ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

С.Н. Мамойленко

КОМАНДНАЯ ОБОЛОЧКА BASH

Новосибирск - 2019

ОГЛАВЛЕНИЕ

ГЛАВА 1. ИСТОРИЯ СОЗДАНИЯ ОПЕРАЦИОННОЙ СИСТЕМЫ LIINUX

«Я стал распространять свою операционную систему прежде всего, чтобы доказать, что все это не пустая болтовня – я действительно что-то сделал».

Линас Торвальдс

Операционная система (ОС) Linux изначально разработана Линасом Торвальдсом (рис. 1), студентом Унивеситета Хелсинки (University of Helsinky, Финляндия).

Рис. 1. Линас Торвальдс. Создатель операционной системы Linux

Линас Торвальдс родился 28 декабря 1969 года в городе Хелсинки (Финляндия). Родители, шведскоговорящие финны Нильс и Анна Торвальдс, были в 60-х годах студентами-радикалами, отец даже был коммунистом, в середине 70-х проведшим год в Москве. Линус был назван в честь Линаса Полинга. В школе преуспевал в физике и математике. Был малообщительным, скромным мальчиком. Его часто дразнили из-за политических взглядов его отца. В 1988 году Линас поступил в Университет Хельсинки, который окончил в 1996, получив степень магистра кибернетики.

Значимым событием в жизни Торвальдса было прочтение им книги Эндрю Таненбаума «Операционные системы: разработка и реализация» (Operating Systems: Design and Implementation). В этой книге, на примере написанной Таненбаумом ОС Minix, представлена структура систем семейства UNIX.

В начале 1991 году Линас Торвальдс приобрел персональный компьютер, собранный на базе центрального процессора Intel 80386, который на тот момент был только что выпущен компанией Intel. В комплекс поставки входила урезанная версия операционной системы DOS, но Линас был очень заинтересован Minix, которую он первым делом и установил.

Изучив систему Minix, а также прочитав обсуждение этой системы в посвещённой ей телеконференции, Линас обнаружил в ней множество недостатков. Больше всего нареканий вызвала программа эмуляции терминала, которую Линас использовал для подключения к университетскому компьютеру.

Несмотря на то, что исходные коды ОС Minix были доступны в полной мере, Линас решил не дорабатывать уже имеющуюся программу, а написать собственную программу эмуляции терминала, которая использовала только возможности аппаратуры и никак небыла связана с Minix. Для чего первым

делом ему пришлось детально изучить как функционирует центральный процессор Intel 80386.

Первая тестовая программа, которую написал Линас, имела два независимых процесса, один их которых использовался для выдачи на экран буквы A, а другой -- для выдачи буквы B. С практической точки зрения это было абсолютно бессмысленно, но зато позволило ему изучить принципы переключения между процессами. На написание этой программы Линасу потребовался почти месяц, потому что во всем ему приходилось разбираться с нуля, читая горы технической документации.

В дальнейшем один процесс был доработан таким образом, чтобы он читал информацию, поступающую от модема и выдавал её на экран, а другой – считывал информацию с клавиатуры и передавал её модему.

Современем эмулятор Линаса обрастал дополнительными функциями: появилась возможность передавать файлы (для чего потребовалось разработать драйвер для доступа к файловой системе ОС Minix), появилась первая версия коммандной оболочки (переделанная из shell) и т.д. Линас называл ее "программой эмуляции терминала типа gnu-emacs". Gnu-emacs начинался как текстовый редактор, но его создатели встроили в него множество разных функций.

Первым упоминанием о том, что Линас задумался о разработке собственной операционной системы, было сообщение, сделанное им 3 июля 1991 года в телеконференции Minix, которое содержало следующее:

> *Привет, сетяне!*
> *Я сейчас делаю один проект (под minix), и мне нужно определение стандартов POSIX. Кто-нибудь знает, где можно взять их последнюю версию, желательно в электронном виде? Ftp-сайты годятся.*

Стандарты POSIX – это подробнейшие правила для каждого из системных вызовов в Unix. Они вырабатываются специальной организацией, состоящей из представителей компаний, которые хотят договориться об общих стандартах для Unix.

Это сообщени не вызвало особого интереса, однако было замечено Ари Лемке преподавателем из Технического университете Хелсинки, который обратился к Линасу с предложение выложить его операционную систему (когда она будет готова) на FTP-сайт, чтобы все желающие могли бы ознакомиться с ней.

Первым именем для нынешней операционной системы Linux было Freax (от англ. freak - фанат - и на конце x от Unix). Это имя непонравилось Ари Лемке. Ему больше нравилось рабочее название – Linux, которое Линас использовал в процессе разработки своей операционной системе.

Получив письмо от Ари Лемке, Линас ещё больше воодушевился созданием новой операционной системы, о чем свидетельствует его сообщение в телеконференции:

Привет всем пользователям minix! Я тут пишу (бесплатную) операционную систему (любительскую версию - она не будет такой большой и профессиональной, как gnu) для 386-х и 486-х AT. Я вожусь с этим с апреля, и она, похоже, скоро будет готова. Напишите мне, кому что нравится/не нравится в minix, поскольку моя ОС на нее похожа (кроме всего прочего, у нее – по практическим соображениям - то же физическое размещение файловой системы).
Пока что я перенес в нее bash (1.08) и gcc (1.40), и все вроде работает. Значит, в ближайшие месяцы у меня получится уже что-то работающее, и мне бы хотелось знать, какие функции нужны большинству. Все заявки принимаются, но выполнение не гарантируется : -)

Линус.

PS. Она свободна от кода minix и включает мультизадачную файловую систему. Она НЕ переносима (используется переключение задач 386 и пр.) и, возможно, никогда не будет поддерживать ничего, кроме AT-винчестеров - потому что у меня больше ничего нет : - (.

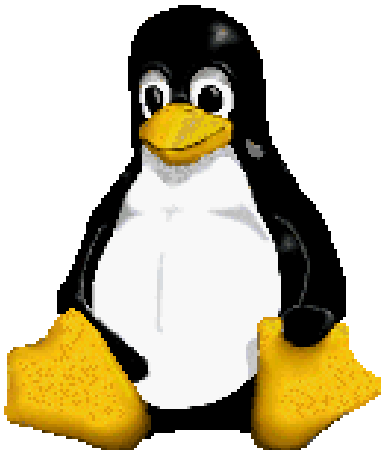
Рис. 2. Пингвин Такс. Личный талисман Линаса Торвальдса, ставший эмблемой ОС Linux

17 сентября 1991 года Линус выложил исходный код программы (версии 0.01) для общедоступной загрузки. Система сразу же вызвала большой интерес. Линасу начали приходит сообщения об ошибках. В начале октября 1991 года была выпущена версия 0.02, с исправлением ошибок и добавлением некоторых программ. В ноябре была выпущена версия 0.03. В конце ноября появилась версии 0.10. Такой скачек нумерации был связан с тем, что Линасу пришлось полностью отказаться от Minix (так как он по ошибке испортил раздел жесткого диска на которой она находилась) и начать использовать только собственную операционную систему. Ещё через несколько недель появилась версия 0.11. В начале января 1992 года была выпущена версия 0.12, в которой Линас реализовал механизмы страничной подкачки. После этого популярность свободно распространяемой операционной системы Linux начала расти не по дням, а по часам.

В течении нескольких лет ядро новой системы дорабатывалось и в марте 1994 года в аудитории Факультета информатики Университет Хелсинки была представлена версия 1.0.

С тех пор прошло уже больше десяти лет. Сотни, потом тысячи программистов стали интересоваться системой и работать над её улучшением и дополнением. Она распространялась и по сей день распространяется на условиях общественной лицензии GNU — GPL. Появилось немело

коммерческих версий этой операционной системы. По сути, сегодня под название Linux скрывается ядро операционной системы, которое снабжается множеством прикладных программ, которые создаются коллективом программистов со всего мира.

В заключении истории Linux следует сказать, что эмблемой ОС Linux является пингвин Такс (Tux) - личный талисман Линуса Торвальдса (рис. 2).

ГЛАВА 2. ПЕРВОЕ ЗНАКОМСТВО С LINUX

«Вы скорбите о тех временах, когда мужчины были настоящими мужчинами и сами писали драйверы устройств?»

Из объявления Линаса Торвальдса о выпуске Linux версии 0.0.2.

Первое впечатление от работы с любым компьютером связано с тем, каким образом он «общается» с пользователем. Способ общения определяется ***интерфейсом*** используемой операционной системы.

2.1. Интерфейс операционной системы

В Linux, как и в любой другой ОС семейства Unix, может применяется два типа интерфейса: *текстовый* и *графический*.

В первом случае (рис. 3), пользователь «общается» с компьютером вводя команды с клавиатуры и получает ответ в виде текстовых сообщений на экране монитора. Во втором случае (рис. 4) пользователь имеет возможность просматривать графическую информацию на экране и управлять работой компьютера при помощи клавиатуры и дополнительных средств ввода, таких как манипулятор «мышь», световое перо, сенсорный экран и т.п.

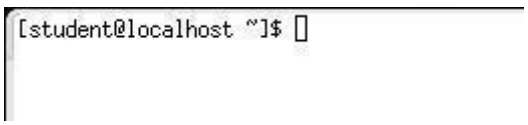

Рис. 3. Вид текстового интерфейса операционной системы (командная оболочка)

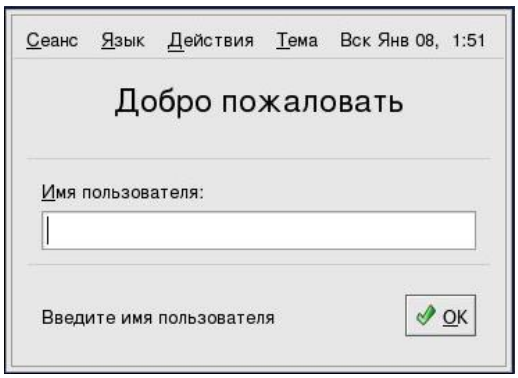

Рис. 4. Вид графического интерфейса операционной системы (окно входа в систему)

Чаще всего пользователям предоставляется несколько виртуальных терминалов[1], каждый из которых может использовать свой тип интерфейса. Переключение между терминалами производится комбинацией клавиш <CTRL+ALT+F?>[2], где F? означает одну из функциональных клавиш F1 – F12. Обычно (хотя и не всегда) первые шесть терминалов используют текстовый режим, а седьмой терминал – графический.

[1] Что такое терминал, смотрите в курсе «Организация ЭВМ и систем».

[2] Если текущий терминал использует текстовый интерфейс, то для переключения можно использовать комбинацию клавиш <Alt+F?>.

2.2. Процедура регистрации пользователя в системе

Не зависимо от используемого типа интерфейса, первое, что необходимо сделать прежде, чем работать с операционной системой Linux - это пройти ***процедуру идентификации*** (регистрации), которая преследует две цели: во-первых, система проверяет, тот ли Вы, за кого себя выдаете, и, во-вторых, она открывает сеанс работы и настраивает для пользователя рабочую среду.

Независимо от типа используемого интерфейса в процессе регистрации, называемой так же "logging in", система запрашивает имя пользователя[3] (*login*) и пароль (*password*).

Имя пользователя чаще всего вводится с клавиатуры. Иногда (если число пользователей компьютера невелико) и используется графический интерфейс, то предоставляется возможность выбрать пользователя из списка. Пароль всегда вводится с клавиатуры, причем набираемые символы на экране не отображаются (или отображаются в виде каких-либо других символов, например символа * - «звездочка»). Ввод имени пользователя и пароля завершается нажатием клавиши Enter[4]. Позиция (или поле), куда будет вводиться информация на экране, обычно указывается при помощи ***курсора*** – мигающей вертикальной или горизонтальной чертой.

