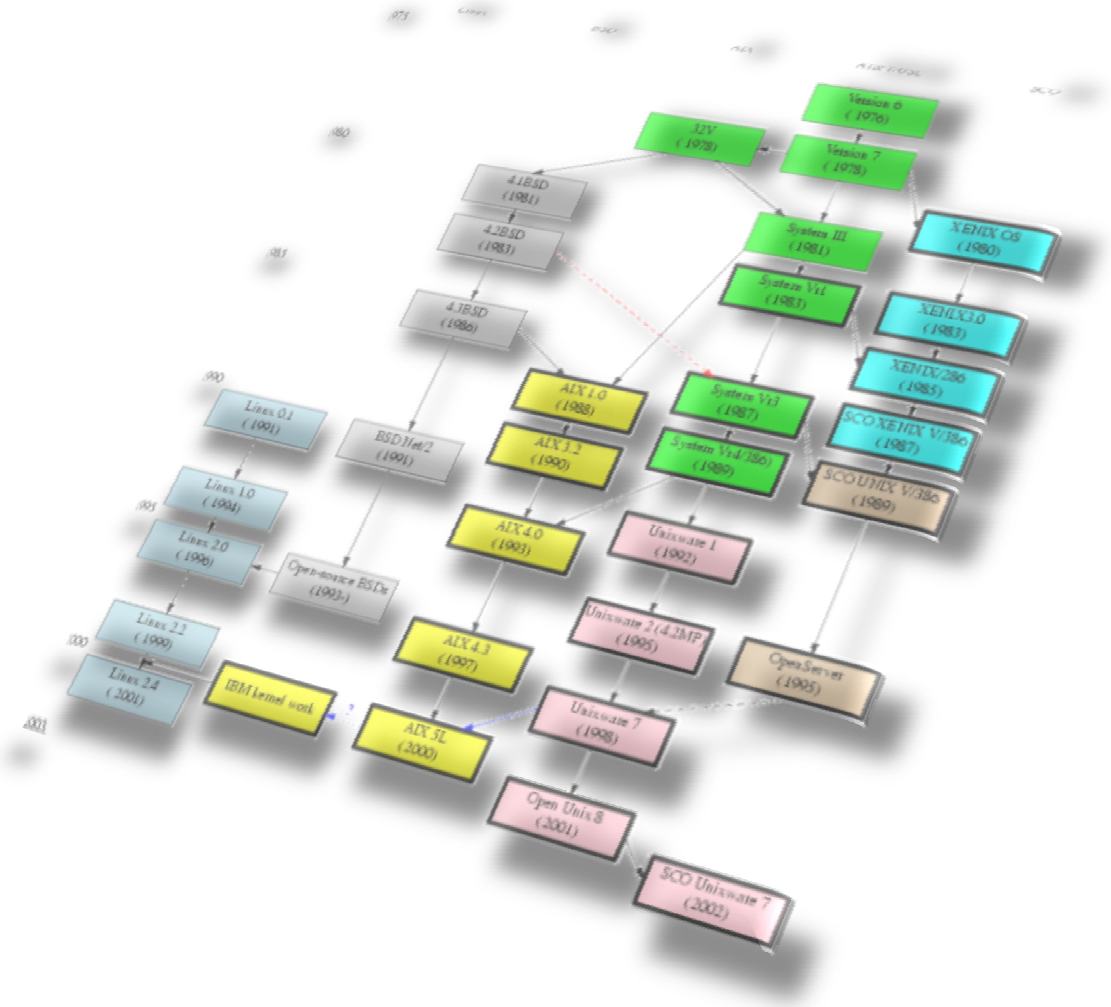
М.В. Ляховец

ПРОГРАММИРОВАНИЕ В UNIX-СИСТЕМАХ

Учебное пособие



Новокузнецк
2011

М.В. Ляховец

ПРОГРАММИРОВАНИЕ В UNIX-СИСТЕМАХ

Учебное пособие

Новокузнецк
2011

Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
«Сибирский государственный индустриальный университет»

М.В. Ляховец

ПРОГРАММИРОВАНИЕ В UNIX-СИСТЕМАХ

Рекомендовано Сибирским региональным учебно-методическим центром высшего профессионального образования для межвузовского использования в качестве учебного пособия для студентов, обучающихся по направлению подготовки 080800 Прикладная информатика

Новокузнецк
2011

УДК 004.451(075)
ББК 32.973.202-018.2я7
Л 985

Рецензенты:
Кафедра автоматизации производственных процессов Кузбасской государственной педагогической академии (заведующий кафедрой – доктор технических наук, профессор В.И. Верёвкин)

Кандидат технических наук, доцент Новокузнецкого филиала-института Кемеровского государственного университета С.А. Шипилов

Ляховец М.В.
Л 985 Программирование в UNIX-системах: учеб. пособие / М.В. Ляховец; Сиб. гос. индустр. ун-т. – Новокузнецк: СибГИУ, 2011. – 136 с.

Рассмотрены практические аспекты использования средств операционных систем семейств UNIX. Произведено введение в среду выполнения системы UNIX в целом, способы обращения пользователей к ее различным частям. Представлены средства для практических каждодневных задач по сопровождению файлов – для копирования и сохранения файлов и для удаления ненужных файлов. Также обращено внимание на собственную среду пользователей и её администрирование. Рассмотрены способы программирования в UNIX-системах с помощью скриптового языка программирования Shell.

Предназначено для студентов всех форм обучения направления подготовки 080800 Прикладная информатика.

УДК 004.451(075)
ББК 32.973.202-018.2я7

Содержание

Предисловие

Операционные системы семейства UNIX очень надежны в плане функционирования, но требуют повышенного внимания при их использовании и администрировании. Задачей данного учебного пособия является ознакомление с основными командами и служебными программами, используемыми как простыми пользователями, так и системными администраторами, а также ознакомление с возможностями по автоматизации рутинных операций с помощью скриптового языка программирования командного интерпретатора Shell.

Учебное пособие состоит из трёх разделов. Первый раздел посвящён рассмотрению истории возникновения и развития как самой операционной системы UNIX, так и значительных ветвей её развития (BSD, Linux и др.). Рассмотрены основные особенности UNIX-систем, позволившие завоевать признанную популярность среди ИТ-профессионалов.

Во втором разделе учебного пособия рассмотрены принципы работы с пользовательским интерфейсом операционной системы, приведены основные команды для выполнения каждодневных пользовательских задач по сопровождению файлов и управлению процессами.

Третий раздел пособия посвящён принципам создания скриптовых программ для командного интерпретатора shell. Рассмотрены возможности работы с переменными, принципы выполнения арифметических операций, основные программные структуры последовательного, условного и итерационного управления, а также средства отладки скриптов и обработки ошибок.

Учебное пособие предназначено для студентов всех форм обучения направления подготовки 080800 Прикладная информатика и будет полезно студентам, аспирантам и преподавателям вузов, использующих для решения различных задач операционные системы семейства UNIX.

Введение

Среди операционных систем (ОС) особое место занимает UNIX. Беспрецедентным является то, что ОС UNIX может работать практически на всех аппаратных платформах, получила распространение на компьютерах с различной мощностью обработки – от мобильных компьютеров до больших вычислительных комплексов. Причём UNIX практически изначально создавалась как сетевая операционная система, что позволило ей занять лидирующие позиции на рынке серверов для интернет-приложений и дало мощные встроенные средства удалённого администрирования. Можно выделить ряд преимуществ использования UNIX: поддержка широкого набора сетевых средств передачи информации и сетевых протоколов; поддержка широкого спектра средств системного и сетевого управления; поддержка всех основных аппаратных процессорных архитектур; поддержка большинства популярных языков разработки приложений, средств работы с базами данных и промежуточного программного обеспечения; полностью стандартизированная система.

Понятие «UNIX» давно уже не означает какую-то конкретную операционную систему, а объединяет все ОС этого семейства, отвечающие определённым требованиям. Первая система UNIX была разработана в 1969 г. в подразделении Bell Labs компании AT&T. С тех пор было создано большое количество различных UNIX-систем. Юридически лишь некоторые из них имеют полное право называться «UNIX»; остальные же, хотя и используют сходные концепции и технологии, объединяются термином «UNIX-подобные» [1].

К главным частям ОС UNIX относятся [2]: ядро, управляющее основными ресурсами и периферийными устройствами обмена и хранения данных; файловая система, организующая структуры данных на устройствах хранения; и командный интерпретатор shell, транслирующий команды пользователя с терминала в запросы к ядру и файловой системе. Shell является мощным скриптовым языком программирования, не похожим на язык программирования Си, который обеспечивает условное выполнение и управление потоками данных. Для успешного управления работой, регулярного обслуживания ОС и выполнения рутинных операций пользователям необходимо знать и уметь использовать данный язык.

1 История операционной системы UNIX

Операционная система UNIX была создана Кеном Томпсоном и Деннисом Ритчи в Bell Laboratories (одно из подразделений корпорации AT&T). Широко распространяться UNIX/v7 (версия 7) начала в конце 70-ых годов двадцатого века. Вручение создателям UNIX в 1983 году Международной премии А.Тьюринга в области программирования ознаменовало признание этой системы мировой научной общественностью.

На текущий момент история UNIX-систем насчитывает порядка 40 лет. Кратко опишем историю развития этой системы.

1.1 Создание операционной системы UNIX

Операционная система UNIX была разработана в лаборатории Bell Labs, входившей в состав американской компании Bell Systems. В 1957 году этой компании потребовалась операционная система для автоматизации запуска некоторых программ и управления ресурсами вычислительного центра, в котором использовалась ЭВМ второго поколения. Новую систему назвали BESYS. [3-5]

В 1964 году, с появлением в лаборатории мощной ЭВМ третьего поколения, возникла потребность в новой операционной системе. К разработке новой системы были привлечены специалисты из Массачусетского института, работающие над проектом MAC, и корпорации General Electrics. В результате совместной работы была создана операционная система Multics (Multiplexed Information and Computing System), которая базировалась на другой экспериментальной системе Массачусетского института – CTSS. Новая операционная система, в отличие от старой, была многозадачной, многопользовательской системой с разделением времени и пользовательским интерфейсом. Однако она не обеспечивала выполнение главных вычислительных задач, для решения которых она предназначалась, получилась сложной в использовании, громоздкой и дорогой, в которой, к тому же, существовал ряд ошибок, связанных, в основном, с неудачно выбранным языком программирования PL/I. Кроме этого, среди разработчиков возникли некоторые организационные разногласия, и проект был закрыт. [3, 4]

После закрытия проекта сотрудники Bell Labs некоторое время использовали созданную компанией General Electrics систему GECOS. Данная система имела довольно узкие возможности. В связи с этим Кеннет Томпсон вместе со своим коллегой Денисом Ритчи начали разработку собственной универсальной операционной системы для «небольшого» компьютера PDP-7, имевшего подходящий объём оперативной памяти, обладавшего графическим дисплеем и имеющего дешёвое машинное время. Разрабатываемая система должна была воплотить все самые удачные идеи, которые появились при разработке Multics, а именно: иерархическая древовидная структура файловой системы, концепции файла и процесса, командный интерпретатор для пользователя, многопользовательский режим работы и другое. С помощью компьютера General Electrics 635 были написаны ядро будущей системы, текстовый редактор, несколько утилит и собственный Ассемблер. Первоначальное название, которое было придумано для новой системы, – UNICS (Uniplexed Information and Computing System). Позднее это название претерпело сокращение до UNIX. Произошло это в 1969 году, а официальной датой «рождения» UNIX стало 1 января 1970 года.

Несмотря на то, что эта ранняя версия системы UNIX уже была многообещающей, она не могла реализовать свой потенциал до тех пор, пока не получила применение в реальном проекте. Так, для того, чтобы обеспечить функционирование системы обработки текстов для патентного отдела фирмы Bell Laboratories, в 1971 году система UNIX была портирована на ЭВМ PDP-11 [6].

В 1971 году UNIX был выбран руководством лаборатории Bell Labs в качестве платформы для создания системы обработки текстов. В это время К.Томпсон, работая над компилятором языка программирования Fortran, создал новый язык программирования, названный Си, с использованием которого в 1973 году была полностью переписана ОС UNIX. А в 1974 исходные коды UNIX стали распространяться в университетах за символическую плату, что обеспечило дальнейшую популярность системы. Небольшая цена, понятный и доступный для изучения исходный код, гибкость и переносимость, возможность настроить систему под любую конфигурацию сделали её привлекательной для большого количества не

только профессионалов-программистов, но и любителей, которые добавляли новые возможности в систему.

Корпорация AT&T, ставшая владельцем прав на исходный код ОС UNIX, решила внести некоторый порядок в выход новых версий, и в 1982 году несколько последних версий были объединены в одну, получившую название UNIX System III. В 1983 году вышла первая коммерческая версия UNIX, названная System V, в которой появились такие принципиальные возможности, как механизм взаимодействия процессов, замещение страниц и семафоры. К 1989 году вышла новая версия System V Release 4, вновь объединившая достоинства последних версий. Самыми значительными возможностями этой версии стали сокеты, сетевая файловая система (NFS) и новые интерпретаторы командного языка ksh и csh. В 1993 году права на UNIX были проданы компании Novell, которая потом передала их X/Open и Santa Cruz Operation (SCO).

Обретя популярность, UNIX получила множество параллельных веток развития, которые до сих пор развиваются силами многочисленных компаний. ОС UNIX и её «клоны» хоть и обладают посвоему уникальными возможностями, но все они отвечают требованиям POSIX (Portable Operating System Interface, переносимый интерфейс операционной системы), соответствуют X/Open Portability Guide, Edition 4 (XPG 4) и сертифицированы консорциумом X/Open на соответствие стандартам UNIX 93. Таким образом, принципиальных различий с точки зрения пользователя между разными ветками ОС UNIX не существует. [3, 4]

Таким образом, история создания и развития UNIX-подобных систем насчитывает несколько десятилетий и множество веток развития, в результате чего появились и используются большое количество UNIX-подобных систем (рисунок 1).

1.2 BSD, Solaris и другие

Одной из значительных ветвей развития UNIX стала ОС BSD (Berkley Software Distribution) [1, 3, 7]. В 1976 году сотрудник Калифорнийского университета города Беркли Билл Джой на основе шестой редакции UNIX разработал свою версию, назвав её BSD. В 1979 году была выпущена новая версия, названная 3BSD, основанная на седьмой редакции UNIX. BSD поддерживал такие полезные

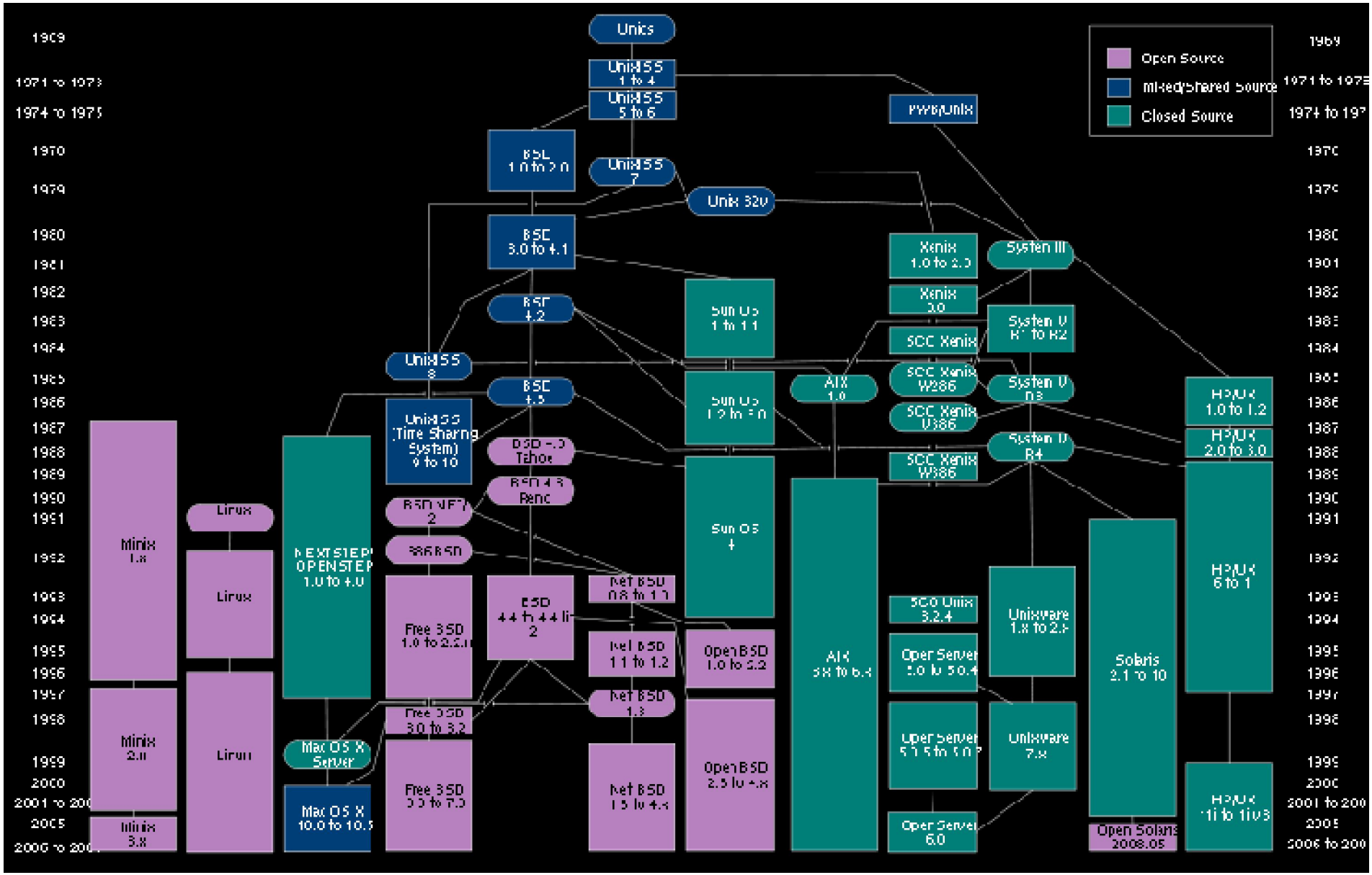


Рисунок 1 – Генеалогическое древо UNIX-систем [1]

свойства, как виртуальную память и замещение страниц по требованию.

В 1980 году, при непосредственном участии министерства обороны США, а именно агентства DARPA, был разработан протокол TCP/IP, что дало эффективное межмашинное взаимодействие и возможность работы в локальной сети. В середине 1983 года была выпущена версия BSD 4.2, поддерживающая работу в сетях Ethernet и Arpanet. Эта ОС распространялась практически бесплатно и имела много последователей, которые на основе BSD выпускали свои версии *nix-систем: FreeBSD, OpenBSD, NetBSD, ClosedBSD, MirBSD, DragonFly BSD, PC-BSD, DesktopBSD, SunOS, TrueBSD, Frenzy, Ultrix и частично Darwin (ядро Mac OS X) [7]. К особенностям FreeBSD можно отнести: приоритетную многозадачность с динамическим регулированием приоритетов; многопользовательская доступность; TCP/IP-стек с поддержкой промышленных стандартов; защита памяти; поддержка симметричной многопроцессорности; поддержка сетевой и виртуальную файловой системы (NFS и VFS, соответственно).

Позже Билл Джой основал фирму Sun Microsystems и занялся на основе UNIX System V Release 4 разработкой SunOS, позднее ставшей известной как Solaris. К особенностям Solaris можно отнести поддержку многопоточности, симметричной многопроцессорной обработки, режима реального времени, а также высокую степень защищённости (встроенные межсетевые экраны, технология безопасности IPSec, стандарт Kerberos 5, контроль над правами доступа).

Кроме BSD и SunOS, появились и другие подвиды UNIX, выпускаемые различными фирмами. Среди них необходимо упомянуть такие системы, как AIX (выпущенная IBM для ЭВМ RS/6000); HP-UX (выпущенная Hewlett Packard для мультипроцессорных ЭВМ с поддержкой больших файловых систем и высокой степенью защищённости от аппаратных и программных сбоев); IRIX (разработанная Silicon Graphics для графических станций и суперкомпьютеров); Digital UNIX (фирмы DEC, предназначенная для мощных серверов, с поддержкой практически всех сетевых интерфейсов и улучшенными драйверами для работы с винчестерами) и многие другие. [3, 4]

Отличия BSD от «классической» системы UNIX состоят в отсутствии переключения уровней выполнения, системе печати, файловой системе и командного процессора. Пользовательские команды практически идентичны. [3, 4, 7]

Современные реализации UNIX, как правило, не являются системами UNIX System V или BSD в чистом виде. Они реализуют возможности как System V, так и BSD. [1]

1.3 GNU/Linux

Говоря об ОС UNIX, необходимо упомянуть и об ОС GNU/Linux (или Linux) [8], которая в настоящий момент является одной из наиболее популярных версий UNIX. ОС Linux поддерживает большинство свойств, присущих другим реализациям UNIX, а также поддерживает другие специфические функции. Linux – это полная многозадачная многопользовательская операционная система.

Сама ОС Linux появилась в начале 90-х годов двадцатого века, но свою историю она берёт с ОС Minix – маленькой операционной системы, которая была создана известным датский профессором Эндрю Танненбаумом в 1987 году для своей книги «Операционные системы». На основе этой ОС в 1991 году студент второго курса Линус Торвальдс из Хельсинки приступил к разработке собственной операционной системы. В это время Ричард Столлмен, занимающийся своим проектом GNU (Gnu is Not UNIX), основная идея которого – бесплатное программное обеспечение, разработал собственный вариант компилятора языка Си. Вместе с Л.Торвальдсом они объединили усилия, и с этого момента началась история ОС Linux. [4]

В сентябре 1991 года появилась Linux 0.01, в октябре – появилась версия 0.02. Почти каждый месяц появлялись более доработанные версии Linux. Так как ОС Linux распространялась всегда бесплатно, с исходными текстами и документацией, то начали появляться различные варианты Linux, собранные энтузиастами и профессионалами по всему миру.

В настоящее время Linux продолжает успешно развиваться и привлекает к себе внимание всё новых и новых пользователей. Именно эта операционная система, а также FreeBSD, стали основ-

ным выбором администраторов web-серверов и корпоративных систем.

К достоинствам ОС Linux можно отнести и то, что Linux работает на множестве архитектур процессора, таких как [8]: Intel x86, x86-64, PowerPC, ARM, Alpha AXP, Sun SPARC, Motorola 68000, Hitachi SuperH, IBM S/390, MIPS, HP PA-RISC, AXIS CRIS, Renesas M32R, Atmel AVR32, Renesas H8/300, NEC V850, Tensilica Xtensa и многих других.

Наиболее известными дистрибутивами Linux являются [8]: Arch Linux, CentOS, Debian, Fedora, Gentoo, Mandriva, Mint, openSUSE, Red Hat, Slackware, Ubuntu. Российские дистрибутивы – ALT Linux, ASPLinux, Calculate Linux, НауЛинукс, AgiliaLinux (ранее MOPSLinux), Runtu и Linux XP.

1.4 Основные особенности UNIX-систем

Популярность и успех системы UNIX можно объяснить несколькими причинами [5, 6]:

- система написана на языке высокого уровня Си, благодаря чему её легко читать, понимать, изменять и переносить на другие машины;
- система поддерживает:
  - вытесняющую многозадачность,
  - многопользовательский режим,
  - виртуальную память и свопинг,
  - разнообразные средства взаимодействия процессов;
- наличие встроенных средств защиты информации;
- наличие единой иерархической файловой системы, имеющей древовидную структуру независимо от количества и типа физических носителей информации, установленных в системе;
- кэширование физического диска для увеличения скорости доступа к данным;
- унификация операций ввода/вывода;
- обеспечение согласования форматов в файлах, работа с последовательным потоком байтов;
- наличие простого, последовательного интерфейса с периферийными устройствами;

- наличие довольно простого пользовательского интерфейса, в котором имеется возможность предоставлять все необходимые пользователю услуги;
- мощный командный язык;
- архитектура машины скрыта от пользователя, благодаря этому облегчен процесс написания программ, работающих на различных конфигурациях аппаратных средств.
- открытый исходный код как самой системы, так и большинства программ для неё;
- бесплатное распространение большинства UNIX-систем.

Простота и последовательность вообще отличают систему UNIX и объясняют большинство из вышеприведенных доводов в её пользу. [6]

По материалам Википедии [1] в настоящее время UNIX используются в основном на серверах, а также как встроенные системы для различного оборудования. На рынке ОС для рабочих станций и домашнего применения UNIX занимает только второе (Mac OS X), третье (GNU/Linux) и многие последующие места.

Вопросы и задания для самоконтроля

1. В каком году началась разработка ОС UNIX? Назовите официальную дату «рождения» ОС UNIX.
2. Какая операционная система легла в основу ОС Linux?
3. Кто создал ОС UNIX?
4. Какое название первоначально получила ОС UNIX?
5. С помощью какого языка программирования была переписана ОС UNIX в 1973 году?
6. Назовите разработчика ОС BSD. В каком году была создана данная версия ОС UNIX?
7. Перечислите основные особенности ОС FreeBSD.
8. Кто создал ОС Linux? В каком году это случилось?
9. Что предопределило популярность ОС UNIX?
10. Перечислите основные особенности ОС UNIX.

2 Основы работы в UNIX

2.1 Начальные сведения

Операционная система UNIX – это набор программ, который управляет компьютером, осуществляет связь между пользователем и компьютером и обеспечивает пользователям инструментальные средства, чтобы помочь выполнить любую необходимую работу. Рассмотренные в данном разделе команды позволяют комфортно работать пользователю с компьютером под управлением операционной системы семейства UNIX, выполняя задачи по сопровождению файлов и собственной среды пользователя.

2.1.1 Начало и конец сеанса работ

Каждый пользователь UNIX наделён строго определёнными атрибутами безопасности, определяющими его работу в операционной системе и доступ к её ресурсам. [5]

Каждый пользователь имеет:

- имя пользователя (для установления взаимодействия пользователей и начисления расходов ресурсов компьютера);
- пароль пользователя (для контроля входа в систему и защиты пользовательских данных).

Пользователи могут быть объединены в группы (например, для работы над проектами и т.п.) для разделения общих ресурсов. В этом случае ещё есть имя группы пользователей.

Один пользователь, называемый суперпользователь (superuser), является администратором системы (его имя – root). В частности, он добавляет всех прочих пользователей, регистрируя их в системе.

Любой сеанс работы в UNIX начинается с входной регистрации пользователя в системе. Во время регистрации пользователю присваивается уникальный идентификатор сеанса и права доступа, без которых невозможно выполнить ни одну команду.

При входе в систему необходимо на запрос

```
login:
```

ввести имя пользователя, например:

```
login: student
```

и нажать на клавишу Enter. На следующий запрос надо ввести пароль пользователя. Необходимо заметить, что при вводе пароля

символы не отображаются, однако пароль вводится в память компьютера.

По окончании успешного входа в систему пользователю предоставляется командная оболочка для выполнения команд, визуально отображающаяся с помощью системного приглашения, которое, по умолчанию, состоит из знака $.

Можно сменить свой пароль в любое время с помощью команды passwd, которая сначала запрашивает у пользователя ввод старого пароля для идентификации пользователя, а затем просит ввести новый пароль и повторить его ввод:

```
login: student
password:
$ passwd
Changing password for student
Old password:
New password:
Retype new password:
$
```

Остановить ввод в систему или выйти из системы (завершить работу) можно с помощью нажатия сочетания клавиш Ctrl-d.

