времени создания файлов на удалённых серверах, которые поставляют *HTTP* и *FTP* серверы, и записывать эту информацию локально. Таким образом, **Wget** имеет возможность *видеть* изменился ли файл на удалённом сервере с момента предыдущего просмотра и копировать лишь те файлы, которые изменились. Это позволяет использовать **Wget** в качестве средства зеркального копирования содержания удалённых серверов.

- **Wget** оказывается очень полезной на медленных или нестабильных линиях связи, повторяя запрос на еще не переданные части документов до тех пор, пока не будут скопированы все запрошенные файлы или пока не будет превышен лимит повторов, установленный пользователем. Программа пытается продолжить копирование файлов с точки, где ранее произошло прерывание.
- **Wget** позволяет использовать, как прокси-серверы (proxy), так и охранные серверы (firewalls).
- Встроенные механизмы позволяют настроить программу на поиск тех типов линков (ссылок), которые вам нужны.
- Процесс передачи данных отображается на экране (или в файле протокола) выводом точек. Каждая точка обозначает 1 КБ. Однако, вы можете сами установить какие символы и что будут обозначать. Большая часть возможностей программы может быть изменена, как с помощью параметров, так и с помощью специального файла для индивидуальной настройки `.wgetrc`, который находится в вашем главном каталоге. Вы можете поддерживать глобальный специальный конфигурационный файл `/usr/local/etc/wgetrc`.
- **Wget** оказывается очень полезным средством и для локального использования (в локальной сети предприятия). Так, если вам необходимо скопировать каталог отдельного пользователя или часть каталога, то это удобнее сделать посредством **Wget**.

15.2 Вызов программы Wget

Вызов программы **Wget** очень прост:

wget [OPTION]... [URL]...

По умолчанию программа передаёт файлы, указанные в *URL*, с удалённого сервера на вашу машину (*URL* означает *Uniform Resource Locator* или Универсальный Указатель Ресурсов). Однако, вы можете вносить коррекции в поведение программы путём изменения параметров программы. Сделать это можно двумя способами: постоянно, введя изменения в файл `.wgetrc` или определяя изменения в командной строке.

15.2.1 Синтаксис параметров Wget

Поскольку **wget** использует **getopts** для анализа параметров, то каждый параметр может быть представлен в двух формах: короткой и длинной. Короткая форма удобнее для ввода, но длинную легче запомнить. Можно использовать и смесь обоих форм и даже определять параметры после поля аргументов. таким образом вы можете писать:

```
wget -r -tries=10 http://fly.cc.fer.hr/ -o log
```

Пробелы между параметрами и их значениями могут быть опущены. Например, вместо *-o log* можно писать *-olog*. Вообще, все соглашения и правила действующие для команды **getopts** являются верными для **Wget**.

В нижеследующих описаниях значений параметров будет использоваться знак | (вертикальная черта), которая будет разделять возможные варианты написания одного и того же параметра.

15.3 Параметры Wget

15.3.1 Общие параметры Wget

-V | **-version** вывести на экран версию программы **Wget**.

-h | **-help** вывести на экран информацию о всех параметрах, которые могут быть использованы в командной строке.

-b | **-background** перейти в фоновый режим сразу после старта программы. Если не был указан файл протокола работы программы с помощью параметра **-o**, то протокол будет записан в файл **wget–log**. Нетребуется никаких дополнительных команд, например, *nohup*, чтобы запустить **Wget** вфоновом режиме и сразу освободить терминал для дальнейшего ввода или для выхода из системы.

-e *command* | **-execute** *command* выполнить команду *command* перед началом работы **Wget** как будто *command* является частью файла `.wgetrc`. Следует заметить, что *command* будет выполнена **после** команд из `.wgetrc`.

15.3.2 Параметры управления протоколированием выполнения Wget

-o *logfile* | **-output-file**=*logfile* записывать все сообщения в файл с именем *logfile*. Обычно такие сообщения направляются в стандартный файл сообщений об ошибках.

-a *logfile* | -**append-output**=*logfile* выводить сообщения о выполнении программы в конец файла *logfile.* Если файл с именем *logfile* не существует, то он будет создан.

-d | -**debug** включить отладочный вывод, означающий различную информацию, которая может быть важной для разработчика **Wget**. Если ваш администратор при компилировании программы **Wget** не включил возможность отладки, то, быть может, программа не будет работать. Если же при компилировании такая возможность была включена, то она не оказывает никакого действия, пока не включён параметр **-d**.

-q | -**quiet** Выключить весь вывод программы.

-v | -**verbose** Включить вывод подробных сообщений. По умолчанию — включено.

-nv | -**non-verbose** Выключить вывод подробных сообщений. Однако, выполнение не будет молчаливым -**quiet**. Вывод основной информации и сообщений об ошибках будет продолжаться.

15.3.3 Ввод для программы Wget

Программа **Wget** может иметь вводной файл, в котором печислены *URL*, которые следует скопировать на локальную машину в данном сеансе выполнения.

-i *file* | -**input-file**=*file* Читать список *URL* из файла с именем *file.* В этом случае нет необходимости задавать *URL* в командной строке. Если *URL* имеются в командной строке и в вводном файле, указанном параметром **-i**, то сначала будут обработаны те *URL*, которые находятся в командной строке. Нет необходимости, чтобы входной файл, указанной с помощью **-i**, имел вид *HTML* страницы. Достаточно, чтобы *URL* в файле были представлены в виде простого списка. Однако, вводной файл может иметь формат *HTML* страницы.

Если вы указали параметр -**force-html**, вводной документ будет рассматриваться как *HTML* страница. В этом случае возможны проблемы с относительными линками, которые вы сможете разрешить добавив в документ строку `<base href="URL»` или определив параметр -**base**=**URL** в командной строке.

-F | -**force-html** Когда ввод *URL* производится посредством чтения файла, следует обрабатывать этот файл как файл в формате *HTML*. Это позволяет вам копировать относительные линки в существующих на вашем компьютере файлах *HTML*, добавляя строку `<base href="URL»` к локальному файлу *HTML* или определив параметр -**base=URL** в командной строке.

15.3.4 Общие параметры копирования файлов

-t *number* | -**tries**=*number* Установить число попыток выполнить копирование равным *number*. Значение **0** (нуль) или **inf** означают неограниченное число попыток.

-O *file* | -**output-document**=*file* Копируемые документы не будут записываться отдельно в соответствующие файлы, а все вместе будут записываться в один файл с именем *file*. Если файл *file* уже существует, то он будет целиком перезаписан (старое содержание будет уничтожено). Если *file* есть **-** (знак минус), документы будут выводиться на стандартный вывод. Использование этого параметра устанавливает число попыток копирования равным 1.

-nc | -**no-clobber** Не расширять существующие Проверять существуют ли имена файлов перед тем как они сохраняются в иерархии каталогов во время рекурсивного копирования файлов. Эта возможность полезна, когда вы хотите продолжать копирование после прерывания в процессе копирования многих файлов. Если файлы имеют окончание **.html** или **.htm** они будут загружены с локального диска и обработаны как будто они скопированы из Интернет.

-c | -**continue** Продолжить приём существующего файла. Это полезно, когда вы желаете закончить копирование удалённого файла, копирование которого было начато другой программой или предыдущим запуском **Wget**. Таким образом, вы можете написать:

```
wget -c ftp://sunsite.doc.ic.ac.uk/ls-lR.Z
```

Если в текущем локальном каталоге имеется файл с именем **ls-lR.Z**, **Wget** будет предполагать, что это есть первая порция удалённого файла и запросит удалённый сервер продолжить копирование с байта, который следует за длиной имеющегося файла.

Заметим, что вам не обязательно определять этот параметр, если вы хотите продолжить копирование файла после обрыва связи – **Wget** делает это по умолчанию. Этот параметр нужен в случаях, когда **Wget** был абортирован или когда начало файла было скопировано другой программой, например, **ncftp**.