Пример регистрации в системе с использованием текстового режима. После загрузки операционной системы на экране появится приглашение для ввода имени пользователя (login):

```
ASPLinux release 10.0
Kernel 2.6.20 on an i686
Welcome to Computer systems chair.
login: _
```

В этом случае в поле имя (*login*) надо ввести с клавиатуры строку, соответствующую Вашему системному имени (например, *student)* и нажать клавишу Enter. После этого система попросит ввести пароль (*password*):

```
ASPLinux release 10.0
Kernel 2.6.20 on an i686
Welcome to Computer systems chair.
login: student
password:
```

Если имя пользователя и пароль введены неправильно, то будет выведено сообщение об ошибке (login incorrect) и процедура ввода повториться заново[5]. В случае правильного ввода, на экран будет выдано приглашение к вводу команд (если иное не предусмотрено рабочей средой пользователя) и система будет готова к их исполнению:

---

[3] Пользователи создаются администратором системы.

[4] Обратите внимание, что в именах пользователей и паролях учитывается регистр символов. Поэтому под именами Student и student могут работать два совершенно разных пользователя.

[5] Следует помнить, что в случае нескольких неудачных попыток входа в систему компьютер может быть заблокирован.

```
ASPLinux release 10.0
Kernel 2.6.20 on an i686
Welcome to Computer systems chair.
login: student
password:
[student@rwp1 student]$_
```

При использовании графического интерфейса процедура регистрации производится аналогичным образом, а после её завершения появится графическое рабочее пространство, называемое рабочим столом (рис. 5).

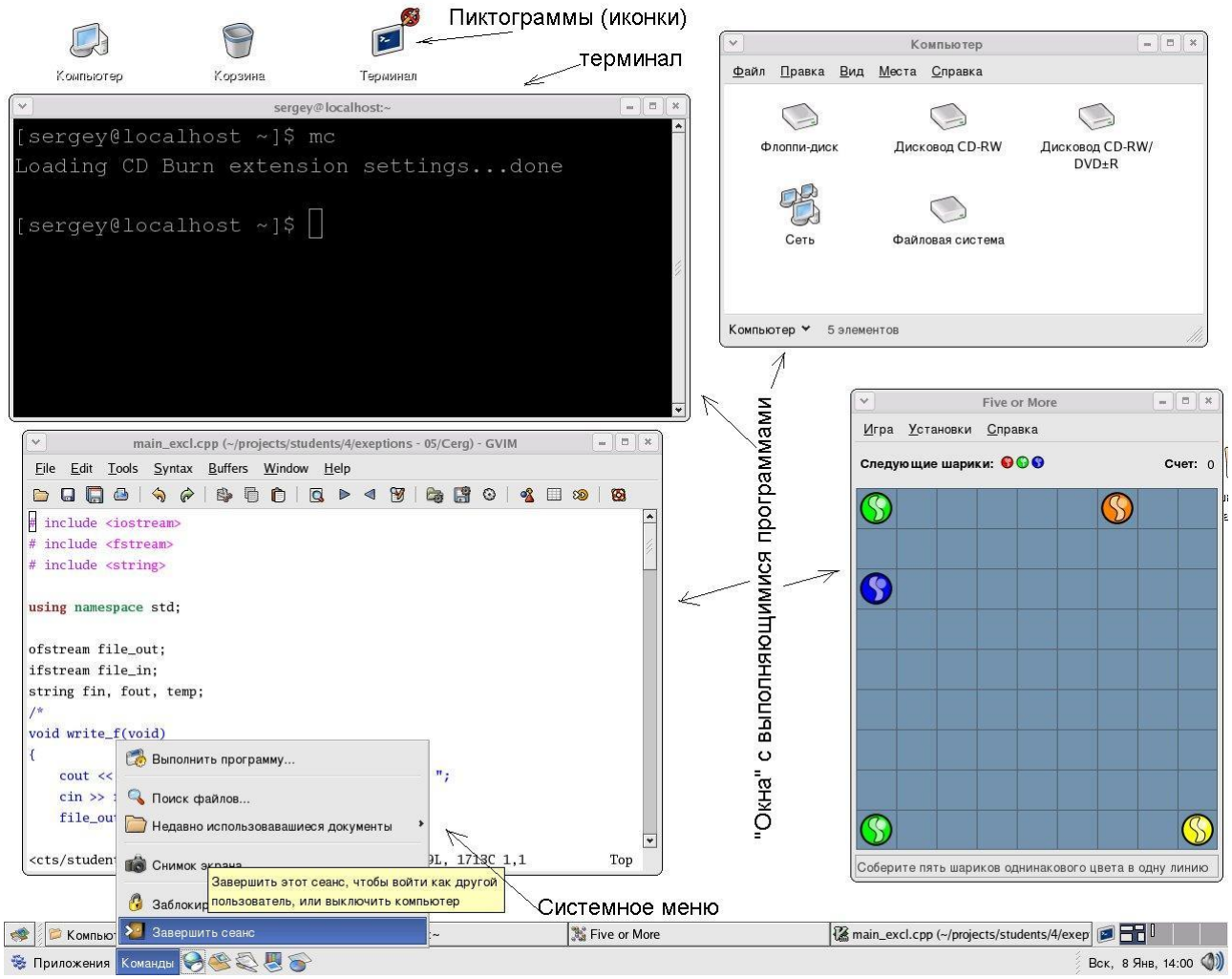

Рис. 5. Вид рабочего стола после регистрации в системе и системное меню с пунктом «Завершить сеанс»

2.3. Процедура завершения сеанса работы с системой

Когда пользователь закончил работу с системой, он должен завершить свой сеанс работы. Для этого он должен ввести команду ***logout*** и нажать клавишу <Enter>, после чего операционная система завершит текущий сеанс, очистит экран и вернётся в режим регистрации пользователя. В графическом режиме обычно вместо команды logout пользователь выбирает один из пунктов системного меню (рис. 5).

ГЛАВА 3. СИСТЕМА ХРАНЕНИЯ ИНФОРМАЦИИ

«Мой эмулятор терминала обрастал наворотами. Я регулярно использовал его, чтобы подключиться к университетскому компьютеру и получить почту или поучаствовать в конференции по Minix. Беда была в том, что я хотел скачивать и закачивать файлы. То есть мне нужно было уметь писать на диск. Для этого моей программе эмуляции нужен был драйвер дисковода. А еще ей был нужен драйвер файловой системы, чтобы она могла вникать в организацию диска и записывать скачиваемые файлы».

Линас Торвальдс. Из книги «Just for fun».

Основным назначением любой вычислительной машины является обработка информации. Этот процесс требует наличия двух составляющих – саму информацию и правила (алгоритмы, программы), согласно которым эта информация будет опрабатываться. Поэтому приступая к работе с компьютером важно четко определиться как в нём хранятся данные.

В современных компьютерах вся хранимая информация представляется в виде ***файлов***, каждый из которых имеет своё имя, содержимое и описывается определённым набором атрибутов.

3.1. Имена файлов. Маски файлов.

Имя файла – последовательность символов из заданного алфавита, включающего русские и английские буквы, цифры и символы: подчеркивание (_), тире (-) и точка (.). В операционной системе Linux *длинна имени* файла ограничена 255 символами. Часто (хотя это и не обязательно) файлам дают такие имена, чтобы они отражали их содержание. Например, файл с именем *lab1.c* – хранит в себе программу на языке Си.

Если в имени файла присутсвует символ точка, то часть имени, следующая после него, называется **расширением** или **суффиксом**. Обычно (хотя это и не обязательно) суффиксы используются для того, чтобы указать тип содержимого файла. В Linux имя файла может содержать несколько символов точка, что используется для указания на изменения типов содержимого файла. Например, имя *lab1.c.gz*, говорит о том, что файл *lab1.c*, содержащий программу на языке Си, был сжат при помощи программф gzip.

Если **символ точка является первым** в имени файла, то он указывает на специальное назначение этого файла[6]. Например файл *.bashrc*.

[6] Обычно такие файла содержат настройки программ пользователя.

В операционной системе Linux в именах файлов кроме символов из заданного алфавита возможно использованиее других (специальных) символов, например «пробел», «звездочка», «процент» и т.п. В этом случае[7], чтобы указать специальный символ, необхоимо перед ним поставить знак ***«\» (обратный слеш)*** или заключить имя файла в **одинарные опострофы или двойные кавычки**[8]. Такой способ также называют **цитированием символа**. Например, *Имя\ файла\ со\ спецаиальными\ знаками\ *_и_\?.doc* или *'Имя файла со специальными знаками *_и_?.doc'* или *"Имя файла со специальными знаками *_и_?.doc"*.

Часто при работе в коммандной оболочке пользователям необходимо в одном имени указать сразу несколько файлов (группу)[9]. Для этого в именах файлов включаются специальные символы[10] «*» (звезда), «?» (вопрос), «[]» (квадратные скобки), которые называются ***символами расширения***. Имя, содержащее эти символы преобразуется коммандной оболочкой в список имен.

Знак ***«*» (звездочка)*** применяется для того, чтобы указать что в этом месте имени файла может находиться любое число (включая ноль) любых символов. Например, имя *а** определяет все файлы, начинающиеся с буквы "а". Имя **xx*.dat* включает любое имя файла, оканчивающееся символами *.dat*, в имени которого присутствуют буквы "xx". Это могут быть имена *abxx.dat*, *lxx33.dat*, *xxyyzz.dat* или даже *xx.dat*.

Знаком ***«?» (вопрос)*** указывается, что на его месте в имени файла может находиться один (не больше, и не меньше) любой символ. Например, имя ??? определяет все файлы, имена которых состоят только из трех символов. Сгенерировать список имен, окачивающихся тремя символами, отделенными от остальной части имени точкой, позволяет выражение .???.

Выражение «[]» используется для указание множества символов, которые могут находиться в имени файла на том месте, где располагается открывающаяся квадратная скобка. Между скобками указываются необходимые символы или их диапазон (с использованием символа «-» (тире)). Например имя f[124], соответсвует файлам f1, f2, f4. А имя f[1-4] – файлам f1,f2,f3,f4. Имя f[abc-f], файлам fa,fb,fc,fd,fe,ff. С помощью квадратных скобок и знака '-' можно указывать сразу несколько диапазонов символов. Например, [123, 5-8, abe, A-E]. Следует отметить, что если символы «*» и «?» указаны внути квадратных скобок, то они являются обычными символами, а не символами расширения. А символ «-» наоборот имеет смысл «диапазон» только внутри скобок. Например, имя -[*?]abc будет определять группу из всего лишь двух имён: '-*abc' и '-?abc'.

---

[7] При работе с файлами, содержащими в своих именах специальные символы, с использованием графического интерфейса или специальных программ оболочек этого делать нет необходимости (операционная система сама преобразует имя файла в нужную форму)

[8] Об отличии одинарных апострофов от двойных кавычек будет рассказано далее.

[9] Например, для того, чтобы удалить несколько ненужных файлов с похожими именами.

[10] Эти символы указываются без обратного слеша. Если перед ними поставить обратный слеш, то они потеряют свою функцию символов расширения, а станут просто символом «звездочка», «вопросик», «квадратные скобки».

3.2. Система именования файлов. Каталоги. Путь файла.

Для структурирования хранимой информации файлы, в свою очередь, объединяются в ***каталоги*** (директории или папки), организованые в виде древовидной структуры (дерева). **Имя каталога**, как и имя файла, - это последовательность символов из заданного алфавита.

Основным каталогом для файловой системы является корневой каталог, обозначаемый символом “/” (слеш). Все остальные файлы и каталоги располагаются в рамках структуры, порожденной корневым каталогом, независимо от их физического местонахождения[11].

Чтобы указать в каком каталоге находитс файл, нужно определить его **путь** – перечень каталогов, которые небходимо пройти до файла. При перечислении каталоги разделяются символом “/” (**слеш**). Например, путь “кат1/кат2/кат3/файл“ – означает, что «файл» находится в каталоге «кат3», который находится в каталоге «кат2», находящемся в каталоге «кат1».

Указывать путь файла можно либо ***относительно*** (начало пути находится в каком-то каталоге), либо ***абсолютно*** (началом пути является корневой каталог). Например, кат1/кат2 – это относительный путь, в котором говориться, что надо перейти в кат1, который находится в текущем каталоге, а затем в кат2. А /кат1/кат2 - это абсолютный путь, в котором говориться, что надо перейти в кат1, который находится в корневом каталоге, а затем в кат2.