2.1.2 Идентификация пользователей в системе

Чтобы узнать всех активных пользователей, работающих в системе в данное время, можно воспользоваться командой who, которая выводит на экране: список пользователей; название терминала, на котором работает тот или иной пользователь; дата и время входа в систему пользователя; а также IP-адрес или имя компьютера, с которого подключился пользователь, если он работает с системой удалено. Например:

```
$ who
root     tty0   Apr 24  14:38
student  ttyp0  Apr 24  14:53 (10.1.2.1)
student  ttyp1  Apr 24  14:48 (10.1.2.10)
. . .
$
```

Чтобы узнать под каким именем пользователя находитесь, можно воспользоваться следующей командой:

```
$ who am i
student  ttyp0  Apr 24  14:53 (10.1.2.1)
```

2.1.3 Формат команд

Управлять работой системы можно, вводя с клавиатуры команды. Различные команды имеют свой синтаксис, определяющийся наличием или отсутствием различных опций и аргументов. Рассмотрим формат вызова команд.

В простейшем случае, когда используется одна команда и без каких-либо опций, в командной строке указывается только название команды, например, для вывода текущей даты на экран:

```
$ date
Mon Apr 24 14:42:25 UTC 2006
$
```

Командная строка – последовательность слов, разделенных пробелами. Первое слово командной строки есть команда; остальные – параметры.

Типы параметров:

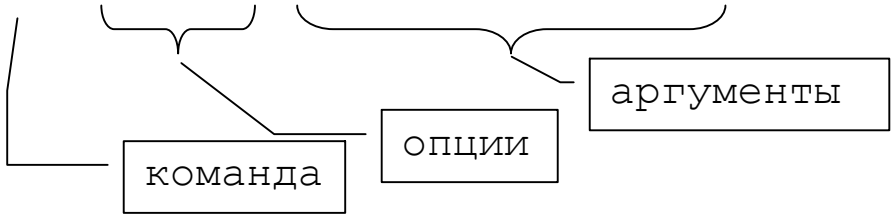
- имя файла = идентификатор (использует символы `a-z`, `A-Z`, `0-9`, `_`, `.`,`-`);
- опция (ключ) – уточняет смысл команды; начинается обычно с минуса. Например `-al` (может быть со знаком `+` или без знака); смысл опции зависит от команды;
- выражение – описывает обычно строку символов или является строкой.

Порядок параметров в команде:

```
command options expression filename(s)
```

Примеры команд:

```
ls  -l -i  file1 file2 file3
```




```
rm old.news bod.news
rm -fr goodies.c baddies.o
grep -o "mary" people
```

Количество параметров у команды может быть переменным. Например, команда `cal` позволяет вывести календарь. Причём, если не указывать параметры, то на экран будет выведен календарь текущего месяца текущего года:

```
$ cal
```

```
   February 2009
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

Если указать один параметр, то он будет интерпретироваться как год:

```
$ cal 2009
                                2009

      January               February              March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
              1  2  3   1  2  3  4  5  6  7   1  2  3  4  5  6  7
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   8  9 10 11 12 13 14
11 12 13 14 15 16 17  15 16 17 18 19 20 21  15 16 17 18 19 20 21
18 19 20 21 22 23 24  22 23 24 25 26 27 28  22 23 24 25 26 27 28
25 26 27 28 29 30 31                        29 30 31

       April                 May                  June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
           1  2  3  4                  1  2      1  2  3  4  5  6
 5  6  7  8  9 10 11   3  4  5  6  7  8  9   7  8  9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30        24 25 26 27 28 29 30  28 29 30
                      31

        July                August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
           1  2  3  4                     1         1  2  3  4  5
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   6  7  8  9 10 11 12
12 13 14 15 16 17 18   9 10 11 12 13 14 15  13 14 15 16 17 18 19
19 20 21 22 23 24 25  16 17 18 19 20 21 22  20 21 22 23 24 25 26
26 27 28 29 30 31     23 24 25 26 27 28 29  27 28 29 30
                      30 31

      October              November             December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
              1  2  3   1  2  3  4  5  6  7         1  2  3  4  5
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   6  7  8  9 10 11 12
11 12 13 14 15 16 17  15 16 17 18 19 20 21  13 14 15 16 17 18 19
18 19 20 21 22 23 24  22 23 24 25 26 27 28  20 21 22 23 24 25 26
25 26 27 28 29 30 31  29 30                 27 28 29 30 31
```

Если же указать два параметра, то они интерпретируются как месяц и год:

```
$ cal 10 2009
   October 2009
Su Mo Tu We Th Fr Sa
              1  2  3
```

```
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Группировка команд. Если необходимо выполнить последовательно несколько команд, то можно воспользоваться специализированными средствами группировки, простейшим из которых является точка с запятой (`;`) [5, 11]. В следующем примере выполняется последовательное выполнение двух команд – идентификации пользователя и вывода текущей даты:

```
$ who am i; date
student  ttyp0  Apr 24  14:53 (10.1.2.1)
Mon Apr 24 14:42:25 UTC 2006
```

Существующие средства группировки представлены в таблице 1.

Таблица 1 – Средства группировки команд

Символы	**Назначение**
`;` (точка с запятой) или `<перевод строки>`	Определяют последовательное выполнение команд.
`&`	Асинхронное (фоновое) выполнение предшествующей команды.
`&&`	Выполнение последующей команды при условии нормального завершения предыдущей, иначе игнорировать.
`||`	Выполнение последующей команды при ненормальном завершении предыдущей, иначе игнорировать.
`|`	Конвейер; стандартный вывод одной команды замыкается на стандартный ввод другой. Если после конвейера стоит `;` (точка с запятой) командный интерпретатор `shell` ждёт его завершения. Если `&` – то не ждёт.

При выполнении команды в асинхронном режиме (после команды стоит один амперсанд) на экран выводится номер процесса,

соответствующий выполняемой команде, и система, запустив этот фоновый процесс, вновь выходит на диалог с пользователем.

Например, команда поиска файлов и каталогов, рассматриваемая далее, занимает очень длительное время, особенно при поиске из корневого каталога, таким образом, лучше запустить программу в фоновом режиме. Например, необходимо найти все файлы с именем `passwd`, а результаты поиска записать в файл `passwd.res`:

```
$ find / -name passwd > passwd.res &
```

После работы команды результаты поиска можно посмотреть в соответствующем файле:

```
$ cat passwd.res
/usr/bin/passwd
/usr/src/etc/pam.d/passwd
/usr/src/release/picobsd/tinyware/passwd
/usr/src/usr.bin/passwd
/etc/pam.d/passwd
/etc/passwd
```

Более подробно о фоновом выполнении программ будет рассмотрено ниже в соответствующем разделе.

Если необходимо определить более сложную логику исполнения команд, то для группировки команд можно использовать фигурные «`{}`» и круглые «`()`» скобки.

```
{список_команд}
```

или

```
(список_команд)
```

В первом случае просто выполняется `список_команд`. Во втором – `список_команд` выполняется с помощью отдельного процесса оболочки командного интерпретатора `shell` и при выходе из круглых скобок происходит возврат в старый командный интерпретатор `shell`. При этом `список_команд` может иметь сколь угодно сложную структуру, состоящую из рассмотренных прежде символов.

2.1.4 Клавиши управления временем исполнения команд

Приостановка-продолжение вывода на экран. Для временного останова вывода на экран результатов выполнения команды служит сочетание клавиш `Ctrl-s`.

Для продолжения вывода на экран информации, которая была остановлена при помощи Ctrl-s, служит сочетание клавиш Ctrl-q.

Останов выполнения команды. Для останова выполнения команды служит сочетание клавиш Ctrl-c или клавиша BREAK.

2.1.5 Документация

Пользователь имеет доступ более чем к 150 различным системным командам. Полные описания системных команд содержится в UNIX Programmer's Manual (Руководство для программиста системы UNIX). Эту документацию можно получить прямо из системы при помощи команды man, которая по заданному в ней имени команды выдаёт её описание из руководства для пользователя [2, 5]. Синтаксис команды

```
man [ том ] заголовок ...
```

Если задан том, man ищет заданные заголовки в этом томе. Том задаётся цифрой – номером тома (например, 3) и, возможно, уточняется одной буквой (например, 3M означает математическую подпрограмму из третьего тома). Если том опущен, man просматривает все тома руководства и выводит первое найденное описание с заданным заголовком (если такое есть). Предпочтение отдаётся описаниям команд по сравнению с описанием подпрограмм из системных библиотек.

Пример использования команды:

```
$ man cal
```

здесь команда man выведет страничку руководства с описанием команды cal (команда cal выводит на экран календарь).

Руководство подразделяется на восемь томов:

1. Команды для пользователей.
2. Системные функции, интерфейс UNIX/C.
3. Библиотечные программы для языка Си, в том числе стандартный пакет ввода-вывода (stdio) и библиотека математических функций.
4. Специальные файлы.
5. Форматы файлов и соглашения о файлах.
6. Игры.
7. Пакеты обработки текстов.
8. Команды и процедуры для администратора системы.

Все описания в руководстве имеют стандартный формат. Описание состоит из нескольких разделов:

1. **Имя** (NAME) – приводится имя команды и её значение.
2. **Резюме** (SYNOPSIS) – содержится краткое описание способа использования команды (её формат). Первое слово обозначает имя команды. Остальные слова обозначают параметры команды. Параметры, начинающиеся со знака минус, часто обозначают опции данной команды. Квадратные скобки обозначают, что заключённые в них параметры можно опускать – необязательные параметры. Многоточие (`...`) указывает, что предшествующий ему параметр можно повторить произвольное число раз.
3. **Описание** (DESCRIPTION) – приводится описание действий, выполняемых командой, и смысл её параметров.
4. **Примеры** (EXAMPLES) – указываются примеры использования команды с различными вариантами параметров.
5. **Диагностика** (DIAGNOSTICS) – приводятся диагностические коды завершения команды, которые могут быть получены при выполнении команды.
6. **Среда исполнения** (ENVIRONMENT) – указываются специфические настройки среды исполнения, при которых может быть выполнена команда.
7. **Файлы** (FILES) – перечисляются используемые командой файлы.
8. **Смотри также** (SEE ALSO) – содержатся ссылки на родственные команды.
9. **Стандарты** (STANDARDS) – перечисляются стандарты, поддерживаемые командой.
10. **История** (HISTORY) – указывается история разработки версий команды.
11. **Недостатки** (BUGS) – перечисляются известные недочёты и ошибки, допущенные при разработке и реализации команды.

Перечисленные здесь разделы являются необязательными, то есть при выводе справочной информации о некоторых командах не все разделы будут представлены.

2.2 Каталоги и файлы

При регистрации пользователя ему назначается администратором собственный каталог пользователя, называемый «домашним»

(Home directory). Кратко рассмотрим структуру файловой системы ОС UNIX и основные команды, используемые для работы с ней.

2.2.1 Файловая система

Файловая система является краеугольным камнем операционной системы UNIX. Она обеспечивает логический метод организации, восстановления и управления информацией. Файловая система имеет иерархическую структуру. Файл, который является основной единицей системы UNIX, может быть: обыкновенным файлом, каталогом, специальным файлом или символическим каналом связи. [2, 5, 9]

- **Обыкновенные файлы** являются набором символов. Они используются для хранения любой информации. Они могут содержать тексты для писем или отчетов, коды пользовательских программ либо команды для запуска программ. Однажды создав обыкновенный файл, пользователь может добавить нужный материал в него, удалить материал из него, либо удалить файл целиком.
- **Каталоги** (каталоги, оглавления, директории) являются суперфайлами, которые могут содержать файлы или другие каталоги. Обычно файлы, содержащиеся в них, устанавливают отношения каким-либо способом. Пользователь может создать каталоги, добавить или удалить файлы из них или удалить каталоги. Все каталоги, которые создаёт пользователь, будут размещены в домашнем каталоге пользователя. Этот каталог назначается системой при входной регистрации пользователя в системе. Никто кроме привилегированных пользователей не может читать или записывать файлы в этот каталог без разрешения пользователя-владельца. Система UNIX также содержит несколько каталогов для собственного использования. Структура этих каталогов аналогична во всех системах UNIX. Этот каталог, включающий в себя несколько системных каталогов, размещается непосредственно под каталогом root (/), который является исходным в файловой структуре UNIX. Все каталоги и файлы иерархически располагаются ниже.
- **Специальные файлы** соответствуют физическим устройствам, таким как терминал, дисковое устройство, магнитная лента или канал связи. Система читает и записывает из/в специальные файлы также как и в обыкновенные файлы. Однако запросы системы

на чтение и запись не приводят в действие нормальный механизм доступа к файлу. Вместо этого они активизируют драйвер устройства, связанный с файлом, приводя, возможно, в действие аппаратуру компьютера, например, головки диска или магнитной ленты.

- **Символические каналы связи** – это файлы, которые указывают на другие файлы.

Структура корневого каталога. Логическая файловая система UNIX организована в виде иерархической структуры, представляющей собой дерево каталогов. Основой любой файловой системы является корневой каталог (обозначается как /). Как правило, корневой каталог имеет следующую структуру (рисунок 2), но администратор системы может изменять эту структуру. [2, 6, 9-11]

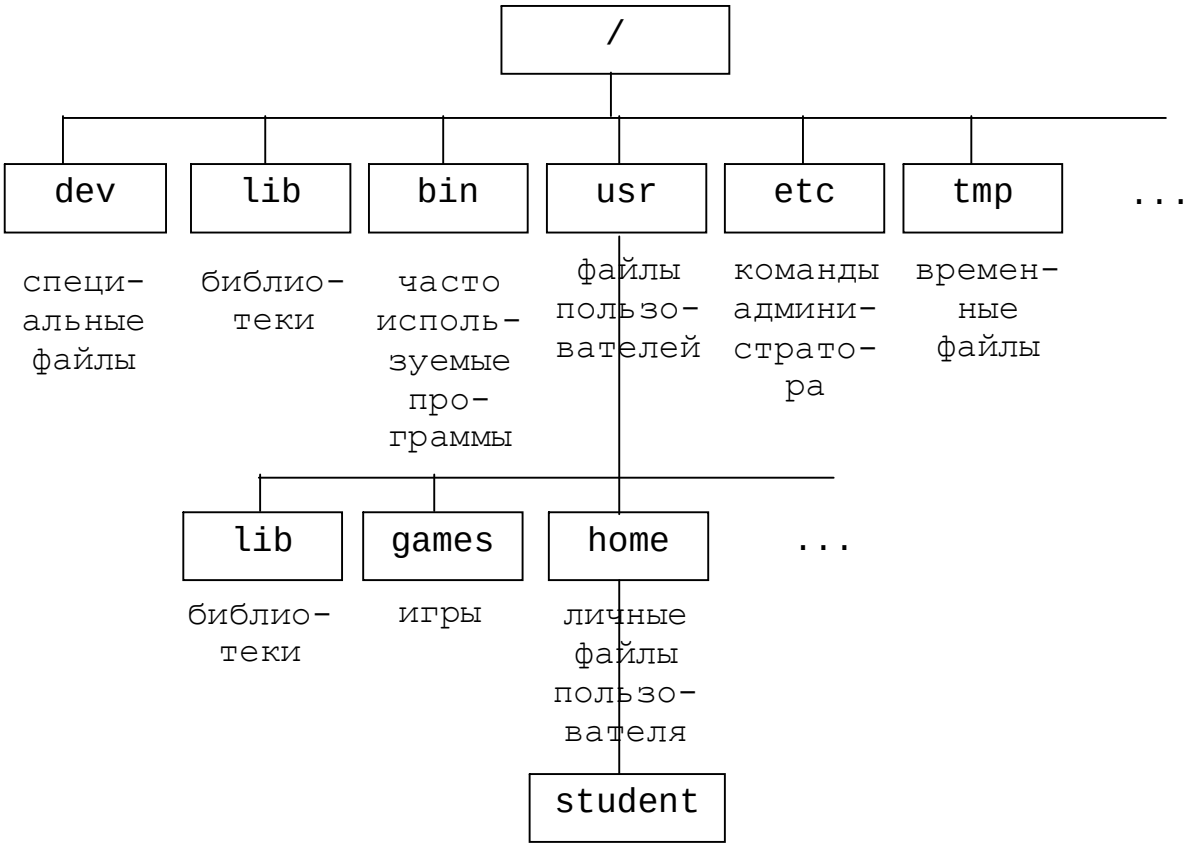
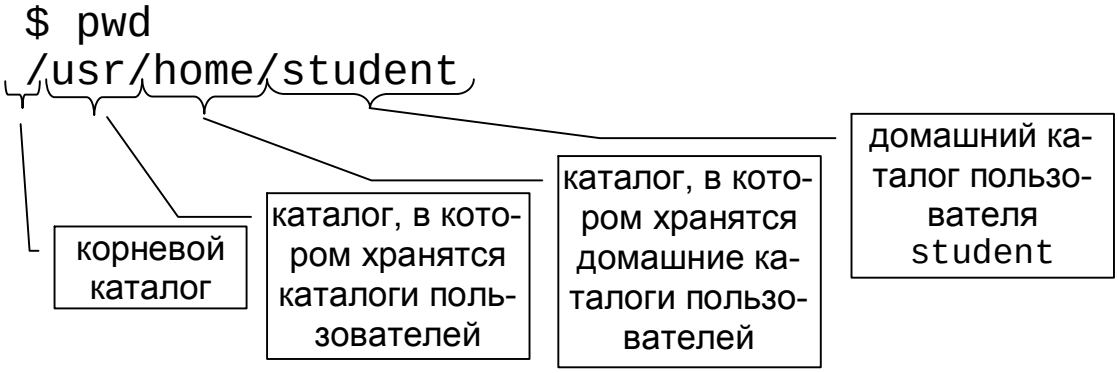


Рисунок 2 – Пример структуры корневого каталога

2.2.2. Основные команды для работы с файлами

Печать рабочего каталога. Узнать полное имя текущего каталога можно с помощью команды pwd:



В данном случае, команда pwd сообщает полное имя текущего рабочего каталога, который совпадает с «домашним» каталогом пользователя с именем student. Каждый файл и каталог системы UNIX идентифицируется уникальным именем пути. Имя пути показывает местоположение файла или каталога и обеспечивает направление поиска его. Существует 2 типа имени пути: полное и родственное.

Полное имя пути (иногда называемое абсолютным именем пути) даёт направление, которое начинается в каталоге root и показывает путь далее по уникальной последовательности каталогов к конкретному каталогу или файлу. Отличительной особенностью полного пути является то, что это имя начинается с символа косая черта (/). Последнее имя в полном имени пути может быть либо именем файла, либо именем каталога. Все другие имена пути должны быть каталогами.

Родственное имя пути даёт направления, которые начинаются в текущем рабочем каталоге и ведут вверх или вниз через серию каталогов к конкретному файлу или каталогу. Двигаясь вниз из текущего каталога, можно получить доступ к своему файлу или каталогу. Двигаясь вверх из текущего каталога, можно пройти через родительские каталоги к родителю всех системных каталогов, то есть к root.

Родственное имя пути начинается с имени каталога или файла.

Одна точка (.) означает текущий каталог, две точки (..) – каталог, непосредственно находящийся выше текущего каталога в иерархии файловой системы. Каталог, представленный двумя точками, называется родительским для каталога, обозначенного одной точкой.

Например, пользователь находится в каталоге student, который содержит каталоги draft, letters и файл file1. Родственным именем пути для каждого из них является просто его имя.

Каталог draft, принадлежащий каталогу student, содержит файлы outline и table. Родственное имя пути от student к файлу outline:

```
draft/outline
```

Чтобы подняться к родителю текущего каталога, можно ввести две точки (..). Это означает, что если пользователь находится в каталоге draft, то .. является именем пути к student, и ../.. является именем пути к родительскому каталогу для student, т.е. к home.

Правила именования каталогов и файлов. Пользователь может давать каталогам или файлам любые имена в соответствии со следующими правилами:

- допустимы все символы, за исключением /;
- некоторые символы лучше не использовать, такие как пробел, табуляция и следующие: ? " # $ ^ ( ) ; < > [ ] | \ * @ ' ~ &. Если необходимо воспользоваться символами пробел или табуляция в имени файла или каталога, то пользователь должен заключить имя в двойные кавычки в командной строке;
- необходимо избегать использования знаков +, - или . в качестве первого символа в имени файла;
- операционная система UNIX различает заглавные и строчные символы.

Печать содержимого каталога. Все каталоги в файловой системе имеют информацию о содержащихся в них файлах и каталогах, такую как: имя, размер и дата последней модификации. Можно получить эту информацию о каталогах, используя команду ls. Команда ls перечисляет имена всех файлов и подкаталогов в указанном каталоге. Если каталог не указан, то команда ls напечатает информацию о файлах и каталогах в текущем каталоге. Чтобы напечатать имена файлов и подкаталогов в каталоге, отличном от текущего без перехода из текущего каталога, необходимо указать имя каталога.

Синтаксис команды:

```
ls [опции] [имя_пути]
```

Параметр имя_пути может быть либо полным именем пути требуемого каталога, либо родственным.

```
$ ls            - печать текущего каталога;
$ ls /          - печать корневого каталога;
```

Часто используемые ключи команды `ls`. Команда `ls` может содержать ключи, которые перечисляют специфичные атрибуты файла или подкаталога. Рассмотрим часто используемые опции:

Распечатать содержание в расширенном формате

Ключ `-l` позволяет отобразить содержание каталога в расширенном формате. Этот формат включает в себя: режим доступа, число связей, имя владельца, название группы, размер в байтах и время последней модификации каждого файла. Например, команда:

`$ ls -l /    - печать корневого каталога полная.`

Выдаст на экран:

```
total 49
drwxr-xr-x   2 root  wheel  1024 Feb 27 16:56 bin
drwxr-xr-x   5 root  wheel   512 Feb 27 17:29 boot
dr-xr-xr-x   4 root  wheel   512 Apr 24 14:35 dev
drwxr-xr-x  17 root  wheel  2048 Apr 12 09:36 etc
drwxr-xr-x   3 root  wheel  1024 Feb 27 16:56 lib
drwxr-xr-x   2 root  wheel   512 Nov  5  2004 mnt
drwxrwxrwt   4 root  wheel   512 Apr 24 14:35 tmp
drwxr-xr-x  17 root  wheel   512 Feb 27 17:47 usr
drwxr-xr-x  22 root  wheel   512 Apr 24 14:35 var
...
```

Необходимо отметить, что здесь вывод команды `ls` показан не полностью. Рассмотрим вывод этой команды более подробно.

Первая выводная строка (`total 49`) показывает объём дискового пространства в байтах. Последующие строки дают представление о каталогах и файлах в указанном каталоге (в данном случае `/`). Первый символ в каждой строке (`d`, `-`, `l`, `b` или `c`) указывает тип файла:

- `d` – каталог
- `-` – обыкновенный файл
- `l` – символическая связь (канал)
- `b` – специальный блочный файл
- `c` – специальный символьный файл.

Следующие девять символов, которые являются либо буквами (`r`, `w`, `x`), либо дефисами, идентифицируют право на чтение/запись и использование файла или каталога.

Далее следует цифра – счётчик связей. Для каталога этот счётчик показывает число каталогов, расположенных в иерархии под ним, плюс два (для самого каталога и каталога-родителя).

Следующим является регистрационное имя владельца файла (в данном случае `root`), и за ним – групповое имя файла или каталога (`wheel`).

Следующее число показывает длину файла или каталога в байтах. Месяц, день и время последней модификации файла – в предпоследней колонке. В последней колонке представлено имя каталога или файла.

Перечислить все файлы в каталоге

Имена некоторых файлов начинаются с точки (например, `.profile`). Когда имя файла начинается с точки, файл не включается в список, распечатываемый командой `ls`. Чтобы распечатать такие файлы, необходимо использовать ключ `-a`.

Например, команда:

```
$ ls –a
```

Наряду с другими файлами текущего каталога покажет и файлы:

```
.           - ссылка на текущий каталог
..          - ссылка на родительский каталог
.profile    - файл, используемый системой в
              служебных целях
```

и другие.

Распечатать содержание в укороченном формате

Ключи `-C` и `-F` команды `ls` используются достаточно часто. Вместе эти ключи распечатывают подкаталоги и файлы каталогов и помечают исполняемые файлы символом `*`, каталоги – символом `/`, символическую связь (каналы) – символом `@`.

Так, самостоятельно проанализируйте вывод команды

```
$ ls –F
```

выделив из полученного списка каталоги, исполняемые файлы и обычные файлы.

Смена рабочего каталога. Изменение рабочего каталога производится командой `cd`.

Синтаксис команды:

```
cd имя_пути_нового_каталога
```

Любое допустимое имя пути (полное или родственное) может использоваться в качестве аргумента команды cd. Указанный каталог, в случае успешного перехода, становится текущим каталогом.

Например, чтобы перейти из каталога student в подчинённый ему каталог work, введите команду cd work. Проверить новое местоположение можно, введя команду pwd. Экран терминала будет выглядеть следующим образом:

```
$ cd work
$ pwd
/usr/home/student/work
$
```

То есть этот каталог стал текущим.

В команде cd, наравне с родственным именем пути, можно использовать и полное имя. Например, чтобы перейти из каталога work обратно, необходимо использовать команду:

```
cd /usr/home/student
```

или

```
cd ..
```

Приведём ещё несколько примеров работы с каталогами:

```
$ cd /etc
$ ls –l        - печать команд администратора
$ cd /usr
$ ls -l bin    - редко используемые команды
$ cd           - без параметров - возврат к собственному домашнему каталогу (/usr/home/student).
```

Создание каталога пользователем. Домашний каталог пользователя, являющийся корнем его ветви файловой системы, создаётся администратором операционной системы при создании учётной записи пользователя. Пользователь может создать нижележащие каталоги командой mkdir.

Синтаксис команды:

```
mkdir имя_нового_каталога(ов)
```

Например, подкаталог draft создаётся при помощи следующей команды, выдаваемой из каталога student (/usr/home/student):

```
$ mkdir draft
```

Если необходимо создать несколько каталогов одновременно, то можно перечислить имена вновь создаваемых каталогов через пробел, так команда:

```
$ mkdir draft1 letters bin
```

создаст три каталога (draft1,letters и bin).

Если необходимо построить иерархический список каталогов, то надо перейти к подкаталогу с помощью команды cd и в нём построить дополнительные подкаталоги.

Печать содержимого файла. В системе UNIX существует несколько команд для распечатки содержимого файлов: cat, more (less), pr:

- Команда cat выводит содержимое файла на экран терминала или в другой файл или новую команду.
- Команда more особенно полезна, если необходимо прочитать содержимое большого файла, так как она отображает текст файла постранично.
- Команда pr форматирует указанные файлы и отображает на терминал или направляет вывод на печать.

Команда cat. Отобразить содержимое указанного файла или файлов на терминал (стандартный вывод, обычно, дисплей) можно с помощью команды cat.

Синтаксис команды:

```
cat [-u] имя_файла ...
```

Команда cat последовательно читает каждый файл (если необходимо вывести на экран содержимое нескольких файлов, то их перечисляют в командной строке через пробел) и записывает их в файл стандартного вывода. Если в качестве стандартного файла вывода задан не терминал, то осуществляется поблочная буферизация выводимых данных, иначе – построчная буферизация. Опция –u отменяет всякую буферизацию.

Пример:

```
$ cat /etc/motd  - посмотреть файл (message of today)
FreeBSD 5.3-RELEASE (GENERIC) #0: Fri Nov  5 04:19:18
UTC 2004

Welcome to FreeBSD!
```

и далее.