Без параметра **-c**, команда из предыдущего примера начнёт копировать удалённый файл и даст ему локальное имя **ls-lR.Z.1**. Параметр **-c** приложим и к HTTP серверам, которые поддерживают заголовок **Range**.

-**dot-style**=*style* Установить стиль отображения процесса копирования в *style*. **Wget** обозначает процесс копирования каждого документа выводом точек на экран, где каждая точка соответствует определённому объёму данных. Любое число точек могут быть объединены в *кластеры*, чтобы сделать вывод более выразительным (лёгким для оценки). Этот параметр позволяет вам выбрать один из встроенных стилей, где вы сможете установить число байтов соответствующих одной точке, число точек в кластере, а также число точек в строке.

По умолчанию, стиль **default**, каждая точка соответствует 1K байтов, 10 точек объединены в кластер, а в строке имеется 50 точек. Стиль **binary** имеет 8K байтов на точку, 16 точек в кластере и 48 точек в строке (384K на строку). Стиль **mega** подходит для очень больших пересылок – каждая точка соответствует 64K, восемь точек на кластер и 48 точек в строке, итого, 3МБ на строку. Стиль **micro** хорош для маленьких файлов: 128 байтов на точку, 8 точек на кластер и 48 точек на строку (6K на строку).

-N | -**timestamping** Включить использование отметок времени в оглавлении каталогов в процессе копирования.

-S | -**server-response** Напечатать на экране заголовки, посланные HTTP серверами и ответы, посланные FTP серверами.

-**spider** Когда программа вызвана с этим параметром, то **Wget** ведет себя как **spider**, т.е. никакие документы не загружаются на локальные диски, а только проверяется, что удалённые документы существуют. Это свойство можно использовать для проверки файла `bookmarks.html`:

```
wget -spider -force-html -i bookmarks.html
```

Эта особенность требует значительно больше работы от **Wget**, чтобы вести себя функционально идентично реальному роботу типа **spider**.

-T | -**timeout**=*seconds* Установить максимальное ожидание до фактического начала операции **чтение** равным значению *seconds* секунд. Когда выдана сетевая операция чтения, файловый описатель проверяется на превышение времени ожидания. В противном случае могло бы остаться зависшее **чтение**. По умолчанию максимальное время ожидания составляет 900 секунд. Установка **timeout** равным 0 выключает проверку.

Пожалуйста не выключайте проверку на **timeout**, если вы не понимаете отчетливо, что вы делаете.

-w *seconds* | -**wait**=*seconds* Ждать определённое количество секунд (*seconds*) между попытками чтения документов. Использование этого параметра можно рекомендовать, чтобы разгрузить удалённый сервер или подождать, пока сервер снова станет доступным. Вместо секунд можно определить интервал в минутах используя суффикс **m**, в часах использую суффикс **h**, или в днях использую суффикс **d**.

Использование больших значений полезно, если сеть или сервер вышел из строя. Таким образом, **Wget** может ждать такое время, за которое сеть или сервер будут приведены в рабочее состояние.

-Y on/off | -**proxy**=**on/off** Включить прокси или выключить прокси. Прокси включено по умолчанию, если определена соответствующая переменная окружения.

-Q *quota* | -**quota**=*quota* Определить квоту для копируемых документов. Значение квоты может быть определено в байтах (по умолчанию), килобайтах (суффикс **k**), или мегабайтах (суффикс **m**).

Заметим, что квота никогда не действует на одиночный файл. Так что, если вы определили

```
wget -Q10k ftp://wuarchive.wustl.edu/ls-lR.gz
```

весь файл `ls-lR.gz` будет передан и записан на диск, даже если его объём оказался намного больше 10КБ. То же самое будет, если несколько *URL* определены в командной строке. Однако, квота будет принята во внимание, когда имеет место рекурсивный поиск и передача файлов или, если *URL* прочитаны из вводного файла. Так, вы можете предусмотрительно записать

```
wget -Q2m -i sites
```

передача и запись файлов будет завершена, если квота 2МБ будет превышена.

Установив значение квоты в 0 или **inf** вы отключите проверку на квоту.

15.3.5 Параметры создаваемых каталогов

-nd | **-no-directories** Не создавать иерархию каталогов копируя удалённую структуру. Если этот параметр включён, то все файлы будут сохранены в текущем каталоге без расширения имён (если имя появляется более, чем один раз, то имя файла получит окончание *.n*, где *n* есть целое.

-x | **-force-directories** Создать иерархию каталогов (противоположно параметру **-nd**), даже если иерархия не была бы создана в другом случае (без использования параметра **-x**). Например,

```
wget -x http://fly.cc.fer.hr/robots.txt
```

сохранит загружаемый файл под именем `fly.cc.fer.hr/robots.txt`.

-nH | **-no-host-directories** Запретить генерацию каталогов с именами хостов. По умолчанию **Wget** вызывается с параметром

```
wget -r http://fly.cc.fer.hr/
```

и будет создавать каталожную структуру начиная с каталога с именем `fly.cc.fer.hr/`.

-**cut-dirs**=*number* Игнорировать *number* компонентов иерархии каталогов удалённого сервера. Параметр используется, чтобы реализовать точное управление процессом создания каталогов на локальной машине во время рекурсивного копирования с удалённого сервера. Например,

```
wget -r ftp://ftp.xemacs.org/pub/xemacs/
```

сохранит все файлы в иерархии каталогов, где верхний каталог имеет имя `ftp.xemacs.org/pub/xemacs/`. В то время как параметр **-nH** может удалить часть *ftp.xemacs.org/*, у вас останется каталожная структура `pub/xemacs`. Это тот случай когда становится ясной роль параметра -**cut-dirs**: с его использованием **Wget** пропускает на удалённом сервере *number* каталожных уровней (естественно, сверху). Ниже приведены несколько примеров как действует параметр -**cut-dirs**:

```
нет управления -> ftp.xemacs.org/pub/xemacs/
-nH            -> pub/xemacs/
-nH –cut-dirs=1 -> xemacs/
-nH –cut-dirs=2 -> .
```

-cut-dirs=1 -> ftp.xemacs.org/xemacs/
...

Если желаете только прочесть каталожную структуру, то это подобно комбинации **-nd** и **-P**. Однако, в противоположность **nd**, параметр -**cut-dirs** не пропускает подкаталоги. Например, с использованием комбинации **-nH** -**cut-dirs=1**, подкаталог `beta/` будет помещен, как можно ожидать, в `xemacs/beta`.

-P *prefix* | -**directory-prefix**=*prefix* Установить *каталожный префикс* в значение *prefix*. Здесь *каталожный префикс* означает каталог, где будут записаны все другие файлы и подкаталоги, т.е. вершина каталожного дерева. Умолчание есть "."(точка), т.е. текущий каталог.

15.3.6 Параметры копирования файлов с использованием протокола *HTTP*

-**http-user**=*user*

-**http-passwd**=*password* определить имя пользователя как *user* и пароль как *password* на *HTTP* сервере. В соответствии с типом обращения, **Wget** будет кодировать эти параметры с использованием одной из схем аутентификации: **basic** (не секретная) и **digest**.

Другим способом определения имени пользователя и пароля является определение этих аргументов в самом *URL*.

-C | -**cache=on/off** Когда установлен в **off** – выключает кэш на стороне сервера. В этом случае **Wget** посылает удалённому серверу соответствующую директиву (**Pragma: no-cache**), чтобы получить файл с удалённого сервера, а не выдать сохранённую в кэше копию. Это полезно использовать для документов, которые относительно часто меняются, чтобы избежать получения устаревшей копии из прокси сервера.

По умолчанию -**cache=on**.

-**ignore-length** К несчастью, некоторые *HTTP* серверы (точнее CGI программы) посылают странные заголовки `Content-Length`, которые вводят программу **Wget** в заблуждение, что не весь ещё документ прочитан. Вы можете опознать такое поведение **Wget**: программа запрашивает один и тот же документ снова и снова, сообщая каждый раз, что соединение прервалось на том же самом байте (всё остальное в норме).

С данным параметром **Wget** будет игнорировать заголовок `Content-Length` как будто его нет.