Для указания в относительном пути на текущий или на родительский каталоги используются соответсвенно символы «.» (**точка**) и «..» (**две точки**). Путь «./файл» - означает, что файл находится в текущем каталоге. Путь «../кат2/кат3/файл» - означает, что файл находится в каталоге «кат3», который находится в каталоге «кат2», находящемся в родительском (для текущего) каталоге. Можно перемещаться сразу на несколько уровней по дереву каталогов, задавая имя «..» соответствующее количество раз. Например, путь «../../», означает родительский каталог, расположенный на два уровня вверх.

В операционной системе Linux, как и в любой другой ОС, предусмотрет ряд обазательных каталогов:

- каталог **/bin.** В нём находятся наиболее часто употребляемые программы и утилиты системы, как правило, общего пользования;
- каталог **/dev.** Содержит специальные файлы устройств, являющиеся интерфейсом доступа к периферийным устройствам. Каталог */dev* может содержать несколько подкаталогов, группирующих специальные файлы устройств одного типа;
- каталог **/etc.** В этом каталоге находятся системные конфигурационные файлы и многие утилиты администрирования;
- каталог **/lib.** В нем находятся библиотечные файлы языка Си и других языков программирования. Стандартные названия библиотечных файлов имеют вид *lib*.a* (или *lib*.so*). Например, стандартная библиотека Си

[11] Подробнее о физической организации смотри ниже

называется libc.a, библиотека системы X Window System имеет имя *libX11.a*. Часть библиотечных файлов также находится в каталоге */usr/lib*;

- каталог **/lost+found.** Каталог "потерянных" файлов. Ошибки в файловой системы, возникающие при неправильном выключении компьютера или аппаратных сбоях, могут привести к появлению т.н. "безымянных" файлов. Структура и содержимое файла являются правильными, однако для него отсутствует имя в каком-либо из каталогов. Программы проверки и восстановления файловой системы помещают такие файлы в каталог */lost+found* под системными числовыми именами;
- каталог **/mnt.** Стандартный каталог для временного связываения (монтирования) физических файловых систем с корневой для получения дерева логической файловой системы. Обычно содержимое каталога */mnt* зависит от назначения системы и полностью определяется администратором;
- каталог **/home**. Содержит домашние каталоги пользователей. Например, для домашнего каталога пользователя *student* будет называться */home/student*;
- каталог **/usr.** В этом каталоге находятся подкаталоги различных сервисных подсистем. */usr/bin* исполняемые утилиты, */usr/local* дополнительные программы, используемые на данном компьютере, файлы заголовков */usr/include, /usr/man* электронные справочники и т.д.;
- каталог ***/var.*** Этот каталог используется для хранения временных файлов различных сервисных программ (системы печати, почтового сервиса и т.п.);
- каталог **/tmp.** Общедоступный каталог. Используется для хранения временной информации.

Как и в именах файлов при указании путей возможно использовать спецаильные символы и символы расширения. В дополнении к этому, для указания абсолютного пути файла, находящегося внутри домашнего каталога пользователя можно использовать специальный символ **(~) (тильда).** Например, путь «***~имя_пользователя***», будет равен пути */home/имя_пользователя*. А путь «~/кат1/файл», обозначает файл, находящийся в каталоге кат1, располагающемся в домашнем каталоге текущего пользователя.

3.3. Типы файлов

В зависимости от содержимого в операционной системе Linux различают несколько типов файлов, например:

- *обычный файл* (regular file). Содержит информационные данные. Для операционной системы такие файлы представляют собой просто последовательность байтов. Вся интерпретация содержимого файла производится прикладной программой, обрабатывающей файл, для которой содержимое файла представляется в определённом формате. К

этим файлам относятся текстовые файлы, бинарные данные, исполняемые программы и т. п.

- *специальный файл устройства* (special device file). Является интерфейсом для взаимодействия с устройствами вычислительной машины. Различают *символьные* (character) и *блочные* (block) файлы устройств. Символьные файлы устройств используются для небуферизированного обмена данными с устройством, в противоположность этому блочные файлы позволяют производить обмен данными в виде пакетов фиксированной длины - *блоков*.
- *файлы взаимодействия между процессами* - именованный канал FIFO (named pipe) и сокет (socket)[12].
- *ссылка* (link). Содержит указатель на другой файл или каталог. Операционная система Linux позволяет создавать указатели (ссылки) на файлы или каталоги, которые позволяют одним и тем же файлам иметь несколько имёт. Указатели бывают двух типов: жесткие и символьные (символические). ***Жесткие ссылки***, по сути, являются именами файла или каталога. Пока существует хотябы одна жесткая ссылка существует и сам файл или каталог. При создании файла для него обязательно создаётся одна жесткая ссылка. ***Символьная ссылка*** является файлом, который содержит лишь путь, указывающий на другой файл или каталог. Если удалить символьную ссылку, то файл, на который она указывает, останентся не трунотым. И обратно, если удалить файл, на который указывает символьная ссылка, то он останется, но будет «неразрешённой».

3.4. Атрибуты файлов. Контроль доступа к файлам и каталогам

Кроме имени и содержимого каждый файл и каталог имеют ряд атрибутов, предназначенных для описания их свойств. К атрибутам относятся:

- дата и время создания файла;
- дата и время последней модификации файла;
- пользователь и группа пользователей (используется для организации доступа к файлу или каталогу);
- права доступа к файлу или каталогу;
- тип содержимого файла;
- и т.д.

Атрибуты «Дата и время создания» и «Дата и время модификации» задаются и изменяются операционной системой соответственно при создании файла или каталога или их модификации.

Контроль доступа к файлам и каталогам в операционной системе Linux осуществляется путём указания ***прав на чтение, записи и исполнение (изменения)***. Указанные права определяются для трех типов пользователей – владельца файла, группы пользователей, всех остальных.

[12] Подробнее о способах межпроцессного взаимодействия будет сказано далее.

Права владельца определяют, что может делать с файлом тот пользователь, которому этот файл непосредственно принадлежит (кто указан в атрибуте «пользователь»). ***Права группы*** определяют, что могут делать с файлом члены группы, указанной в атрибуте «группа». И, наконец, ***права прочих пользователей*** определяют возможные действия тех пользователей, которые не принадлежат к предыдущим двум категориям.

Пользователь, имеющий право чтения, может просматривать содержимое файла или каталога. Пользователь, имеющий право записи, может редактировать файл или изменять содержимое каталога. И, наконец, пользователь, имеющий право выполнения, может запускать программу, содержащуюся в этом файле или делать каталог текущим (переходить в него).

ГЛАВА 4. КОМАНДНАЯ ОБОЛОЧКА BASH

«В Unix оболочка – это своего рода мать всех программ».
Линас Торвальдс, из книги «Just for fun».

При использовании текстового интерфейса пользователь взаимодействует с компьютером (а точнее с операционной системой) при помощи команд, которые вводятся им с клавиатуры. Введённые команды интерпретируются специальной программой, называемой ***командной оболочкой***, которая проверяет их правильность и организовывает запуск соответствующих программ. После завершения выполнения одной команды оболочка ждет следующую, и так до тех пор, пока пользователь не выйдет из системы (т.е. не выполнит команду *logout*).

В современных UNIX системах, к которым относится операционная система Linux, применяются различные оболочки, отличающие набором функций. Наиболее популярными являются:

- Bourne shell (sh), названная в честь своего создателя Стивена Борна (Steven Bourne) из AT&T Bell Labs;
- Bourne Again Shell (bash), расширенна версия предыдущей оболочки;
- C shell (csh), разработанная Билом Джоем (Bill Joy) и реализующая спецаильный набор команд в стиле, соответсвующим языку программирования Си;
- Korn shell (ksh), созданная Дэвидом Корном (David Korn) на базе Bourne shell, но также реализующая некоторые возможности оболочки C shell.

Пользователи могут использовать любую из имеющихся в системе командных оболочек и даже переключаться между ними[13]. Однако в один момент времени пользователь может взаимодействовать только с одной оболочкой. После завершения регистрации в системе[14] запускается та командная оболочка, которая назначена администратором системы.

Оболочка - это одна из многих программ. Программные файлы для всех командных оболочек обычно находяся в каталоге */bin*. Так, например, программный файл оболочки Bourne Again Shell будет выглядеть так */bin/bash*. Пользователь временно может изменить тип используемой оболочки запустив ту, которая ему нужна.

4.1. Понятие команды. Ввод и редактирование команд

Командой называется символьная строка, вводимая пользователем для управления операционной системой и завершаемая символом перевода каретки (клавиша <ENTER>). Команды вводятся в коммандной строке, которая обычно

[13] Например, если пользователю потребуются какие-либо функции отсутствующие у текущей оболочки и имеющиеся у другой оболочки.
[14] В случае, если для входа используется текстовый интерфейс.

содержит приглашение, за которым установлен курсор. Форма приглашения может быть различной[15], но по умолчанию она имеет следующий вид:

```
[student@wp1 student]$_
```

Приглашение *[student@wp1 student]$*, содержит информацию о текущем пользователе (*student*), имени рабочей станции (*wp1*), и текущем каталоге (*student*). Далее идет символ $, после которого установлен курсор.

Каждая команда состоит минимум из одного поля – *имени команды*, представляющее собой (полное) имя файла (или внутренней команды), которую требуется выполнить. Полное имя файла, т.е. содержащее путь к нему, указывается редко. Зачастую командная оболочка ищет указанный без пути файл в изветных ей каталогах. Подробнее об этом будет сказано ниже.

Кроме имени команда может ещё содержать разделяемые пробелами параметры. Если параметр начинается с символа минус (или тире), то он называется ***опцией***. Если с двух минусов, то ***длинной опцией***. Количество и значение параметров зависят от конкретной команды.

Например, команда для вывода содержимого домашнего каталога[16] может иметь вид:

```
ls /home/STUDENTS/student
```

Здесь *ls* – «имя команды», а */home/STUDENTS/student* – параметр, конкретизирующий работу команды, т.е. указывающий имя каталога, содержимое которого надо вывести.

Если в процессе ввода команды была допущена ошибка, то её можно исправить, используя определённые клавиши управления курсором или специальные комбинации клавиш (таблица 1). Также можно воспользоваться историей команд, в которую помещается некоторое число ранее введенных пользователем команд.

Таблица 1. Клавиши редактирования командной строки в оболочке BASH

Клавиша	Назначение
СТРЕЛКА ВПРАВО (CTRL-F)	перемещение вправо по командной строке на один символ (в пределах уже введённой строки)
СТРЕЛКА ВЛЕВО (CTRL-B)	перемещение на один символ влево
ESC-F	перемещение на одно слово вправо
ESC-B	перемещение на одно слово влево
HOME (CTRL-A)	перемещение в начало строки
END (CTRL-E)	перемещение в конец строки
DEL	Удаление символа справа от курсора (текущего симво-

[15] Подробнее о формате командной строке будет рассказано ниже.
[16] Подробнее о команде *ls* будет рассказано ниже.