Этот файл, создаваемый администратором для текущих объявлений, обычно печатается автоматически при входе в систему (`login`).

Необходимо отметить, что рассматриваемая команда выводит содержимое указанных файлов на стандартный вывод, который обычно является дисплеем. Пользователь всегда может воспользоваться переадресацией ввода-вывода и перенаправить, например, вывод команды `cat` не на дисплей, а в файл. Рассмотрим подробнее возможности переадресации ввода-вывода.

Переадресация ввода-вывода. Существует три направления ввода-вывода:

- стандартный ввод (`stdin`, дескриптор 0),
- стандартный вывод (`stdout`, дескриптор 1),
- стандартный протокол (`stderr`, дескриптор 2).

Как правило, команды берут исходные данные из стандартного ввода и помещают результаты в стандартный вывод.

Стандартные ввод, вывод и протокол можно переназначить. Делается это с помощью следующих структур:

1. Для переназначения стандартного ввода (дескриптор файла 0):

```
< имя_файла
```

(теперь стандартный ввод будет браться из указанного файла, то есть те данные, которые находятся там, будут считаться, грубо говоря, набранными на клавиатуре).

2. Для переназначения стандартного вывода (дескриптор файла 1):

```
> имя_файла
```

(теперь результат выполнения команды будет записан в указанный файл; если он не существует, то будет создан).

3. Многострочный стандартный ввод:

```
<< строка
```

(ввод происходит со стандартного ввода, пока не встретится указанная `<строка>` (не включая её) или конец файла; если `<строка>` является пустой цепочкой литер, то вводимый текст завершается пустой строкой).

4. Для стандартного вывода:

```
>> имя файла
```

(если файл существует, то выводимая информация добавляется в конец этого файла, иначе файл создаётся).

5. Указание стандартного ввода с помощью дескриптора:

```
<&  цифра
```

(в качестве стандартного ввода объявляется файл, ассоциированный с дескриптором <цифра>).

6. Аналогично для стандартного вывода:

```
>&  цифра
```

(в качестве стандартного вывода объявляется файл, ассоциированный с дескриптором <цифра>).

7. Закрытие стандартного ввода:

```
<&-
```

8. Закрытие стандартного вывода:

```
>&-
```

Если любой из этих конструкций предшествует цифра, то с указанным файлом будет ассоциирован дескриптор, равный указанной цифре, вместо 0 и 1 по умолчанию. Например,

```
2>имя_файла
```

при выполнении команды диагностическое сообщение будет направляться в указанный файл, следовательно:

```
2>&1
```

совместит файлы стандартного вывода и диагностики при выполнении команды.

Таким образом, команда

```
$ cat /etc/motd > myfile1
```

не выведет содержимое файла `motd` на экран, а запишет его в файл с именем `myfile1`, причём содержимое последнего будет переписано.

Также с помощью переадресации ввода-вывода возможно создать новый файл, например, команда:

```
$ > myfile2
```

создаст пустой файл с именем `myfile2`.

Важен порядок переназначения: командный интерпретатор производит переназначение слева направо по указанному списку.

Команда `more`. Команда `more` позволяет распечатывать содержимое файла или файлов на терминал. Синтаксис команды:

```
more имя-файла(ов)
```

Команда последовательно выводит указанные файлы (имена файлов указываются через пробел) на стандартный терминал. Основной особенностью данной команды является возможность вывода на терминал файлов частями – поэкранно. После того, как more отобразит страницу текста, она напечатает процент вывода файла. Для продолжения вывода части файла необходимо нажать клавишу Enter (для вывода следующей строки файла) или клавишу Пробел (для вывода следующий страницы файла). Нажатие клавиши b возвращает текст на один экран назад.

Например, рассмотренный ранее вывод файла /etc/motd с помощью команды cat имеет один недостаток, заключающийся в невозможности отображения всего файла на экране компьютера (отображается только концовка файла). Таким образом, данный файл лучше отобразить на экране с помощью команды more следующим образом:

```
$ more /etc/motd
```

Улучшенной версией команды more является команда – less, позволяющая отображать содержимое файла на экране компьютера с возможностью прокрутки. С помощью команды man выведите справочную информацию о данной команде и самостоятельно ознакомьтесь с её возможностями и способами применения.

Команда pr. Команда pr используется для форматирования и печати содержимого файла. Она форматирует заголовки, количество страниц и печатает файл на экране терминала. Синтаксис команды:

```
pr [ опция ] [ файл ] ...
```

Если не указана ни одна из допустимых опций, то команда pr сформирует вывод в одну колонку, страница будет содержать 66 строк, и тексту будет предшествовать короткий заголовок. Заголовок состоит из 5 строк: две пустые строки; строка, содержащая дату, время, имя файла и номер страницы; две пустые строки. Возможные опции команды представлены в таблице 2. Опции применяются ко всем выводимым на терминал файлам, но их можно переустанавливать, задавая между файлами.

Таблица 2 – Опции команды pr

Опция	Функция
-число	Выполнять вывод в указанное количество колонок.
+число	Пронумеровать страницы, начиная с указанного номера.
-h	Использовать следующий параметр в качестве заголовка страниц.
-w число	Установить ширину страницы равной указанному числу литер вместо 72 по умолчанию.
-f	Использовать для перехода на новую страницу литеры перевода формата вместо литер новой строки.
-l число	Установить длину страницы равной указанному числу строк вместо 66 по умолчанию.
-t	Не выводить пятистрочные «шапку» и «концевик», обычно сопровождающие каждую страницу.
-s символ	Отделять колонки одной указанной литерой вместо подходящего количества пробелов. Если символ опущено, используется литера табуляции.
-m	Печатать все файлы одновременно, каждый в своей колонке.

Команда pr часто используется с командой lp для получения копии текста на бумаге в том виде, в каком он был введён в файл.

Просмотр начала файла. Команда head печатает в стандартный вывод начало указанного файла. Синтаксис команды:

```
head [-n число-строк | -c число-байт] [имя_файла ...]
```

По умолчанию команда head печатает 10 первых строк. Например:

```
$ head /etc/motd
```

выведет на экран терминала десять первых строк файла /etc/motd. Если необходимо вывести другое число строк, то его надо указать с помощью опции -n. Например, следующая команда напечатает первых три строки файла:

```
$ head –n 3 /etc/motd
```

С помощью опции -c можно указать необходимое для вывода количество байт файла, например:

```
$ head –c 3 /etc/motd
```

выведет только три первых символа данного текстового файла.

Если при вызове команды передать несколько имён файлов, то вывод каждого будет начинаться с конструкции

`==> имя_файла <==`

Просмотр конца файла. Команда `tail` печатает конец указанного файла. Синтаксис команды:

```
tail [-F | -f | -r] [-b число | -c число | -n число] [имя_файла ...]
```

По умолчанию команда `tail` печатает 10 последних строк. Например:

```
$ tail /usr/src/README

usr.bin         User commands.

usr.sbin        System administration commands.


For information on synchronizing your source tree with one or more of
the FreeBSD Project's development branches, please see:

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/synching.html
```

Явно можно задать количество (со знаком `-`) или номер строки, от которой печатать до конца (со знаком `+`):

```
$ tail -3 /usr/src/README  - три последние строки;
$ tail +6 /usr/src/README  - последние строки, начиная с 6-й
```

Если задать опцию `–r`, то команда выведет строки с конца файла в обратном порядке:

```
$ tail -r /usr/src/README
```

Определение типа файла. Для определения типа файла используется команда `file`. При использовании этой команды файлы, тип которых надо узнать, перечисляются через пробел в командной строке после команды, например:

```
$ file /bin/ls /usr/bin etc/passwd usr/include/stdio.h
/bin/ls: pure executable                - исполняемый;
/usr/bin: directory                     - каталог;
/etc/passwd: ascii text                 - текст в коде ASCII;
/usr/include/stdio.h:C program text     - текст C-программы.
```

Копирование файлов. При работе с системой может появиться необходимость сделать копию файла. Команда `cp` полностью копирует содержимое одного файла в другой. Также она позволяет скопировать один или более файлов из одного каталога в другой, оставив оригинал файла на прежнем месте. Синтаксис команды.

```
cp старый_файл новый_файл
```

или

```
cp файл ... оглавление
```

Содержимое `старый_файл` копируется в `новый_файл`. Если `новый_файл` уже существует, его полномочия и владелец не изменяются. В противном случае полномочиями `новый_файл` становятся полномочия файла `старый_файл`.

Команда `cp` во второй форме копирует один или несколько файлов в заданное оглавление, сохраняя их исходные имена. Подготовленные к копированию файлы должны быть перечислены через пробел.

Команда `cp` отказывается копировать файл в себя.

Например, командой

```
$ cp /usr/src/README /usr/home/student
```

будет произведено копирование файла `README` в домашний каталог пользователя. А с помощью команды

```
$ cp /usr/src/README /usr/home/student/README.new
```

будет произведено копирование файла `README` в новый файл `README.new`, находящийся в домашнем каталоге пользователя.

При копировании и других операциях, связанных с групповыми операциями над файлами и каталогами, удобно для задания групп имён файлов использовать метасимволы. Рассмотрим принципы работы с метасимволами подробнее.

Использование метасимволов. Метасимволы служат для подстановки любых строк и символов в именах файлов:

- `*` – представляет произвольную строку (возможно, пустую);
- `?` – любой одиночный знак;
- `[C1 - C2]` – любая литера из диапазона `C1` – `C2` (в стандарте `ASCII`). Пара символов, разделенных знаком `–`, означает любой символ, который находится между ними, включая и их самих. Если первым символом после «`[`» идёт «`!`», то указанные символы не должны входить в имя файла.

Например:

```
$ cat g*          напечатает все файлы каталога,
                  начинающиеся с «g»;
$ cat *g*         напечатает все файлы, содержащие «g»;
$ cat program.?   напечатает файлы данного каталога с
                  однобуквенными расширениями, например
                  «program.c» или «program.o», но не
                  напечатает «program.com»;
$ cat [a-d]*      напечатает файлы, которые начинаются
                  с «a», «b», «c», «d». Аналогичный
                  эффект дадут и команды «cat [abcd]*»
                  и «cat [bdac]*».
```

Перенос и переименование файлов. Команда mv позволяет пользователю переименовать файл в том же каталоге или переместить его из одного каталога в другой. Когда перемещают файл в другой каталог, то можно переименовать его или оставить прежнее имя. Синтаксис команды:

```
mv старый_файл новый_файл
```

или

```
mv файл ... оглавление
```

Команда mv пересылает (переименовывает) старый_файл в новый_файл. Если новый_файл уже существует, он предварительно исключается. Если право записи в новый_файл отсутствует, mv выводит полномочия этого файла и спрашивает подтверждение переноса. Например, скопируем файл /usr/home/student/README.new в файл /usr/home/student/README.old, изменим права доступа файла README.new и переместим README.old обратно:

```
$ cp README.new README.old
$ chmod 444 README.new    - только чтение
$ mv README.old README.new
README.new: mode 444? y
```

Команда при подтверждении перемещения считывает одну строку из стандартного файла ввода. Если эта строка начинается с буквы y, то пересылка (переименование) выполняется; иначе mv заканчивает работу.

Команда mv во второй форме пересылает один или несколько файлов в заданное оглавление, сохраняя их имена. Некоторые версии этой команды позволяют переименовывать оглавления.

Команда mv отказывается пересылать файл в себя.

Удаление файлов и каталогов. Удаление ненужных каталогов и файлов возможно с помощью двух команд: rmdir (для удаления пустых каталогов); rm (для удаления файлов и каталогов).

Уничтожение пустого каталога.

Для удаления пустых каталогов используется команда rmdir. Синтаксис команды:

```
rmdir имя(имена) каталога(ов)
```

Для удаления нескольких каталогов необходимо указать их имена в командной строке.

Командой rmdir нельзя удалить каталог, если пользователь не является его владельцем или он не пустой. Если одному пользователю необходимо удалить файл из каталога другого пользователя, то владелец должен дать право на запись для родительского каталога этого файла.

Если попытаться удалить каталог, в котором содержатся подкаталоги и/или файлы, то команда rmdir напечатает сообщение:

```
rmdir: имя-каталога not empty
```

Например, если попробовать удалить каталог /usr/home/student/temp1, который содержит два файла README и README.new, то на терминале появится сообщение:

```
$ rmdir temp1
rmdir: temp1 not empty
```

Таким образом, сначала необходимо удалить всё содержимое каталога, а затем удалять его. Так каталог /usr/home/student/temp уже пуст, поэтому команда:

```
$ rmdir temp
```

удалит указанный каталог.

Удаление файлов

Команда rm исключает из каталога одну или несколько ссылок на файлы и удаляет файлы. Синтаксис команды:

```
rm [ опция ] файл ...
```

Если исключаемая ссылка была последней ссылкой на файл, файл уничтожается. Для исключения файла из каталога необходимо иметь право на запись в этот каталог, но не требуется права ни на чтение, ни на запись в сам файл.

Если право на запись в файл отсутствует, а стандартным файлом ввода является терминал, на него выводятся полномочия файла

и считывается одна строка. Если эта строка начинается с символа y, файл уничтожается, иначе – сохраняется. Если задана опция -f, вопросы не задаются и сообщения не выводятся.

Если указанный файл является каталогом, то кроме случая, когда задана опция -r, выводится сообщение об ошибке. Если задана опция -r, rm рекурсивно уничтожает содержимое всего указанного каталога и его самого.

Так, например, команда

```
$ rm -rf /usr/home/student/temp1
```

уничтожит все файлы, входящие в каталог temp1, а затем удалит и сам каталог, не выдавая на терминал никаких сообщений.

Если задана опция -i, команда rm спрашивает про каждый файл, требуется ли его исключить, и про каждый каталог (если задана опция -r), требуется ли его просматривать. Например:

```
$ rm -i file1 file2
file1 : n              (no - нет)
file2 : y              (yes - да)
```

Пустая опция (знак минус) указывает, что все следующие параметры следует считать именами файлов. Это позволяет задавать имена файлов, начинающиеся со знака минус.

2.2.3 Управление правами доступа к файлам

Операционная система UNIX является многопользовательской ОС, что означает наличие средств защиты ресурсов (файлов) одного пользователя от другого. Каждому файлу соответствует набор прав доступа, представленный в виде девяти битов режима. Он определяет, какие пользователи имеют право читать файл, записывать в него данные или выполнять его. Вместе с другими тремя битами, влияющими на запуск исполняемых файлов, этот набор *образует код режима доступа к файлу*. [2, 5, 6, 9, 10]

Владелец файла и защита файла. Каждый файл и каталог имеют владельца – обычно это пользователь, создавший их в первый раз. Владелец может затем назначить защиту файла со стороны трёх классов пользователей:

- владелец (сам);
- группа – пользователи этой же группы, где владелец;
- остальные – все, имеющие доступ к системе.

Каждый файл имеет три вида разрешения на доступ:

- чтение (r) – можно читать (смотреть) содержимое файла или каталога (читать с ключом -l в ls);
- запись (w) – можно менять содержимое файла или каталога (создавать или удалять файлы в каталоге);
- выполнять (x) – использовать файл как команду UNIX и искать в каталоге.

Все комбинации трёх видов разрешения доступа для трех классов пользователей (9 комбинаций) записываются в формате:

```
(если все права есть)
  rwx                 rwx                 rwx      или 777
   ¦                   ¦                   ¦
Владелец               ¦                   ¦
                     Группа                ¦
                                        Остальные
```

Отсутствие права доступа указывается минусом:

```
r--r--r-      или 444
```

Пример:

```
$ ls -l /bin
-r-xr-xr-x     1 bin  2005  Nov.26    12:00 ar
   ...
```

Эта команда показывает режимы доступа.

Установка и изменение режима доступа к файлу. Изменить существующие права можно с помощью команды chmod. Синтаксис команды chmod для установки режима:

```
chmod [-fR] <режим> <файлы>
```

Пример:

```
$ chmod 644 f1 f2 f3
```

где 644 соответствует rw-r--r--. А сама команда устанавливает указанные права для трёх файлов: f1, f2, f3.

Также для изменения прав доступа к файлам и оглавлениям можно воспользоваться другой формой команды chmod. Синтаксис команды chmod для изменения режима:

```
chmod [-fR] <изменения> <файлы>
```

В изменениях используются обозначения:

```
u – user        (установка для пользователя-владельца)
g – group       (установка для членов группы пользователя)
o – other       (установка для остальных пользователей)
a – all         (установка для всех групп)
r – read        (право на чтение)
```

```
w – write       (право на запись)
x – execute     (право на исполнение или поиск)
= - назначить   (назначить права вновь)
+ - добавить    (добавить новые права)
- - отнять      (исключить старые права)
```

Например, имеется какой-то каталог с тремя файлами `f1 f2 f3`:

```
$ ls -l
-r-------- ... f1
-r-------- ... f2
-r-------- ... f3
$ chmod  a = r, u + w  f1 f2 f3
```

или (эквивалентный вариант изменения прав доступа)

```
$ chmod  u = rw, go = r  f1 f2 f3
$ ls -l
-rw-r--r-- ... f1
-rw-r--r-- ... f2
-rw-r--r-- ... f3
$ chmod o-r f1 f2 f3
$ ls -l
-rw-r----- ... f1
-rw-r----- ... f2
-rw-r----- ... f3
```

Другие пользователи, не входящие в группу, потеряли право читать файлы.

У команды `chmod` можно использовать две опции:

- `-f` – означает, что команда не будет сообщать о невозможности установки прав доступа;
- `-R` – предполагает, что данное изменение прав доступа будет применяться рекурсивно для всех подкаталогов, указанных в списке файлов.

Владелец файла, а также суперпользователь могут изменять владельца и группу-владельца файла с помощью команды `chown` со следующим синтаксисом:

```
chown [-h] [-R] <владелец>[:<группа>] <файл> ...
```

Для изменения только группы, владеющей файлом, предназначена команда:

```
chgrp [-h] [-R] <группа> <файл> ...
```

2.2.4 Команды обработки информации

Подсчёт строк, слов и символов. С помощью команды wc можно подсчитать число строк, слов и символов в указанном файле [2, 5, 9, 10]. Если указано более одного файла в командной строке, то программа wc осуществляет подсчёт строк, слов и символов в каждом файле и затем выдаёт общее число. Словами считаются цепочки литер, разделённые литерами типа пробел, табуляция, новая строка. С помощью ключей можно указать или подсчёт только строк, или только слов, или символов.

Синтаксис команды:

```
wc [ –lwc ] [ файл ] ...
```

Если задана опция, начинающаяся с одной из букв lwc, то буквы l, w, c в ней задают подсчёт строк, слов и литер соответственно. По умолчанию берётся -lwc.

Система отвечает строкой в следующем формате:

```
l       w       c       имя_файла
```

где l – число строк в файле;

w – число слов в файле;

c – число символов в файле.

Например, чтобы подсчитать число строк, слов и символов в файле /usr/home/student/temp/README, необходимо использовать команду:

```
$ wc README
      80     380    2749 README
```

Система отвечает, что в файле README 80 строк, 380 слов и 2749 символов.

Чтобы подсчитать число строк, слов и символов в нескольких файлах, используйте следующий формат:

```
wc файл1 файл2
```

Система отвечает следующим образом:

```
l       w       c       файл1
l       w       c       файл2
l       w       c       total
```

Число строк, слов и символов для файл1 и файл2 отображается на отдельных строках. На последней строке отображается общее число строк, слов и символов в двух файлах.

Например, подсчитаем число строк, слов и символов в файлах README и myfile1, который заранее создадим. Экран будет выглядеть следующим образом:

```
$ cal > myfile1          -создадим файл myfile1
$ wc README myfile1
      80     380    2749 README
       8      39     145 myfile1
      88     419    2894 total
$
```

Поиск различий в файлах. Команда diff обнаруживает и сообщает обо всех различиях между двумя файлами и указывает, как изменить первый файл, чтобы он был дубликатом второго. [2, 5, 9, 10]

Синтаксис команды:

```
diff [ опция ] файл1 файл2
```

Команда diff выводит те строки файлов, которые нужно изменить для того, чтобы привести файлы в соответствие друг с другом. За исключением редких случаев, diff находит минимальный достаточный набор различий. Если файл1 или файл2 задан знаком минус, то используется файл стандартного ввода. В стандартном формате выводимые строки имеют вид

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

Номера после буквы относятся к файлу2. Буквы означают: a – добавить строки/символы; d – удалить; c – заменить. Заменяя a на d, и читая наоборот (справа налево), можно также выяснить, как преобразовывать файл2 в файл1. Вслед за каждой такой строкой идут все затрагиваемые заданным в ней действием строки из первого файла с пометкой <, а затем – все затрагиваемые данным действием строки из второго файла с пометкой >.

Опции, доступные при выполнении команды diff, представлены в таблице 3.

Таблица 3 – Опции команды diff

Опция	Функция
-e	Сгенерировать для редактора ed командный файл с запросами a, end, который будет воссоздавать файл2 из файла1
-h	Выполнить задание быстрым, но не очень надёжным методом. Этот метод годится, когда файлы имеют неограниченную длину, а отличия в них невелики и хорошо различаются
-b	Проигнорировать пробелы и табуляции в конце строк; остальные цепочки пробелов и табуляций считать одинаковыми
Примечание: за исключением опции -b, которую можно задавать вместе с любой другой опцией, опции команды diff взаимоисключают друг друга.	

Рассмотрим применение команды diff на примере. Для чего произведём копирование файла README в файлы README1 и README2, отредактируем их с помощью текстового редактора ee, удалив некоторые строки, и сравним эти файлы:

```
$ cp /usr/src/README /usr/home/student/temp/README1
$ cp /usr/src/README /usr/home/student/temp/README2
$ cd /usr/home/student/temp
$ ee README1                  - удаляем некоторые строки
$ ee README2                  - удаляем некоторые строки
$ diff README1 README2
2a3
> $FreeBSD: src/README,v 1.22 2003/03/08 10:01:26 markm
Exp $
36,37d36
< Source Roadmap:
< ---------------
79c78
<               http://www.freebsd.org/doc/en_US.ISO8859-
1/books/handbook/synching.html
---
>
$
```

Поясним полученный вывод команды подробнее. В первой строке (2a3) указано, что вторая строка файла README1 и третья строка файла README2 отличаются; в следующей строке вывода

команды указано, какую строку текста надо добавить в файл README1, чтобы файлы были одинаковыми (направление изменения указано символом >, а характер изменения – символом a в предыдущей строке вывода команды). Далее, три строки вывода команды, начиная с «36,37d36...», указывает: чтобы рассматриваемые файлы стали одинаковыми, надо либо удалить две указанные строки текста из файла README1, либо добавить их же в файл README2, в указанные позиции (строки под номером 36 и 37). И последняя группа указаний, начинающаяся со строки «79c78», показывает, что существует различие в 78 строке файла README2 и в строке 79 файла README1, и указывает какие именно отличия.

Использование текстового редактора ee. При стандартной установке UNIX в системе пользователю доступно несколько текстовых редакторов: ed, vi, ee, – а также форматоры текста nroff и troff. С помощью указанных текстовых редакторов и форматоров можно создать как простые текстовые файлы (например, исходный текст программы или какую-нибудь заметку), так и документы с достаточно сложным форматированием. Основным недостатком редакторов со стороны пользователя является тот факт, что редакторы не поддерживают какой-либо графический интерфейс, а документы создаются без использования технологии создания WYSIWYG (сокращение от *What You See Is What You Get*, англ. *что видишь, то и получишь*, произносится как «*ви-зи-виг*») – способа редактирования, при котором редактируемый материал в процессе редактирования выглядит в точности так же, как и конечный результат. Особенно, данное замечание можно отнести к текстовым редакторам ed, vi и форматорам nroff/troff, для работы с которыми пользователю необходимо знать большой список управляющих команд. Текстовый редактор ee, в отличие от других, представляет пользователю некоторый текстовый интерфейс со списком наиболее часто используемых команд (рисунок 3).

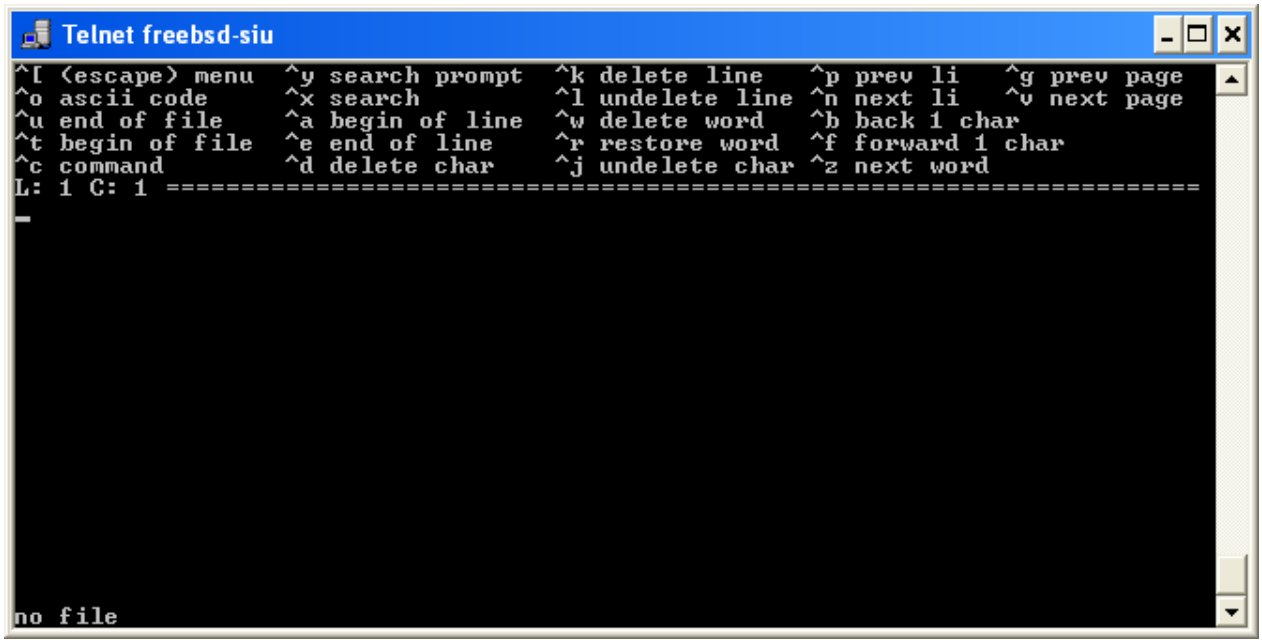


Рисунок 3 – Интерфейс текстового редактора ee

В верхней части окна расположен список управляющих клавиш и краткое описание их действий (таблица 4).