-**header**=*additional-header* Определить дополнительный заголовок *additional-header*, чтобы пройти к *HTTP* серверу. Заголовки должны содержать **:** (двоеточие), за которым следуют несколько символов (не пробелов) и не должны содержать символ **newline**.

Вы можете определить более, чем один заголовок с использованием параметра -**header** более, чем один раз.

```
wget --header='Accept-Charset: iso-8859-2' \
     --header='Accept-Language: hr'        \
       http://fly.cc.fer.hr/
```

Определение пустой строки в качестве заголовка очистит все предыдущие заголовки, определённые пользователем.

-**proxy-user**=*user*

-**proxy-passwd**=*password* Определить имя пользователя как *user* и пароль пользователя как *password* для аутентификации на прокси сервере. **Wget** будет кодировать эти параметры с использованием схемы **basic**.

-s | -**save-headers** Сохранить заголовки, посланные *HTTP* сервером, в файле перед данными, разделив заголовок и данные пустой строкой.

-U *agent-string* | -**user-agent**=*agent-string* Определить *agent-string* для HTTP сервера.

Протокол *HTTP* разрешает клиенту идентифицировать себя с использованием поля заголовка *User-Agent*. Это позволяет распознавать какое использовалось программное обеспечение, для статистических целей или для целей отладки. **Wget** идентифицирует себя как *Wget/version*, где *version* есть текущий номер версии программы **Wget**.

Однако, как известно некоторые серверы выдают информацию в зависимости от значения поля *User-Agent*. Концептуально в этом ничего нет плохого, тем не менее печально, если сервер не выдаёт никакой информации клиентам, которые не идентифицируют себя, например, как `Internet Explorer` или `Mozilla`. Этот параметр позволяет вам сменить поле *User-Agent*, которое выдаёт **Wget**. Использование такой возможности трудно приветствовать, но вы знаете что делаете.

Заметим попутно, что компания Netscape Communications Corp. настояла на том, что использование значения поля *User-Agent* равное `Mozilla` является знаком авторских прав и его несанкционированное использование может преследоваться по закону. **Не используйте** в параметре *User-Agent* программы **Wget** значение `Mozilla`.

15.3.7 Параметры копирования файлов с использованием протокола *FTP*

-retr-symlinks Произвести чтение символического линка на удалённом сервере так, как будто это обычный файл. Иными словами не создавать символического линка локально.

-g on/off | -glob=on/off Разрешает и запрещает расширение имён файлов в соответствии с метасимволами типа `*` (звёздочка), `?` (знак вопроса), `[` и `]` (открывающая и закрывающая квадратные скобки), чтобы расширить число файлов переписываемых с удалённого сервера. Например,

```
wget ftp://glueck.cern.ch/*.msg
```

По умолчанию установлено **-g on**, т.е. включено расширение.

Возможно вы должны будете заключить ваши *URL* в кавычки, чтобы защитить их от автоматического расширения в вашей оболочке **UNIX**. Когда возможность расширения имён включена, **Wget** просматривает оглавление каталога, который зависит от типа операционной системы. Это определяет, что данная возможность успешно работает с **FTP** серверами под управлением диалектов **UNIX**.

-passive-ftp Использовать *пассивную* схему **FTP** соединения, в которой клиент инициирует соединение. Это временами требуется для **FTP**, чтобы успешно работать через охранные серверы (firewalls).

15.3.8 Параметры рекурсивного поиска и копирования файлов

-r | -recursive Включить рекурсивный просмотр каталогов и подкаталогов на удалённом сервере.

-l *depth* | **-level**=*depth* Определить максимальную глубину рекурсии равной *depth* при просмотре каталогов на удалённом сервере. По умолчанию *depth*=5.

-delete-after Этот параметр заставляет **Wget** удалять с локальной машины каждый одиночный файл **сразу после** копирования этого файла с удалённого сервера. Это может оказаться полезным для того, чтобы загрузить прокси сервер какой-то популярной страницей, например,

```
wget -r -nd -delete-after http://www.cern.ch/~popular/page/
```

здесь **-r** – рекурсивный просмотр каталогов; **-nd** – не создавать каталогов на локальной машине.

-k | **-convert-links** Превратить абсолютные линки в *HTML* документе в относительные линки. Преобразованию подвергнутся только те линки, которые указывают на реально загруженные страницы; остальные не будут преобразовываться.

Заметим, что лишь в конце работы **Wget** сможет узнать какие линки были реально загружены. Следовательно, лишь в конце работы **Wget** будет выполняться окончательное преобразование.

-m | **-mirror** Включить режим зеркального отображения. Этот параметр включает рекурсию, временные отметки и поддерживает расширение имён. Иными словами, устанавливается **-r -N -l inf -nr**. Здесь **-r** – рекурсия по каталогам; **-N** – включить отметки времени создания файла; **-l inf** – глубина рекурсии по просмотру каталогов и подкаталогов не ограничена; **-nr** – не удалять временные файлы типа `.listing`, которые генерирует **Wget** при *FTP* поиске. Обычно, такие файлы содержат оглавления каталогов на удалённых *FTP* серверах. Сохранив такие файлы, позже их можно использовать для целей отладки. Кроме того, часто удобно иметь оглавление удалённого FTP сервера.

15.3.9 Параметры управления просмотром удаленных серверов

-A *acclist* | **-accept** *acclist*

-R *rejlist* | **-reject** *rejlist* Определить разделённые запятыми списки суффиксов имён файлов или шаблонов, чтобы принять или отвергнуть копирование файлов. Подробнее смотрите раздел 15.4.3.

-D *domain-list* | **-domains**=*domain-list* Определить приемлемые имена доменов и имена просматриваемых серверов *DNS*. Здесь *domain-list* есть список, разделённый запятыми. Заметим, что данная возможность не включается в **-H**. Использование данной возможности ускоряет работу, даже если если имеется в виду один хост. Подробнее смотрите раздел 15.4.2.

-**exclude-domains** *domain-list* Исключить имена доменов, перечисленные в *domain-list* из списка просматриваемых на серверах *DNS*.

-L | -**relative** Читать лишь те файлы на удалённом сервере, на которые указывают только относительные линки. Файлы с абсолютными линками не будут приниматься во внимание, даже если они находятся на том же сервере.

-**follow-ftp** Читать все FTP линки, которые имеются в HTML документе. Без этого параметра **Wget** будет игнорировать все FTP линки.

-H | -**span-hosts** Разрешить переходы с хоста на хост, когда выполняется рекурсивный поиск.

-I *list* | -**include-directories**=*list* Определить разделённый запятыми список каталогов, которые вы желаете переписать на свою машину. Элементы списка *list* могут содержать метасимволы.

-X *list* | -**exclude-directories**=*list* Определить разделённый запятыми список каталогов, которые вы не желаете копировать на свою машину. Элементы списка *list* могут содержать метасимволы.

-nh | -**no-host-lookup** Выключить занимающий время просмотр *DNS* почти всех хостов (смотрите раздел 15.4).

-np | -**no-parent** Не переходить в родительский каталог во время поиска файлов. Это очень полезное свойство, поскольку оно гарантирует, что будут копироваться только те файлы, которые расположены **ниже** определённой иерархии. Дополнительные детали могут быть найдены в 15.3.5.

15.4 Рекурсивный поиск и копирование файлов

GNU Wget способна проходить части мировой паутины (или одиночный *HTTP* или *FTP* сервер) следуя линкам имеющимся в документах и структуре каталогов. Это называется в данном случае **рекурсивным** поиском или просто **рекурсией**.

Wget разыскивает и анализирует страницу *HTML* данного *URL*, находя файлы и *HTML* документы, на которые имелись ссылки посредством тегов (или описателей) типа **href** или **src**. Если вновь скопированный файл также оказался типа **text/html**, он будет проанализирован и, если в нём оказались ссылки, поиск будет продолжен.

Максимальная **глубина**, на которую может распространяться поиск определяется параметром **-l** (по умолчанию максимальная глубина равна 5). Смотрите также раздел 15.3.8.