	ла)
BACKSPASE	Удаление символа в позиции, предшествующей курсору
CTRL-K	удалить правую часть строки, начиная с символа, на который указывает курсор
CTRL-U	удалить левую часть строки, включая символ, который находится слева от курсора
ENTER (CTRL-M)	запуск на выполнение команды, определяемой набранной цепочкой символов
CTRL-L	очистить экран и поместить текущую команду в верхней строке экрана
CTRL-T	поменять местами два символа: символ, на который показывает курсор, и символ слева от курсора, затем, курсор переместить на один символ вправо
ESC-T	поменять местами два слова: слово, на которое указывает курсор и слово слева от первого
CTRL-K	вырезать часть строки от текущей позиции курсора до конца строки (вырезанная часть строки сохраняется в буфере, ее можно вставить в другое место строки)
ESC-D	вырезать часть строки от текущей позиции курсора до конца текущего слова (если курсор указывает на пробел между словами, то вырезается все слово справа от курсора)
CTRL-W	вырезать часть строки от текущей позиции курсора до предыдущего пробела
CTRL-Y	вставить последний вырезанный текст в позицию курсора
ESC-C	символ, на который указывает курсор, заменить на тот же символ, но заглавный, а курсор переместить на первый пробел справа от текущего слова
ESC-U	сделать символы данного слова заглавными, начиная с символа, на который указывает курсор, а курсор установить на пробел справа от слова
ESC-L	превратить символы, начиная с символа, на который указывает курсор, до конца данного слова в прописные (маленькие) буквы, а курсор установить на пробел справа от слова
TAB	Продление текущего слова с использованием имен файлов, расположенных в каталоге или в путях поиска. Если таких файлов несколько, то повторной нажатие TAB выдаст их список.
СТРЕЛКА ВВЕРХ (CTRL-P)	переход к предыдущей команде в списке (движение назад по списку)
СТРЕЛКА ВНИЗ (CTRL-N)	переход к следующей команде в списке (движение вперед по списку)

PGUP	переход к (вызов в командную строку) самой первой команды, сохраненной в истории команд
CTRL-R	Поиск в истории команд начинающихся с указанных после символов

Если команда не помещается в коммандную строку, то её можно продолжить на следующей строке, для чего текущую строку надо завершить символом “\” (обратный слеш) и нажать <ENTER>. Коммандная оболочка после нажати <ENTER> определит, что последним символом в команде был символ обратный слешь, и будет ожидать продолжение команды на новой строке. Точно так же можно завершить вторую, третью и т.д. строки.

4.2. Электронный справочник

Для удобства пользователей в операционной системе Linux имеется электронный справочник **man**, содержащий информацию о всех командах[17]. Чтобы получить справку по какой-либо команде, необходимо вызывать электронный справочник, указав ему в качестве параметра имя команды. Например, если необходимо получит справку по команде *echo*, то вызвать справочник можно так:

```
[student@wp1 student]$man echo
```

После этого на экране появится текст справки, который можно пролистывать с использованием клавиш “стрелка вниз”, “стрелка вверх”, PgUp, PgDown. Чтобы выйти из справочника нужно нажать клавишу *q*.

Электронный справочник *man* имеет несколько разделов, каждый из которых содержит информацию о командах или функциях, имеющих соответсвуюшее назначение.

Часто в разных разделах может располагаться информация о командах или функциях, имеющих одинаковые имена. Например, *read* – это команда получения информации от пользователя (раздел 1) и функция чтения информации из файла (раздел 2). Чтобы указать в каком разделе необходимо искать требуемую команду, следует в команде запуска электронного справочника явно указать имя раздела. Например,

```
[student@wp1 student]$man 3 echo
```

4.3. Команды работы с файлами и каталогами

4.3.1. Определение текущего каталога

Для определения имени текущего каталога можно использовать команду **pwd**. В любой момент только один каталог является текущим, и команды оболочки по умолчанию применяются к файлам или подкаталогам этого каталога. После регистрации в системе, текущим будет домашний каталог

[17] Большинство справочной информации приводится на английском языке.

пользователя. Перед выполнением каких-либо команд необходимо убедиться, что вы находитесь в нужном каталоге. Например,

```
[student@wp1 student]$pwd<Enter>
/home/student
[student@wp1 student]$_
```

4.3.2. Изменение текущего каталога

С помощью команды **cd** вы можете сделать текущим другой каталог, указанный в качестве параметра командной строки. Например,

```
[student@wp1 student]$pwd<Enter>
/home/student
[student@wp1 student]$cd /home/anotherstudent<Enter>
[student@wp1 anotherstudent]$pwd<Enter>
/home/anotherstudent
[student@wp1 anotherstudent]$_
```

Если в выполнить команду *cd* без параметров, то текущим станет домашний каталог пользователя.

В качестве параметра команды *cd* может быть задан либо абсолютный путь (как в предыдущем примере), либо относительный. Например:

```
[student@wp1 student]$pwd<Enter>
/home/student
[student@wp1 student]$cd ../anotherstudent<Enter>
[student@wp1 anotherstudent]$pwd<Enter>
/home/anotherstudent
[student@wp1 anotherstudent]$_
```

4.3.3. Вывод информации о содержимом каталога.

Для отображения содержимого каталога (имен расположенных в нём файлов и каталогов) используется команда **ls**. Если не будет указано ни одного параметра, то будет выведенео сдержимое текущего каталога. Чтобы вывести содержимое другого каталога, необходимо указать соотвествующий путь в качестве параметра. Например:

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  lab1.c  lab2.txt  nsmail
[student@wp1 student]$_
```

Кроме этого, команда *ls* обрабатывает ряд опций, определяющих внешний вид списка файлов и информацию, отображаемую для каждого из них (таблица 2). Более подробный перечень используемых опций можно найти в электронном справочнике man.

Опции команды *ls*, могут быть указаны как по отдельности, так и в виде одной последовательности символов. Это означает, что выражения *ls -l -F* и *ls -lF* дадут одинаковый результат.

Таблица 2. Опции команды *ls*

Опция	**Описание**
-a	Выводит информацию обо всех файлах. По умолчанию команда *ls* не отображает файлов, имена которых начинаются с точки (".")
-d	Выводит информацию не только для каталога, но и для его содержимого.
-l	Выводит информацию о атрибутах файлов и каталогов. Если данная опция не указана,выводятся только имена файлов.
-r	Изменяет порядок сортировки на обратный.
-t	Располагает файлы по дате их изменения. Если содержимое файла изменялось недавно, он будет отображаться в начале списка.
-u	Сортирует файлы по времени последнего обращения к ним.
-R	Показывает содержимое указанного каталога и всег его подкаталогов.

4.3.4. Просмотр содержимого файла

Посмотреть содержимое файла в неинтерактивном режиме (т.е. просто выдает его на экран) можно с использованием команды **cat** с указанием требуемого файла в качестве параметра. Если размер файла большой, то начало окажется за пределами экрана. Чтобы посмотреть его можно использовать комбинацию клавишь <SHIFT+PgUp> для прокрутки вверх и <SHIFT+PgDown> для прокрутки вниз.

Чтобы **просмотреть файл в интерактивном режиме** (т.е. иметь возможность при просмотре перемещаться по файлу), то можно использовать команду **less**. При этом на экран будет выдана помещающаяся на него часть файла и *less* будет ждать от пользователя нажатия клавиши (в левом нижнем углу будет выведен символ “:” двоеточие). Перемещаться по файлу можно с использованием клавиш управления курсором «стрелка вверх», «стрелка вниз», «PgUp», «PgDown». Для завершения просмотра файла необходимо нажать клавишу q. Более подробную информацию о возможностях программы *less* можно найти в электронном справочнике man.

4.3.5. Создание файлов

Файлы обычно создаются при помощи прикладных программ. Коммандная оболочка BASH позволят создать пустой файл при помощи встроенной команды **touch**. Например создать файл «file1» в текущем каталоге можно так:

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  lab1.c  lab2.txt  nsmail
[student@wp1 student]$touch file1<Enter>
```

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  file1  lab1.c  lab2.txt  nsmail
[student@wp1 student]$_
```

Если файл с таким именем уже существует, то команда *touch* не будет выполнять никаких действий.

4.3.6. Создание каталогов

Для создания каталогов используется команда **mkdir**, которой в качестве параметра указывается имя требуемого каталога. Если имя указано без пути, то каталог будет создан в текущем каталоге. В противном случае каталог будет создан в указанном месте.

В процессе создания каталога система автоматически создает в нем две записи, описывающие каталоги с именами "." и "..". Такие записи содержатся во всех каталогах, и пользователи не вправе их удалить. Каталог, который содержит только пункты "." и "..", считается пустым.

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  lab1.c  lab2.txt  nsmail
[student@wp1 student]$mkdir dir<Enter>
[student@wp1 student]$ls<Enter>
Desktop  Dir  Mail  file1  lab1.c  lab2.txt  nsmail
[student@wp1 student]$cd dir<Enter>
[student@wp1 dir]$ls -A<Enter>
. ..
[student@wp1 dir]$_
```

4.3.7. Удаление файлов

Команда **rm** с указанным именем файла позволяет удалить его. При использовании команды *rm* необходимо помнить, что удалённые файлы не подлежат восстановлению. Единственный способ вернуть информацию - обратиться к системному администратору и восстановить файл, воспользовавшись последней резервной копией (версия файла, хранящаяся в ней, не обязательно окажется самой последней).

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  lab1.c  lab2.txt  nsmail
[student@wp1 student]$rm lab2.txt<Enter>
[student@wp1 student]$ls<Enter>
Desktop  Dir  Mail  lab1.c  nsmail
[student@wp1 student]$_
```

4.3.8. Удаление каталогов

Для удаления каталога прежде следует удалить все его содержимое (включая все подкаталоги со всем их содержимым), и только после этого использовать команду **rmdir**: *rmdir имя_каталога*. Либо сразу для удаления

каталога и всего его содержимого можно воспользоваться и командой *rm* с опцией *–r*, например *rm –r имя_каталога*.

```
[student@wp1 student]$ls<Enter>
Desktop  Dir  Mail  lab1.c  lab2.txt  nsmail
[student@wp1 student]$rmdir dir<Enter>
[student@wp1 student]$ls<Enter>
Desktop  Mail  lab1.c  lab2.txt  nsmail
[student@wp1 student]$_
```

4.3.9. Копирование файлов и каталогов

Для копирования файлов и каталогов используется команда **cp**, которой в качестве первого параметра указывается что надо скопировать, а в качестве второго – куда. Например, команда «*cp файл1 файл2*», создаст копию файла с именем «файл1» и назовёт её «файл2» (оба файла находятся в текущем каталоге). Если в качестве файла назначения указано имя каталога, команда *cp* скопирует исходный файл в этот каталог. Чтобы скопировать каталог вместе с его содержимым необходимо указать опцию –r. Например, *cp –r ../cat1/ ../../cat2*.

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  file1  lab1.c  lab2.txt  nsmail
student@wp1 student]$cp file1 file2<Enter>
[student@wp1 student]$ls<Enter>
Desktop  Mail  file1  file2  lab1.c  lab2.txt nsmail
[student@wp1 student]$_
```

4.3.10. Переименование (перемещение) файлов и каталогов

Для переименования файлов и каталогов используется команда **mv**, которой в качестве первого параметра указывается что переименовать, а вторым во что. Если в качестве второго параметра указан файл, находящийся в другом каталоге (или просто другой каталог), то указаный в первом параметре файл будет перемещён в указанное место с указанным файлом (с тем же именем).