Таблица 4 – Управляющие клавиши

Сочетание клавиш	**Действие**
Esc (escape)	Вызов основного меню; доступны следующие пункты: a) покинуть редактор (если текст не сохранён, то предлагается сохранение); b) вызов помощи по командам и управляющим клавишам; c) вызов подменю для работы с файлами (открытие файла, сохранение файла, печать файла); d) перерисовка экрана; e) настройка параметров редактора; f) вызов подменю поиска в тексте; g) вызов подменю дополнительных возможностей редактора (форматирование текста, проверка правописания, вызов командного интерпретатора)
Ctrl-o	Вставка в текущую позицию указанного ASCII кода
Ctrl-u	Перемещение курсора в конец файла
Ctrl-t	Перемещение курсора в начало файла

Продолжение таблицы 4

Сочетание клавиш	**Действие**
`Ctrl-c`	Вызов консоли для ввода команды
`Ctrl-y`	Вызов консоли для задания шаблона поиска в файле и осуществление поиска
`Ctrl-x`	Поиск в файле по заданному ранее шаблону
`Ctrl-a`	Перемещение курсора в начало строки
`Ctrl-e`	Перемещение курсора в конец строки
`Ctrl-d`	Удаление следующего за курсором символа
`Ctrl-k`	Удаление строки, находящейся за курсором, включая символ конца строки
`Ctrl-l`	Отмена удаления строки
`Ctrl-w`	Удаление слова начиная от текущей позиции курсора до разделительного символа (например, пробела)
`Ctrl-r`	Отмена удаления слова
`Ctrl-j`	Отмена удаления символа
`Ctrl-p`	Перемещение на предыдущую строку текста
`Ctrl-n`	Перемещение на последующую строку текста
`Ctrl-b`	Перемещение на один символ влево
`Ctrl-f`	Перемещение на один символ вправо
`Ctrl-z`	Перемещение на следующее слово
`Ctrl-g`	Перемещение на предыдущий экран
`Ctrl-v`	Перемещение на последующий экран
стрелки	Выводит помощь по командам и управляющим клавишам

С помощью сочетания клавиш `Ctrl-c` пользователь может вызвать консоль для ввода команды (таблица 5).

Таблица 5 – Список команд

Команда	**Действие**
`help`	Вызов помощи по командам и управляющим клавишам
`read`	Открытие файла и добавление в редактируемый файл с текущей позиции курсора
`write`	Сохранение файла

Продолжение таблицы 5

Команда	Действие
exit	Выход из редактора с сохранением сделанных изменений
quit	Выход из редактора без сохранения сделанных изменений
line	Отображение информации о текущей строке текста – номер строки и количество символов в строке
expand	Активация режима замены символа табуляции в тексте на символы пробела
noexpand	Деактивация режима замены символа табуляции в тексте на символы пробела
file	Отображение названия файла
char	Отображение ASCII кода текущего символа
case	Поиск чувствителен к регистру символов шаблона
nocase	Поиск нечувствителен к регистру символов шаблона
!cmd	Выполнение указанной команды командным интерпретатором shell (команду надо указывать вместо cmd)
0-9	Переход на указанную строку текста (можно указывать любой номер, включая двухзначные и более)

Запуск текстового редактора осуществляется с помощью команды ee. Однако при запуске можно указать некоторые опции и имена файлов, которые необходимо открыть в редакторе. Синтаксис команды следующий:

```
ee [+#] [-i] [-e] [-h] [file(s)]
```

где +# – при открытии файла переходит на указанную строку;

-i – отключает окно помощи вверху окна редактора;

-e – при запуске деактивирует режим замены символа табуляции в тексте на символы пробела;

-h – снижает яркость отображаемых символов.

Например, следующая команда запустит текстовый редактор без окна помощи, откроет файл README и переместит курсор на 15 строку:

```
ee +15 -i README
```

Поиск в файле по шаблону. Пользователь может произвести поиск в файле указанного слова, фразы, группы символов с помощью команды `grep` [2, 5, 9, 10]. Поиск осуществляется по шаблону. Синтаксис команды:

```
grep [опции] шаблон файл(ы)
```

Команда `grep` сканирует заданные входные файлы (по умолчанию – файл стандартного ввода) в поисках строк, соответствующих некоторому образцу. Каждая найденная строка, как правило, копируется в файл стандартного вывода (то есть выводится на терминал). Образцы, используемые командой `grep`, являются ограниченными регулярными выражениями. Наиболее часто используемые опции приведены в таблице 6.

Таблица 6 – Опции команды `grep`

Опция	Функция
`-c`	Напечатать только общее число найденных строк
`-e шаблон`	То же самое, что и просто параметр `шаблон`; применяется, если выражение начинается со знака минус
`-i`	При выполнении сравнений игнорировать размер букв. В некоторых версиях системы `UNIX` эта опция называется `-y`
`-l`	Напечатать имена файлов, содержащих строки, сопоставляющиеся с образцом
`-n`	Перед каждой строкой выводить её номер в данном файле
`-s`	Выполнить команду «молча». Никаких сообщений не выводится (кроме сообщений об ошибках)
`-v`	Напечатать все строки, которые не сопоставляются с данным образцом

Если входных файлов несколько, то во всех случаях перед выводимыми строками выводится имя соответствующего файла. Следует соблюдать осторожность при использовании в регулярном выражении литер `$ * ? [ ^ | ( )` и `пробел`, так как они имеют особый смысл и в языке-оболочке. Самое надежное – заключить весь параметр в одиночные кавычки (апострофы).

Например, команда

```
$ grep –n FreeBSD README
```

выведет на экран все строки, содержащие слово FreeBSD в файле README:

```
1:This is the top level of the FreeBSD source directory.  This file
3:$FreeBSD: src/README,v 1.22 2003/03/08 10:01:26 markm Exp $
11:building components (or all) of the FreeBSD source tree, the most
13:everything in the FreeBSD system from the source tree except the
78:the FreeBSD Project's development branches, please see:
```

А команда

```
$ grep FreeBSD README > myfile
```

сохранит те же строки в новый файл myfile, но уже без указания номеров строк.

Если же шаблон поиска будет содержать, например, знак пробела, то весь шаблон надо заключить в одинарные кавычки:

```
$ grep –c ‘source tree’ README
3
```

в данном примере на экран будет выведено число строк (в данном случае 3), в которые входит шаблон поиска.

Сортировка и слияние файлов. Система обеспечивает эффективное средство для сортировки и слияния файлов с помощью команды sort [2, 5, 9, 10]. Синтаксис команды:

```
sort [ опции ] [ +поз1 [ –поз2 ]]...[ опции ] [ имя ] ...
```

Команда sort считывает поименованные файлы, сортирует (упорядочивает) их и помещает результат в файл стандартного вывода. Имя – обозначает файл стандартного ввода. Если не задано ни одного имени входного файла, то сортируется файл стандартного ввода.

Подразумеваемым по умолчанию ключом сортировки является вся строка; подразумеваемым порядком сортировки является лексикографический порядок:

- строки, начинающиеся с цифры, будут отсортированы по цифрам и перечислены после строк, начинающихся с буквы;
- строки, начинающиеся с большой буквы, перечисляются до строк, начинающихся с маленькой буквы;
- строки, начинающиеся с таких символов, как «%», «*» сортируются на основе символьного представления ASCII.

Часто используемые опции, воздействующие на процесс сортировки глобальным образом, представлены в таблице 7. Можно задавать одну или несколько этих опций.

Таблица 7 – Опции команды sort

Опция	Функция
-b	Игнорировать начальные пробелы и табуляции при сравнении полей.
-f	Преобразовывать при сравнениях прописные буквы в строчные.
-r	Изменить порядок сортировки на противоположный.
-t символ	Использовать в качестве разделителя полей (табуляции) символ
-m	Только слияние, входные файлы уже упорядочены.
-o имя	Следующий параметр является именем выходного файла, который будет использоваться вместо файла стандартного вывода. Этот файл может совпадать с одним из входных файлов.
-T имя	Следующий параметр является именем оглавления, в котором следует создавать временные файлы.

Конструкция +поз1 -поз2 делает ключом сортировки поле, начинающееся с позиции поз1 и оканчивающееся непосредственно перед позицией поз2. Оба параметра поз1 и поз2 имеют вид m.n; далее может следовать один или несколько флагов. Число m задаёт количество полей, которое нужно пропустить от начала строки, а n задаёт количество литер, которое требуется пропустить в дополнение к этому. Опущенное n означает 0; опущенное – поз2 означает конец строки.

Когда имеется несколько ключей сортировки, последующие ключи сравниваются, только если все предшествующие ключи равны. Строки, которые при выполнении операции сравнения оказались одинаковыми, упорядочиваются по всем значимым байтам.

Рассмотрим применение команды sort на практике. Допустим, имеется два файла group1 и group2, каждый из которых содержит перечень имён. Необходимо отсортировать каждый список по алфавиту и затем объединить два списка в один. Вначале отобразим содержание файлов, выполнив команду cat для каждого файла. Экран будет выглядеть следующим образом:

```
$ cat group1
```

```
Фамилия Имя     Отчество
Петрова Ольга   Ивановна
Васина  Ирина   Васильевна
Антонов Иван    Михайлович
Сидоров Антон   Петрович
Иванов  Николай Федорович
$ cat group2
Фамилия Имя     Отчество
Орлов   Петр    Григорьевич
Яковлев Роман   Васильевич
Жданов  Олег    Алексеевич
Рыкова  Жанна   Николаевна
$
```

Теперь отсортируем и склеим эти два файла, выполнив команду `sort`:

```
$ sort group1 group2
```

Однако результат выполнения этой команды будет только распечатан на экране терминала. Для того, чтобы отсортированный список был сохранен в файл для дальнейшего его использования, необходимо использовать опцию `–o`:

```
$ sort -o group12 group1 group2
$ cat group12
Фамилия Имя     Отчество
Фамилия Имя     Отчество
Антонов Иван    Михайлович
Васина  Ирина   Васильевна
Жданов  Олег    Алексеевич
Иванов  Николай Федорович
Орлов   Петр    Григорьевич
Петрова Ольга   Ивановна
Рыкова  Жанна   Николаевна
Сидоров Антон   Петрович
Яковлев Роман   Васильевич
$
```

В данном случае отсортированный список будет сохранён в файл с именем `group12`.

В случае если необходимо произвести сортировку, например, по имени, то команда будет выглядеть следующим образом:

```
Шаг 1. Избавимся от строки «Фамилия Имя     Отче-
ство»
$ grep –v ‘Фамилия’ group1 > group1.new
Шаг 2. Избавимся от строки «Фамилия Имя     Отче-
ство»
$ grep –v ‘Фамилия’ group2 > group2.new
Шаг 3. Добавим соответствующую строку в результи-
рующий файл
$ echo ‘Фамилия Имя     Отчество’ > group12
Шаг 4. Сортируем по имени (по второму полю)
$ sort +1 group1.new group2.new >> group12
Шаг 5. Удаляем временные файлы
$ rm group1.new group2.new
Шаг 6. Выведем результирующий файл на экран
$ cat group12
Фамилия Имя     Отчество
Сидоров Антон   Петрович
Рыкова  Жанна   Николаевна
Антонов Иван    Михайлович
Васина  Ирина   Васильевна
Иванов  Николай Федорович
Жданов  Олег    Алексеевич
Петрова Ольга   Ивановна
Орлов   Петр    Григорьевич
Яковлев Роман   Васильевич
```

Поиск и обработка файлов и каталогов. Поиск файлов и каталогов, удовлетворяющих определённым критериям, выполняется стандартными средствами операционной системы [2, 5, 9, 10]. Обычно для этих целей используется команда `find` с соответствующими опциями. Это довольно мощное средство, но имеет и свои ограничения. Синтаксис команды:

```
find список_составных_имен выражение
```

Команда `find` для каждого составного имени, заданного в `списке_составных_имен` (обычно, указывается каталог, из которого будет начата процедура поиска), рекурсивно обходит определяемую этим именем иерархию оглавлений в поисках файлов, удовлетворяющих заданному логическому выражению. Логическое выражение строится из первичных выражений, описания которых приводятся ниже (таблица 8). В этих описаниях параметр `n` исполь-

зуется для указания целого десятичного числа, причем +n означает «больше чем n», -n означает «меньше чем n», а =n означает «в точности n».

Таблица 8 – Опции команды find

Опция	Функция
-name шаблон	Истина, если имя текущего файла совпадает с шаблоном поиска. При использовании метасимволов необходимо маскировать шаблоны
-type тип	Истина, если файл имеет тип, указанный параметром тип. Типы файлов задаются символами b, c, d, f, l, p и s, обозначающими специальное блочное устройство, специальное символьное устройство, каталог, обычный файл, символическую ссылку, именованный канал и сокет
-user имя	Истина, если владельцем файла является пользователь имя, где имя – либо регистрационное имя пользователя, либо его числовой идентификатор
-group группа	Истина, если файл принадлежит группе группа, где группа – либо имя группы, либо её числовой идентификатор
-perm [-] права	Если дефис не задан, то условие истинно, только если права доступа в точности соответствуют указанным (как в команде chmod). Если задан дефис, то условие истинно, если в правах доступа файла, как минимум, установлены те же биты, что и в указанных правах
-links n	Истина, если на файл имеется n ссылок
-size n [c]	Истина, если файл занимает n 512-литерных блоков или символов (если указан суффикс c). Перед размером можно указывать префикс + (не меньше), - (не больше) или = (в точности равен)
-atime n	Истина, если к файлу были обращения в течение последних n дней

Продолжение таблицы 8

Опция	Функция
`-mtime n`	Истина, если файл изменился в течение последних `n` дней
`-ctime n`	Истина, если файл был создан в течение последних `n` дней
`-newer файл`	Истина, если файл – более новый, чем указанный в параметре `файл`
`-ls`	Условие истинно всегда (выдаёт информацию о файле, аналогичную длинному листингу)
`-print`	Условие истинно всегда (выдаёт полное имя файла в стандартный выходной поток)
`-exec команда {} \;`	Истина, если выполненная `команда` имеет код возврата 0. Команда заканчивается замаскированной точкой с запятой. В команде можно использовать конструкцию `{}`, заменяемую полным именем рассматриваемого файла
`-ok команда {} \;`	Аналогично `exec`, но полученная после подстановки имени файла вместо `{}` команда выдаётся с вопросительным знаком и выполняется, если пользователь ввел символ `y`
`-print`	Истина всегда; вызывает печать составного имени текущего файла

В различных версиях ОС `UNIX` могут поддерживаться и другие компоненты выражений в команде `find`. Если командная строка сформирована неправильно, команда немедленно завершает работу.

Рассмотренные в таблице 8 первичные выражения (`p`) можно объединять при помощи следующих операций (в порядке уменьшения приоритета):

- `(...)` – заключённая в скобки группа первичных выражений, связанных операциями (скобки имеют специальное значение для программы-оболочки, которое должно быть отменено);
- `!p` – знак `!` представляет собой унарную операцию отрицания;

- `p p` – конкатенация первичных выражений. Если первичные выражения располагаются друг за другом, подразумевается, что они связаны логической операцией «и»;
- `p -o p` – альтернативные первичные выражения. Операция `-o` означает логическое «или».

Рассмотрим несколько примеров применения команды `find`.

Вначале – простой пример поиска файлов в каком-либо каталоге и вывод содержимого на консоль. Например, чтобы получить содержимое рабочего каталога программы (в нашем примере это `/usr/home/student/temp`), достаточно выполнить команду

```
$ find `pwd` -print
```

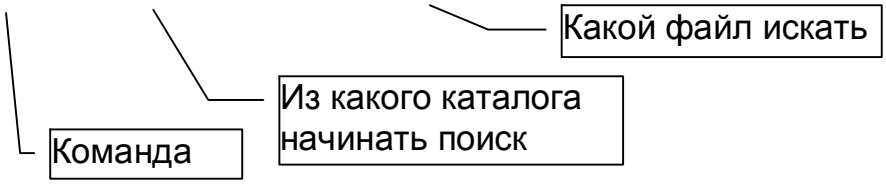
На экране дисплея будет отображено содержимое текущего каталога:

```
/usr/home/student/temp
/usr/home/student/temp/README.new
/usr/home/student/temp/README
/usr/home/student/temp/myfile
/usr/home/student/temp/group1
/usr/home/student/temp/group2
/usr/home/student/temp/group12
/usr/home/student/temp/temp1
/usr/home/student/temp/group.new
```

В приведённом примере команда `pwd` заключается в обратные одинарные кавычки (``); и смысл использования указанной команды в том, что `pwd` выполняется раньше команды `find`, возвращая в результате своей работы наименование текущего каталога, имя которого и подставляется в качестве параметра в команду `find`.

С помощью следующей команды на консоли будут перечислены все файлы с именем `motd`, присутствующие в каталогах ниже по иерархии каталога `/usr`.

```
$ find /usr -name motd
```




```
/usr/share/examples/etc/motd
/usr/src/etc/motd
/usr/src/etc/rc.d/motd
/usr/src/release/picobsd/mfs_tree/etc/motd
```

```
$
```

В следующем примере с помощью команды `find` в указанном каталоге выполняется поиск файлов, начинающихся с `temp` или `READ`:

```
$ find /usr/home/student/temp \( -name 'temp*'
-o -name 'READ*' \)
/usr/home/student/temp
/usr/home/student/temp/README.new
/usr/home/student/temp/README
/usr/home/student/temp/temp1
$
```

Ещё одной, часто используемой опцией является `-size`. Следующий пример демонстрирует поиск в каталоге `usr/home/student/temp` файлов, размер которых превышает 200 байтов, и файлов, размер которых меньше 200 байтов. Однако сначала покажем какие файлы находятся в указанном каталоге:

```
$ ls -l
total 20
-rw-r--r--  1 student  wheel  2749 Apr 26 12:07 README
-rw-r--r--  1 student  wheel  2749 Apr 26 10:20
README.new
-rw-r--r--  1 student  wheel   119 Apr 26 16:34
group.new
-rw-r--r--  1 student  wheel   140 Apr 26 15:56 group1
-rw-r--r--  1 student  wheel   235 Apr 26 17:00 group12
-rw-r--r--  1 student  wheel   116 Apr 26 15:58 group2
-rw-r--r--  1 student  wheel   313 Apr 26 12:13 myfile
-rw-r--r--  1 student  wheel   203 Apr 26 17:00 temp1
$ find usr/home/student/temp -size +200c
/usr/home/student/temp
/usr/home/student/temp/README.new
/usr/home/student/temp/README
/usr/home/student/temp/myfile
/usr/home/student/temp/group12
/usr/home/student/temp/temp1
$ find usr/home/student/temp -size -200c
/usr/home/student/temp/group1
/usr/home/student/temp/group2
/usr/home/student/temp/group.new
$
```

Команда `find` обладает ещё одной возможностью – выполнять команду или группу команд, передавая в качестве параметра

результат поиска файлов. Рассмотрим следующий пример. Пусть в каталоге usr/home/student/temp необходимо удалить все файлы, имя которых начинается с temp. Для выполнения команды служит опция -exec. Команда должна заканчиваться пробелом и символами \;.

```
$ ls                          -узнаем, какие файлы есть
README          group.new       group12         myfile
README.new      group1          group2          temp1
$ find usr/home/student/temp -name 'temp*' -print -exec
rm {} \;
/usr/home/student/temp
rm: /usr/home/student/temp: is a directory
/usr/home/student/temp/temp1
$ ls                          -узнаем, какие файлы остались
README          group.new       group12         myfile
README.new      group1          group2
$
```

Рассмотрим ещё один пример с использованием опции –exec:

```
$ find `pwd` -name 'group*' -exec ls -l {} \;
-rw-r--r--    1  student    wheel    140  Apr  26  15:56
/usr/home/student/temp/group1
-rw-r--r--    1  student    wheel    116  Apr  26  15:58
/usr/home/student/temp/group2
-rw-r--r--    1  student    wheel    235  Apr  26  17:00
/usr/home/student/temp/group12
-rw-r--r--    1  student    wheel    119  Apr  26  16:34
/usr/home/student/temp/group.new
$
```

В данном примере на консоль были выведены атрибуты всех файлов, удовлетворяющих шаблону поиска.

Ещё одной часто используемой опцией является опция –user, с помощью которой можно искать файлы определённых пользователей. Например, следующая команда удалит все файлы пользователя student1:

```
$ find / -user student1 –exec rm {} \;
```

Выбор частей строк файлов. Для выбора и печати на стандартный вывод частей строк файлов используется команда cut. Задание частей строк для последующей выборки может осуществляться пользователем тремя различными способами (в зависимости от выбранного способа используется разный синтаксис команды):

1. Искомая часть задаётся позицией байт в строке:

cut {-b список-байтов, --bytes=список-байтов} [-n] [--help] [--version] [имя-файла(ов)]

2. Искомая часть задаётся номерами символов в строке:

cut {-c список-символов, --characters=список-символов} [--help] [--version] [имя-файла(ов)]

3. Искомая часть задаётся номером поля в строке:

cut {-f список-полей, --fields=список-полей} [-d разделитель] [-s] [--delimiter=разделитель] [--only-delimited] [--help] [--version] [имя-файла(ов)]

Таким образом, с помощью опции -b или --bytes задаётся список байтов, которые необходимо выбрать из каждой строки файлов, указанных в параметре имя-файла(ов). Список задаётся с помощью перечисления диапазонов и/или номеров байт, разделённых запятыми[1], например, команда:

```
$ cut –b 1-3,5 README
```

выведет на экран первый, второй, третий и пятый байты из каждой строки файла README. Так как текст чаще всего представляется с помощью однобайтных символов, то в данном случае на экран будут выведены соответствующие символы. Символы табуляции и backspace (возврат на символ) трактуются подобно другим символам и занимают один байт. В случае представления текста с помощью двухбайтовых символов, например, Unicode, лучше использовать опции -c или -- characters, которые задают список символов:

```
$ cut –c 1-3,5 README
```

Необязательная опция -n при использовании опции -b или --bytes указывает не разбивать на части многобайтовые символы.

С помощью опции -f или --fields задаётся список номеров полей, которые необходимо выбрать из каждой строки файлов, указанных в параметре имя-файла(ов). Например, следующая команда

```
$ cut –f 1 group12 | grep –v "Фамилия" > Surname_group
```

сохранит в файле Surname_group фамилии студентов, содержащиеся в файле group12, созданном при рассмотрении принципов

[1] Байты, символы и поля нумеруются, начиная с 1. Могут задаваться неполные диапазоны: если опустить нижнюю границу (-5), то будет использоваться диапазон (1-5) включительно; если опустить верхнюю границу (5-), то диапазон будет ограничиваться концом строки или последним полем.

работы команды sort. Причём, с помощью команды grep удаляется слово «Фамилия».

Если разделитель полей в файле отличается от умалчиваемого (символ табуляции), то его необходимо задать с помощью опции -d или --delimiter. Например, следующая команда выведет на стандартный вывод (экран терминала) имена пользователей и их домашние каталоги в формате «name:home» из системного файла /etc/passwd, содержащего сведения о всех пользователях, которые могут работать в системе:

```
$ cut -d : -f 1,6 /etc/passwd
```

Опция -s или --only-delimited используется вместе с опцией -f и позволяет не выводить строки, в которых отсутствует разделитель полей.

Опции --help и --version выводят, соответственно, краткую справку о программе и информацию о её версии.

Если в командной строке отсутствуют имена файлов или в качестве имени определен дефис (-), команда cut выполняет обработку стандартного ввода.

Преобразование символов. Для выполнения преобразования, подстановки, замены, сокращения и/или удаления символов, поступающих со стандартного ввода с последующей записью результата преобразования на стандартный вывод используется команда tr, имеющая следующий синтаксис:

```
tr строка1 строка2
```

или

```
tr -d строка1
```

или

```
tr -s строка1
```

Если не указана ни одна опция, то на стандартный вывод будет напечатано содержимое стандартного ввода, в котором все вхождения символов, указанных в параметре строка1, будут заменены на символы, указанные в параметре строка2. Например:

```
$ echo qwerty | tr rw 23
q3e2ty
```

Здесь вывод команды echo подаётся на вход команды tr, которая заменяет все вхождения символов r на символ 2, а w – на 3.

Следующий пример заменит все символы в файле /etc/motd на заглавные:

```
$ tr a-z A-Z < /etc/motd
```

В данном примере для задания набора символов используется диапазон их значений (a-z и A-Z), кроме того параметры строка1 и строка2 можно указывать и с помощью специальных метасимволов. Способы задания диапазонов и метасимволы, которые можно использовать в работе команды tr, представлены в таблице 9.

Таблица 9 – Способы задания диапазонов и метасимволы команды tr

Способ задания	Описание
Задание диапазонов	
символ	Любой символ алфавита (неслужебный символ)
символ1-символ2	Все символы из диапазона от символ1 до символ2 включительно
[символ*]	Указывается в параметре строка2; команда выполняет копирование символ в количестве равном длине параметра строка1
[символ*N]	Указывается в параметре строка2; команда выполняет копирование символ N раз
[=символ=]	Все символы, которые эквивалентны символ
[:alnum:]	Все буквы и цифры
[:alpha:]	Все буквы
[:blank:]	Все горизонтальные символы пробела (пробел, табуляция)
[:cntrl:]	Все управляющие символы
[:digit:]	Все цифры
[:graph:]	Все печатные символы (исключая пробел)
[:lower:]	Все строчные буквы
[:print:]	Все печатные символы (включая пробел)
[:punct:]	Все знаки пунктуации
[:space:]	Все горизонтальные или вертикальные пробелы
[:upper:]	Все прописные (заглавные) буквы

Продолжение таблицы 9

Способ задания	Описание
`[:xdigit:]`	Все шестнадцатеричные цифры
Задание служебных символов	
`\NNN`	Восьмеричное число, состоящее из трёх цифр `NNN` и представляющее любой действительный символ в коде `ASCII`
`\\`	Символ обратной косой черты
`\a`	Символ-звонок (alert)
`\b`	Символ возврата на одну позицию (backspace)
`\f`	Символ прокрутки страницы
`\n`	Символ новой строки
`\r`	Символ возврата каретки
`\t`	Символ горизонтальной табуляции
`\v`	Символ вертикальной табуляции

Таким образом, используя возможность задания диапазона с помощью специальных метасимволов, предыдущий пример можно представить иным способом:

```
$ tr [:lower:] [:upper:] < /etc/motd
```

Однако необходимо отметить, что данные способы изменения регистра символов могут быть использованы только для символов латинского алфавита, а для кириллических символов необходимо указывать диапазон в явном виде. Например, следующая команда заменит все символы в файле `group1` на заглавные, а результат сохранит в файле `group1_upper`:

```
$ tr а-п,р-я А-П,Р-Я < group1 > group1_upper
```

Команда `tr` с указанной опцией `-d` произведёт удаление из стандартного ввода всех символов, совпадающих с символами, заданными в параметре `строка1`. Например, следующая команда удалит из файла `group1` все управляющие символы:

```
$ tr -d [:cntrl:] < group1
```

Команда `tr` с указанной опцией `–s` произведёт замену в стандартном вводе последовательности повторяющихся символов, указанных в параметре `строка1` на один такой символ (иными слова-

ми произведёт удаление всех повторяющихся символов, кроме первого). Например:

```
$ echo qqqqqwwweeertyrq | tr -s [:alpha:]
```

Результатом работы данной команды будет строка

```
qwertyrq
```

В данном примере в команде `tr` указаны все буквы (`[:alpha:]`), таким образом любые буквы со стандартного ввода будут фильтроваться, и удаляться повторяющиеся символы, стоящие вместе.

2.3 Управление дисковым пространством и процессами

2.3.1 Управление дисковым пространством

Очень важной задачей для пользователя и системного администратора является контроль за используемым дисковым пространством. Даже при больших ёмкостях дисковых накопителей рано или поздно наступает момент, когда места на дисках не хватает. Современные программы потребляют, как правило, значительные ресурсы постоянной памяти, и сохраняется устойчивая тенденция к дальнейшему увеличению такого потребления. Даже небольшая система `UNIX` с малым числом пользователей порождает сотни файлов в ходе обычной работы. В процессе программирования можно создавать множество файлов для различных версий пользовательских программ. Поэтому каждый пользователь должен нести ответственность за расход дискового пространства.

Решение этой проблемы во многих случаях оказывается довольно простым – архивация и/или удаление неиспользуемых или редко используемых файлов и удаление «мусора». Для того чтобы отслеживать используемые ресурсы файловой системы, в операционных системах семейства `UNIX` предусмотрен целый ряд команд. Контроль за использованием ресурсов жёстких дисков можно выполнить с помощью системных команд `du` и `df`.