Когда производится рекурсивный поиск с использованием протокола *FTP*, **Wget** будет копировать данные из данного дерева каталогов (включая все подкаталоги на заданную глубину) на удалённом сервере, создавая локальную зеркальную копию. Просмотр *FTP* серверов также ограничен параметром **-l**.

По умолчанию, **Wget** будет создавать локальную копию дерева каталогов удалённого сервера.

Рекурсивный поиск осуществляют целый ряд приложений, среди которых важным является зеркальное копирование. Оно очень удобно, когда медленная передача данных по сети может быть обойдена созданием локальной копии необходимых файлов.

Следует, тем не менее, помнить о возможных проблемах, связанных с массовым копированием с использованием рекурсии.

Может оказаться, что глубокая рекурсия породит такой поток данных, что он прекратит всякую разумную деятельность на вашем локальном сервере для всех пользователей, например, в связи с переполнением дисковой памяти. Большой поток данных может еще больше загрузить вашу сеть, а не освободить её, как вы предполагали.

Таким образом, следует весьма внимательно относиться к использованию таких параметров, как глубина рекурсии (**-l**). Полезно ограничивать число попыток (**-t**) прочитать большие файлы и, одновременно, увеличивать интервал между попытками (**-w**).

Во всех случаях полезно точно оценивать объём копируемых данных, а также уровень, на который вырастет загрузка вашей компьютерной сети, **ДО** того, как вы начёте копирование.

15.4.1 Следование ссылкам (линкам)

Когда производится рекурсивный поиск и копирование, полезно ограничивать копирование информации, в которой вы не нуждаетесь.

Например, если вы хотели бы иметь копию страниц какого-то проекта, вы, возможно, не захотите иметь копии страниц других проектов, на которые могут быть ссылки на интересных для вас страницах. **Wget** имеет несколько механизмов для точного отбора, какие линки должны приниматься во внимание, а какие – нет.

Относительные ссылки

Когда вы используете только относительные линки (параметр **-L**), рекурсивный поиск и копирование никогда не сменит хост, т.е. копирование будет иметь место лишь с одного сервера. Не будет производиться лишних поисков новых имён хостов через серверы *DNS*, тем самым работа по копированию будет производиться быстрее. такой режим часто будет отвечать вашим интересам, поскольку много документов типа *HTML* получены путём использование разнообразных конвертеров **x2html**, а они, обычно, генерируют относительные линки.

Проверка имен серверов

Недостаток следования только относительным ссылкам очевиден. Часто авторы смешивают относительные линки с абсолютными на одном и том же сервере и, даже, на одной и той же странице. В режиме проверки имён серверов, который включен по умолчанию, будут приниматься во внимание лишь те *URL*, которые указывают на тот же хост.

Проблемы могут быть и здесь для **Wget** нет никакого способа догадаться, что *dbserv.pnpi.spb.ru* и *www.pnpi.spb.ru* есть один и тот же хост. Когда встречается абсолютный линк, то производится поиск с обращением к серверу *DNS*. Даже, учитывая, что ответы сервера запоминаются в кэше, это весьма замедляет копирование файлов.

Чтобы избежать таких задержек, можно использовать параметр **-nh**, который выключает обращение к *DNS* серверу во время копирования файлов. Тем самым, процедура копирования ускорится, но останется риск, что абсолютные ссылки на один хост будут рассмотрены как ссылки к двум разным хостам.

Заметим, кроме того, что современные *HTTP* серверы допускают один *IP* адрес для нескольких **виртуальных серверов**, каждый из которых имеет собственную иерархию каталогов. Такие серверы различаются по именам хостов (каждый сервер имеет своё имя хоста), каждый из которых указывает на один *IP* адрес. В таком случае бесполезно обращаться с одним и тем же именем хоста, значит параметр **-nh должен быть включен**.

Другими словами, параметр **-nh** должен использоваться для организации поиска и копирования с виртуальных серверов, которые различаются лишь именами хостов. Поскольку число таких серверов растёт, то в будущем такой режим станет стандартным умолчанием.

15.4.2 Контроль имен доменов

Параметр **-D** позволяет определить имена доменов, которые будут приняты во внимание при поиске и копировании файлов. Хосты домена, который не включён в список, не будут просматриваться посредством сервера *DNS*. Так, если вы определили **-D***mit.edu*, чтобы гарантировать, что ничего не будет копироваться с серверов вне университета *MIT*. Это очень важно и полезно. Параметр **-D** не подразумевает параметра **-H** (разрешить переходы с хоста на хост), который должен определяться отдельно. Таким образом, вы могли бы вызвать

```
wget -r -D.gov http://www.fnal.gov/
```

чтобы гарантировать, лишь хосты типа *.gov* будут приняты во внимание в данной операции поиска и копирования.

Другой пример.

```
wget -r -H -Dmit.edu,stanford.edu http://www.mit.edu/
```

поиск начнётся с хоста *www.mit.edu*, а продолжится, следуя ссылкам, лишь в доменах *mit.edu* и *stanford.edu*.

Имеется возможность специально исключить некоторые домены с помощью параметра -**exclude-domains**, который принимает те же типы значений, что и параметр **-D**, но исключает все домены, имеющиеся в списке. например, если вы желаете копировать все линки с любых хостов домена *foo.org*, исключая *sun.foo.org*, то можно было бы написать

```
wget -rH -Dfoo.org -exclude-domains sun.foo.org \
http://www.foo.org/
```

Интерпретация такой записи проста: начиная с хоста *www.foo.org* рекурсивно следовать всем линкам в домене *foo.org*, исключая субдомен *sun.foo.org* с разрешением смены хостов.

Все хосты

Когда параметр **-H** определяется без параметра **-D**, то все хосты могут быть приняты во внимание при следовании ссылкам. Нет никаких ограничений какая бы часть мировой сети не была задействована для поиска и копирования файлов, исключая максимальную глубину рекурсии. Это может повлечь мощнейший поток данных, например, если в качестве отправной точки использовался *www.yahoo.com*. Такая возможность весьма редко оказывается полезной сама по себе без связи с другими ограничениями.

15.4.3 Типы файлов

Когда загружаются файлы из мировой паутины, вы можете захотеть ограничить поиск лишь отдельными типами файлов. Например, вы может интересоваться статьями и не интересоваться картинками из Интернет.

Wget предлагает два варианта решения такой проблемы. Каждый вариант включает короткие имена параметров, длинные имена параметров, а также их эквиваленты в файле **.wgetrc** .

-A *acclist*

-**accept** *acclist*

accept = *acclist* Значение параметра -**accept** представляет собой список окончаний имён файлов или шаблонов, которые **Wget** будет рекурсивно искать и копировать. Окончание представляет собой конечную часть имени файла, например, **.gif** или **.ps**. Шаблон представляет собой вариант регулярного выражения подобно тем, что используются в оболочках, например, `man*[3-7]*`. Таким образом команда

```
wget -A ps,pdf http://www.cern.ch
```

будет искать и копировать файлы в форматах PostScript и PDF, в то время как команда

```
wget -A man*[3-7]* http://www.cern.ch
```

будет искать файлы с именами, удовлетворяющими шаблону. Среди таких могут оказаться файлы с именами: `manual41` или `manpower75.ps`. Последняя форма записи параметра **accept** = *acclist* используется в файле `.wgetrc`.

Любое количество окончаний или шаблонов могут быть перечислены через запятую.

-R *rejlist*

-**reject** *rejlist*

reject = *rejlist* Параметр -**reject** действует таким же образом как -**accept**, только с противоположным смыслом: **Wget** будет загружать файлы с любыми именами, **исключая** файлы, имена которых перечисленные в списке или удовлетворяют шаблону. Последняя форма параметра **reject** = *rejlist* используется в файле `.wgetrc`.

Таким образом, комбинируя эти параметры можно весьма точно настроить работу программы **Wget**, чтобы, с одной стороны, получить все необходимые файлы, и минимизировать сетевой трафик и время передачи - с другой.