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  file1  lab1.c  lab2.txt  nsmail
[student@wp1 student]$mv file1 file2<Enter>
[student@wp1 student]$ls<Enter>
Desktop  Mail  file2  lab1.c  lab2.txt nsmail
[student@wp1 student]$_
```

4.3.11. Создание ссылок на файлы и каталоги

Создать ссылку можно с использованием команды **ls** с указанием файла (каталога) на который нужно сделать ссылку и название самой ссылки. Чтобы создать символьную ссылку надо указать опцию –s. Следует помнить, что

жесткие ссылки на каталоги может создавать только администратор системы. Например:

```
[student@wp1 student]$ls<Enter>
Desktop  Mail  file1  lab1.c  lab2.txt  nsmail
[student@wp1 student]$ln file1 file2<Enter>
[student@wp1 student]$ls<Enter>
Desktop  Mail  file1  file2  lab1.c  lab2.txt nsmail
[student@wp1 student]$ln –s file1 file3<Enter>
[student@wp1 student]$ls –l file3<Enter>
file3 –> file1
[student@wp1 student]$_
```

4.3.12. Определение прав доступа к файлам и каталогам.

Определить права доступа к файлу или каталогу можно используя команду *ls* с ключем -l. Например:

```
[student@wp1 student]$ls –l file3<Enter>
-rw-rw-r--  1 student student 0 Янв  9 22:21 file3
[student@wp1 student]$_
```

В первом столбце в символьной форме указаны права на файл. Первый символ определяет тип файла (в данном случае «-» означает обычный файл). Далее указаны **права на файл в виде триад из символов**, где r – право на чтение, w – право на запись, x – право на исполнение. Структура каждой триады одинакова. Сначала идёт право на чтение, затем на запись, после чего – на исполнение. Если вместо символа указан знак -, то это означает отсутсвие соответсвующего права. Триады расположены в порядке: пользователь, группа, остальные.

4.3.13. Изменение прав доступа к файлам и каталогам

Обычно для установки прав доступа к файлам и каталогам используется команда **chmod**, которой в качетсве параметров указывают права доступа и требуемый файл или каталог**.**

Права могут указываться двумя способами: символьным и восьмеричным.

В ***символьных выражениях***, как и следует из их названия, для указания прав доступа используются символы r,w,x. Выражение такого типа имеет следующую структуру: (категория пользователя) (действие) (права). *Категорией* пользователя могут быть следующие: u - владелец файла, g - группа, o - прочие, a - все. Действие: «+» - добавление прав к существующим правам доступа (если они уже существуют, то ничего не изменится), «-» - отмета указанных прав (если их нет, то ничего не изменится), «=» - явное указание прав доступа (т.е. будут установлены именно такие права доступа). Например, чтобы изменить права доступа на файл таким образом, чтобы

владелец мог читать файл, группа записывать в файл, остальные – исполнять файл, то запись будет иметь вид: *chmod u+r,g+w,o+x file*.

При использовании ***восьмиричных выражений*** права указываются явным образом в виде четырёхзначного числа, представляющего права для всех категорий пользователей: владельца (старшая цифра), группы (средняя цифра), прочих(младшая цифра) и специальные права файла. В каждой цифре (кроме первой) право на чтение представляется числом 4 (2^2), на запись 2 (2^1), на выполнение 1 (2^0). Результирующее право формируется как сумма этих цифр. Таким образом значение, определяющее права доступа кокретной категории пользователей, лежит в диапазоне от 0 до 7. Например, если требуется, чтобы пользователь имел права на чтение и выполнение, а остальные не имели ни каких прав, то необходимо выполнить команду: *chmod 0500 file*. Первая цифра в восмиричной записи указывает специальные права файла: установить эффективного пользователя (4), установить эффективную группу (2) и задать флаг «прикреплённости» файла (1). Подробнее об этих правах будет рассказано в п.**Ошибка! Источник ссылки не найден.**.

4.3.14. Изменение владельцев и групп

Права доступа к файлу или каталогу указываются для владельца файла, группе к которой относится владелец файла и всем остальным. Для того, чтобы изменить владельца файла или его группу, используются команды **chown** и **chgrp**, которым в качестве параметров указывают требуемого пользователя (команда *chown*) или группы (*chgrp*) и требуемый файл или каталог.

Например, *chown student /home/student* объявляет пользователя *student* владельцем указанного каталога. Администратор системы (пользователь с именем *root*) может изменять владельца любого файла. Обычный пользователь может сменить только группу файла или каталога, которым он владеет и только на ту группу, членом которой он является.

Если команда *chown* вызывается с опцией -R, она выполняется рекурсивно, т.е. для всех файлов и каталогов указанного каталога.

4.4. Списки команд

Часто бывает необходимо выполнить несколько команд последовательно одну за другой, т.е. *списком*. Для этого команды объединяются **символом ";"**: *команда1* ; *команда2* ; и т.д. Например,

```
[student@wp1 student]$mkdir docs ;cd docs ; ls<Enter>
[student@wp1 docs]$_
```

Списки также применяются в тех случаях, когда выполнение команды должно зависеть от статуса завершения предыдущей команды[18]. При этом

[18] Как определить статус завершения последней программы будет сказано ниже. В данном случае командная оболочка сама определяет статус завершения каждой программы из списка.

используются логические операторы “&&” и “||”: *команда1 && команда2*, *команда1 || команда2*.

В первом случае (**оператор &&**) команды объеденены оператором конъюнкции. При этом комнада2 выполняется только в том случае, когда статус завершения команды1 равен 0, что означает «успешно завершена». Например:

```
[student@wp1 student]$mkdir docs && cd docs<Enter>
[student@wp1 docs]$_
```

В данном примере сначала создаётся каталог docs, а затем, если он создан (т.е. команда mkdir верёнт 0), то этот каталог будет сделан текущим, что и произошло.

Во втором случае (**операция ||**) команды связаны между собой операцией дезъюнкции, и команда2 выполняется только в случае, если команда1 возвращает значение больше 0, т.е. результат отличный от «успешно завершена». Например:

```
[student@wp1 student]$mkdir docs || cd docs<Enter>
[student@wp1 docs]$_
```

В отличии от предыдущего примера каталог docs будет сделан текущим, если произошла ошибка при его создании. Результат будет такой же как и в предыдущем примере, так как каталог docs уже создан и его повторное создание вызовет ошибку.

4.5. Каналы ввода-вывода. Перенаправление каналов. Конвейер

Любая выполняющаяся программа имеет как минимум три канала для взаимодействия с пользователем: поток ввода, поток вывода и поток вывода ошибок. По умолчанию, эти потоки связаны терминалом, т.е. позволяют получать информацию о нажимаемых пользователем клавишах, и выводить пользователю информацию на экран.

При запуске программы эти потоки могут быть перенаправлены на другое устройство или на файл. Это бывает необходимо, например для того, чтобы информацию об ошибках программы выводилась не на экран, а печаталась на принтере. Или для того, чтобы программы получала информацию не с клавиатуры, а считывала её из какого-то файла, содержащего обрабатываемую информацию. Конечно же эти действия могла бы сделать и сама программа. Но в этом случае, при необходимости обрабатывать информацию получаемую и от пользователя и находящуюся в файле, потребовалось бы иметь две программы: одну считывающую информацию с терминала, другую – с файла, что оказывается неэффективно.

Перенаправление потоков ввода/вывода производится с помощью операторов **”>”, ”>>”, ”<”, “<<”**, после которых указывается новые источники или получатели информации. Например, *команда > файл*.

Оператор “>” перенаправляет поток вывода. Например *команда > файл*, приведет к созданию (если такой файл не существует) или перезаписи (в противном случае) указанного файла, в который будет помещаться все, что *команда* выводит в поток вывода.

Оператор “>>” также перенаправляет поток вывода, но при этом если указанный приемник уже существует, то производится дозапись в него. Если источник не существет, то он будет создан. Например, *команда >> файл*.

Для перенаправления потока вывода ошибок перед операторами “>” и “>>” ставится цифра 2 (что соответсвует 2-му стандартному потоку). Например, *команда 2> файл*.

Поток ввода может быть перенаправлен, используя **оператор "<"**. к*оманда < файл*. В этом случае любая попытка получения командой данных со стандартного потока ввода приведёт к чтению необходимой инфомации из указанного файла.

Если требуется передать программе несколько следующих команд, то используется **оператор “<<”**, после которого указывается команда, которая будет являться завершающей. команда << разделитель. В этом случае после нажания клавиши <ENTER> коммандная оболочка будет ждать от пользователя следующие команды и помещать их во временный файл до тех пор, пока пользователь не введёт команду-разделитель. После этого *команда* будет запущена на выполнение и её поток ввода будет связан с созданным временным файлом, который по завершению *команды* будет удалён. Например,

```
[student@wp1 student]$sort << END<ENTER>
>Апельсин
>Мандарин
>Банан
>END
Апельсин
Банан
Мандарин
[student@wp1 student]$_
```

В примере запускается команда *sort*[19], вход которой связан с файлом, содержащим три строки: Апельсин, Мандарин. Банан. Результат работы команды – отсортированные по алфавиту строки файла.

Следует отметить, что при ожидании команд пользователя, помещяемых во временный файл, формат командной строки изменился на “>”.

Операторы перенаправления ввода и вывода могут присутствовать одновременно в одной команде. Например, в предыдущем примере отсортированные строки можно было бы поместить в файл. Сделать это можно так:

[19] Подробнее о команде sort смотрите в электронном справочнике man.

```
[student@wp1 student]$sort << END > file<ENTER>
>Апельсин
>Мандарин
>Банан
>END
[student@wp1 student]$cat file<ENTER>
Апельсин
Банан
Мандарин
[student@wp1 student]$_
```

При обработке информации в UNIX системах выходные данные одной программы могут поступать на вход другой. Это необходимо, например для того, чтобы вывод одной программы был автоматически обработан другой программой[20]. Такой способ взаимодействия программ называется *конвейром*. Для организации конвейера команды объединяются **оператором** «|», т.е. формируют **ковейер**: *команда1* | *команда2* | и т.д. Обратите внимание, что при организации конвейера используется оператор "|", а не "||" как в случает списка команд. Например:

```
[student@wp1 student]$cat file | sort –r>file1<ENTER>
[student@wp1 student]$cat file1<ENTER>
Мандарин
Банан
Апельсин
[student@wp1 student]$_
```

В этом примере поток вывода команды *cat* (в который она помещает содержимое файла file) соединяется с потоков ввода команды *sort,* вызванной с опцией –r (обратная сортировка). Вывод команды *sort* помещается в файл file1, который следующей командой выводится на экран.

4.6. Изменение пароля пользователя

При входе в систему каждый пользователь чтобы идентифицировать себя должен ввести своё имя (или выбрать его из списка) и пароль. Пользователи задаются (создаются) администратором системы, который и назначает каждому из них первоначальный пароль. Пользователи меют возможность самостоятельно изменить свой пароль, чтобы гарантировать что никто кроме них не сможет войти в систему под их именем.

Изменить пароль можно используя либо специальные команды, либо графические оболочки, которые, по сути, просто спрашивают у пользователя необходимую информацию и вызывают соответсвующие команды.

[20] Например, отфильтрованы лишни сообщения. Или сообщения оформлены определённым образом и т.п.

Если информация о пользователях хранится на этом же компьютере[21], то изменить пароль можно при помощи комманды **passwd**. Если компьютер подключен к сети, в которой используется сестевая система паролей (NIS), то смена пароля производится с помощью команды **yppasswd**[22].

При изменении пароль система сначала запросит текущий пароль пользователя (чтобы кто-то другой не смог его сменить), затем новый пароль и его подтверждение (т.е. новый пароль ещё раз).

После ввода нового пароля система проверит подходит ли он под имеющиеся требования безопасности и, если все в порядке, то изменит его. Если же новый пароль неудовлетворяет этим требованиям или новый пароль несовпадает с подтверждением, то система сообщит об ошибке и попросит ввести новый пароль заново. Например,

```
[student@wp1 student]$yppasswd<ENTER>
Old password:
New password:
Re-type password:
Your password has been changed to ...
[student@wp1 student]$_
```

При нескольки неудачных попытках смены пароля процедура будет завершена и пароль останется без изменений.

Обычно администратор системы кроме самого пароля определяет требования о частоте его смены и о том, что этот пароль может содержать. Например, «*пароль должен меняться не реже, чем раз в месяц и содержать минимум 6 символов, включающих маленькие и большие латинские буквы и цифры*».

4.7. Получение информации о пользователях системы

Каждый пользователь операционной системы Linux описывается как минимум следующими характеристиками:

- системное имя пользователя;
- пароль;
- идентификационный номер пользователя UID;
- идентификационный номер основной группы пользователя GID.
- информация о пользователе (обычно это человеческое ФИО);
- домашний каталог;
- основная командная оболочка.

Информация о пользователях хранится либо в специальном файле с именем **/etc/passwd**, либо на сервере сети (если используется сетевая информационная служба).

[21] Место хранения информации о пользователях определяется администратором системы.
[22]Первые две буквы (yp) показывают, что используется система NIS, которая ещё называется yellow pages.

В файле */etc/*passwd содержится несколько строк, каждая из которых состоит из 7 полей содержащих сведения об одном пользователе и разделённых символом ":" (двоеточие). Формат строки следующий: *имя:пароль:UID:GID:информация о пользователе:home_dir:shell*. Например, строка «*student:*:500:500:Иванов Иван Иванович:/home/student:/bin/bash*» - содержит информацию о пользователе с именем *student*, паролем[23] *, его UID равен *500*, GID равен *500*, ФИО пользователя – *Иванов Иван Иванович*, домашний каталог */home/student*, командная оболочка, запускаемая при входе в систему - */bin/bash*.

В случае использования сетевой службы паролей (NIS) информация о пользователях системы храниться на соответствующем сервере и получить её можно с использованием команда *ypcat* с параметром *passwd*. В результате получится список пользователей в таком же формате, как и файл /etc/passwd.

Информацию о текущем пользователе можно получить, используя команду **id**[24]. Например,

```
[student@wp1 student]$id<ENTER>
uid=500(student) gid=500(students)
группы=500(students),550(ftp_students)
[student@wp1 student]$_
```

Чтобы получить информацию о том, какие пользователи зарегистрированы в системе в данный момент времени можно использовать команды **whoami** и **who**. Первая команды возвращает имя текущего пользователя[25]. Её вызов аналогичен вызову команды *id* с параметром *–un*. Вторая – список зарегистрированных в данный момент в системе пользователей. Например, вызов:

```
[student@wp1 student]$who<ENTER>
student1   pts/1  Jan 18 19:24 (192.168.1.38)
student2   pts/0  Jan 18 20:25 (192.168.1.25)
[student@wp1 student]$_
```

показывает, что в системе зарегистрировано два пользователя: *student1* и *student2*. Первый зашел в систему 18 января в 19:24 с использованием текстового терминала pts/1(сетевой вход с компьютера с адресом 192.168.1.38). Второй – 18 января в 20:25, с использование текстового терминала pts/0 (сетевой вход с компьютера с адресом 192.168.1.25).

Чтобы определить, какой пользователь когда был зарегистрирован в системе используется команда **last**, которая выдаёт содержимое журнала регистрации. Например, вызов,

[23] Для повышения безопасности системы пароли могут храниться в другом файле (/etc/shadow) в зашифрованном виде. Доступ к такому файлу обычно разрешен далеко не каждому пользователю.
[24] Информацию о параметрах команды смотрите в электронном справочнике man.
[25] Эта команда часто применяется в совместно используемых скриптах, о чем будет сказано далее.