Справка об использовании дискового пространства. В системе `UNIX` существует команда, выдающая отчёт об использовании дискового пространства заданными файлами, а также каждым каталогом иерархии подкаталогов каждого указанного каталога. Здесь под использованным дисковым пространством понимается про-

странство, занятое всей иерархией подкаталогов указанного каталога [2, 6, 10]. Синтаксис команды:

du [опции] [имя ...]

Команда du для каждого заданного имени файла или каталога выдаёт количество блоков, занимаемых на диске этим файлом или всеми файлами и (рекурсивно) всеми подкаталогами данного каталога. Если имена файлов не заданы, выдаётся информация для файлов из текущего каталога.

Команда du не показывает фактический размер файла в байтах, а отображает только количество выделенных блоков или байтов на диске. Размер блока зависит от настроек операционной системы и может варьироваться в широком диапазоне. Часто используют блоки размером в 1 Кбайт. Минимальное дисковое пространство, выделяемое для файла и каталога, обычно составляет 4 блока или (при размере блока в 1 Кбайт) 4 Кбайт. Обычно такую группу блоков называют *кластером*. Если, например, для файла выделено 8 Кбайт дискового пространства, то говорят, что файл занимает 2 кластера. Размер кластера может также варьироваться, хотя обычно он принимается равным 4 блокам. Именно поэтому для файлов, имеющих разные размеры, может быть выделено одинаковое дисковое пространство. Например, для файлов с размерами 3 байта и 3 Кбайт будет выделено на жестком диске одинаковое пространство в 4 Кбайт, то есть 1 кластер.

Запущенная без аргументов, команда du выдаёт отчет о дисковом пространстве для текущего каталога.

Перейдем к рассмотрению опций команды du. Некоторые наиболее часто используемые опции приведены в таблице 10.

Таблица 10 – Опции команды du

Опция	Функция
-s	Выдаётся только суммарный итог для каждого аргумента
-a	Информация выдаётся для каждого встретившегося файла, а не только для каталогов, причём производится рекурсивный обход всех каталогов, заданных в командной строке
–c	Выдаётся общий итог по всем аргументам с конкретизацией по подкаталогам после того, как все аргументы будут обработаны. Этот параметр может

Продолжение таблицы 10

Опция	Функция
	быть использован для выяснения суммарного использования дискового пространства для всего списка заданных файлов и каталогов
`-I` шаблон	При рекурсивном выполнении пропускает каталоги или файлы, чьи имена совпадают с шаблоном. Это шаблон может быть любым файловым шаблоном
`-h`	Добавляет букву размера, например M, для двоичного мегабайта к каждому размеру
`-k`	Выдаёт размеры в бинарных килобайтах (1024 байт)
`-m`	Выдаёт размеры в бинарных мегабайтах (1048576 байт)

Рассмотрим некоторые практические аспекты применения команды du. Команда du без параметров показывает размер дискового пространства, занимаемого файлом или каталогом в целом:

```
$ pwd
/usr/home/student
$ du temp              -узнаем размер каталога temp
2       temp/temp1     -размер подкаталога temp1 (2 блока)
22      temp           -размер каталога temp (22 блока)
$
```

Информация выдаётся в блоках, размер которых определяется установками операционной системы. В приведённом примере информация о занимаемом дисковом пространстве подразумевает размер блока в 1 Кбайт.

Команда du выдаёт информацию в виде `[размер_используемого_дискового_пространства] [полное_имя_файла]`

Если команда du задана с опцией `-c`, то помимо информации по отдельным каталогам выдаётся суммарная величина используемого дискового пространства. Это можно увидеть в следующем примере:

```
$ du -c temp
2       temp/temp1
22      temp
22      total
```

```
$
```

В приведённых примерах в качестве параметров команды du используются имена каталогов. В этих случаях информация отображается в целом для каталога. Для того чтобы просмотреть размер файлов, необходимо ввести полное путевое имя файла. Следующие примеры демонстрируют применение команды du для работы с файлами:

```
$ du /usr/home/student/temp/README
4       temp/README
```

Использование опции –a, также позволяет вывести на экран информацию о файлах, хранящихся в указанном каталоге:

```
$ du -a temp
4       temp/README.new
4       temp/README
2       temp/myfile
2       temp/group1
2       temp/group2
2       temp/group12
0       temp/temp1/file1 -файл подкаталога temp1
2       temp/temp1       -подкаталог temp1
2       temp/group.new   -файл каталога temp
22      temp             -общий размер каталога temp
```

Файлы каталога ***temp***

Команда du, применённая с опцией -I, позволяет исключить из рассмотрения отдельные файлы:

```
$ ls temp
README        group.new      group12        myfile
README.new    group1         group2         temp1
$ du -a -I 'READ*' temp
2       temp/myfile
2       temp/group1
2       temp/group2
2       temp/group12
0       temp/temp1/file1
2       temp/temp1
2       temp/group.new
14      temp
```

в этом примере из результата выполнения команды исключены данные по файлам, удовлетворяющим шаблону READ*. Два таких файла находятся в каталоге /usr/home/student/temp.

Применение опции –h позволяет сделать вывод команды du более понятным для пользователя – после размера файла добавляется буква, характеризующая этот размер (например, K – для обозна-

чения двоичного килобайта). Например, если изменить команду du, используемую в предыдущем примере, добавив опцию –h, получим:

```
$ du -a -I 'READ*' -h temp
2.0K    temp/myfile
2.0K    temp/group1
2.0K    temp/group2
2.0K    temp/group12
  0B    temp/temp1/file1
2.0K    temp/temp1
2.0K    temp/group.new
 14K    temp
```

Информация о свободном пространстве на диске. Системная команда df выдаёт отчёт о доступном и использованном дисковом пространстве на файловых системах [2, 10]. Синтаксис команды может быть представлен как

```
df   [опции]   [файл]
```

Единицей измерения пространства в команде df являются физические блоки файловой системы. В некоторых версиях системы UNIX физические блоки состоят из 1024 байт и вмещают два (логических) блока, используемых в качестве единицы измерения пространства в команде du.

При запуске без аргументов df выдаёт отчёт по доступному и использованному пространству для всех смонтированных файловых систем (всех типов). В противном случае df для каждого файла, указанного в командной строке, выдаёт отчёт по файловой системе, которая его содержит. Если аргумент файл является дисковым файлом устройства, содержащим смонтированную файловую систему, то df показывает доступное пространство этой файловой системы, а не той, где содержится файл устройства.

Рассмотрим несколько примеров использования команды df:

```
$ df /usr/home/student/temp
Filesystem  1K-blocks   Used   Avail Capacity  Mounted on
/dev/ad0s1f   2814526 948770 1640594    37%    /usr
```

в этой команде вывод на консоль выполняется в пересчете на однокилобайтные блоки. Вывод команды df состоит из шести колонок, указывающих имя специального файла или имя смонтированного ресурса (Filesystem), общий объём (1K-blocks), использован-

ный объём (Used), доступный объём для использования обычными пользователями (Avail), процент свободного места в файловой системе (Capacity), точка монтирования (Mounted on). Для каждой физической файловой системы выдаётся отдельная строка. В предыдущем примере каталог temp находится в файловой системе, представленной в операционной системе специальным файлом /dev/ad0s1f, которая имеет объём в 2814526 килобайт, из которых используется пользователями 948770 килобайт, что составляет 37%. Точка монтирования файловой системы – каталог /usr.

В следующем примере задаётся опция -a, поэтому на дисплей выводится список всех файловых систем, включающий специальные файловые системы:

```
$ df -a
Filesystem  1K-blocks   Used   Avail Capacity  Mounted on
/dev/ad0s1a    253678  35406  197978    15%    /
devfs               1      1       0   100%    /dev
/dev/ad0s1e    253678      6  233378     0%    /tmp
/dev/ad0s1f   2814526 948770 1640594    37%    /usr
/dev/ad0s1d    253678    636  232748     0%    /var
```

Описание всех стандартных файловых систем находится в специальной таблице, которая представлена в операционной системе System V файлом /etc/vfstab, в других версиях UNIX, например FreeBSD, – файлом /etc/fstab [10]. В этих файлах описываются стандартные параметры для физических файловых систем. Поля в таблице разделены пробелами и символами табуляции и представляют: специальное блочное устройство или имя монтируемого ресурса; стандартный каталог монтирования; тип файловой системы; опции доступа к хранимой информации и другую служебную информацию. Так, например, содержимое файла /etc/fstab будет следующим:

```
$ cat /etc/fstab
# Device      Mountpoint  FStype Options    Dump   Pass#
/dev/ad0s1b   none        swap   sw         0      0
/dev/ad0s1a   /           ufs    rw         1      1
/dev/ad0s1e   /tmp        ufs    rw         2      2
/dev/ad0s1f   /usr        ufs    rw         2      2
/dev/ad0s1d   /var        ufs    rw         2      2
/dev/acd0     /cdrom      cd9660 ro,noauto  0      0
```

Применение опции –h позволяет сделать вывод команды df более понятным для пользователя – после размера файла добавляет-

ся буква, характеризующая этот размер (например, M – для обозначения двоичного мегабайта):

```
$ df -h temp
Filesystem     Size    Used   Avail Capacity  Mounted on
/dev/ad0s1f    2.7G    927M    1.6G    37%    /usr
```

2.3.2 Управление процессами

Получение состояние процессов. С помощью команды ps можно выяснить, какие программы и как расходуют процессорное время, определить «зависшие» процессы и вообще исследовать производительность системы [2, 5, 9, 10]. Синтаксис команды:

```
ps [ опция ]
```

По умолчанию (без параметров) команда ps печатает только процессы, инициированные пользователем; задание опции -a вызывает печать всех процессов и других пользователей, и системные процессы. Например:

```
$ ps
  PID  TT  STAT      TIME COMMAND
 1074  p0  S      0:00.23 -sh (sh)
 1081  p0  R+     0:00.03 ps
```

в данном случае показаны процессы пользователя, запустившего команду ps. В данном примере показано, что в настоящий момент времени выполняются два процесса (sh – командный интерпретатор; ps – список процессов).

При использовании опции –a вывод команды изменится:

```
$ ps -a
  PID  TT  STAT      TIME COMMAND
 1073  p0  Is     0:00.35 login [pam] (login)
 1074  p0  S      0:00.25 -sh (sh)
 1122  p0  R+     0:00.05 ps -a
  426  v0  Is+    0:00.03 /usr/libexec/getty Pc ttyv0
  427  v1  Is+    0:00.03 /usr/libexec/getty Pc ttyv1
  428  v2  Is+    0:00.03 /usr/libexec/getty Pc ttyv2
  429  v3  Is+    0:00.03 /usr/libexec/getty Pc ttyv3
  430  v4  Is+    0:00.03 /usr/libexec/getty Pc ttyv4
  431  v5  Is+    0:00.03 /usr/libexec/getty Pc ttyv5
  432  v6  Is+    0:00.03 /usr/libexec/getty Pc ttyv6
  433  v7  Is+    0:00.03 /usr/libexec/getty Pc ttyv7
```

Для каждого процесса во всех форматах вывода команды ps выводятся следующие сведения: PID – идентификатор процесса;

TT – управляющий терминал процесса (если стоит ?, то процесс не имеет управляющего терминала); TIME – время центрального процессора, использованное данным процессом (сюда входит время, затраченное на выполнение как программ пользователя, так и системных функций); STAT – состояние процесса (состояние процесса представляется последовательностью из четырёх букв, например RWNA); COMMAND – команда, выполняемая процессом в данный момент (если использовать команду ps с опцией -c, то на экран выводится имя команды без аргументов). Обозначение каждого из символов в поле STAT, в качестве самостоятельной работы, можно узнать в Руководстве пользователя с помощью команды man.

Необходимо отметить, что существует версии команды ps для операционных систем System V и BSD. Они отличаются друг от друга, в первую очередь, используемыми параметрами и выводом результата работы. Здесь рассмотрим команду ps, присущую операционным системам BSD, так как на компьютере с выполняющейся операционной системой UNIX, как правило, присутствуют одновременно обе версии ps.

Рассмотрим некоторые примеры использования команды ps с различными опциями. Так, например, опция -l позволяет вывести на экран, так называемый, длинный листинг:

```
$ ps -l
UID   PID   PPID CPU  PRI  NI   VSZ   RSS   MWCHAN STAT  TT  TIME     COMMAND
1001  1074  1073  0     8    0    1684  1260  wait       S     p0  0:00.28  -sh (sh)
1001  1140  1074  22    98   0    1360  836   -          R+    p0  0:00.15  ps -l
```

В длинном листинге добавляются несколько полей к рассмотренным ранее, а именно: UID – показывает идентификатор пользователя – владельца процесса (в нашем случае, пользователь с идентификатором 1001); PPID – указывает идентификатор родительского процесса (в нашем случае, процесс ps -l был порождён процессом с номером 1074, то есть командным интерпретатором sh); CPU – представляет информацию об использовании процесса для планировщика; PRI – содержит значение приоритета процесса: меньшее число означает больший приоритет и наоборот; NI – содержит поправку приоритета; VSZ – показывает размер исполняемого образа процесса в 512-байтовых блоках; RSS – показывает

фактический размер памяти, занимаемый процессом; MWCHAN – указывает на то, что процесс ожидает наступления события.

При использовании опции -u, вывод команды ps также несколько изменится:

```
$ ps -u
USER    PID  %CPU %MEM  VSZ RSS  TT STAT STARTED  TIME     COMMAND
student 1074 0.0  1.1   1684 1260 p0  S    1:29PM   0:00.36  -sh (sh)
student 1241 0.0  0.7   1360 836  p0  R+   2:43PM   0:00.03  ps -u
```

Назначение большинства полей уже ранее раскрывались, однако появились и новые поля: USER – указывает имя пользователя, запустившего процесс; %CPU – показывает долю процессорного времени, используемого процессом; %MEM – показывает долю памяти, используемую процессом; STARTED – показывает момент времени, когда процесс был запущен.

Запуск процессов в фоновом режиме. Выполнение некоторых процессов (копирование больших объёмов информации, поиск, сортировка и др.) может занять продолжительное время, причём, пока выполняется процесс, терминал (командная строка) занят, и невозможно выполнять другие команды. В связи с этим для выполнения продолжительных по времени задач лучше запускать их в фоновом режиме. Для этого в командной строке после набора команды указывается символ &. Например, процедура поиска файла с именем motd во всей файловой системе занимает продолжительное время, поэтому его лучше запустить в фоновом режиме:

```
$ find / -name motd > temp/find.motd &
```

в примере вывод результатов поиска перенаправлен с терминала в файл с именем temp/find.motd. При выполнении команды в фоновом режиме результаты выводятся сразу, как только они получены (даже если команда ещё не завершила своей работы). Это может помешать набору пользователем в командной строке текстов и команд. Таким образом, вывод результатов работы команды лучше перенаправить в какой-нибудь файл, а после окончания выполнения команды просмотреть результирующий файл, например, командой cat temp/find.motd. Однако необходимо отметить, что вывод сообщений об ошибках, возникающих при выполнении фоновых команд, немедленно выводятся на экран. Вывод сообщений об ошибках также можно переопределить. Попробуйте самостоятельно

сделать переопределение вывода сообщений об ошибках при осуществлении поиска файла с именем motd.

Окончание выполнения команды в фоновом режиме сопровождается выводом на терминал сообщения, например:

```
[1]   Done (1)        find / -name motd >temp/find.motd
```

Текущее состояние процесса можно узнать с помощью команды ps:

```
$ ps
  PID  TT  STAT      TIME COMMAND
 1074  p0  S      0:00.64 -sh (sh)
 1311  p0  D      0:03.55 find / -name motd
 1316  p0  R+     0:00.02 ps
```

Иногда необходимо, чтобы все фоновые процессы завершились, прежде чем будет выполняться какой-то расчёт. Для этого служит специальная команда `wait [PID]`. Эта команда ждёт завершения указанного идентификатором (числом) фонового процесса. Если команда без параметра, то она ждёт завершения всех фоновых процессов. При этом вся остальная работа с этой командной строкой пользователю недоступна.

Изменение приоритета выполнения процесса. Во всех операционных системах семейства UNIX пользователи могут при запуске процесса задавать значение поправки приоритета с помощью команды nice [2, 5, 9, 10]. Синтаксис команды:

```
nice [ -n [-|+]инкремент ] команда [ параметры ...]
```

Диапазон значений инкремента в большинстве систем варьируется от -20 до +20. Если инкремент не задан, используется стандартное значение +10. Положительный инкремент означает снижение текущего приоритета. Обычные пользователи (непривилегированные) могут задавать только положительный инкремент и, тем самым, только снижать приоритет. Например, команду поиска можно задать следующим образом:

```
$ nice -n +20 find / -name motd > temp/find.motd &
```

здесь команда поиска запускается в фоновом режиме и с поправкой приоритета +20:

```
$ ps -l
UID   PID   PPID CPU  PRI  NI   VSZ   RSS   MWCHAN STAT  TT  TIME     COMMAND
1001  1074  1073  1     8    0    1684  1260  wait       S     p0  0:00.74   -sh (sh)
1001  1394  1074  13   -8   20   1336  880   biord      DN    p0  0:02.48   find / -name
1001  1396  1074  3    96   0    1360  836   -          R+    p0  0:00.02   ps -l
```

Для того чтобы изменить значение поправки приоритета во время выполнения процесса, следует применить команду `renice`. Команда `renice` в качестве аргументов принимает поправку приоритета (первый аргумент) и идентификатор процесса (второй аргумент). Например:

```
$ renice +10 1394
```

2.3.3 Завершение процесса

Для завершения выполнения (уничтожения) процессов используется команда `kill` [2, 5, 6, 9, 10]. С помощью команды `kill` пользователь может отправить сигналы процессам, которые могут не только уничтожать процессы, но и выполнять другие действия по управлению процессами. Синтаксис команды:

```
kill [ –сиг ] идентификатор_процесса ...
```

Команда `kill` посылает указанному процессу сигнал `сиг`. Если сигнал не задан (то есть первый параметр – не сигнал), посылается сигнал программного завершения. Сигнал программного завершения уничтожит процесс, если последний его не перехватит. С помощью команды `kill` с опцией `–l` можно получить список сигналов, поддерживаемых системой:

```
$ kill -l
hup  int  quit  ill  trap  abrt  emt  fpe  kill
bus  segv  sys  pipe  alrm  term  urg
stop  tstp  cont  chld  ttin  ttou  io  xcpu
xfsz  vtalrm  prof  winch  info  usr1  usr2
```

При использовании команды `kill` можно указывать как символьное наименование сигнала, так и его числовую интерпретацию. Например, для безусловного уничтожения процесса используется сигнал `kill`, который имеет порядковый номер `9`. Таким образом, команда

```
kill -9 ...
```

гарантирует уничтожение процесса, так как этот сигнал не может быть перехвачен ни каким процессом. По соглашению, если в качестве номера процесса задан `0`, сигнал передаётся всем процессам данной группы (то есть всем процессам, порождённым в ходе текущего сеанса работы в системе). Уничтожаемые процессы должны принадлежать тому же пользователю.

В следующем примере покажем, каким образом можно завершить процесс, запущенный в фоновом режиме:

```
$ find / -name motd > temp/find.motd &
$ ps
  PID  TT  STAT      TIME COMMAND
  436  p0  S      0:00.08 -sh (sh)
  437  p0  D      0:00.14 find / -name motd
  438  p0  R+     0:00.02 ps
$ kill -9 437
$
[1]   Killed           find / -name motd >temp/find.motd
$
```

2.4 Средства связи

Система UNIX предоставляет пользователям простые средства общения друг с другом и с системой. Рассмотрим некоторые команды, позволяющие пользователям, работающим на одном и том же или на разных компьютерах, посылать почтовые сообщения другим пользователям или общаться друг с другом. Команда `mail` посылает сообщения (письма) другим пользователям, а команда `write` используется для связи в интерактивном режиме с пользователем, работающим за другим терминалом.

2.4.1 Почтовая служба

Команда `mail` позволяет принимать сообщения и посылать сообщения другим пользователям системы [5]. Эти сообщения хранятся в некотором файле до тех пор, пока адресат не прочитает их и не уничтожит или не сохранит. Синтаксис команды:

```
mail [ –r ] [ –f файл ]
```

Если пользователя ожидает почта, то при входе в систему он получит сообщение об этом:

```
you have mail
```

Оболочка также сообщит пользователю о поступлении любой новой почты перед выдачей на терминал очередного приглашения.

Команда `mail` без параметров печатает, письмо за письмом, текущую корреспонденцию пользователя. Первым выдаётся письмо, полученное последним. Опция `-r` меняет порядок просмотра корреспонденции на обратный.

```
$ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/student": 1 message 1 new
>N  1 student@virtServer.s  Fri May  5 15:09 18/786  "Hello"
&
```

После вывода заголовков писем или после вывода каждого письма выводится приглашение &, которое ожидает от пользователя ввода команд для работы с каждым письмом. Для уничтожения полученного письма необходимо набрать команду `d`; для повторной печати этого же письма надо набрать литеру `p`; для печати следующего письма из текущей почты – нажать клавишу возврата каретки (`Enter`), будет напечатано содержимое письма, например:

```
Message 1:
From  student@virtServer.siu.sibsiu.ru  Fri  May  5
15:09:36 2006
Date: Fri, 5 May 2006 15:09:35 GMT
From: User Student <student@virtServer.siu.sibsiu.ru>
To: student@virtServer.siu.sibsiu.ru
Subject: Hello

Hello, how are you?

&
```

Чтобы сохранить письмо в некотором файле, следует воспользоваться запросом

```
s [ имя_файла ]
```

Если в запросе `s` не указано имя файла, письмо будет сохранено в файле `mbox` в домашнем каталоге пользователя. Сохранённые и уничтоженные письма удаляются из почтового файла при выходе из команды `mail`. Выход из команды `mail` осуществляется при помощи команды `q`. Чтобы выйти из команды `mail` без каких-либо изменений в почтовом файле, следует воспользоваться командой `x`.

Почту можно послать одному или нескольким пользователям с помощью команды:

```
mail имя_пользователя ...
```

```
Subject: тема_письма
Текст_письма
.
```

которая отправит письмо указанным пользователям, дополнив его именем отправителя и темой письма. Текст письма заканчивается признаком конца файла или литерой . (точка), введённой в отдельной строке. Например, письмо для пользователей student и root:

```
$ mail student root     -кому отправить письмо
Subject: Hello          -тема письма
How are you?            -текст письма
.                       -признак конца тела письма
EOT
$
```

Если в процессе составления письма произойдет какое-либо прерывание, то недописанное письмо будет сохранено в файле dead.letter в домашнем каталоге пользователя, и произойдет выход из команды mail.

Для создания письма для пользователя, работающего на другом компьютере, используют сеть. В этом случае почтовый адрес состоит из имени компьютера и регистрационного имени пользователя, разделённых восклицательным знаком. Например, для отправки почты пользователю student, находящемуся на компьютере с именем class542, достаточно ввести команду

```
$ mail class542!student
```

с последующим вводом текста письма.

2.4.2 Передача сообщений

Помимо почтовой службы имеется команда write, позволяющая пользователям устанавливать связь между терминалами и непосредственно общаться друг с другом [5]. Синтаксис команды

```
write пользователь [ имя_tty ]
```

Команда write копирует строки с терминала пользователя на терминал указанного с помощью параметра пользователь пользователя. При вызове она посылает сообщение

```
Message from ваше_имя on имя_вашего_терминала at время_отправления
("Сообщение от пользователя ... с терминала ..., время ...").
```

Получателю в этот момент следует вызвать write для передачи сообщений в обратном направлении. Передача сообщений продолжается до тех пор, пока не будет введён признак конца файла (Ctrl-D) или получено прерывание. Когда это происходит, write посылает на другой терминал сообщение EOF и завершает работу.

Если необходимо послать сообщение пользователю, вошедшему в систему одновременно с нескольких терминалов, то можно воспользоваться параметром имя_tty для указания имени нужного терминала.

Иногда, например, в случаях, когда терминал используется как печатающее устройство, или при работе с экранным текстовым редактором появление случайного сообщения не желательно. Для запрета передачи сообщений на пользовательский терминал используется команда

```
mesg n
```

Для разрешения передачи сообщений на пользовательский терминал используется команда

```
mesg y
```

Например, обмен приветствиями между двумя пользователями student1 и student2 выглядит следующим образом:

Терминал пользователя student1:

```
$ who
student2          ttyp0    May  5 12:51 (10.1.2.133)
student1          ttyp1    May  5 16:51 (10.1.2.133)
$ who am i
student1          ttyp1    May  5 16:51 (10.1.2.133)
$ mesg y
$ write student2
Hello, student2
```

Терминал пользователя student2:

```
Message from student1@virtServer.siu.sibsiu.ru on ttyp1
at 16:52 ...
Hello, student2
$ write student1
Hello, student1!!!!!
:-)
^D
```

Терминал пользователя student1:

```
Message from student2@virtServer.siu.sibsiu.ru on ttyp0
at 16:55 ...
Hello, student1!!!!!
```

```
:-)
EOF
$
```

Вопросы и задания для самоконтроля

1. Объясните порядок выполнения следующих команд:

а) `k1 && k2; k3`

б) `k1 && {k2; k3}`

в) `{k1; k2} &`

где `k1`, `k2` и `k3` – какие-то команды.

2. Выведите справку о команде `cal`. Какая опция позволяет вывести календарь на текущий год?

3. Выведите справку о команде `date`. Как можно изменить формат вывода текущей даты и времени?

4. С помощью документации определите назначение команды `units`, ее опции.

5. Создайте иерархическую структуру каталогов, аналогичную представленной на рисунке 4.

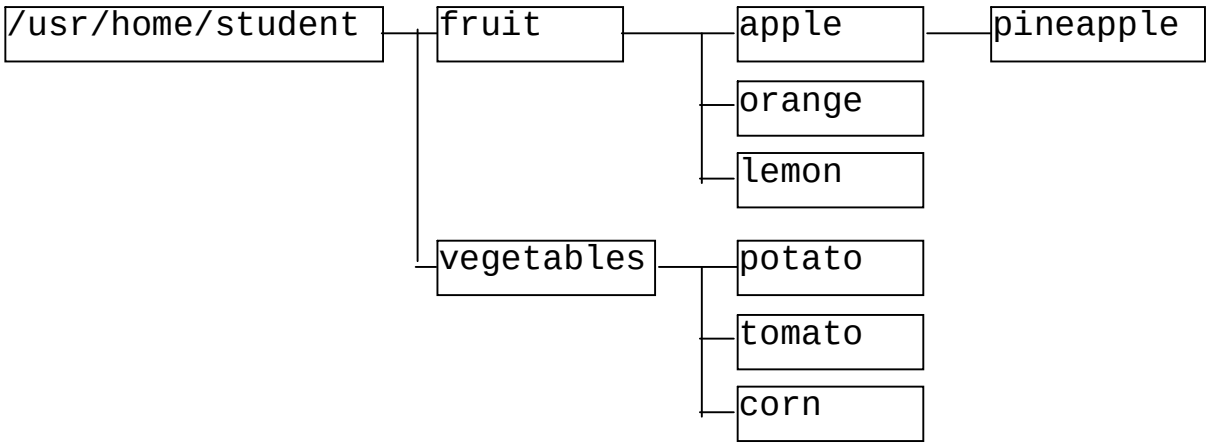


Рисунок 4 – Пример иерархической структуры каталогов

6. Выведите на экран и в файл содержимое таких файлов, как: `.login`, `.profile` (находящихся в домашнем каталоге пользователя), `/usr/src/README`.