Заметим в заключение, что данные параметры не действуют на файлы типа *HTML*; **Wget** должна загрузить все файлы типа HTML, чтобы понять куда двигаться дальше – без этого рекурсивный поиск теряет смысл.

15.5 Отметки времени в записях о файлах в оглавлении каталогов

Одим из наиболее важных аспектов зеркального копирования информации из Интернет является изменение ваших архивов.

Копирование полного архива снова и снова, чтобы заменить несколько обновлённых файлов является дорогим мероприятием, как в смысле потребляемой пропускной способности канала связи, так и времени на выполнение. Вот почему все средства, предназначенные для выполнения зеркального копирования, предлагают методы частичной замены файлов.

Такой механизм обновления обозначает, что удалённый сервер сканируется на предмет определения **новых** файлов. Только эти файлы будут копироваться на ваш компьютер, замещая прежние версии этих файлов.

Файл рассматривается как **новый** в следующих случаях:

1. Файл с данным именем не существует на локальной машине.
2. Файл с данным именем существует, но он **старее**, чем файл с таким же именем на удалённом сервере.

Таким образом, для определения какой файл страше, программе, которая это сравнение делает, необходимы сведения о времени модификации, как локальных, так и удалённых файлов. Такая информация именуется **отметки времени**.

В GNU **Wget** режим проверок отметок времени включается параметром **-timestamping** (**-N**) или с помощью директивы

timestamping = on

в файле `.wgetrc`. Если такой параметр использован, то каждый раз когда конкретный файл планируется программой **Wget** к копированию, программа проверяет существует ли уже файл с таким именем, а если существует, то проверяет какой файл старше (удалённый или локальный). Копирование производится лишь тогда, когда локальный файл **старше** удалённого. Если локальный файл с таким именем не существует или его размер отличается от удалённого, то производится копирование независимо от времени модификации.

15.5.1 Использование отметок времени для файлов в оглавлении каталогов во время копирования

Использование отметок времени модификации файла весьма просто. Скажем, вы желаете скопировать удалённый файл так, чтобы он сохранял время модификации

```
wget -S http://www.gnu.ai.mit.edu/
```

По завершении копирования простой командой

```
ls -l
```

вы увидите, что время модификации локальной копии равно времени модификации удалённой копии. Как вы обратили внимание, это произошло даже без использования параметра **-N**.

Несколько дней спустя, вы проверяете изменилась ли информация, которую вы уже копировали

```
wget -N http://www.gnu.ai.mit.edu/
```

В данном случае **Wget** сначала проверит, какие файлы старше, удалённые или локальные. Скопированы будут лишь те файлы, которые изменились за эти несколько дней.

То же самое имеет место и для *FTP*.

Если вы желаете получать зеркальную копию каждую неделю, вы могли бы выдавать каждую неделю следующую команду:

```
wget -timestamping -r ftp://prep.ai.mit.edu/pub/gnu/
```

15.5.2 Техника использования отметок времени для файлов, копируемых по протоколу *HTTP*

Отметки времени в документах *HTTP* реализована посредством проверки заголовка *Last-Modified.* Если вы хотите скопировать файл `feee.html` посредством *HTTP* с использованием отметок времени, то **Wget** сначала проверит существует ли файл `feee.html` локально. Если нет, то он будет скопирован.

Если файл существует локально, **Wget** проверит время модификации файла, затем пошлёт запрос на удалённый сервер. Если локальная копия файла окажется **старее**, то файл будет скопирован. Если нет, то **Wget** перейдёт к следующим файлам. В дополнение, **Wget** проверит размер файла, если размеры различны, файл будет скопирован.

15.5.3 Техника использования отметок времени для файлов, копируемых по протоколу *HTTP*

Отметки времени в документах *HTTP* реализована посредством проверки заголовка *Last-Modified.* Если вы хотите скопировать файл `feee.html` посредством *HTTP* с использованием отметок времени, то **Wget** сначала проверит существует ли файл `feee.html` локально. Если нет, то он будет скопирован.

Если файл существует локально, **Wget** проверит время модификации файла, затем пошлёт запрос на удалённый сервер. Если локальная копия файла окажется **старее**, то файл будет скопирован. Если нет, то **Wget** перейдёт к следующим файлам. В дополнение, **Wget** проверит размер файла, если размеры различны, файл будет скопирован.

15.5.4 Техника использования отметок времени для файлов, копируемых по протоколу *FTP*

Теоретически отметки времени в *FTP* обрабатываются также как и в файлах *HTTP*, только реализовано это иначе. В *FTP* отметки времени получаются из оглавления *FTP* сервера.

Для каждого каталога *FTP* **Wget** использует команду `LIST`, чтобы получить оглавление каталога. Программа пытается анализировать оглавление, предполагая, что оно имеет такой же формат как ответ команды

```
ls -l
```

в операционной системе **UNIX**. В остальном – все происходит так же как и в *HTTP*.

15.6 Конфигурационный файл .wgetrc : установка умолчаний

Раз вы знаете какие параметры принимает **Wget**, вы могли бы пожелать установить часть параметров постоянно в специальном конфигурационном файле `.wgetrc`, чтобы использовать конкретный набор параметров постоянно. Файл `.wgetrc` является главным конфигурационным файлом для **Wget**, однако, если имеется файл `$HOME/.netrc`, то он будет принят во внимание. Формат файла `$HOME/.netrc` может быть найден в описании системы Linux, смотрите, например,

man ftp

Wget читает файл `$HOME/.wgetrc` и интерпретирует ограниченный набор директив.

15.6.1 Где может находиться файл .wgetrc

Во время старта программа **Wget** читает сначала глобальный конфигурационный файл, который находится по умолчанию в `/usr/local/etc/wgetrc` и выполняет его.

Затем **Wget** пробует прочесть пользовательский конфигурационный файл. Имя файла может содержаться в переменной окружения WGETRC. Если эта переменная установлена, то **Wget** пытается прочесть пользовательский конфигурационный файл. Если файл не найден, то более не предпринимается никаких попыток разыскать пользовательский конфигурационный файл.

Если переменная WGETRC не установлена, то **Wget** пытается прочесть файл `$HOME/.wgetrc`. Если содержание глобального и локального конфигурационного файла противоречат друг другу, то принимаются значения из пользовательского конфигурационного файла.

15.6.2 Синтаксис директив в файле .wgetrc

Синтаксис директив в файле `.wgetrc` прост:

variable = *value*

variable может также именоваться **командой**. Правильные значения для *value* различны для разных команд. Значения нечувствительны к используемому регистру и знакам подчёркивания. Например, **DiR___Prog** полностью эквивалентно **dirprog**.

Пустые строки, строки содержащие в первой позиции знаки # (решётка), а также строки, содержащие лишь пробелы, игнорируются.

Если вы хотите очистить какой-то параметр, то это можно сделать, например, так

```
reject =
```

тем самым параметру **reject** присваивается пустое значение.

15.6.3 Директивы в файле .wgetrc

Полный список директив приведён ниже. Буквы после знака = (равно) означают новое значение, которое принимает параметр. Это может быть **on** или **off**, а также **0** или **1**. **STRING** означает непустую строку, а **N** – неотрицательное целое. К примеру, вы можете написать **use_proxy** = **off**, чтобы запретить использование прокси сервера по умолчанию. Вы можете использовать **inf** для чисел без ограничений по величине.

Большинство, но не все, директивы из файла **wgetrc** имеют свои эквиваленты в параметрах командной строки.

accept/reject = *string* То же самое, что **-A**/**-R** (типы файлов).

add_hostdir = **on/off** Разрешить/запретить создание файлов, имеющих префикс в виде имени хоста. Параметр **-nH** выключает это.

continue = **on/off** Разрешить/запретить продолжение поиска. Параметр **-c** разрешает это.

background = **on/off** Разрешить/запретить работу в фоновом режиме, то же, что параметр **-b**, который разрешает его.

base = *string* Установить базу для относительных URL, то же, что параметр **-b**, который разрешает это.

cache = **on/off** Когда установлено **off**, то запрещает кэширование, то же, что **-B**.

convert links = **on/off** Включить/выключить преобразование абсолютных линков в относительные. То же, что **-k**.

cut_dirs = *n* Игнорировать *n*-ые компоненты удалённых каталогов.

debug = **on/off** Включить/выключить режим отладки. То же что **-d**.

delete_after = **on/off** Включить/выключить удаление файла сразу после завершения копирования. То же, что -**delete-after**.

dir_prefix = *string* Вершина иерархии каталогов. То же, что **-P**.

dirstruct = **on/off** Включить/выключить структуру оглавления. То же, что **-x** или **-nd** соответственно.

domains = *string* То же самое, что **-D**.

dot_bytes = *n* Определить количество байтов на точку, которые выводятся на экран, чтобы показать процесс копирования (умолчание: 1024 байта).

dots_in_line = *n* Определить число точек, которые будут печататься в каждой строке (умолчание: 50 точек).

dot_spacing = *n* Определить число точек в кластере (умолчание: 10 точек).

dot_style = *string* Определить стиль вывода точек, то же, что -**dot-style**.

exclude_directories = *string* Определить через запятую список каталогов, которые вы хотите исключить из списка на копирование (то же что **-X**).

exclude_domains = *string* То же, что -**exclude-domains**.

follow_ftp = **on/off** Следовать всем FTP линкам в документе HTML, то же, что **-f**.

force_html = **on/off** Если включено (**on**) рассматривать входной документ как документ HTML, то же что **-F**.

ftp_proxy = *string* Использовать *string* как FTP прокси, вместо определённого в переменной окружения.

glob = **on/off** Включить/выключить расширение имён файлов, то же что **-g**.

header = *string* Определить дополнительный заголовок, то же что -**header**.

http_passwd = *string* Установить пароль для HTTP.

http_proxy = *string* Использовать *string* в качестве HTTP прокси вместо прокси сервера, который определён в переменной окружения.

http_user = *string* Установить имя пользователя HTTP.

ignore_length = **on/off** Когда включено (**on**), игнорировать содержимое поля заголовка *Content-Length*, то же самое что -**ignore-length**.

include_directories = *string* Определить разделённый запятыми список каталогов, которые вы хотите скопировать, то же самое что **-I**.

input = *string* Читать URL из *string*, то же что **-i**.

kill_longer = on/off Рассматривать данные с размером более, чем определены в поле заголовка *Content-length*, как неверные и попробовать скопировать их снова. По умолчанию данные сохраняются если их размер равен или больше содержимого поля заголовка *Content-length*.

logfile = *string* Установить имя файла для сохранения протокола выполнения программы. То же, что **-o**.

login = *string* Установить имя пользователя на удалённом FTP сервере. Умолчание – *anonymous*.

mirror = **on/off** Включить/выключить зеркальное копирование. То же, что **-m**.

netrc = **on/off** Включить/выключить чтение конфигурационного файла **$HOME/.netrc**.

noclobber = **on/off** То же, что **-nc**.

no_parent = **on/off** Запретить поиск вне каталожной иерархии, то же что -**no-parent**.

no_proxy = *string* Использовать *string* как разделённый запятыми список доменов, которые не должны загружаться через прокси сервер вместо списка определённого в переменной окружения.

output_document = *string* Установить имя выводного файла, то же что **-O**.

passive_ftp = **on/off** Установить режим пассивного FTP, то же что -**passive-ftp**.

passwd = *password* Установить пароль для FTP. Умолчание – *username@hostname.domainname.*

proxy_user = *string* Установить имя пользователя как *string*, то же что -**proxy-user**.

proxy_passwd = *string* Установить пароль для прокси как *string*, то же что -**proxy-passwd**.

quiet = **on/off** Включить/выключить режим работы **Wget** без выдачи диагностических сообщений, то же что **-q**.

quota = *quota* Определить квоту для суммарного объёма копируемых данных. Когда квота определена, **Wget** остановит поиск и копирование после превышения квоты. Квота может быть определена в байтах (умолчание), в Кбайтах (использовать букву **k**), или в мегабайтах (использовать букву **m**). Иными словами, **quota** = **5m** означает квоту в 5 мегабайтов.

reclevel = *n* Определить уровень рекурсии, то же что **-l**.

recursive = **on/off** Включить/выключить рекурсию, то же что **-r**.

relative_only = **on/off** Следовать лишь относительным линкам в документе HTML, то же что **-L**.

remove_listing = **on/off** Если **on**, то удаляется оглавление каталогов, полученное программой **Wget** с удалённого FTP сервера, по завершении работы программы. То же что **-nr**.

retr_symlinks = **on/off** Если установлено **on**, то рассматривать символический линк как обычный файл; то же что -**retr-symlinks**.

robots = **on/off** Использовать (**on**) или не использовать файл /**robots.txt**. Умолчание: **on**. Если задумаете менять умолчание, то постарайтесь ясно понять результат.

server_response = **on/off** Включить/выключить печати ответов серверов HTTP и FTP; то же что **-S**.

simple_host_check = **on/off** То же, что **-nh**.

span_hosts = **on/off** То же, что **-H**.

timeout = *n* Установить значение максимального ожидания ответа; то же что **-T**.

timestamping = **on/off** Включить/выключить использование отметок времени; то же что **-N**.

tries = *n* Установить максимальное число повторов на URL; то же что **-t**.

use_proxy = **on/off** Включить/выключить использование прокси; то же что **-Y**.

verbose = **on/off** Включить/выключить подробную/сокращённую диагностику времени выполнения программы **Wget**; то же что **-v**/**-nv**.

wait $= n$ Ждать n секунд между новыми попытками поиска; то же что **-w**.

15.6.4 Пример файла .wgetrc

Ниже приведён пример конфигурационного файла для программы **Wget**.

```
###
###             Пример файла инициализации .wgetrc
###

## Вы можете использовать данный файл для изменения поведения программы
## wget, принятого по умолчанию.
##
## Инициализационный файл программы wget может находиться в
## /usr/local/etc/wgetrc  глобальные умолчания)
## или в $HOME/.wgetrc (персональные умолчания для отдельного
## пользователя).


# Вы можете установить квоту (полезно для начинающих) на суммарный объем
# скопированных файлов в одном сеансе работы программы. За числом может
# следовать 'K' (килобайты) или 'M' (мегабайты). По умолчанию, квота
# не ограничена.
#quota = inf

# Вы можете уменьшить или увеличить число повторов попыток скопировать
# файл с удаленного сервера. Умолчание: 20 попыток.
#tries = 20

# Глубина рекурсии при просмотре удаленных документов HTTP и FTP.
# Умолчание: 5.
#reclevel = 5

# Многие места (серверы, организации) имеют защитные серверы, которые не
# разрешают инициализацию FTP соединений извне организации. Для таких
# серверов вы может использовать режим 'passive' для FTP. Если вы
# находитесь в таком положении, то вам можно установить режим 'passive' по
# умолчанию.
#passive_ftp = off
passive_ftp = on

##
## Локальные установки (для конкретного пользователя этот файл находится в
## $HOME/.wgetrc. Крайне неудобно устанавливать эти умолчания в глобальном
## инициализационном файле (обычно в /usr/local/etc/wgetrc), поскольку
## данные установки могут оказаться неудобными и неприемлемыми для других
## пользователей.
##

# Установить в 'on', чтобы использовать отметки времени по умолчанию.
```

```
#timestamping = off
timestamping = on

# Очень хорошая идея установить этот заголовок. Wget будет посылать его
# вместе с вашими запросами. Поэтому в случае возникновения проблем,
# администратор сервера сможет связаться с вами. Wget никогда не посылает
# 'From:' по умолчанию.
#header = From: Your Name <username@site.domain>

# Вы можете установить и другие заголовки, например, Accept-Language
# (приемлемый язык). Заголовок Accept-Language не посылается по умолчанию.
#header = Accept-Language: en

# Вы можете установить по умолчанию прокси сервер, который будет
# использовать Wget. Эта установка заменит прокси сервер, который
# содержится в переменной окружения.
#http_proxy = http://proxy.yoyodyne.com:18023/
http_proxy = http://proxy.pnpi.spb.ru:81/
ftp_proxy = http://proxy.pnpi.spb.ru:81/

# Если вы не хотите использовать прокси по умолчанию, то установите 'off'.
#use_proxy = on
use_proxy = on

# Вы можете сконфигурировать внешний вид диагностики о процессе
# копирования. Правильные значения: default, binary, mega, micro.
#dot_style = default

# Установка данного параметра в 'off' будет означать, что Wget не примет
# во внимание файл /robots.txt на удаленном сервере. Прежде чем менять
# это умолчание, пожалуйста ознакомьтесь, что это будет означать конкретно
# для вас.
#robots = on

# Может оказаться полезным установить некоторый интервал между очередными
# попытками скопировать файл с удаленного сервера. Интервал может
# измеряться в секундах (без обозначений), 'm' - минуты, 'h' - часы, 'd' -
# дни.
#wait = 0

# Вы можете потребовать создать каталожную структуру, даже если вы
# копируете единственный файл. Для этого надо установить 'on'.
#dirstruct = off

# Вы можете включить рекурсивный поиск по умолчанию ('on').
#recursive = off

# Чтобы сказать Wget, что надо следовать по FTP линкам из документов HTML,
# установите 'on'.
#follow_ftp = off
```