7. Создайте пустой файл с именем `proba.txt` в подкаталоге `Folder` домашнего каталога пользователя.

8. С помощью переадресации вывода скопируйте файл `/usr/src/README` в файл `/usr/home/student/Folder/README.OLD`.

9. Для самостоятельного изучения работы команды pr выполните печать файла /usr/src/README и проанализируйте вывод команды.

10. Определите тип следующих файлов: /usr/src/README, /lib/libmd.so.2, /dev/zero, /root.

11. С помощью команды cp произведите копирование файлов /usr/home/student/README и /usr/home/student/README.new в созданный Вами каталог /usr/home/student/temp.

12. С помощью команды cp, используя метасимволы для задания шаблона имён копируемых файлов, произведите копирование файлов /usr/home/student/README и /usr/home/student/README.new в созданный Вами каталог /usr/home/student/temp.

13. Произведите перемещение файлов README и README.new из каталога /usr/home/student/temp в созданный Вами каталог /usr/home/student/temp1.

14. Самостоятельно создайте несколько вложенных друг в друга каталогов, скопируйте туда несколько файлов, а затем удалите все созданные каталоги.

15. С помощью команды grep узнайте в каких строках файла README находится интернет-адрес www.freebsd.org.

16. С помощью команды grep узнайте встречается ли в файле README слово GENERICS.

17. С помощью команд cut и ps вывести на экран все идентификационные номера процессов текущего пользователя. (для выполнения данного задания можно использовать и другие команды)

18. С помощью текстового редактора ee создайте текстовый файл, аналогичный представленному на рисунке 5. Подсказка: линии таблицы рисуются с помощью псевдографики ASCII кодами.

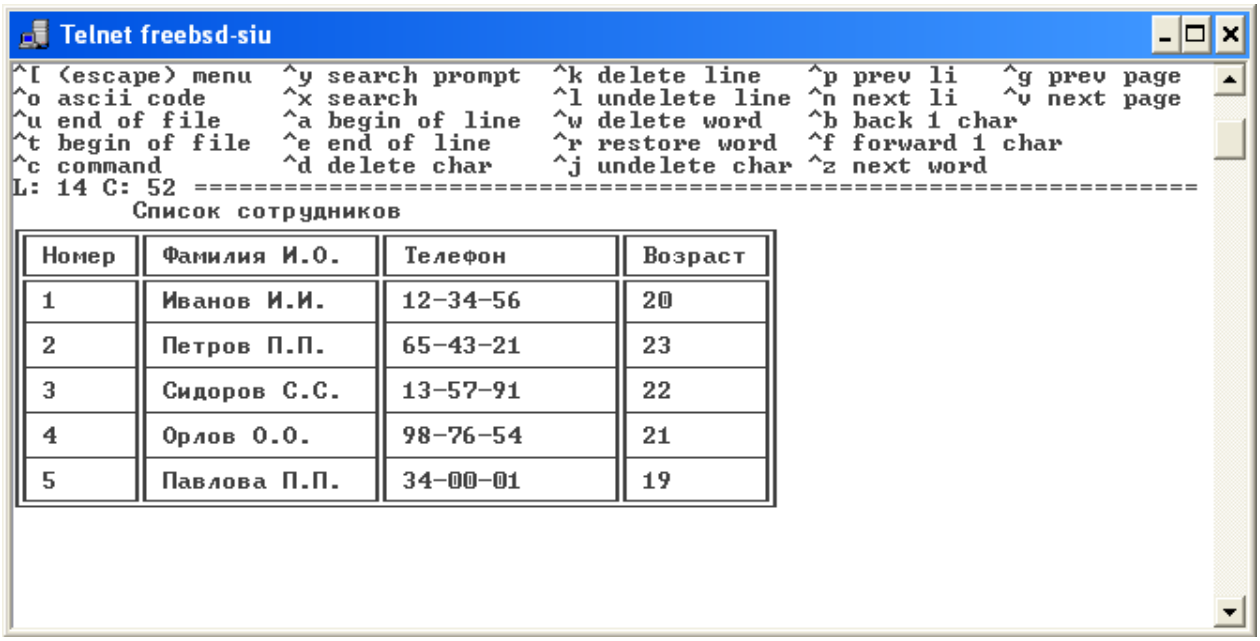


Рисунок 5 – Пример табличного форматирования в редакторе ee

3 Основы программирования на языке командного интерпретатора Shell

3.1 Основные понятия языка Shell

Операционная система UNIX предоставляет большие возможности для создания прикладных программ. Сама оболочка операционной системы, в сущности, представляет собой командный язык, который не только обеспечивает интерфейс пользователя с операционной системой, но и позволяет создавать программы (командные процедуры), дополняющие возможности «системных» команд пользовательскими. Вновь создаваемые командные файлы имеют тот же статус, что и «системные» команды. Таким образом, можно построить операционную среду, отвечающую потребностям отдельного человека или группы пользователей и автоматизировать многие рутинные операции администратора.

Достоинства использования интерпретируемых языков программирования очевидны [11, 12]:

- переносимость: написанный в одной операционной системе командный файл может быть выполнен на другой операционной системе семейства UNIX (главное, чтобы в системе был установлен командный интерпретатор для языка, на котором написана командная процедура);
- простота написания кода: нет необходимости специально обучаться сложному программированию на компилирующих языках, таких как C, C++, Pascal, Fortran (такое программирование наиболее очевидно, поскольку программа пишется в пошаговом режиме, то есть человек принимает решение о своем следующем шаге в зависимости от реакции системы на предыдущий шаг);
- быстрота написания кода благодаря простоте синтаксиса языка и мощным средствам отладки;
- большие функциональные возможности: с помощью командного языка можно использовать любые существующие файлы и команды системы.

В существующей среде программирования ОС UNIX сложилась методика эффективной разработки программ, включающая в себя следующие рекомендации [11, 12]:

- желательно, чтобы одна программа выполняла одну функцию (необходимо избегать избыточной функциональности);
- необходимо избегать избыточного, хаотичного и неструктурированного вывода в программе (надо помнить, что вывод любой программы может быть вводом для другой);
- по мере возможности необходимо использовать или переделывать уже существующее средство.

3.1.1 Исполнение командных файлов Shell

Файл, написанный для выполнения командным интерпретатором, – это командный файл, представляющий собой текст из последовательного набора команд. Существует несколько способов его вызова на выполнение [5]:

1. явно вызвать оболочку shell командой sh и передать в качестве параметра имя командного файла:
   ```
   $ sh имя_файла [параметры]
   ```
   или
   ```
   $ sh < имя_файла [параметры]
   ```
   или
   ```
   $ . имя_файла [параметры]
   ```
   (в отличие от первых двух – файл будет исполняться в текущем экземпляре shell)
2. сделать командный файл исполняемым, для чего достаточно поменять атрибуты (чтение, запись, выполнение) этого файла командой chmod:
   ```
   $ chmod 711 имя_файла
   ```
   а потом вызвать файл на исполнение, просто указав его имя
   ```
   $ имя_файла [параметры]
   ```

При вызове командного файла можно указать параметры, которые в дальнейшем будут доступны внутри командного файла в виде переменных.

При исполнении команд происходит поиск последних по каталогам в следующем порядке (по умолчанию): сначала – текущий каталог; затем – системный /bin; в конце – системный /usr/bin. Таким образом, если имя пользовательского командного файла дуб-

лирует имя команды в системных каталогах, то последняя станет недоступной (если только не набирать её полного имени).

3.1.2 Простейшие средства Shell

Комментарии. Как во всяком языке программирования в тексте скрипта, написанного на языке командного интерпретатора shell, могут быть использованы комментарии. Для этого используется символ «#». Все, что находится в строке (в командном файле) правее этого символа, воспринимается интерпретатором как комментарий. Например,

```
# Это комментарий.
## И это комментарий.
### И это тоже.
```

Структура команд. Команды в shell имеют аналогичный системным командам формат, то есть после имени команды можно указать параметры и аргументы через пробел:

```
Имя_команды  флаги  аргумент(ы)
```

Например,

```
kill -9  866
```

Метасимволы, генерация имён файлов. Метасимволы – символы, имеющие специальное значение для интерпретатора:
? * ; & ( ) | ^ < > пробел табуляция возврат_каретки

Однако каждый из этих символов может представлять самого себя, если перед ним стоит символ \(бэк-слэш). Все символы, заключённые между кавычками ' и ', представляют самих себя. Между двойными кавычками (") выполняются подстановки команд и параметров, а символы \, `, " и $ могут экранироваться предшествующим символом \.

Бэк-слэш (\) экранирует следующий за ним символ, что позволяет использовать специальные символы просто как символы, представляющие сами себя (он может экранировать и сам себя – \\), так же в командном файле бэк-слэш позволяет объединять строки в одну (экранировать конец строки).

Двойные кавычки (") экранируют пробелы, представляя несколько слов, разделённых пробелами, как одно слово. Однако ка-

вычки позволяют подставлять команды, параметры и переменные в строку текста.

После всех подстановок в каждом слове команды ищутся символы *, ?, и [. Если находится хотя бы один из них, то это слово рассматривается как шаблон имён файлов и заменяется именами файлов, удовлетворяющих данному шаблону (в алфавитном порядке). Если ни одно имя файла не удовлетворяет шаблону, то он остаётся неизменным. Значения указанных символов было подробно рассмотрено ранее.

Вывод на экран. Для вывода текстовой информации на стандартный вывод (по умолчанию – экран терминала) используется команда echo, имеющая следующий синтаксис:

```
echo [-n] [строка текста...]
```

В качестве параметра строка текста указывается текст, которой надо вывести на экран. При указании параметра -n после вывода на экран текста не производится переход на следующую строку. Например, действие команды echo без параметра -n:

```
$ echo Строка текста
Строка текста
$
```

и действие команды echo с параметром -n:

```
$ echo -n Строка текста
Строка текста$
```

3.2 Переменные Shell

Командные процедуры могут манипулировать с четырьмя типами переменных [5, 11-15]:

- позиционные параметры;
- специальные параметры;
- именованные (пользовательские) переменные;
- именованные (специальные) переменные.

3.2.1 Позиционные параметры

Позиционные параметры являются переменными в командной процедуре. Их значение устанавливается из аргументов, указанных в командной строке при запуске программы на исполнение. Позиционные параметры нумеруются от 0 до 9, и на них ссылка производится с помощью символа $:

`$0` соответствует имени данного командного файла;

`$1` первый по порядку параметр;

`$2` второй по порядку параметр;

`...` …

`$9` девятый по порядку параметр.

В программе можно обратиться непосредственно только к девяти позиционным параметрам одновременно.

Например, если программа вызывается с помощью командной строки, подобной следующей:

```
shell.prog pp1 pp2 pp3 pp4 pp5 pp6 pp7 pp8 pp9
```

то позиционному параметру `$1` в программе присваивается значение `pp1`, `$2` – значение `pp2` и т.д. во время вызова программы. Чтобы практически рассмотреть это замещение позиционных параметров, создайте файл `pp`:

```
echo Первый позиционный параметр: $1
echo Второй позиционный параметр: $2
echo Третий позиционный параметр: $3
echo Название файла: $0
```

Если выполнить эту программу с аргументами `один`, `два`, `три`, то будет следующий результат:

```
$ sh pp один два три
Первый позиционный параметр: один
Второй позиционный параметр: два
Третий позиционный параметр: три
Название файла: pp
$
```

Для примера создадим командный файл `AddToPhones`, который записывает переданные в командной строке параметры (ФИО, пол человека, его телефон) в специальный файл, например `telephones`. Данный командный файл будет содержать только одну строку:

```
echo "$1        $2      $3" >> $HOME/telephones
```

При запуске вводим после имени программы необходимые сведения

```
$ sh AddToPhones "Иванов И.И." муж 12-34-56
$ cat telephones
Иванов И.И.     муж         12-34-56
$
```

В данном случае первый параметр – это фамилия и инициалы человека. Чтобы инициалы из-за разделяющего их с фамилией пробела не были интерпретированы как второй параметр, необходимо заключить их в двойные кавычки. В дальнейшем на других примерах продолжим добавлять функциональность к этой программе.

Сдвиг параметров. Командный интерпретатор `Shell` позволяет указывать в командной строке до 128 аргументов, но ссылаться можно не более чем на 9 позиционных параметров. Однако существует команда `shift`, с помощью которой можно сдвинуть имена на остальные аргументы, если их больше 9 (причём окно доступа к переменным остается шириной 9). Синтаксис команды:

```
shift [сдвиг]
```

если параметр сдвиг не указан, то величина сдвига принимается равной одному. Параметр `сдвиг` может быть задан только положительным числом.

Для большего понимания действия данной команды создайте следующий файл с именем `ManyParametrs`:

```
echo "Исходное состояние: $1 $5 $9 "
shift
echo "1 сдвиг: первый=$1 пятый=$5 девятый=$9"
shift 2
echo "1 + 2 = 3 сдвига: первый=$1 пятый=$5 девятый=$9"
```

вызовите его с 13-ю параметрами:

```
ManyParametrs 10 20 30 40 50 60 70 80 90 100 110 120 130
```

Проанализируйте самостоятельно работу данной программы и команды `shift` в частности.

Переопределение параметров внутри программы. Своеобразный подход к параметрам даёт команда `set`. Используя данную команду, можно установить новые параметры в уже выполняемой программе. Можно использовать следующие виды синтаксиса:

```
set значения_параметров_через_пробел
```

или

```
set `команда`
```

где после выполнения `команда`, её результат будет рассматриваться выполняющейся программой как параметры.

Например, следующая программа:

```
echo "Первый = $1; Второй = $2"
```

```
set два один
echo "Первый = $1; Второй = $2"
```

будет переназначать параметры следующим образом:

```
$ sh SetParametrs один два
Первый = один; Второй = два
Первый = два; Второй = один
$
```

Другой способ использования команды `set` для назначения параметров может быть проиллюстрирован следующим командным файлом:

```
set `date`
cal $6
```

Данный командный файл сначала вызывает команду `date`, возвращающая текущую дату, состоящую из шести слов, последним из которых является текущий год, который, в свою очередь, передаётся команде `cal`, выводящей на экран календарь на указанный год.

3.2.2 Специальные параметры

В командном интерпретаторе доступны специальные параметры, автоматически устанавливаемые самим командным интерпретатором. В таблице 11 перечислены все доступные специальные параметры и дано пояснение по каждому из них.

Таблица 11 – Специальные параметры командного интерпретатора

Параметр	Описание
#	Количество позиционных параметров передаваемых в `shell` (содержит десятичное значение)
-	Флаги, указанные при запуске командного интерпретатора или установленные командой `set`
?	Десятичное значение, возвращённое предыдущей синхронно выполненной командой (код завершения программы – код «0» соответствует нормальному завершению процесса)
$	Номер текущего процесса
!	Номер последнего асинхронного (фонового) процесса
@	Перечень параметров, как совокупность слов (эквивалентно $1 $2 $3 ...)
*	Перечень параметров, как одна строка (эквивалентно «$1 $2 $3 ...»)

Чтобы получить значения этих переменных, перед ними нужно поставить знак $.

Разберём действие каждого параметра на простом примере. Создайте следующую программу с именем SpecParametrs:

```
echo Имя программы - $0
echo Код завершения предыдущего процесса - $?
echo Идентификатор процесса - $$
echo Идентификатор последнего фонового процесса - $!
echo Количество параметров - $#
echo Значения параметров, как строки, - $*
echo Значения параметров, как слов, - $@
set -au
echo Режимы работы командного интерпретатора - $-
```

Результат работы программы будет следующий:

```
$ sh SpecParametrs один два три
Имя программы - SpecParametrs
Код завершения предыдущего процесса - 0
Идентификатор процесса - 20255
Идентификатор последнего фонового процесса -
Количество параметров - 3
Значения параметров, как строки, - один два три
Значения параметров, как слов, - один два три
Режимы работы командного интерпретатора – au
$
```

3.2.3 Именованные (пользовательские) переменные

Все переменные в командном языке – текстовые. В Shell используется всего два типа данных: строка символов и текстовый файл. Их имена должны начинаться с буквы или знака подчеркивания и состоять из латинских букв, цифр и знака подчеркивания. Чтобы воспользоваться значением переменной, надо перед ней поставить символ $. Использование значения переменной называется подстановкой.

Определение переменной содержит её имя (без символа $) и значение:

```
переменная=значение
```

Таким образом, для присваивания значений переменным используется оператор присваивания «=». При этом необходимо запи-

сывать как переменную, так и её значение без пробелов относительно символа присвоения. Например, следующее объявление

```
var_1=13
```

присвоит переменной значение «13» (однако это не число, а строка, состоящая из двух символов – двух цифр).

Если необходимо присвоить значение, включающее в строке пробелы или другие разделяющие знаки, то надо воспользоваться двойными кавычками:

```
var_2="Операционная система FreeBSD"
```

Следующее объявление присвоит переменной пустую строку:

```
var_3=
```

Как отмечалось выше, при обращении к пользовательской переменной необходимо перед именем ставить символ $. Например, команды

```
var_2="Операционная система FreeBSD"
echo $var_2
echo var_2
```

выдадут на экран

```
Операционная система FreeBSD
var_2
```

Также необходимо помнить, что не может быть одновременно функции и переменной с одинаковыми именами.

Конструкции подстановки переменных. Для того чтобы имя переменной не сливалось со строкой, следующей за именем переменной, используются фигурные скобки. В таблице 12 представлены конструкции, используемые для подстановки значений переменных.

Таблица 12 – Конструкции подстановки

Конструкция	Назначение
`${переменная}`	Если значение `переменная` определено, то оно подставляется. Скобки применяются лишь в том случае, если за переменной следует символ, который без скобок «приклеится» к имени
`${переменная:-слово}`	Если переменная определена и не является пустой строкой, то подставляется её значение; иначе подставляется слово

Продолжение таблицы 12

Конструкция	Назначение
`${переменная:=слово}`	Если `переменная` не определена или является пустой строкой, ей присваивается значение `слово`; после этого подставляется её значение
`${переменная:?слово}`	Если `переменная` определена и не является пустой строкой, то подставляется её значение; иначе на стандартный вывод выводится `слово` и выполнение командного интерпретатора завершается. Если `слово` опущено, то выдается сообщение «`parameter null or not set`»
`${переменная:+слово}`	Если `переменная` определена и не является пустой строкой, то подставляется `слово`; иначе подставляется пустая строка

Например, можно проверять в программе существует ли переменная и определена ли она с помощью такой конструкции:

```
${var_1:?"Такого параметра нет"}
```

в этом случае на экран выведется соответствующее сообщение и выполнение программы прекратиться, если переменная `var_1` не существует или она не была определена до указанной конструкции.

Определение переменных с помощью команд. Возможны и иные способы присваивания значений пользовательским переменным. Так в качестве значения можно присвоить переменной результат выполнения какой-либо команды. Для этого необходимо использовать следующий синтаксис:

```
имя_переменной=`имя_команды`
```

то есть обратные кавычки указывают на то, что сначала должна быть выполнена заключённая в них команда, а результат её выполнения, вместо выдачи на стандартный вывод, присваивается в качестве значения переменной.

Например, запись

```
CurrentPath=`pwd`
```

приводит к тому, что сначала выполняется команда pwd, а результат присваивается переменной CurrentPath.

Ввод значений переменных с клавиатуры. Можно присвоить значение переменной и с помощью команды read, которая обеспечивает приём значения переменной с дисплея (клавиатуры) в диалоговом режиме [5, 11-15]. Для этого необходимо использовать следующий синтаксис:

```
read переменная1 переменная2 … переменнаяN
```

Обычно команде read в командном файле предшествует команда echo, которая позволяет предварительно выдать какое-то поясняющее сообщение на экран. Например:

```
echo -n "Введите трехзначное число: "
read x
```

Введённое с клавиатуры число будет присвоено переменной x.

Одна команда read может прочитать (присвоить) значения сразу для нескольких переменных. Если переменных в read больше, чем их введено (через пробелы), оставшимся присваивается пустая строка. Если передаваемых значений больше, чем переменных в команде read, то лишние – игнорируются. На самом деле интерпретатор для продолжения работы ждёт лишь нажатия клавиши <CR> (перевод каретки – Enter). Введённое число воспринимается им не как число, а как последовательность символов. Интерпретатор не проверяет, что введено, поэтому в качестве значения переменной может оказаться любая введенная «абракадабра» или просто нажатие <CR>, как значение пустой строки.

В качестве примера использования команды read переработаем ранее рассмотренный пример добавления записи в телефонный справочник. В рассмотренном ранее примере сведения для добавления новой записи берутся из параметров командной строки, что достаточно неудобно для конечного пользователя, поэтому напишем программу, которая будет в интерактивном режиме спрашивать пользователя о добавляемой записи:

```
echo Добавление сведений в телефонный справочник
echo -n Введите имя:
read FIO
echo -n "Введите пол (муж/жен): "
read pol
echo -n Введите телефон:
read phone
```

```
echo "$FIO      $pol    $phone" >> $HOME/telephones
```

Использование пользовательских переменных для промежуточного хранения сведений позволит в дальнейшем производить некоторую дополнительную обработку, если потребуется. Вывод программы следующий:

```
$ sh AddToPhones
Добавление сведений в телефонный справочник
Введите имя: Сидоров С.С.
Введите пол: муж
Введите телефон: 98-76-65
$ cat telephones
Иванов И.И.     муж     12-34-56
Сидоров С.С.    муж     98-76-65
$
```

3.2.4 Именованные (специальные) переменные

Ранее в разделах 3.2.1 и 3.2.3 были приведены примеры записи в телефонный справочник, в которых использовалась переменная `$HOME`. Данная переменная является специальной именованной переменной операционной среды. Эти переменные обычно устанавливаются в файле `.profile` и `rc` в регистрационном оглавлении пользователя. Вызвав команду `set` без параметров, можно получить информацию о стандартных переменных, созданных при входе в систему (и передаваемых далее всем новым процессам «по наследству»), а также переменных, созданных и экспортируемых процессами, созданными самим пользователем. В таблице 13 представлены основные специальные переменные, которыми оперирует операционная среда.

Такие переменные имеют для оболочки `shell` особое значение, их не следует использовать для других целей.

Таблица 13 – Описание специальных переменных

Переменная	Описание	Пример
$MAIL	При работе с терминалом оболочка перед выдачей очередного приглашения обследует состояние файла, определяемого этой переменной. Если с момента предыдущего обследования в этот файл были внесены изменения, оболочка выдает сообщение you have mail (для Вас почта) перед выдачей приглашения для ввода очередной команды.	MAIL=/var/mail/student
$HOME	Это имя домашнего каталога, в котором пользователь оказывается после входа в систему.	HOME=/home/student
$CDPATH	Список каталогов, просматриваемых командой cd. Имена каталогов разделяются двоеточиями.	CDPATH=:..:$HOME/desk
$PATH	Задаёт последовательность файлов, которые просматривает оболочка в поисках команды. Имена файлов разделяются двоеточиями. Последовательность просмотра соответствует очередности следования имён в переменной. Однако первоначально поиск происходит среди так называемых встроенных команд. В число встроенных команд входят наиболее часто используемые команды, например echo, cd. После набора командной строки и нажатия <CR> оболочка (после выполнения необходимых подстановок) распознаёт имя, соответствующее команде и осуществляет её поиск в директориях, перечисленных в переменной. Если команда размещена вне этих директорий, она не будет найдена. Если присутствует несколько команд с одинаковым именем, то вызвана будет та, которая расположена в директории, просматриваемой первой.	PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/usr/X11R6/bin:/home/student/bin

Продолжение таблицы 13

Переменная	Описание	
$IFS	Внутренний Разделитель Полей – перечисляет символы, которые служат для разделения слов. Таковыми являются «пробел», «табуляция» и «перевод строки».	IFS=' '
$LOGNAME	Имя входа («имя» пользователя).	LOGNAME=s
$PWD	Имя текущего каталога	PWD=/usr/
$PS1	Вид промтера – приглашения – которое печатается в командной строке.	PS1='$ '
$PS2	Этот промтер используется как приглашение к продолжению ввода незаконченной команды в очередной строке.	PS2='> '
$SHELL	Указывает оболочку, которую использует пользователь.	SHELL=/bi
$TERM	Указание типа терминала.	TERM=ansi
$TERMCAP	Строка задания параметров терминала.	TERM-CAP=conso :li#25:co
$UID	Идентификатор пользователя.	UID=501
$TZ	Определяет зону времени	TZ= EST5E

3.2.5 Манипуляции с переменными

Экспорт переменных. Основополагающим понятием в ОС UNIX, как и в любой другой ОС, является понятие процесса. Процесс возникает тогда, когда запускается на выполнение какая-либо команда. У каждого процесса есть своя среда – множество доступных ему переменных. Переменные локальны в рамках процесса, в котором они объявлены, то есть где им присвоены значения. Для того чтобы они были доступны и другим порождаемым процессам, надо передать их явным образом. Для этого используется встроенная команда export, имеющая следующий синтаксис:

```
export [имя_переменной1 имя_переменной2 …]
```

если параметры не заданы, выдаётся список имён экспортируемых переменных. [5, 11-15]

Механизм работы данной команды рассмотрим на примере, описанном в [11].

Пусть командный файл pr1 объявляет и определяет две переменных – var1 и var2, экспортирует переменную var2, а затем последовательно вызывает два других командных файла pr1_1 и pr1_2, которые переопределяют эти переменные:

1. pr1:

```
echo Расчет pr1
var1=1 var2=2
echo var1=$var1 var2=$var2
export var2
sh pr1_1
sh pr1_2
echo Снова расчет pr1: var1=$var1 var2=$var2
```

2. pr1_1:

```
echo "  Расчет pr1_1"
echo "  var1=$var1 var2=$var2"
var1=a var2=b
echo "  var1=$var1 var2=$var2"
export var1
```

3. pr1_2:

```
echo "  Расчет pr1_2"
echo "  var1=$var1 var2=$var2"
var1=A var2=B
```

```
echo "  var1=$var1 var2=$var2"
export var2
```

Запустив на исполнение первый файл, получим следующий вывод:

```
$ sh pr1
Расчет pr1
var1=1 var2=2
        Расчет pr1_1
        var1= var2=2
        var1=a var2=b
        Расчет pr1_2
        var1= var2=2
        var1=A var2=B
Снова расчет pr1: var1=1 var2=2
$
```

Из примера видно, что значения переменных экспортируются только в вызываемые расчёты. Таким образом, если необходимо чтобы все вновь запускаемые программы «видели» какую-либо переменную, достаточно в командной строке объявить и инициализировать эту переменную, а потом экспортировать её. Например, создадим переменную `MyGlobalVariable`:

```
$ MyGlobalVariable=true
$ export MyGlobalVariable
```

Экспортировать переменные можно и командой `set` с параметром `-a`. После вызова такой команды, все вновь объявляемые переменные будут сразу же экспортироваться.

Ограничение доступа к переменным. Переменные, значение которых желательно сохранить неизменными, можно объявить доступными только для чтения с помощью команды `readonly`. Эта команда имеет аналогичный вид команде `export`, а именно:

`readonly [имя_переменной1 имя_переменной2 …]`

если параметры не заданы, выдаётся список всех переменных, доступных только для чтения.

Последующие попытки присвоить значение таким переменным отвергаются.