15.7 Примеры использования Wget

Примеры разбиты на три части, которые совершенно очевидны по своим названиям.

15.7.1 Простые примеры

Предположим, вы хотите скопировать на свою машину удалённый файл, с определённым URL. Тогда просто печатайте:

`wget http://www.cern.ch`

Ответ будет примерно таким:

```
11:09:55>bash>pcfarm>wget http://www.cern.ch
--11:10:03--  http://www.cern.ch:80/
           => ‘index.html’
Connecting to proxy.pnpi.spb.ru:81... connected!
Proxy request sent, awaiting response...
```

Иными словами, запрос послан.

А вот что происходит, когда канал не работает более часа. В запросе было указано, что новую попытку чтения следует делать через один час **-w1h**.

```
dbserv: 11:48 /usr/users/shevel > wget -Yoff -w1h http://www.cern.ch
--11:48:48--  http://www.cern.ch:80/
           => ‘index.html’
Connecting to www.cern.ch:80...
connect: Connection timed out
Retrying.

--12:50:03--  http://www.cern.ch:80/
  (try: 2) => ‘index.html’
Connecting to www.cern.ch:80...
connect: Host is unreachable
Retrying.

--13:51:17--  http://www.cern.ch:80/
  (try: 3) => ‘index.html’
Connecting to www.cern.ch:80... connected!
HTTP request sent, fetching headers... done.
Length: unspecified [text/html]

    OK -> ........

13:51:38 (533.79 B/s) - ‘index.html’ saved [10165]
```

А что произойдёт, если соединение с удалённым сервером очень медленное, а файл велик по размеру? Естественно, что во время передачи соединение может быть потеряно. Если соединение будет потеряно до того, как будет закончено копирование файла, то **Wget** попытается возобновить копирование. По умолчанию делается 20 попыток. Вы можете установить своё значение.

```
wget -tries=45 http://www.cern.ch
```

Наконец, вы можете запустить **Wget** в фоновом режиме и направить вывод в протокольный файл *log*.

```
wget -t 45 -o log http://fly.cc.fer.hr/jpg/flyweb.jpg &
```

Как мы помним **-t** означает то же самое, что **-tries**. Если вы желаете установить неограниченное число повторов, то используйте **-t inf**.

Использование FTP также очень просто.

```
wget ftp://ftp.desy.de/ls-lR
```

Если вы укажете не отдельный файл, а каталог, то **Wget** скопирует оглавление каталога и преобразует его в документ вида *HTML*, который запишет в ваш рабочий каталог под именем `index.html`.

15.7.2 Более сложные примеры

Например, у вас есть список *URL*, которые вам надо скопировать из Интернет. Допустим, список находится в файле `URL_list`, тогда

```
wget -i URL_list
```

выполнит вашу задачу. А если вы укажете `-` (знак минус) вместо имени файла, то **Wget** будет читать список *URL* со стандартного устройства ввода.

Вы можете создать зеркальную копию сервера **GNU** с одной попыткой чтения каждого документа и записью всего, что происходило за время копирования в файл `gnulog`:

```
wget -r -t1 http://www.gnu.ai.mit.edu/ -o gnulog
```

Вы можете скопировать первый уровень всех линков с хоста *http://www.yahoo.com/*:

```
wget -r -l1 http://www.yahoo.com/
```

Если вы, кроме того, захотите видеть на экране все заголовки, то используйте

```
wget -r -l1 -S http://www.yahoo.com/
```

Если вы пожелаете записать все заголовки в начале файлов, которые вы копируете, то это делается так:

```
wget -s http://www.lycos.com/
more index.html
```

В файле `index.html` в начале будут заголовки, которые послал сервер.

Другая задача. Скопировать первые два уровня документов, сохранив их в каталоге `/tmp`

```
wget -P/tmp -l2 ftp://wuarchive.wustl.edu/
```

Допустим, вам необходимо скопировать все *PDF* файлы с удалённого сервера. В этом случае команда

```
wget http://host/dir/*.pdf
```

не будет работать, поскольку *HTTP* протокол не поддерживает подобное расширение имён файлов. В этом случае, вы должны использовать:

```
wget -r -l1 -no-parent -A.pdf http://host/dir/
```

Комбинация параметров **-r -l1** означает рекурсивный поиск с глубиной 1. А выражение -**no-parent** означает, что ссылки к родительскому каталогу должны игнорироваться. Наконец, **-A.***pdf* означает, что следует копировать только файлы типа *PDF*. Кстати, если написать **-A** *"*.pdf"*

(с кавычками), то это тоже будет работать.

Предположим, **Wget** была прервана в середине процесса копирования группы файлов. Вы не желаете, чтобы имело место автоматическое расширения имён файлов, которые успели скопироваться на ваш компьютер до прерывания **Wget**.

```
wget -nc -r http://www.gnu.ai.mit.edu/
```

Программа **Wget** распознаёт `URL`, указываюшие на личные данные конкретного пользователя, а не только anonymous. Общий формат проиведён здесь

```
ftp://user:password@host/path
http://user:password@host/path
```

Если вы хотите скопировать файл из вашего собственного каталога, то

```
wget ftp://shevel:mypass@rec03.pnpi.spb.ru/lecture
```

Однако, если файл находится не в вашем каталоге, а, например, выше по каталожной иерархии, то следует использовать другую форму команды

```
wget ftp://shevel:mypass@rec03.pnpi.spb.ru/../../data/lecture
```

В показанном примере вы берёте файл из каталога `data`, который находится на один уровень выше, чем основной каталог пользователя `shevel` и находится в соседней ветви дерева каталогов. В ряде случаев, `URL` полезно взять в кавычки:

```
wget "ftp://shevel:mypass@rec03.pnpi.spb.ru/../../data/lecture"
```

Если вы желаете сменить стиль визуализации процесса копирования, то можно, например, установить:

```
wget -dot-style=binary ftp://prep.ai.mit.edu/pub/gnu/README
```

Такой стиль часто оказывается удобнее (8K байтов на точку или 512K байтов на строку). Стиль, который вам оказался более по вкусу, можно поместить в ваш файл `.wgetrc`, сделав тем самым этот стиль стандартным умолчанием.

15.7.3 Нетривиальные примеры использования Wget

Примеры, приведённые здесь, являются полезными способами применения **Wget**.

- Если вы хотите иметь постоянно зеркальную копию удалённой *HTML* страницы или каталога *FTP*, полезно использовать параметр **-mirror** или **-m**, который включает в себя **-r -N**. Вы можете поместить командную строку с вызовом программы **Wget** в **crontab**, определив, что она должна выполняться каждое воскресенье:

  ```
  crontab
  0 0 * * 0 wget -m ftp://ftp.xemacs.org/pub/xemacs/ \
  -o /home/shevel/weeklog
  ```

- Если вы хотите сделать то же самое, но не желаете копировать всякие картинки, а только документы *HTML*, то можно поступить так

  ```
  wget -m -A.html http://www.w3.org/
  ```

- Если вы хотели бы убрать все абсолютные линки, то можно сделать так:

  ```
  wget -k -r URL
  ```

- Вы хотели бы выдать документ *HTML* на стандартный вывод: `wget -O - http://www.fnal.gov`

  При этом **Wget** автоматически предполагает молчаливое выполнение (**-q**).

- Вы может также комбинировать различные возможности, чтобы найти нечто на удалённых серверах:

  ```
  wget -O - http://www.fnal.gov | grep -i cern | \
  wget -force-html -i -
  ```

  Здесь первый вызов **Wget** порождает вывод домашней страницы с сервера *http://www.fnal.gov* на стандартный вывод. Далее, команда **grep** отбирает те строки, которые содержат комбинацию `cern`. Далее, следующая команда **Wget** рассматривает ввод со стандартного ввода как список *URL* и производит копирование документов в соответствии со списком.

15.8 Использование серверов прокси

Прокси – это серверы специального назначения, которые служат для передачи данных с удалённых серверов на локальные клиенты. Одно из важных свойств серверов **прокси** – это снижение загрузки компьютерной сети, а, следовательно, сокращение времени ожидания завершения передачи данных по сети. Достигается снижение загрузки путём кэширования (запоминания) всех передаваемых файлов. Таким образом, если какой-то файл будет запрошен снова, то он не будет передаваться по медленной медленной линии связи, а будет выдан пользователю из из локального дискового кэша.

Другой мотив использования серверов **прокси** – разделить внутрифирменные компьютерные сети и Интернет в целях безопасности. При этом все выходы в Интернет для поиска информации производятся только через сервер **прокси**.

Wget поддерживает работу через серверы **прокси** для протоколов FTP и HTTP. Существует несколько способов, чтобы сообщить программе **Wget**, где находятся соответствующие серверы **прокси**. Один из них – это использование переменных окружения.

http_proxy Эта переменная окружения содержит *URL* сервера **прокси** для протокола *HTTP*.

ftp_proxy Эта переменная окружения содержит URL сервера **прокси** для протокола FTP. Достаточно общим является то, что стандартные переменные окружения http_proxy и ftp_proxy содержат один и тот же *URL*.

no_proxy Эта переменная должна содержать разделённый запятыми список доменов, для которых сервер **прокси** не должен использоваться. Например, если переменная no_proxy содержит *pnpi.spb.ru*, то **прокси** не будет использоваться, для чтения документов с сервера Петербургского Института Ядерной Физики *www.pnpi.spb.ru*.

В дополнение к переменным окружения **Wget** распознаёт и другие способы указания серверов **прокси**.

Так, в параметрах **Wget** может быть указано **-Y on/off** или **-proxy=on/off**. Это значит включить/выключить использование сервера **прокси**. То же самое может быть обозначено в инициализационном файле **wgetrc** следующим образом: **proxy = on/off**.

В то же время, в инициализационном файле **.wgetrc** могут быть заданы URL для серверов **прокси**:

```
http_proxy = url
ftp_proxy = url
no_proxy = string
```

Эти установки могут быть переустановлены в переменных окружения.

Некоторые серверы **прокси** требуют аутентификации (использования пароля). Процесс аутентификации предполагает, что программа **Wget** должна послать на сервер прокси имя пользователя **username** и пароль **password** для проверки. Для протоколов *HTTP* реализованы несколько протоколов аутентификации. **Wget** использует пока простую схему **Basic**.

Вы можете определить **прокси** следующим образом. Допустим, ваша компания имеет сервер **прокси** *http://proxy.company.com:8001.* Тогда полный *URL* вместе с данными для аутентификации имеет вид:

http://username:password@proxy.company.com:8001/

Вы также имеете возможность использовать параметры **proxy–user** и **proxy–password**, а также их эквиваленты в файле `.wgetrc`: **proxy_user** и **proxy_passwd**, чтобы установить имя и пароль пользователя для проверки на сервере **прокси**.

15.9 Прочие сведения о Wget

Программу **Wget** можно взять как и другие **GNU** программы на основном сервере **GNU** *prep.ai.mit.edu* и множестве зеркал этого сервера. Например, **Wget 1.5.3** находится в `ftp://prep.ai.mit.edu/pub/gnu/wget-1.5.3.tar.gz`.

Программа **Wget** имеет свой список рассылки *wget@sunsite.auc.dk.* Чтобы получать сообщения из этого списка рассылки, вы можете подписаться, послав письмо по адресу *wget-subscribe@sunsite.auc.dk* с магическим словом **subscribe** в поле *Subject.* Архив списка рассылки находится на сервере *http://fly.cc.fer.hr/archive/wget.*

Над программой **Wget** работали (добавляли различные свойства, тестировали, исправляли ошибки) несколько десятков человек. Если вы посмотрите на своей системе Linux описание **Wget** с помощью команды

```
info wget
```

то сможете найти полный список лиц, участвовавших в создании программы. Программа не является застывшим продуктом, она постоянно улучшается, появляются новые версии.

15.9.1 Дополнения о файле `/robots.txt`

Поскольку **Wget** производит поиск по ссылкам в документах *HTML*, то эта программа может рассматриваться как один из **Интернетровских поисковых роботов**, или просто **роботов**. Такие поисковые системы часто именуют *путешественниками (wanderer)* или *пауками (spider)* А раз так, то **Wget** понимает содержание файла `/robots.txt`, который администраторы серверов используют, чтобы закрыть часть каталогов от **Wget** и других поисковых систем.

Файл `/robots.txt` принимается во внимание только тогда, когда **Wget** производит рекурсивный поиск (**-r**) и он считывается с хоста лишь однажды во время поиска. Например,

```
wget -r http://fly.cc.fer.hr/
```

Индекс с сервера будет загружен. Далее, если **Wget** найдёт что-то для загрузки с того же сервера, лишь тогда будет считан файл `/robots.txt`, чтобы решить, следует ли загружать очередной файл, на который имеется ссылка в индексном файле. **Wget** не поддерживает тег `META` в файле `/robots.txt`.

Подробное описание содержания файла `/robots.txt` находится на страницах *http://info.webcrawler.com/mak/projects/robots/norobots.html.*

15.9.2 Родственники программы wget

Wget имеет полезных родственников, т.е. систем, которые выполняют подобные функции. Некоторые из этих систем являются дополнительными по отношению к **Wget**, иными словами, реализуют те функциональные возможности, которые отсутствуют или недостаточно проработаны в **Wget**.

К таким системам можно отнести **cURL** (`http://www.fts.frontec.se/~dast/curl/manual.html`). Эта система позволяет не только копировать информацию с удалённого сервера на локальный, но даёт возможности загружать информацию с локальной машины на удалённый сервер (*ftp, http, https*). Можно обратить внимание на систему *Паук - Pavuk* (`http://www.idata.sk/~ondrej/pavuk/`), indexwget!Pavuk которая также выглядит весьма привлекательно.

Глава 16

Список литературы и индексы

Литература

[1] Matt Welsh and Lar Kaufman
Running LINUX
Second Edition
1996
ISBN 1-56592-151-8
O'Reilly & Associates, Inc.
630 pages.

[2] Jessica Perry Hekman
Linux in a Nutshell
A Desk Quick Reference
ISBN: 1-56592-167-4
O'Reilly & Associates,Inc.
1997
424 pages.

[3] Ричард Петерсен
LINUX: руководство по операционной системе
Второе издание, переработанное и дополненное
в двух томах, перевод
Киев, BHV, 1998
1000 стр.

[4] Андрей Робачевский
Операционная система UNIX
BHV-Санкт-Петербург
1997
500 стр.

[5] Paul W. Abrahams, Bruce R. Larson
UNIX for the Impatient
Addison-Wesley Publishing Company
1992
560 pages.

Предметный указатель