3.3 Программные структуры

При рассмотрении программных структур, используемых в командном интерпретаторе, необходимо определиться с некоторыми терминами:

Простая команда – это последовательность слов, разделённая пробелами. Первое слово является именем команды, которая будет выполняться, а остальные будут переданы ей как аргументы. Имя команды передаётся ей как аргумент номер `0` (т.е. имя команды является значением параметра `$0`). Значение, возвращаемое простой командой, – это её статус завершения, если она завершилась нормально, или (восьмеричное) `200+статус`, если она завершилась аварийно.

Список – это последовательность одного или нескольких конвейеров, разделённых символами `;`, `&`, `&&` или `||` и быть может заканчивающаяся символом `;` или `&`.

Команда – это либо простая команда, либо одна из управляющих конструкций. Кодом завершения команды является код завершения её последней простой команды.

3.3.1 Арифметические операции

Разнообразные возможности для выполнения арифметических операций над числами или значениями переменных имеет команда `expr` [5, 11-15]. Результат вычислений выводится на стандартный вывод. Операнды выражения должны быть разделены пробелами. Метасимволы, используемые в выражениях, должны быть экранированы. Строки, содержащие пробелы или другие специальные символы, должны быть заключены в кавычки. Целые рассматриваются как 32-битные числа.

Список операторов в порядке возрастания приоритета (операции с равным приоритетом заключены в фигурные скобки, перед символами, которые должны быть экранированы, стоит `\`) представлен в таблице 14.

Таблица 14 – Список операторов команды `expr`

Операторы	Описание
`<выр1> | <выр2>`	Если `<выр1>` не пустое и не нулевое, то возвращает его, иначе возвращает `<выр2>`

Продолжение таблицы 14

Операторы	Описание
<выр1> \& <выр2>	Если оба <выр1> и <выр2> не пустые и не нулевые, то возвращает <выр1>, иначе возвращает 0
<выр1> { =, \>, \>=, \<, \<=, != } <выр2>	Возвращает результат целочисленного сравнения если оба <выр1> и <выр1> целые; иначе возвращает результат лексического сравнения
<выр1> { +, - } <выр2>	Сложение и вычитание целочисленных аргументов
<выр1> { *, /, % } <выр2>	Умножение, деление и получение остатка от деления целочисленных аргументов. Операция умножения («*») обязательно должна быть экранирована
<выр1> : <выр2>	Оператор сопоставления : сопоставляет первый аргумент со вторым, который должен быть регулярным выражением. Обычно оператор сравнения возвращает число символов, удовлетворяющих образцу (0 при неудачном сравнении). Для выделения части первого аргумента могут применяться символы \(и \).

Регулярное выражение строится следующим образом:

. – обозначает любой символ;

* – обозначает предыдущий символ, повторенный несколько раз;

[]– обозначают любой один из указанных между ними символов; группа символов может обозначаться с помощью знака «-»; если после [стоит ^, то это эквивалентно любому символу, кроме указанных в скобках и <CR>; для указания] в качестве образца, на-

до поставить ее сразу за [(т.е. []...]); . и * внутри квадратных скобок обозначают самих себя.

Все остальные символы (и ^, если стоит не в квадратных скобках) обозначают самих себя. Для указания символов ., *, [и] надо экранировать их.

Примеры использования:

1. a=`expr $a + 1`

производится увеличение на 1 переменной a

2. b=`expr $a : '.*/\(.*\)' \| $a`

выделяет из имени файла короткое имя (т.е. из /usr/util/ena выделяется ena). Надо помнить, что одиночный символ / будет воспринят как знак операции деления.

3. count=`expr $VAR : '.*'`

получение количества символов переменной VAR.

4. a=`expr 'полукеды' : 'пол'`

получение количества одинаковых символов с начала слова.

Обратите внимания, что команда expr при вызове указывается в обратных кавычках.

В качестве побочного эффекта expr возвращает следующие коды завершения:

- 0 – если выражение не нуль и не пустая строка;
- 1 – если выражение нуль или пустая строка;
- 2 – для некорректных выражений.

Команда expr также выдаёт следующие сообщения об ошибках:

- syntax error – для ошибок в операторах или операндах;
- non-numeric argument – для попыток применения арифметических операций к нечисловым строкам.

3.3.2 Пустой оператор

Пустой оператор – эквивалент операции «NOP» (*no op*, нет операции) – имеет формат

```
:
```

Он ничего не делает и возвращает значение 0. Например, в конструкции

```
:
```

```
echo $?
```

На экране отобразится ноль.

Данный оператор может использоваться в следующих случаях [14]:

- для организации бесконечного цикла (рассмотрению использования циклических операторов посвящён раздел 3.3.5 настоящего пособия);
- как символ-заполнитель в условном операторе (рассмотрению использования условных операторов посвящён раздел 3.3.4 настоящего пособия);
- как символ-заполнитель в операциях, которые предполагают наличие двух операндов. Например:

```
: ${username=`who am i`}
```

без символа `:` выдаётся сообщение об ошибке, если `username` не является командой;

- Как символ-заполнитель для оператора вложенного документа. Например, можно создать блочный комментарий:

```
: << COMMENTBLOCK
echo "Эта строка не будет выведена."
Эта строка комментария не начинается с символа "#".
&*@!!++= - просто символы
COMMENTBLOCK
...
```

Между словами `COMMENTBLOCK` находится многострочный блок комментариев. В данном случае весь блок комментариев рассматривается как встроенный документ и его вывод перенаправляется в пустой оператор `:`;

- в операциях с подстановкой параметров. Например:

```
: ${HOSTNAME?} ${USER?} ${MAIL?}
  echo
  echo "Имя машины: $HOSTNAME."
  echo "Ваше имя: $USER."
  echo "Ваш домашний каталог: $HOME."
  echo "Ваш почтовый ящик: $MAIL."
```

Если хотя бы одна переменная не определена, то произойдёт вывод сообщения об ошибке;

- в операциях замены подстроки с подстановкой значений переменных (см. раздел 3.2.3).

- в комбинации с оператором перенаправления вывода (>), усекает длину файла до нуля[2]. Если указан несуществующий файл, то он создаётся.

```
: > File.new
```

То же самое можно осуществить и с помощью команды

```
cat /dev/null > File.new
```

Однако при использовании пустого оператора не производится создание нового процесса;

- в комбинации с оператором перенаправления с добавлением в конец файла (>>) обновляет время последнего доступа к файлу, не меняя содержимое указанного файла[2]. Например:

```
: >> File.old
```

Если задано имя несуществующего файла, то он создается;

- может использоваться как разделитель полей в файле пользовательских паролей `/etc/passwd` и именованных специальных переменных `$PATH` и `$CDPATH` (см. раздел 3.2.4).

3.3.3 Оператор сравнения test («[ ]»)

Команда `test` проверяет выполнение некоторого условия и обычно используется в командных процедурах [5, 11-15]. С использованием этой команды формируются операторы выбора и цикла командного языка. Существует два возможных формата команды:

```
test условие
```

или

```
[ условие ]
```

между скобками и содержащимся в них условием обязательно должны быть пробелы. Также пробелы должны быть и между значениями и символом сравнения или операции (как, кстати, и в команде `expr`).

В результате выполнения команды будет возвращён код завершения: если он равен `0` – то истинность условия подтверждено, если равен `1` – то истинность условия не подтверждена. В случае возникновения ошибки при сравнении код завершения равен `2`.

Имеется четыре типа проверок [5, 11-15]:

- оценка числовых значений;

[2] Такое использование пустого оператора применимо только к обычным файлам и неприменимо к конвейерам, символическим ссылкам и другим специальным файлам.

- оценка типа файла;
- оценка строк;
- сложные операции.

Для каждого типа применяются свои примитивы.

Оценка числовых значений. Для сравнения числовых значений используется следующий синтаксис:

```
X op Y
```

где X, Y – числа или числовые переменные;

op – соответствующая операция сравнения, которая может принимать одно из значений, представленных в таблице 15.

Таблица 15 – Операции сравнения числовых значений

Операция	Описание
`X -eq Y`	Сравнение на равенство
`X -ne Y`	Сравнение на неравенство
`X -gt Y`	X больше Y
`X -ge Y`	X больше или равно Y
`X -lt Y`	X меньше Y
`X -le Y`	X меньше или равно Y

При сравнении числовых значений с помощью указанных операторов команда `test` воспринимает строки символов как целые числа. Таким образом, если надо обнулить переменную, скажем, X, то необходимо присвоить `X=0`. Отметим, что во всех остальных случаях (кроме операций сравнения числовых значений) «нулевому» значению соответствует пустая строка.

Рассмотрим в качестве примера работы команды сравнения числовых значений следующий командный файл:

```
x=abc ; [ abc -eq $x ] ; echo $?
x=10 ; [ 10 -eq $x ] ; echo $?
x=10.1 ; [ 10.1 -eq $x ] ; echo $?
x=10 ; [ 12 -lt $x ] ; echo $?
```

при выполнении которого на стандартном выводе будут получены следующие результаты:

```
[: abc: bad number
2
0
[: 10.1: bad number
```

```
2
1
```

Поясним результаты работы команды сравнения. В первой строке выводится сообщение об ошибке (в примере вместо числового значения передаются текстовые символы), а во второй – код завершения команды сравнения (так как команда равнения завершилась аварийно с ошибками, то код завершения равен 2). В третьей строке вывод результата сравнения – истина. Так как команда сравнения ожидает ввод целых чисел, а 10.1 является дробным числом, то в четвёртой и пятой строке вывода сообщается о соответствующей ошибке. В последней строке выводится результат сравнения – ложь (так как 12 не меньше 10).

Оценка типа файла. Для оценки типа файла используют следующий синтаксис:

```
op filename
```

где `op` – операция сравнения, которая может принимать одно из значений, представленных в таблице 16.

Таблица 16 – Основные операции оценки типа файлов

Операция	Описание
`-f filename`	Файл `filename` существует и является обычным файлом
`-d filename`	Файл `filename` существует и является каталогом
`-c filename`	Файл `filename` существует и является специальным символьно-ориентированным файлом
`-b filename`	Файл `filename` существует и является специальным блок-ориентированным файлом
`-r filename`	Файл `filename` существует и имеется разрешение на чтение
`-w filename`	Файл `filename` существует и имеется разрешение на запись в файл
`-x filename`	Файл `filename` существует и является выполняемым
`-s filename`	Файл `filename` существует и не пустой (имеет ненулевую длину)

Продолжение таблицы 16

Операция	Описание
`-p filename`	Файл `filename` существует и является именованным каналом (pipe)
`-t дескрип-тор_файла`	Открытый файл с указанным дескриптором (по умолчанию 1) существует и ассоциирован с терминалом

Для рассмотрения примера использования операторов оценки типа файла создайте командный файл с именем `File1`, в котором будет только одна команда `date`. Также создайте командный файл `TestFiles`, в котором будет выполняться проверка типа файла, переданного в качестве параметра:

```
[ -s $1 ]; echo "Файл существует: $? "
[ -f $1 ]; echo "Файл: $? "
[ -d $1 ]; echo "Каталог: $? "
[ -r $1 ]; echo "Чтение из файла: $? "
[ -w $1 ]; echo "Запись в файл: $? "
[ -x $1 ]; echo "Исполняемый файл: $? "
```

Запустив файл `TestFiles` с параметром `File1`, получим следующие результаты:

```
$ sh TestFiles File1
Файл существует: 0
Файл: 0
Каталог: 1
Чтение из файла: 0
Запись в файл: 0
Исполняемый файл: 1
```

С помощью команды `chmod` измените атрибут файла на исполняемый:

```
chmod  711  File1
```

и снова запустите файл `TestFiles`. Вывод будет следующий:

```
$ sh TestFiles File1
Файл существует: 0
Файл: 0
Каталог: 1
Чтение из файла: 0
Запись в файл: 0
Исполняемый файл: 0
```

Самостоятельно проверьте вывод результатов работы данного командного файла, если передать в качестве параметра имя несуществующего файла.

Оценка строк. Для оценки и сравнения строк синтаксис в зависимости от используемого оператора сравнения следующий:

- унарный оператор сравнения:
  `op1 S`
- бинарный оператор сравнения:
  `S op R`

где `S`, `R` – строки или строковые переменные;

`op`, `op1` – операции сравнения, которые могут принимать одно из значений, представленных в таблице 17.

Таблица 17 – Операции оценки и сравнения строк

Операция	Описание
`str1 = str2`	Строки `str1` и `str2` совпадают
`str1 != str2`	Строки `str1` и `str2` не совпадают
`-n str1`	Строка `str1` существует (непустая)
`-z str1`	Строка `str1` не существует (пустая)

В качестве примера представим следующие команды сравнения строк:

```
x=abc ; [ -n $x ] ; echo $?
```

Результат: `0`

```
x=abc; [ abc = $x ]; echo $?
```

Результат: `0`

```
x="" ; [ -n $x ] ; echo $?
```

Результат: `1`

```
[ abc ] ; echo $?
```

Результат: `0`

```
[ ] ; echo $?
```

Результат: `1`

В двух последних командах такой результат получается в связи с тем, что в первом случае в скобках стоит не пустая строка, а во втором – пустая строка.

Кроме того, существуют два стандартных значения условия, которые могут использоваться вместо условия (для этого не нужны скобки).

```
true ; echo $?
```

Результат: `0`

```
false ; echo $?
```

Результат: `1`

Сложные операции. Несколько проверок разных типов могут быть объединены логическими операциями в сложные операции:

`!` – инвертирует значение кода завершения (операция отрицания);

`-o` – дизъюнкция (логическое «ИЛИ»);

`-a` – конъюнкция (логическое «И»).

Следующие команды иллюстрируют использование сложных операций:

- `[ ! abc ]`
- `[ $x –qe 0 –a $x –le 10 ]`
- `[ -w File2 -a -r File1 ]`
- `[ $x –eq 0 –o $y –eq 0 ]`

3.3.4 Условное управление

В командном интерпретаторе используются два оператора условного управления [5, 11-15]:

- оператор условного перехода `if`
- оператор выбора `case`

Условный оператор `if`. В общем случае оператор `if` имеет синтаксис:

```
if  условие1
  then  список1
    [elif  условие2
      then  список2]
    [else  список3]
fi
```

Ключевые слова `if`, `then`, `else`, `elif` и `fi` пишутся с начала строки. Допускается вложение произвольного числа операторов `if` (как и других операторов). Разумеется, `список` в каждом случае должен быть осмысленный и допустимый в данном контексте. Если

выполнено условие (как правило, это команда с кодом завершения 0), то выполняется список, иначе он пропускается.

Например:

```
echo -n "А какую оценку получил на экзамене?:"
read z
if [ $z = 5 ]
    then echo  Молодец !
    elif [ $z = 4 ]
        then echo  Все равно молодец !
        elif [ $z = 3 ]
            then echo  Все равно !
            elif [ $z = 2 ]
                then echo  Все !
                else echo   !
fi
```

Как правило, в качестве команды, входящей в условие используют команду test. Однако, возможно использование любой другой команды, главное требование – код завершения должен быть 0, чтобы выполнить список команд, написанных после слова then; либо любой другой код завершения, чтобы выполнить команды после слова else. Например, следующий командный файл:

```
    echo "Введите слово и имя файла:"
    read word file
    if grep $word $file > /dev/null
        then echo "$word есть в $file"
        else echo "$word нет в $file"
    fi
```

использует команду grep для поиска слова в файле. Результаты работы grep выводит на экран.

Также можно использовать и сложные условия. Например, в рассматриваемом ранее примере добавления записи в телефонный справочник можно реализовать дополнительную проверку вводимых сведений о человеке:

```
echo Добавление сведений в телефонный справочник
echo -n Введите имя:
read FIO
echo -n "Введите пол (муж/жен): "
read pol
if [ $pol != муж –a $pol != жен ]
```

```
    then echo "Неправильно указан пол"
    exit
fi
echo -n Введите телефон:
read phone
echo "$FIO      $pol    $phone" >> $HOME/telephones
```

Если пользователь неправильно введёт обозначение пола человека, то на экран будет выведено соответствующее сообщение и выполнение файла прекратится.

Оператор выбора case. Оператор выбора case имеет структуру:

```
    case   строка  in
    шаблон11|шаблон12)  список_команд1;;
    шаблон2)  список_команд2;;
        ...
    *)  список_командN;;
    esac
```

Здесь case, in и esac – служебные слова, строка (это может быть и один символ) сравнивается с шаблонами. Затем выполняется соответствующий список_команд выбранной строки, и управление передаётся на команду, идущую за ключевым словом esac. В конце строк выбора необходимо писать ;;. Для каждой альтернативы может быть выполнено несколько команд. Если эти команды будут записаны в одну строку, то символ ; будет использоваться как разделитель команд. Обычно последняя строка выбора имеет шаблон *, что в структуре case означает «любое значение». Эта строка выбирается, если не произошло совпадение значения переменной ни с одним из ранее записанных шаблонов, ограниченных скобкой). Значения просматриваются в порядке записи. Для организации нескольких вариантов выбора в одном шаблоне необходимо разделять варианты в шаблоне символом | без пробелов между ними. Также в шаблонах могут использоваться и другие метасимволы ?, [].

В качестве примера использования данного оператора модифицируем рассмотренную ранее задачу:

```
echo -n " А какую оценку получил на экзамене?: "
read z
case $z in
```

```
    5) echo Молодец !             ;;
    4) echo Все равно молодец !   ;;
    3) echo Все равно !           ;;
    2) echo Все !                 ;;
    *) echo  !                    ;;
esac
```

Как отмечалось выше, в шаблонах могут быть использованы метасимволы. Так, в следующем примере метасимволы применяются для задания диапазона символов:

```
echo "Нажмите клавишу и затем клавишу Enter."
read Key
case $Key in
  [a-z]   ) echo "Латинская буква в нижнем регистре";;
  [A-Z]   ) echo "Латинская буква в верхнем регистре";;
  [0-9]   ) echo "Цифра";;
  *       ) echo "Знак пунктуации, пробел или что-то другое";;
esac
```

В качестве самостоятельной работы дополните данный командный файл возможностью определения кириллических букв верхнего и нижнего регистра.

Оператор выбора case удобно использовать для реализации простейших меню. Например, функционал рассмотренного ранее командного файла по добавлению сведений в телефонный справочник можно расширить, добавив несколько различный действий в один командный файл:

```
echo Работа с телефонным справочником:
echo "1) Просмотр всех данных."
echo "2) Просмотр данных по указанному абоненту."
echo "3) Добавление данных."
echo "4) Удаление данных по указанному абоненту."
echo "5) Выход."
echo -n Ваш выбор:
read Action
Telephones=$HOME/telephones
case $Action in
    1)  cat $Telephones ;;
```

```
    2)  echo -n Введите имя:
        read FIO
        if grep $FIO $Telephones
            then :
            else  echo Сведений  о  таком  абоненте
нет
        fi
        ;;
    3)  echo -n Введите имя:
        read FIO
        echo -n "Введите пол (муж/жен): "
        read pol
        if [ $pol != муж –a $pol != жен ]
            then echo "Неправильно указан пол"
                exit
        fi
        echo -n Введите телефон:
        read phone
        echo "$FIO      $pol    $phone" >> $Tele-
phones
        ;;
    4)  echo -n Введите имя:
        read FIO
        cp $Telephones ${Telephones}.temp
        grep  -v $FIO ${Telephones}.temp > $Tele-
phones
        rm ${Telephones}.temp
        ;;
    5)  exit ;;
    *)  echo Команда выдрана неправильно ;;
esac
```

В данном примере пользователь, запустив командный файл, может выполнить одно из следующих действия: просмотреть весь справочник или только сведения об одном абоненте, удалить сведения об одном абоненте или добавить новые сведения. Добавление данных уже рассматривалось ранее, поэтому подробное описания этих действий приводить не будем. Для просмотра всех записей, содержащихся в справочнике, используется команда `cat`; для вывода сведений о конкретном человеке используется команда `grep`, при-

чём в случае необнаружения данной записи, производится вывод соответствующего сообщения. Удаления записи о каком-либо абоненте производится в три этапа: копирование справочника во временный файл; выборка всех записей, не содержащих удаляемую запись, и перенос их в старый справочник с заменой его содержимого; удаление временного справочника.

3.3.5 Циклическое управление

В командном интерпретаторе shell используются три оператора циклического управления [5, 11-15]:

- оператор цикла с перечислением for
- оператор цикла с истинным условием while
- оператор цикла с ложным условием until.

Оператор цикла с перечислением for. Оператор цикла for имеет структуру:

```
for имя [in список_значений]
do
    список_команд
done
```

где for – служебное слово, определяющее тип цикла,

do и done – служебные слова, выделяющие тело цикла.

Ключевые слова for, do, done пишутся с начала строки. Фрагмент [in список_значений] является необязательным и может отсутствовать. Если часть [in список_значений] опущена, то это означает in "$@" (то есть все позиционные параметры: in $1 $2 ... $n). Имя является параметром цикла и в теле цикла последовательно принимает значения, указанные в список_значений.

Например, командный файл, определяющий сумму последовательности чисел, будет иметь следующий вид:

```
for i in 1 2 3 4 5
do
    summa=`expr $summa + $i`
done
echo "Сумма равна $summa"
```

В этом примере имя i играет роль параметра цикла. Это имя можно рассматривать как shell-переменную, которой последова-

тельно присваиваются перечисленные после служебного слова in значения (i=1, i=2, …, i=5)

Часто используется форма

```
for i in *
```

означающая «для всех файлов текущего каталога».

Использование данной формы оператора for можно проиллюстрировать следующим кодом, выводящем на экран список файлов – исходных текстов программ, написанных на языке программирования Pascal, – содержащих объявление типов данных запись:

```
for f in $HOME/*.pas
do
    if [ -f  $f ]
    then
        if grep record $f > /dev/null
        then
            echo $f
        fi
    fi
done
```

В данном примере последовательно проверяются все файлы в домашнем каталоге пользователя, имеющие расширение pas. Если текущий файл является файлом и содержит слово record, то имя данного файла выводится на экран.

Как было указано ранее, при отсутствии в структуре оператора for фрагмента [in список_значений], значения берутся из параметров вызывающей команды. Например, в качестве параметров вызова в следующей программе указывают список месяцев, и по именам этих месяцев выдаётся название сезонов, в которые они входят:

```
for i
do
    case $i in
        декабрь|январь|февраль) echo $i: ЗИМА ;;
        март|апрель|май) echo $i: ВЕСНА ;;
        июнь|июль|август) echo $i: ЛЕТО ;;
        сентябрь|октябрь|ноябрь) echo $i: ОСЕНЬ ;;
      *) echo $i: Название месяца не определено  ;;
    esac
```

```
done
```

Теперь, при вызове командного файла

```
$season сентябрь май июнь Март
```

на экране будет:

```
сентябрь: ОСЕНЬ
май: ВЕСНА
июнь: ЛЕТО
Март: Название месяца не определено
```

В качестве список_значений может выступать и результат выполнения каких-либо команд. Например, следующий командный файл производит широковещательную рассылку сообщения из файла, переданного в качестве позиционного параметра:

```
if test $# -eq 0
then
    echo Не указан файл с сообщением
    exit
fi
for i in `who|cut -b 1-17`
do
    echo Отправляю сообщение $i....
    cat $@|write $i
done
```

В данном примере оператором if проверяется количество позиционных параметров, переданных при вызове командного файла. Если параметров нет, то выдаётся соответствующее сообщение и прекращается дальнейшее выполнение файла. Список_значений циклического оператора получается посредством выполнения конвейера команд who и cut -b 1-17. Первая команда возвращает список всех пользователей системы, конвейер передаёт их на вход второй команды, которая вырезает указанные подстроки из строк (в демонстрационном примере – это имена пользователей, которые указываются в первых 17 символах каждой строки вывода команды who). В теле цикла производится пересылка содержимого файла, указанного с помощью позиционного параметра, с помощью команды write. Конечно, для успешной работы указанного командного файла необходимо чтобы пользователь, который получает сообщения, разрешил их приём с помощью команды mesg.

Оператор цикла с истинным условием `while`. Структура `while`, также обеспечивающая циклическое выполнение расчётов, чаще всего используется в том случае, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле. Оператор цикла `while` имеет структуру:

```
while
    список_команд_1
do
    список_команд_2
done
```

где `while` – служебное слово, определяющее тип цикла с истинным условием.

Значением, проверяемым в цикле `while`, является код последней из списка команд, следующих за словом `while`. При каждой итерации выполняется `список_команд_1`. Список команд в теле цикла (между `do` и `done`) повторяется до тех пор, пока сохраняется истинность условия (то есть код завершения последней команды в теле цикла равен `0`) или цикл не будет прерван изнутри специальными командами (см. раздел 3.3.5.4). При первом входе в цикл условие должно выполняться.

Например, определение суммы последовательности чисел:

```
n=0
while [ $n -lt 5 ]    # пока n < 5
do
    n=`expr $n + 1`
    summa=`expr $summa + $n`
done
echo "Сумма равна $summa"
```

Как отмечалось выше, `список_команд_1` может состоять из нескольких операторов. Так следующий командный файл использует цикл `while` для ввода списка имён в файл:

```
echo "Введите список имён, разделяя Enter"
echo "Закончите ввод списка Ctrl-d"
while
    echo -n "Следующее имя:"
    read x
do
```

```
    echo $x >> xfile
done
echo "Файл содержит следующие имена:"
cat xfile
```

Результаты выполнения программы:

```
$ enter.name
Введите список имён, разделяя Enter
Закончите ввод списка Ctrl-d
Следующее имя: Вася
Следующее имя: Петя
Следующее имя: <^d>
Файл содержит следующие имена:
Вася
Петя
$
```

Если в качестве условия выполнения тела цикла установить оператор, всегда возвращающий код завершения `0`, то цикл будет бесконечным. Например:

```
while true
do
    ...
done
```

или, используя пустой оператор:

```
while :
do
    ...
done
```

Бесконечные циклы могут найти своё применение при решении различных задач. Например, в разделе 3.3.4 был рассмотрен пример создания командного файла для работы с телефонным справочником с использованием интерактивного меню. Однако недостатком такого способа организации работы со справочником является тот факт, что за один запуск файла производится только одно действие со справочником. Используя бесконечный цикл, можно исправить данную ситуацию:

```
echo Работа с телефонным справочником:
while :
do
```

```
    echo "1) Просмотр всех данных."
    echo "2) Просмотр данных по указанному або-
ненту."
    echo "3) Добавление данных."
    echo "4) Удаление данных по указанному або-
ненту."
    echo "5) Выход."
    echo -n Ваш выбор:
    read Action
    Telephones=$HOME/telephones
    case $Action in
        1)  cat $Telephones ;;
        2)  echo -n Введите имя:
            read FIO
            if grep $FIO $Telephones
            then :
            else echo Сведений о таком абонен-
те нет
            fi
            ;;
        3)  echo -n Введите имя:
            read FIO
            echo -n "Введите пол (муж/жен): "
            read pol
            if [ $pol != муж –a $pol != жен ]
            then echo "Неправильно указан пол"
                exit
            fi
            echo -n Введите телефон:
            read phone
            echo "$FIO    $pol   $phone" >> $Tel-
ephones
            ;;
        4)  echo -n Введите имя:
            read FIO
            cp $Telephones ${Telephones}.temp
            grep -v $FIO ${Telephones}.temp > $Tele-
phones
            rm ${Telephones}.temp
            ;;
```

```
        5)  exit ;;
        *)  echo Команда указана неправильно ;;
    esac
done
```

Также оператор цикла с истинным условием удобно использовать и для построчного чтения файла от начала до конца:

```
i=1
while
    read line
do
    echo "Строка №$i: $line"
    i=`expr $i + 1`
done
```

Вызов данного командного файла производится следующим образом:

```
$ sh ReadFileToLine < OutFile
```

где ReadFileToLine – имя командного файла;

OutFile – имя файла, который необходимо прочитать построчно.

Оператор цикла с ложным условием until. Структура until, также обеспечивающая выполнение циклических расчётов, схожа с рассмотренным раннее оператором while. Оператор until используется тогда, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле. Оператор цикла until имеет структуру:

```
until
    список_команд_1
do
    список_команд_2
done
```

где until – служебное слово, определяющее тип цикла с ложным условием. Список команд список_команд_2 в теле цикла (между do и done) повторяется до тех пор, пока сохраняется ложность условия (условие продолжения или окончания выполнения итераций определяется результатом выполнения последней команды в список_команд_1, причём список_команд_1 выполняется в каждой итерации) или цикл не будет прерван изнутри специальны-

ми командами. При первом входе в цикл условие не должно выполняться.

Отличие от оператора `while` состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) проверяется после каждого (в том числе и первого) выполнения команд тела цикла.

Например, рассмотренный ранее пример определения суммы последовательности чисел при использовании оператора цикла `until` примет вид:

```
n=0
until [ $n -gt 5 ]      # пока n не станет больше 5
do
    n=`expr $n + 1`
    summa=`expr $summa + $n`
done
echo "Сумма равна $summa"
```

В следующем примере программа ждёт, пока не будет создан файл `file1`.

```
until [ -f file1]
do
    echo "Файл ещё не создан. Подожду..."
    sleep 300
done
```

Здесь условием завершения цикла является существование файла с именем `file1`. Каждая итерация цикла заключается в пятиминутном ожидании (300 секунд), после чего цикл повторяется снова. При этом предполагается, что какой-нибудь другой процесс в некоторый момент создаст этот файл.

Оператор цикла удобно использовать для перебора всех позиционных параметров командного файла. Следующий пример производит удаление всех файлов, имена которых переданы через позиционные параметры:

```
if [ $# -eq 0 ]    # проверяем наличие параметров
then
    echo "Нет параметров ..."
    exit
fi
```

```
until [ -z "$1" ]        # перебираем параметры
do
    if [ -f $1 ]         # если файл существует
    then
        echo "Производим удаление файла $1"
        rm –f $1
    else
        echo "Файла $1 не существует"
    fi
    shift              #сдвиг позиционных параметров
done
```

Специальные команды управления циклом. Для прерывания выполнения цикла используют специальные команды `break` и `continue`.

- Команда `break` имеет следующий синтаксис:

  `break [n]`

  Данная команда позволяет выходить из объемлющего цикла, если таковой существует. Необязательный параметр `n` указывает число вложенных циклов, из которых надо выйти, например,

  `break 3`

  означает выход из трёх вложенных циклов. Если параметр `n` отсутствует, то это эквивалентно

  `break 1`.

- Команда `continue` имеет аналогичный синтаксис:

  `continue [n]`

  Эта команда позволяет прекратить выполнение текущего цикла и возвращает управление на начало цикла. Необязательный параметр `n` указывает номер вложенного цикла, на начало которого надо перейти, например,

  `continue 2`

  означает выход на начало второго (если считать из глубины) вложенного цикла. Если параметр `n` отсутствует, то это эквивалентно

  `continue 1`.

Например:

```
while :
do
```

```
    if date|grep 12:00
    then
        echo "Время"
        break
    fi
    sleep 30
done
```

Данная программа каждые 30 секунд проверяет текущее время: если оно равно 12:00, производится вывод фразы и выход из цикла. Данный пример можно использовать для выполнения каких-либо действий в назначенное время.

Наравне с предыдущими командами существует команда `exit`, синтаксис которой

```
exit [n]
```

Эта команда позволяет безусловно прекратить выполнение программы или процедуры и выйти из оболочки с кодом завершения `n` (признак конца файла также приведет к выходу из оболочки). Если необязательный параметр `n` не указан, то кодом завершения будет код завершения последней выполненной команды.

Методы использования команды `exit` были проиллюстрированы примерами предыдущих разделов.

3.3.6 Пользовательские функции

Аналогично любому другому языку программирования, в командном интерпретаторе `shell` можно использовать функции для группировки списка команд более логичным способом для последующего выполнения. [11-15]

Описание функции имеет вид:

```
имя()
{
    список_команд
}
```

Обращение к функции происходит по указанному в заголовке имени. При выполнении функции не создаётся нового процесса. Она выполняется в среде соответствующего процесса. Аргументы функции становятся её позиционными параметрами; имя функции – её нулевой параметр. Все переменные, используемые функцией, яв-

ляются глобальными. Прервать выполнение функции можно оператором return, который имеет следующий синтаксис:

```
return [n]
```

где необязательный параметр n – это код возврата. Код возврата пользовательской функции доступен с помощью $?.

Пример использования функций. В программе создадим пользовательскую функцию, которая выводит диалоговое сообщение на экран и ждёт ответа.

```
message()
{
while true
echo –n "$1 (y-да, n-нет, c-отмена)"
read answer
do
    case $answer in
        y) return 0;;          # ответ - ДА
        n) return 1;;          # ответ - НЕТ
        c) return 2;;          # ответ - ОТМЕНА
    esac
done
}
message "Начать работу\?"
case $? in
    0) echo "Начнем, пожалуй...";;
    1|2) echo "Жаль..."
         exit;;
esac
for i in *
do
    if [ -f $i ]
    then
        message "Удалить файл $i\?"
        case $? in
            0) rm $i >> log.log;;
            1) continue;;
            2) break;;
        esac
    fi
done
```

3.4 Инструменты отладки программы

3.4.1 Обработка прерываний `trap`

Командные процедуры обычно завершаются, если с терминала поступил сигнал прерывания. Однако часто бывает необходимо защитить выполнение программы от прерывания. Сигналы, поддерживающиеся ОС `UNIX`, приведены в таблице 18. [5, 10]

Таблица 18 – Сигналы прерываний `UNIX`.

Сигнал		**Описание**
Символьное представление	**Числовое представление**	
	`0`	Выход из командного интерпретатора
`SIGHUP`	`1`	Отбой (отключение удалённого абонента)
`SIGINT`	`2`	Прерывание от клавиши `<Del>`
`SIGQUIT`	`3`	Нестандартный выход
`SIGILL`	`4`	Неверная команда
`SIGTRAP`	`5`	Ловушка
`SIGFPE`	`8`	Исключительная ситуация при выполнении операций с плавающей запятой
`SIGKILL`	`9`	Уничтожение процесса (не перехватывается)
`SIGBUS`	`10`	Ошибка шины
`SIGSEGV`	`11`	Нарушение сегментации
`SIGSYS`	`12`	Неверный системный вызов
`SIGPIPE`	`13`	Запись в канал без чтения из него
`SIGALRM`	`14`	Будильник
`SIGTERM`	`15`	Программное завершение процесса (окончание выполнения)

Для защиты от прерываний существует команда `trap`, имеющая следующий синтаксис [11, 12]:

```
trap 'список_команд' сигналы
```

Если в операционной системе возникнут прерывания, чьи сигналы перечислены через пробел в параметре `сигналы`, то будет

выполнен список_команд, после чего управление вернётся в точку прерывания и продолжится выполнение командного файла (если в списке команд не была указана команда exit). Команды выполняются по порядку номеров сигналов. Любая попытка установить сигнал, игнорируемый данным процессом, не обрабатывается. Попытка прерывания по сигналу 11 приводит к ошибке.

Если список_команд опущен, то все прерывания устанавливаются в их начальные значения. Если в качестве параметра список_команд указана пустая строка, то этот сигнал игнорируется командным интерпретатором и вызываемыми им программами. Если в качестве сигналы указан 0 (ноль), то список_команд выполняется при выходе из командного интерпретатора. Команда trap без аргументов выводит список команд, связанных с каждым сигналом.

Сигналы могут быть обработаны одним из трёх способов:

- их можно игнорировать (в таком случае сигнал никогда не передаётся в соответствующий процесс);
- их можно перехватывать (в этом случае процесс должен задать действия, которые необходимо выполнить при получении сигнала);
- можно не задавать реакции на сигнал (сигнал вызовет завершение процесса без выполнения каких-либо дальнейших действий).

Если к моменту входа в командную процедуру уже задано игнорирование некоторых сигналов, то команда trap для этих сигналов (как и сами эти сигналы) игнорируется.

Например, если перед прекращением по прерываниям выполнения командного файла необходимо удалить какой-то файл tempor, то это может быть выполнено командой trap:

```
trap 'rm -f tempor; exit 1' 0 1 2 3 15
```

В самой системе UNIX не использует сигнал 0; в оболочке он означает выход из командной процедуры.

Команда trap позволяет игнорировать прерывания, если список_команд пустой. Так, например, команда:

```
trap '' 1 2 3 15
```

позволит игнорировать возникновение перечисленных сигналов как самим командным файлом, так и командами, запускаемые им.

Реакцию на сигналы можно переустанавливать. Например, команда:

```
trap 2 3
```

восстановит для сигналов `2` и `3` стандартную реакцию. Перечень реакций, установленных в текущий момент, можно получить с помощью команды `trap` без параметров:

```
trap
```

3.4.2 Обработка ошибок

Обработка ошибок средствами командной оболочки. Командный интерпретатор не имеет ни своего отладчика, ни отладочных команд или конструкций. Синтаксические ошибки или опечатки часто вызывают сообщения об ошибках, которые практически никак не помогают при отладке. [5, 11, 12, 14]

Одним из инструментов, который может помочь при отладке неработающих сценариев, является запуск командной оболочки со специальными параметрами. Рассмотрим способы запуска командных файлов.

Проверка без выполнения

Данный способ запуска командного файла только проверит наличие синтаксических ошибок, не запуская сам файл на исполнение:

```
sh -n имя_программы
```

Того же эффекта можно добиться, вставив в сам командный файл команду

```
set -n
```

или

```
set -o noexec
```

Однако некоторые из синтаксических ошибок не могут быть выявлены таким способом.

Покомандный вывод

Данный способ запуска командного файла выводит каждую команду прежде, чем она будет выполнена:

```
sh -v имя_программы
```

Того же эффекта можно добиться, вставив в сценарий команду:

```
set -v
```

или

```
set -o verbose.
```

Ключи `-n` и `-v` могут употребляться совместно:

```
sh -nv имя_программы
```

Вывод результата каждой команды

Данный способ запуска командного файла выводит в краткой форме результат исполнения каждой команды:

```
sh -x имя_программы
```

Того же эффекта можно добиться, вставив в сценарий команду:

```
set -x
```

или

```
set -o xtrace
```

Другие параметры команды `set`

Обработка ошибок, обнаруженных командным интерпретатором, зависит от вида ошибки и от того, используется ли интерпретатор в интерактивном режиме. Интерпретатор считается интерактивным, если его ввод и вывод связаны с некоторым терминалом. Интерпретатор, вызванный с флагом `–i`, также считается интерактивным.

Параметр `–e` вызывает выход из оболочки при обнаружении любой ошибки.

Отменяются любые параметры команды `set` с помощью указания снимаемого параметра со знаком + (плюс):

```
set +опция_без_-
```

Например, для отмены параметра проверки исходного кода без выполнения самого командного файла `–n` необходимо вызвать команду `set` следующим образом:

```
set +n
```

Отладка конвейеров. Использовать вышеперечисленные методы для отладки работы конвейеров затруднительно, так как при конвейерном выполнении команд стандартный вывод предыдущей команды перенаправляется в стандартный ввод следующей команды. В случае необходимости отладки конвейеров используется команда `tee`, которая сохраняет копию стандартного ввода в файл,

имя которого дано как аргумент, в тоже время не прерывая нормальной работы конвейера. [11, 12, 14]

Общий синтаксис команды `tee` следующий:

```
команда1 | tee промежуточный_файл | команда2
```

где `промежуточный_файл` – это файл, который сохраняет вывод команды для дальнейшего анализа.

Использование данной команды рассмотрим на следующем примере. Предположим, в командном файле необходимо определить идентификационные номера некоторых процессов (PID), запущенных текущим пользователем. Эту задачу можно решить с помощью такой конвейерной обработки:

```
ps -U $USER|grep sh|cut -b 1-6
```

где `ps -U $USER` – список процессов текущего пользователя;

`grep $1` – фильтр списка процессов текущего пользователя по какому-нибудь условию, задаваемому с помощью позиционного параметра при вызове командного файла;

`cut -b 1-6` – вырезание первых шести байтов – PID.

Можно воспользоваться командой `tee`, чтобы скопировать вывод каждой команды в отдельные файлы (`step1` и `step2`), не разрушая остальной конвейер:

```
ps -U $USER|tee step1|grep $1|tee step2|cut -b 1-6
```

Вызвав данный конвейер, указав вместо `$1`, например, `sh` (то есть определение PID всех процессов, где используется командный интерпретатор), получим следующие временные файлы:

```
$ ps -U $USER|tee step1|grep sh|tee step2|cut -b 1-6
 8772
 8865
$ cat step1
  PID  TT  STAT      TIME COMMAND
 8772  p0  S      0:00.37 -sh (sh)
 8863  p0  R+     0:00.07 ps -U student
 8864  p0  S+     0:00.03 tee step1
 8865  p0  S+     0:00.06 grep sh
 8866  p0  R+     0:00.04 tee step2
 8867  p0  R+     0:00.03 cut -b 1-6
$ cat step2
 8772  p0  S      0:00.37 -sh (sh)
 8865  p0  S+     0:00.06 grep sh
```

3.4.3 Необязательные параметры команды `set`

Необязательные параметры – это дополнительные ключи (опции), которые оказывают влияние на поведение командного файла и/или командного интерпретатора. [5, 11, 12, 14]

Команда `set` позволяет задавать дополнительные параметры прямо в исходном коде командного файла. В том месте программы, где необходимо, чтобы тот или иной параметр вступил в силу, необходимо использовать следующий синтаксис:

```
set -o имя_параметра
```

или в более короткой форме:

```
set –сокращенное_имя_параметра
```

Эти две формы записи совершенно идентичны по своему действию.

Полный синтаксис команды `set`:

```
set [abefhkmnptuvxldCHP] [-o опция] [аргумент]
```

Описание всех параметров представлено в таблице 19.

Таблица 19 – Параметры команды `set`

Сокращённое имя параметра	**Имя параметра**	**Описание**
`-a`	`allexport`	Отмечает переменные, которые модифицированы или созданы для экспорта
`-b`	`notify`	Вызывает прекращение фоновых заданий, о котором сообщает перед выводом следующего приглашения командной строки
`-e`	`errexit`	Немедленный выход, если выходное состояние команды ненулевое
`-f`	`noglob`	Выключает генерацию имени файла
`-h`		Обнаруживает и запоминает команды как определенные функции до того, как функция будет выполнена

Продолжение таблицы 19

Сокращённое имя параметра	**Имя параметра**	**Описание**
`-k`		В окружении команды располагаются все аргументы ключевых слов, не только те, которые предшествуют имени команды
`-m`	`monitor`	Разрешается управление заданиями
`-n`	`noexec`	Читает команды, но не выполняет их
`-o имя_опции`		Устанавливает флаг, соответствующий `имени_опции`
	`braceexpand`	Оболочка должна выполнить brace-расширение (фигурноскобочное расширение) – механизм, с помощью которого можно генерировать произвольные строки
	`emacs`	Использует интерфейс редактирования `emacs`
	`ignoreeof`	Оболочка не выходит при чтении `EOF`
	`interactive-comments`	Позволяет вызывать слово, начинающееся с #, и все оставшиеся символы на этой строке игнорировать в диалоговой оболочке
	`posix`	Изменяет режим интерпретатора в соответствии со стандартом Posix 1003.2 (предназначен для того, чтобы сделать режим строго подчиненным этому стандарту)
	`vi`	Использует интерфейс редактирования строки редактора `vi`

Продолжение таблицы 19

Сокращённое имя параметра	Имя параметра	Описание
`-p`	`privileged`	Включает привилегированный режим. В этом режиме файл `$ENV` не выполняется, и функции оболочки не наследуются из среды. Это включается автоматически начальными действиями, если идентификатор эффективного пользователя (группы) не равен идентификатору реального пользователя (группы). Выключение этой опции присваивает идентификатор эффективного пользователя (группы) идентификатору реального пользователя (группы)
`-t`		Выход после чтения и выполнения команды
`-u`	`nounset`	Во время замещения рассматривает незаданную переменную как ошибку
`-v`	`verbose`	Выдаёт строки ввода оболочки по мере их считывания
`-x`	`xtrace`	Выводит команды и их аргументы по мере выполнения команд
`-l`		Сохраняет и восстанавливает связывание имени в команде `for`
`-d`	`nohash`	Выключает хеширование команд, найденных для выполнения. Обычно команды запоминаются в хеш-таблице и, будучи однажды найденными, больше не ищутся
`-C`	`noclobber`	Не позволяет существующим файлам перенаправление вывода

Продолжение таблицы 19

Сокращённое имя параметра	Имя параметра	Описание
`-H`	`histexpand`	Закрывает замену стиля истории. Этот параметр принимается по умолчанию
`-P`	`physical`	Если установлен, не следует символьному указателю при выполнении команды типа `cd`, которая изменяет текущий каталог. Вместо этого используется физический каталог
`--`		Если нет аргументов, следующих за этим параметром, то не задаются позиционные параметры. В противном случае позиционные параметры присваиваются аргументам, даже если некоторые из них начинаются с `a-`
`-`		Сигнал конца параметра, вызывающего присваивание оставшихся аргументов позиционным параметрам. Параметры `-x` и `-v` выключаются. Если здесь нет аргументов, позиционный параметр не изменяется

При использовании `+` вместо `-` осуществляется выключение этих параметров. Параметры также могут использоваться при вызове интерпретатора. Текущий набор параметров может быть получен пользователем с помощью специальной переменной `$-`.

В синтаксисе команды `set` также используется необязательный параметр `аргумент`, который представляет собой новые позиционные переменные `$1`, `$2` и т.д. командного файла. Если все аргументы опущены, выводятся значения всех переменных.

Вопросы и задания для самоконтроля

1. Выведите на экран содержимое файла `/.profile`. Прокомментируйте его содержимое.

2. Что означает параметр `-n` команды `echo`?

3. Каким образом можно переопределить позиционные параметры?

4. Каким образом можно узнать сколько позиционных параметров было передано в командной строке при запуске командного файла?

5. С помощью какой команды производится ввод значений переменных с клавиатуры?

6. Что будет выведено на экран при выполнении следующего командного файла? Почему?

```
Var_1="Yes";
echo ${Var_1:-"No"}
```

7. С помощью какой команды можно вывести на экран все специальные именованные переменные среды?

8. Для чего производится экспорт переменных?

9. Напишите командный файл, который выводит на экран календарь указанного с клавиатуры года.

10. Напишите программу решения линейного уравнения вида $a \cdot x + b = 0$, где a, b – целочисленные коэффициенты, задаваемые пользователем с клавиатуры. Ответ вывести в виде обыкновенной дроби.

11. В следующих командах используются сложные операции сравнения. Попытайтесь определить коды завершения у каждой команды.

- `x=abc; [ $x -a -f File1 ] ;`
- `x=""; [ "$x" -a -f File1 ] ;`
- `x=""; [ "$x" -a -f File1 -o true ] ;`
- `x=abc; [ "$x" -a -f File1 -o ! true ] ;`

12. Напишите командный файл, который производит простейшие действия с файлами: копирование, переименование, удаление, вывод на экран. Выбор действия осуществлять с помощью оператора выбора `case`. В командном файле необходимо осуществлять проверку на то, чтобы файлы, указанные пользователем, были файлами, а не каталогами, и соответствующие действия над ними разрешены.

13. Модифицируйте программу предыдущего задания так, чтобы имена файлов, над которыми будут производиться действия, передавались с помощью позиционных параметров. Включите в командный файл проверку количества позиционных параметров и правильность используемого синтаксиса при вызове пользователем командного файла.

14. Напишите программу, которая просматривает телефонный справочник в файле `$HOME/telephones` на совпадение с параметрами командного файла и выводит на экран соответствующие записи. В качестве позиционных параметров передаются имена абонентов через пробел.

15. Создайте командный файл, печатающий на экране список всех файлов, указанного каталога. Причём необходимо выводить пояснение о типе выводимого файла: обычный файл, специальный файл, каталог или именованный канал. Предусмотреть параметр вызова данного командного файла, при котором рекуррентно выводится информация не только текущего каталога, но и всех каталогов, входящих в текущий каталог.

16. В разделе 3.3.5 представлен пример использования бесконечного цикла `while` для организации работы с телефонным справочником. Модернизируйте программу, используя условие окончания работы с телефонным справочником как условие окончания выполнения цикла.

17. Командный файл для работы с телефонным справочником, созданный в ходе выполнения предыдущего задания, имеет один весомый недостаток – при удалении записей из справочника удаляются все строки, содержащие указанное имя абонента. Модернизируйте программу таким образом, чтобы удаление каждой конкретной записи из справочника сопровождалось подтверждением пользователя. Для решения данной задачи удобно использовать построчное чтение телефонного справочника и нахождение интересующей записи, а при её нахождении – обращение к пользователю за подтверждением удаления записи.

18. Напишите программу игры «Угадай число», в которой программа загадывает число в диапазоне от 1 до 99, а игрок должен за меньшее количество попыток угадать данное число. В программе необходимо предусмотреть ведение таблицы рекордов, представляющей собой отдельный файл, содержащий имя игрока, число по-

пыток отгадывания числа и само загаданное число. Генерацию псевдослучайного числа можно осуществить следующим образом:

```
date > \dev\null
Random=`echo $$|cut -b 2-3`
```

Переменная `Random` будет содержать искомое число. При обновлении таблицы рекордов необходимо учитывать введённые ранее имена игроков: если игрок в таблице уже есть, то обновить его рекорд, если нет – добавить. Таблица должна быть отсортирована по возрастанию числа попыток угадывания числа. Для сортировки строк можно использовать команду `sort`, а для удаления информации из строк – команду `tr`.

19. В командном файле, предназначенном для работы с телефонным справочником и созданном в рамках выполнения задания для самостоятельной работы №17, необходимо оформить все действия со справочником: удаление, добавление, изменение и вывод на экран – с помощью пользовательских функций.

20. Модернизируйте программу игры «Угадай число», созданную в рамках выполнения задания для самостоятельной работы №18, реализовав сравнение вводимого пользователем числа с загаданным программой с помощью пользовательской функции.

21. С помощью оператора `trap` выведите какое-либо сообщение при завершении выполнения командного файла.

22. С помощью оператора `trap` произведите автоматическую сортировку телефонного справочника (файл `$HOME/telephones`, полученный в ходе выполнения заданий для самостоятельной работы №19) при завершении выполнения командного файла, предназначенного для работы со справочником.

3. С помощью соответствующих параметров команды `set` задайте проверку синтаксиса и вывода каждой команды программы игры «Угадай число», созданной в рамках выполнения задания для самостоятельной работы №19. Прокомментируйте вывод программы.

Заключение

Операционные системы семейства UNIX очень надежны в плане функционирования, являются устойчивыми к сбоям, толерантными к ошибкам оператора и пользователя, но требуют повышенного внимания при их использовании и администрировании, тщательного управления её работой и регулярного обслуживания. Для успешного выполнения задач администрирования необходимы глубокие знания программной архитектуры операционной системы и аппаратной части обслуживаемого оборудования.

С помощью командного интерпретатора shell пользователь может либо давать команды операционной системе по отдельности, либо запускать скрипты, состоящие из списка команд. Он является мощным скриптовым языком программирования и обеспечивает условное выполнение и управление потоками данных. Командный интерпретатор shell является де-факто стандартом командной оболочки, используемой в операционных системах семейства UNIX, и доступна почти в любом дистрибутиве UNIX. Однако существует много разновидностей командных оболочек, как основанных на shell, так и имеющих совершенно иной синтаксис, но не уступающих по функциональности.

Разновидностями оригинального командного интерпретатора являются [16]: ksh (синтаксис совместим, функциональность интерактивности увеличена); pdksh (открытая реализация ksh); bash (расширенная оболочка shell) и др. Разновидности командных интерпретаторов с синтаксисом на основе языка программирования Си [16]: csh (проприетарная оболочка из состава дистрибутива BSD, имеет Си-образный синтаксис и не является POSIX-совместимой); tcsh (свободная реализация csh с интерактивными возможностями, не уступающими bash) и др. Также существуют современные микро-версии, предназначенные для встраиваемых систем: ash, busybox и др.

Таким образом, пользователи операционных систем семейства UNIX имеют большой выбор инструментальных средств администрирования и сопровождения систем, что является ещё один доводом в пользу использования данных систем.

Список литературы

1. UNIX – Википедия – [Электронный ресурс] – Режим доступа – [http://ru.wikipedia.org/wiki/UNIX] – Загл. с экрана.
2. Дегтярев Е.К. Введение в UNIX / Е.К. Дегтярев. – М.: МП «Память», 1991. – 156 с.
3. Roman aka Docent. История UNIX // СпецХакер, – №47. – 2004. – С.4-7.
4. TanaT Вселенная UNIX. Эту историю должен знать каждый! // Xakep, – №49. – 2004. – С.78-83.
5. Баурн С. Операционная система UNIX / С. Баурн. – М.: Мир, 1986. – 464 с.
6. Морис Дж. Бах. Архитектура операционной системы Unix – [Электронный ресурс] – Режим доступа – [http://lib.ru/BACH/] – Загл. с экрана.
7. BSD – Википедия – [Электронный ресурс] – Режим доступа – [http://ru.wikipedia.org/wiki/BSD] – Загл. с экрана.
8. Linux – Википедия – [Электронный ресурс] – Режим доступа – [http://ru.wikipedia.org/wiki/GNU/Linux] – Загл. с экрана.
9. Кристиан К. Операционная система UNIX / К. Кристиан. – М.: Финансы и статистика, 1985. – 320 с.
10. Магда Ю.С. Администрирование UNIX / Ю.С. Магда. – СПб.: БХВ-Петербург, 2005. – 800 с.
11. Соловьев А. Программирование на shell (UNIX) – [Электронный ресурс] – Режим доступа – [http://www.linuxcenter.ru/lib/books/shell/] – Загл. с экрана.
12. Интерпретатор командного языка shell – [Электронный ресурс] – Режим доступа – [http://lib.ru/unixhelp/unixshell.txt] – Загл. с экрана.
13. Семенюченко А. Шелл для кодера // СпецХакер, – №51. – 2005. – С.82-85.
14. Mendel Cooper. Искусство написания Bas-скриптов – [Электронный ресурс] – Режим доступа – [http://www.opennet.ru/docs/RUS/bash_scripting_guide/] – Загл. с экрана.
15. Гребенников Р. Кодим в Bash! // Xakep, – №55. – 2005. – С.64-65.
16. Командная оболочка UNIX – Википедия – [Электронный ресурс] – Режим доступа – [http://ru.wikipedia.org/wiki/Командная_оболочка_UNIX] – Загл. с экрана.

Учебное издание

Ляховец Михаил Васильевич

ПРОГРАММИРОВАНИЕ В UNIX-СИСТЕМАХ

Учебное пособие

Редактор _____

Подписано в печать «___»_____ 20___г.
Формат бумаги 60?84 1/16. Бумага писчая. Печать офсетная.
Усл. печ. л. 7,98 Уч.-изд. л. 8,50 Тираж 320 экз. Заказ ___.

Сибирский государственный индустриальный университет,
654007, г. Новокузнецк, ул. Кирова, 42.
Типография СибГИУ