```
[student@wp1 student]$last<ENTER>
student1 pts/1 1.2.3.4 Wed Jan 18 19:24 still logged in
student1 pts/1 1.2.3.4 Wed Jan 17 19:24 - 20:00 (00:36)
[student@wp1 student]$_
```

показывает, что пользователь student1 был зарегистрирован в системе 18 января в 19:24 и до сих пор работает, а также был зарегистрирован 17 января в 19:24 и вышел из системы в 20:00, т.е. провел в ней всего 36 минут.

4.8. Физическое размещение файлов на носителях информации

В Linux все доступное пользователям файловое пространство объединено в единое дерево каталогов. В большинстве случаев единое дерево, таким какое его видит пользователь системы, составлено из нескольких отдельных файловых систем, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим системам, могут находиться на различных устройствах (например, дискетах, flash-дисках и т.п.).

Чтобы подключить новую файловую систему, а также посмотреть какие файловые системы уже подключены используется команда **mount**.

Если её вызвать без параметров, то на экран будет выдан список уже подключенных файловых систем. В этом списке каждая строка описывает одну подключенную файловую систему. Первым полем строки является устройство, на котором хранится подключенная файловая система (например, /deh/hda4 – 4-й раздел жесткого диска, подключенного к первому контроллеру IDE в режиме master[26]). Второе поле содержит метку общей файловой системы, которая является точкой входа. Далее идет тип подключенной файловой системы и её параметры. Например,

```
[student@wp1 student]$mount<Enter>
/dev/hda4 on / type ext2 (rw) proc on /proc type proc (rw)
/dev/hda2 on /boot type ext2 (rw)
192.168.1.1:/home on /home type nfs (rw,addr=192.168.1.1)
[student@wp1 student]$
```

Этот пример показывает, что корневая файловая система находится на устройстве, описываемом файлом */dev/hda4* (*/dev/hda4* on / type ext2) и имеет файловую систему типа ext2. Каталог */boot* (вторая строка) ссылается на файловую систему 2-го раздела устройства, подключенного к первому контроллеру первым устройством и также имеет файловую систему *ext2*. Каталог */home* ссылается на сетевую файловую систему (тип nfs), находящуюся на компьютере с адресом 1*92.168.1.1* и имеющей имя локальное имя */home*.

Чтобы подключить новую файловую систему необходимо вызвать команду *mount*[27] указав ей в качестве первого параметра имя устройства, где

[26] Подробнее об устройстве жесткого диска и о способах его разделения на разделы смотрите в курсе «Организация ЭВМ и систем»

[27] Подробнее о команде mount смотрите в электронном справочнике man.

она храниться, а вторым параметром – точку в общей файловой системе к которой её надо подключить. Например,

```
[student@wp1 student]$mount /dev/fd0 /mnt/fl<Enter>
[student@wp1 student]$
```

Чтобы отключить файловую систему необходимо использовать команду **umount**, которой в качестве параметра следует указать соответсвующую точку в общей файловой системе. Например,

```
[student@wp1 student]$umount /mnt/fl<Enter>
[student@wp1 student]$
```

4.9. Среда окружения

При запуске любой программы (в том числе и командной оболочки) в системе, создается *среда окружения*, которая кроме прочего включает в себя набор переменных, описывающих текущий сеанс работы пользователя с операционной системой (таблица 3). Все переменные среды окружения доступны всем процессам пользователя, начиная с текущего.

Список всех установленных переменнных можно получить используя команду **env**. Например,

```
[student@wp1 student]$env | head -5<ENTER>
HOSTNAME=wp1.csc.neic.nsk.su
TERM=linux
SHELL=/bin/bash
HISTSIZE=1000
USER=student
[student@wp1 student]$_
```

В примере вывод команды *env* направлен на вход команды *head,* которая выводит заданное опцией число строк (в данном случае 5), начиная с начала файла. В результате выдано пять переменных: HOSTNAME, TERM, SHELL, HISTSIZE, USER, с соответсвующими значениями.

Установка новых и **изменение значения** существующих переменных среды окружения осуществляется путем *экспортирования* (помещения в среду): ***export*** *переменная=значение*.

Чтобы **удалить переменную** используется команда **unset**: *unset имя*.

Таблица 3. Некоторые стандартные переменные среды окружения

Имя	Значение
UID	Содержит числовой идентификатор текущего пользователя. Инициируется при запуске оболочки.
HOME	Домашний каталог текущего пользователя.
PATH	Список каталогов, разделенных двоеточием, в которых

	командная оболочка выполняет поиск файла, в случае если в команде не задат его путь.
PS1	Формат строки-приглашения (первая строка).
PS2	Формат строки-приглашения (вторая строка).
PWD	Текущий каталог.
TERM	Тип используемого терминала.
HOSTNAME	Сетевое имя компьютера.
SECONDS	Число секунд, прошедших с момента запуска оболочки.

4.10. Переменные. Массивы.

Кроме переменных среды окружения командная оболочка позволяет во время своего выполнения хранить данные в виде собственных переменных и даже массивов. Значения этих переменных используются только самой оболочкой и, в отличии от переменных среды окружения, недоступны запускаемым из неё программам.

Значение переменной присваивается следующем образом: *переменная=значение* (т.е. без процедуры эскпортирования). Например, X=1, или X=a, или X=”f” и т.п.

Значение массивов могут задаваться двумя способами:

1. имя[индекс]=значение
2. имя=(значение1 значение2 ... значениеN).

В первом случае в качестве индекса могут быть использованы как числовые значения так и символьные лексемы. Например: FRUIT[0]=apple или FRUIT[first]=apple.

Обратите внимание, что до и после знака равно нет пробелов !!! Если поставить пробелы, например так x = 1, то командная оболочка будет считать, что введена команда x и она имеет два параметра = и 1.

Если в командной оболочке создать переменную с тем же именем, что и переменная среды окружения, то в командной оболочке будет использоваться значение этой переменной, а в запускаемых программах – старое значение переменной среды окружения. Получить список всех переменных можно с использованием команды **declare**.

Удалить значение переменной или массива можно также с использованием команды *unset*.

4.11. Подстановка переменных, команд и арифметических выражений

Камандная оболочка BASH позволяет формировать команды с использованием значений переменных, результатов работы других команд и т.п. Такое формирование называется **подстановкой.** Т.е. в команду «подставляется» что-либо (переменная, вывод другой команды и т.п.). Для подстановки используется либо символ $, либо выражение заключенное в обратные апострофы (`выражение`).

Если в тексте команды встречается **символ “$”,** то следующий за ним текст до пробела или конца команды интерпретируется как имя переменной, значение которой подставляется в текст команды. Например,

```
[student@wp1 student]$FRUIT=Апельсин<ENTER>
[student@wp1 student]$echo “Фрукт ”$FRUIT<ENTER>
Фукт Апельсин
[student@wp1 student]$_
```

В примере создаётся одна переменная командной оболочки (FRUIT), которой присваивается значение «Апельсин». Замет выполняется команда *echo*, которая должна на экран выдать текст указанный ей в качестве параметра. В данном случае параметром является строка *“Фрукт “$FRUIT*, в которой присутсвует символ $. Поэтому прежде, чем выполнить команду echo, командная оболочка «подставить» в её параметр значение переменной FRUIT, сформировав тем самым текст “Фрукт Апельсин”.

Подстановку переменных можно использовать и для формирования новых команд. Например,

```
[student@wp1 student]$DIR=Docs<ENTER>
[student@wp1 student]$COMM=”ls –A”<ENTER>
[student@wp1 student]$COMM=$COMM” “$DIR<ENTER>
[student@wp1 student]$$COM<ENTER>
.  ..
[student@wp1 student]$_
```

Обратите внимание, что при присваивании переменной значения её имя указывается без знака доллара. Т.е. если вы напишите $FRUIT=apple, командная оболочка выдаст ошибку: -bash Апельсин=apple:command not fount. Так как прежде, чем выполнить команду командная оболочка подставить в неё значение переменной FRUIT, а затем только попытается её выполнить. И если вы напишите echo FRUIT, то на экран будет выведено слово FRUIT, а не значение переменной с таким именем.

Для **подстановки элементов массива** используется запись вида *${имя[индекс]}*. Например,

```
[student@wp1 student]$m[0]=Docs<ENTER>
[student@wp1 student]$m[1]=”ls –A”
[student@wp1 student]$COMM=${m[1]}” “${m[0]}<ENTER>
[student@wp1 student]$$COM<ENTER>
.  ..
[student@wp1 student]$_
```

Если в качестве индекса массива указать символ "*" или "@", то результатом будут все элементы массива. Такую операцию можно делать только в том случае, если все элементы массива проинициализированы.

Запись вида **${имя}** можно использовать и для подстановки переменной, например в том случае, если имя переменной содержит символ подчеркивания. Например, *${FRUIT_APPLE}*.

Подстановку можно использовать также в том случае, если требуется в команде использовать то, что некоторая программа помещает в потов вывода. В этом случае программа заключается её в **обратные апострофы.** Прежде, чем выполнить команду, командная оболочка выполнить программу, заключенную в обратные апострофы, все что она поместит в поток вывода будет подставлено в командную строку и только затем команда будет выполнена.

Например,

```
[student@wp1 student]$DATE=`date`<ENTER>
[student@wp1 student]$echo $DATE
Сбт Янв 19 18:39:22 NOVT 2002
[student@wp1 student]$_
```

Переменной DATE присваивается текст, который должна была вывести на экран команда date, т.е. текущую системную дату. Затем на экран выводится значение переменной DATE.

Командная оболочка позволяет выполнять арифметические операции. Для этого, выражение которое необходимо **интерпретировать как арифметическое**, заключается в двойный круглые скобки и перед ним ставится знак доллара. Например в результате команды:

```
[student@wp1 student]$foo=$(((5+3*2)-4)/2))<ENTER>
[student@wp1 student]$echo $foo
3
[student@wp1 student]$_
```

присвоит переменной foo значение равное 3.

Условная подстановка переменных является ещё одним механизмом подстановки значений (таблица 4).

Таблица 4. Виды условной подстановки переменных

Форма подстановки	Описание
${перем:-значение}	Если переменная неопределена или равна NULL, вместо него будет использовано значение. Содержимое переменной при этом не изменяется.
${перем:+значение}	Если переменная установлена, то вместо него подставляется значение. Содержимое переменной при этом не изменяется.
${переменная:=значение}	Если переменная неопределена или равна NULL, то ей присваевается значение, которое сразу и подставляется

${перем:?сообщение}	Если переменная неопределена или равна NULL, в стандартный поток ошибок выводится сообщение.
${перем:смещение}	Выделяет из переменной часть строки, начиная со смещения и заканчивая концом строки.
${перем:смещение:длинна}	Выделяет из переменной часть строки, начиная со смещения и с укзанной длиной.
${#переменная}	Подставляет длину содержащейся в переменной строки в символах.
${перем/что/на_что} или ${перем//что/на_что}	Подставляет значение «перем» с заменой указанной строки «что» на указанную строку «на_что». Значение самой переменной не изменяется. Если перед полем *«что»* указан один слэш, то произойдёт замена только одного вхождения подстроки *«что»*. Если указано два слеша, то произойдет замена всех вхождений подстроки *«что»*.

Например, с использования условной подстановки переменных можно проверить правильность задания некоторой команды:

```
[student@wp1 student]$${COMMA:?Нет команды}<ENTER>
-bash: COMMA: Нет команды
[student@wp1 student]$_
```

Командная оболочка проверит, есть ли переменная с именем *COMMA* и присвоено ли ей какое-либо значение. Если переменная есть и у неё есть значение, то оно будет интерпретировано как команда (так как в команде кроме подстановки ничего больше нет). Если же переменной нет, или ей не присвоено никакого значения, то вместо интерпретации команды на экран будет выдано сообщение «Нет команды».

Заменить символ пробел в строке, хранящейся в переменной *DB* на символ подчеркивание можно так: *DB=${DB// /_}*.

4.12. Формат приглашения командной оболочки

Как уже было сказано выше, командная оболочка предлагает пользователю вводить команды используя определённое приглашение. Формат этого приглашения задается с помощью *переменных окружения:* PS1, PS2, PS3, PS4. Переменная **PS1** определяет обычную командную строку. Переменная **PS2** командную строку выдаваемую при продолжении команды на следующей строке (2-я и последующие строки, если перед нажатием <ENTER> был введен символ '\').

В строку приглашения могут входить любые допустимые символы, команды терминала и специальные символьные последовательности, которые используются для получения информации о системе (таблица 5).

Таблица 5. Специальные последовательности формата командной строки.

Послед.	Выводимое значение
\t	время в формате часы: минуты: секунды
\d	дата в формате день_недели месяц число
\n	перевод строки
\s	имя оболочки, базовое имя $0 (участок, следующий за конечным /)
\w	текущий рабочий каталог
\W	базовое имя $PWD
\u	имя текущего пользователя
\h	сетевое имя компьютера
\#	номер этой команды
\!	номер истории этой команды
\nnn	символ, имеющий код равный nnn в восьмиричной системе
\$	если uid=0, то \#, иначе \$.
\\	обратная косая черта (backslash).
\[	начало терминальной последовательности
\]	конец терминальной последовательности

Чтобы изменить формат приглашения командной оболочки достаточно присвоить необходимое значение соответствующей переменной. Например, выделить текущего пользователя зелёным цветом можно, присвоив переменной PS1 следующее значение:

```
PS1=[\[\033[32m\]\u\[\033[0m\]@\h \W]\$
```

4.13. Получение информации от пользователя

При необходимости командная оболочка позволяет сформировать значение некоторой переменной «спросив» его у пользователя. Для этого используется команда **read**, которой в качестве параметра передаётся имя требуемой переменной.

```
[student@wp1 student]$read CHOICE
Привет !!!<ENTER>
[student@wp1 student]$echo "Вы ввели "${CHOICE}
Привет !!!
[student@wp1 student]$_
```

Чтобы указать какое приглашение должно быть выведно в строке для ввода можно использовать параметр *–p*. Например, *read –p "Введите X" X*.

Время ожидания (в секундах) ввода задаётся или при помощи переменной **TMOUT** или при помощи параметра *–t*. Если переменная *TMOUT* неопределена, или её значение равно 0 и не указан параметр *-t*, то время ожидания считается бесконечным. **Обратите внимание**, что значение

переменной *TMOUT* также влияет на время ожидания командной оболочкой очередной команды !!!

Используя параметр *–s* можно запретить отображение вводимых символов на экране. Это удобно, например, при вводе паролей.

4.14. Управляющие структуры

Командная оболочка bash позволяет организовать ветвление программы в зависимости от различных условий.

4.14.1. Условный оператор if-fi

Выражеиние **if** записывается следующем образом[28]:

```
$if выражени1 ; then \
>выражение2 ; \
>elif выражение3 ; then \
>выражение4 ; \
>else \
>выражение5 ; \
>fi<ENTER>
```

В приведенной выше команде *if* сначала выполняется выражение1. Если код завершения выражения1 равен 0 (что интерпретируется как его истинность), то выполняется выражение2 и команда *if* заканчивается. В противном случае выполняется выражение3 и проверяется код завершени. Если выражение3 возвращает значение равное 0, то выполняется выражение4 и команда *if*. Если выражение3 возвращает ненулевое значение, то выполняется выражение5. Наличие операторов elif и else необязательно. В блоке *if-fi* может содержаться несколько elif.

Часто в блоке if-fi в качестве выражений, результаты которых проверяются, используется команда **test**, которая имеет две формы записи: *test параметры* или *[параметры]*. После интерпретации параметров (таблица 6) как логического выражения команда test возвращает значение 0 - истина, либо 1 - ложь.

Таблица 6. Некоторые параметры команды test

Выражение	Значение
-d файл	существует ли файл и является ли он каталогом?
-e файл	существует ли указанный файл?
-f файл	существует ли файл и является ли он обычным файлом?
-L файл	существует ли файл и символьная ли он ссылка?
-r файл	существует ли файл и разрешен ли он для чтения?
-s файл	существует ли файл и имеет ли он нулевой размер?

[28] В дальнейшем по тексту будем использовать сокращенный формат приглашения командной строки, состоящей только из символа $.

-w файл	существует ли файл и разрешена ли в него запись?
-x файл	существует ли файл и является ли он исполняемым?
-O файл	существует ли файл и приналежит ли он текущему пользователю?
файл1 -nt файл2	был ли файл1 последний раз модифицирован позже, чем файл2?
-z строка	указанная строка имеет нулевую длину?
-n строка	указанная строка имеет ненулевую длину?
стр1 == стр2	указанные строки совпадают?
! выражение	указанное выражение false?(содержит не ноль)
выр1 -a выр2	логическое AND двух выражений
выр1 -o выр2	логическое OR двух выражений
выр1 -eq выр2	выр1 равно выр2?
выр1 -ne выр2	выр1 неравно выр2?
выр1 -lt выр2	выр1 меньше (в арифметическом смысле) выр2?
выр1 -le выр2	выр1 меньше либо равно выр2?
выр1 -gt выр2	выр2 меньше выр1?
выр1 -ge выр2	выр2 меньше либо равно выр1?

Ниже приведены примеры использования команды test в составе выражения *if--fi*:

```
$if [ -d $HOME/bin ] ; then \
>PATH="$PATH:$HOME/bin" ; fi
$if [ -z "$DTHOME" ] && [-d /home/student/dt ] ;then \
>DTHOME=/home/student/dt ; \
>fi
$if [ -z "$DTHOME" -a -d /home/student/dt ] ; then \
>DTHOME=/home/student/dt ; \
>fi
```

В первом примере проверяется существует ли каталог *$HOME/bin*, и, если он существует, то он добавляется к переменной *PATH*.

Во втором и третьем случае проверяется установлено ли значение переменной *DTHOME* и существует ли каталог /home/student/dt. Если переменная неопределена, или имеет пустое значение и указанный каталог существует, то переменной *DTHOME* задаётся новое значение. Во втором примере проверка указанного условия происходит с использованием списка из двух команд test, выполняемых по условию «И». В третьем примере используется один вызов команды test, в параметрах которой указаны все условия (условие «И» реализуется с помощью опции –a).

Следует обратить внимание, что при проверке значения переменной *DTHOME* она взята в кавычки. Это сделано так потому, что если переменная все-таки неопределена или определена, но не имеет значения, то вместо неё в командную строку ничего не будет подставлено. Если кавычки не использовать, то после подстановки потеряется второй параметр команды *test* и

нечего будет проверять (т.е. строка *test –z $DTHOME* будет преобразована к виду *test –z*).

4.14.2. Оператор множественного выбора case-esac

Блок **case-esac** аналогичен оператору if-fi с множеством elfi и предназначен для проверки одной переменной на несколько возможных значений. Блок *case-esac* записывается следющим образом:

```
$case значение in \
>шаблон1) \
>список команд1 ;;
>шаблон2) \
>список команд2 ;;
>esac<ENTER>
```

В данном случае значение - это строка символов, сравниваемая с шаблоном до тех пор, пока она не совпадет с ним. Список команд, следующий за шаблоном, которому удовлетворяет значение, запускается на выполнение. За списком следует команда ;;, которая завершает работу блока case-esac.

Если значение не удовлетворяет ни одному из шаблонов, выражение case завершается. Если необходимо выполнить какие-то действия по умолчанию, следует включить в выражение шаблон "*", которому удовлетворяет любое значению.

В выражении case-esac должен присутсвовать по крайней мере один шаблон. Максимальное число шаблонов неограничено.

Шаблон формируется по правиалам аналогичным именам файлов и каталогов (с учетов символов расширения), а также оператор дизъюнкции "|" (операция или). Ниже приведен пример использования блока case-esac:

```
$case "$TERM" in \
>*term) \
>TERM = xterm \
>;; \
>network | dialup | unknown | vt[0-9]*) \
>TERM=vt100 \
>;; \
>esac
```

4.15. Циклические конструкции

Циклические конструкции применяются в том случае, когда надо многократно повторить одни и те же действия.

4.15.1. Цикл for

Цикл **for** предназначен для выполнения определённых действия над несколькими данными. Формат записи цикла следующий:

```
$for имя_переменной in список_значений ; \
>do \
>команда1 ; команда2 ...\
>done
```

В цикле **for** переменной с указаным именем последовательно присваиваются все значения из списка_значений и для каждого из этих значений выполняется список_команд. Значения в списке_значений перечисляются через пробел. Например следующая команда выдаст на экран десять строк: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

```
$for i in 1 2 3 4 5 6 7 8 9 10 ; \
>do \
>echo $i ; \
>done
```

4.15.2. Цикл while

Если необходимо выполнять какие-либо действия до тех пор, пока истино некоторое выражение, то следует использовать цикл **while**, который записывается следующим образом:

```
$while выражение ; \
>do \
>список_команд ; \
>done
```

На каждой итерации этого цикла выполняется выражение и до тех пор, пока он возвращает значение 0, выполняется список_команд. Если в качестве выражения указать команду /bin/true, то цикл будет бесконечным, и завершить его можно только используя оператор **break.** Пропустить какую-либо часть цикла и перейти на следуюшую его итерацию можно используюя в списке_команд команду **continue**.

Например, вывод последовательности цифр от 1 до 9 можно организовать следующим образом:

```
$x=1
$while [ $x -lt 10 ] ; \
>do \
>echo $x ; \
>x=$(( $x+1 )) ; \
>done
```

4.15.3. Цикл until

Если необходимо выполнять какие-либо действия до тех пор, пока ложно некоторое выражение, то следует использовать цикл **until**, записываемый следующим образом:

```
$until выражение ; \
>do \
>список_команд ; \
>done
```

На каждой итерации этого цикла выполняется выражение и до тех пор, пока он возвращает значение отличное от 0, выполняется список_команд. Если в качестве выражения указать команду /bin/false, то цикл будет бесконечным, и завершить его можно только используя оператор **break.**

Например, вывод последовательности цифр от 1 до 9 можно организовать следующим образом:

```
$x = 1
$until [ $x -ge 10 ] ; \
>do \
>echo $x ; \
>x=$(( $x+1 )) ; \
>done
```

4.15.4. Цикл select

Если имеется набор значений и требуется по желанию пользоваться над некоторыми из них выполнить некоторые операции, то используется цикл **select**, который позволяет создать нумерованное меню значений, пункты которого могут быть выбраны пользователем. Данный цикл записывается следующим образом:

```
$select имя_переменной in список_значений ; \
>do \
>список_команд ; \
>done
```

При запуске цикла *select* в сандартный поток ошибок выводится список_значений. Перед каждым пунктом отображается номер, который и предлагается выбрать пользователю. Затем выводится приглашение ввода номера. Формат приглашения определяется переменной **PS3**. Выбрав номер пункта, пользователь вводит его с клавиатуры. После этого, для указанного элемента спика значений (который помещается в указанную переменную) выполняется список_команд. После этого опять выводится меню цикла *select*.

Завершить цикл *select* можно либо после ввода EOF (нажатия комбинации клавиш <CTRL+D>), либо при помощи команды break. Пример использования цикла select:

```
$select file in ./*.c ; \
>do\
>gcc –Wall -ansi $file –o `basename $file .c` ; \
>done
```

В данном примере пользователю предлагается выбрать один из файлов с суффиксом .c, расположенных в текущем каталоге. Выбранный файл будет скомпилирован и результат компиляции помещён в файл с таким же именем, но без суффикса .c. Имя файла результата формируется командой basename[29], результат которой подставляется в команду компиляции.

4.16. Группирование команд. Функции. Скрипты.

Командная оболочка BASH позволяет группировать несколько команд, выполняющих определенное действие, в функции или специальные файлы, называемые скриптами.

Функции опрделяются так:

```
$function имя () { список_команд ; }
```

В дальнейшем, если будет дана команда *имя*, то вместо неё будет выполнен *список_команд*. Например[30],

```
$ls
$function ls () { /bin/ls –A $@ ; }
$ls
. ..
```

Здесь объявляется функция с именем *ls*, которая содержит всего лишь одну команду */bin/ls* с соответсвующими параметрами. До того, как определена эта функция команда *ls* привозит к запуску файла */bin/ls*, выводящего на экран содержимое каталога.

Скрипт – это обычный текстовый файл[31], содержащий команды оболочке. Такой файл может быть запущен на исполнение следующим образом:

```
$bash имя_файла
```

Другими словами, для выполнения скрипта необходимо запустить командную оболочку, передав ей в качестве параметра имя соответвующего файла.

Другой вариант запуска скрипта - просто указав его имя в командной оболочки (т.е. сделав из него некий вид программы). Для этого надо в параметрах доступа определить файл как исполняемый, и в первых строках этого файла явно указать оболочку, для которой предназначен этот скрипт следующем образом:

```
#!оболочка
```

[29] Подробнее о команде basename смотрите в электронном справочнике man.
[30] Значение переменной @ смотри ниже.
[31] Обычно такие файлы имеют суффикс .sh

В общем случае символ # в скрипте означает комментарий, требующий игнорировать строку. Однако ели он является первым символом файла и за ним следует символ !, то это означает "магическую комбинацию", за которой указывается путь к файлу, который должен быить использован в качестве интерпретатора (например, */bin/bash*, */bin/perl*, */bin/sh* и т.д). Встретив такую комбинацию командная оболочка запустить соответсвуюший файл и передаст ей имя файла в качестве параметра.

Скрипт, в свою очередь, может содержать все конструкции описанные ранее (в том числе и функции).

Выполнение скрипт происходит построчно. При этом, если конструкция включает в себя несколько команд, то они могут располагатья на нескольких строках и указывать символ \ в конце неоконченной строки при этом нет необходимости. Например,

```
#!/bin/bash
X=1
read –p “Введите X = “ X
if [ $X –lt 0 ] ; then
   echo “Вы ввели отрицательное число”
else
   echo “Вы ввели положительное число”
fi
```

4.17. Анализ опций, передаваемых группе команд (функции и скрипту)

Любому скрипту, точто также как и функции, могут быть переданы параметры (так как это делается при запуске любой команды). Передаются параметры в виде специальных переменных (таблица 7).

Таблица 7. Специальные переменные, используемые в скриптах

Перем	Значение
$0	Имя выполняемой команды. Для скрипта -- это путь, указанный при его вызове. Для функции – имя оболочки.
$n	Переменные, соответсвующие параметрам, заданным при вызове сценария. Здесь n - десятичное число, соответсвующее номеру параметра. (Первый параметр $1, второй $2 и т.д.)
$#	Число параметров, указанных при вызове сценария.
$*	Строка параметров, заключенная в двойные кавычки.
$@	Все параметры, каждый заключен в двойные кавычки
$?	Статус завершения последней выполненной команды.
$$	Номер процесса, соответсвующего текущей оболочке.
$!	Номер процесса, соответсвующий команде, запущенной в фоновом режиме.

Последователь просмотреть параметры командной строки можно используя следующую конструкцию:

```
#!/bin/bash
while [ -n "$1" ] ; do
  echo "Имеется параметр - "$1
  shift
done
```

Параметры просматриваются в цикле *while,* условием выполнения которого является результат команды *test* с параметрами, требующими, чтобы длинны строки *"$1"* была отлично от нуля. В строку подставляется значение первого параметра. Если параметр не указан, то переменная *$1* имеет пустое значение, соответсвенно строка будет иметь нулевую длинну и цикл сразу же прекратиться. Если параметр указан, то он выводится в теле цикла и затем выполняется команда **shift**, которая изменяет переменные $n сдвигая их на одну влево (т.е. *$1=$2, $2=$3* и т.д.). Значение первого параметра при этом теряется. Когда сдвигается последний параметр, то переменной *$1* присваивается пустое значение (так как следующей за ней переменной несуществует). В качестве пераметра команды *shift* можно указать на сколько требуется сдвинуть строку параметров. Например, *shift 2* приведёт к следующему изменению: *$1=$3, $2=$4* и т.п.

Если требуется определить есть ли в параметрах определённые короткие опции, то можно использовать специальную команду **getopts,** которая сама просматривает специальные переменные и проверяет является ли параметр короткой опцией.

Напомним, что **опцией называется** параметр командной строки, начинающийся с символов '-' или '--'. Причем, если используется один знак минуса, то опция считается односимвольной или ***короткой***, если же два знака минуса, то многосимвольной или ***длинной***[32].

Команда *getopts* вызывается следующим образом:

```
getopts строка_опций_поиска переменная
```

В *строке_опций_поиска* указываются односимвольные опции, которые должны быть «распознаны» *getopts*. Например, строка *acdf* говорит о том, что требуется найти какую либо из опций: *-a, -c, -d* или *-f.*

Алогритм работы команда *getopts* следующий.

Последовательно просматриваются специальные переменные *$1, $2, $3…$#*, начиная с номера, указанного в переменной **OPTIND**. При первоначальном запуске командной оболочки в *OPTIND* находится 1[33]. Другими словами, если впервые вызвать команду *getopts*, то она начнет просмотр с переменной *$1*.

Если очередной параметр непустой и он не содержит опцию (не начинается с символа «-»), то *getopts* завершается с ненулевым кодом возврата.

[32] Длинные опции в скрипте можно «распознавать» только через специальные переменные.

[33] Если потребуется заново провести проверку опций, то такое присвоение должен будет сделать пользователь.

Иначе считается, что найдена опция и производится сравнение следующего за тире символа с содержимым *строки_опций_поиска*.

Если найденная опция совпадает с одним из символов в *строке_опций_поиска*, то соответсвующий символ записывается в *переменную*.

Если совпадение необнаружено (т.е. в коммандной строке указана опция, которая отсуствует в *строке_опций_поиска*), на экран выводится сообщение об ошибке, *переменной* присваивается значение "?".

Если просматриваемый параметр больше не содержит символов, то номер следующего параметра записывается в переменную *OPTIND* (чтобы при следующем запуске команды продолжить просмотр параметров). В противном случае *OPTIND* остаётся без изменеения, а запоминается позиция внутри текущего просматриваемого параметра (чтобы в следующий раз продолжить просматривать этот параметр, считая его опцией и начиная со следующего символа).

В случае, если найдена «распознаваемая» или «нераспознаваемая» опция, код возврата *getopts* равен 0. В случае если просмотрены все параметры и больше опций нет, или очередной параметр не является опцией, то код возврата *getopts* отличен от 0.

Чтобы **запретить выдачу сообщения об ошибке** при нахождении опции, которая «нераспознаётся», необходимо либо переменной *OPTERR* присвоить 0, либо в *строке_опций* первым символом указать “:” (двоеточие). В последнем случае *getopts* начинает работать в «тихом» (silent) режиме, и, в случае ошибки найденную «нераспознаваемую» опцию она помещает в переменную *OPTARG*, а в *переменную* помещает символ “?”.

Команда *getopts* позволяет «распознать» **опции, для которых требуется дополнительный параметр**. Для этого надо в строке опций после соответсвующего символа включить знак ”:”[34]. Например, в строке *a:cdf:* указывается, что опции *–a* и *–f* должны дополняться обязательным параметрами, например *–a файл* или *–b время*.

Если при «распознавании» найдена подобная опция, и за ней следует непустой параметр, то он помещается в переменную *OPTARG*, а сама опция помещается в *переменную* и *getopts* завершается.

Если опция найдена, но за ней нет параметра, то на экран выдаётся сообщение об ошибке, в *переменную* помещается символ «?» и *getopts* завершается. В «тихом» режиме при нахождении такой опции и осутсвия у неё параметра в *переменную* помещается символ “:”, а в переменную *OPTARG* помещается сама опция.

Например:

```
#!/bin/bash

while getopts ab:cde:f: OPTION ; do
```

[34] Обратите внимание, что в данном случае символ “:” не является первым в строке, а следует за символом. При этом в начале строки символ двоеточие может как присутствовать (чтобы перевести *getopts* в «тихий» режим), так и отсутствовать.

```
  case $OPTION in
     \?)
        echo "Найдена \"нераспознаваемая\" опция"
        ;;
     :)
       echo "Тихий режим и найдена опция "${OPTARG}
       ;;
     ?)
        echo "Найдена опция "${OPTION}
        if [ -n "${OPTARG}" ] ; then
          echo "Она имеет параметр "${OPTARG}
        fi
  esac
  echo "OPTIND = "${OPTIND}
done
echo "Finally OPTIND = "${OPTIND}
```

Скрипт содержит в себе цикл, в котором последовательно запускается команда *getopts* до тех пор, пока код её возврата равен 0. В качестве *строки_опций_поиска* команда *getopts* получает строку вида «*ab:cde:f:*». Результат будет помещаться в переменную *OPTION*.

Запуск скрипта с параметрами *–ab –c –d –e –f –g* приведёт к следующему:

```
[sstuden@wp student]$./getopts.sh -ab -c -d -e –f \
>-g<ENTER>
Найдена опция a
OPTIND = 1
Найдена опция b
Она имеет параметр –c
OPTIND = 3
Найдена опция d
OPTIND = 4
Найдена опция e
Она имеет параметр –f
OPTIND = 6
./getopts.sh: illegal option – g
Найдена "нераспознаваемая" опция
OPTIND = 7
Finally OPTIND = 7
```

При первом запуске (на первой итерации цикла) команда *getopts* начнёт просмотр переменной *$1*, в которой располагается параметр *–ab*. Так как он начинается с символа «-», то считается, что найдена опция. После того, как определено, что параметр содержит опцию, проверяется входит ли она в строку опций. В данном случае она входит в неё и не требует дополнительного параметра (так как за ней нет символа двоеточие). Поэтому в переменную

OPTION помещается символ *a*, и getopts завершается. Значение переменной OPTIND не изменяется, так как параметр содержит ещё другие символы (b).

На следующей итерации цикла getopts продолжает просматривать первый параметр (так как *OPTIND* = 1). В нём также оказывается опция, но она требует параметр (в *строке_опций_поиска* после символа b стоит двоеточие). Поэтому *getopts* проверяет указан ли следом за этой опцией параметр. В данном случае он указан и равен *–c*, поэтому в *OPTION* помещается *b*, а *OPTARG* приравнивается к *–c*. Переменная OPTIND при этом указывает на параметр с номером 3, так как текущий параметр больше не содержит символов, а 2-й параметр относился к найденной опции.

Далее аналогично просматриваются остальные параметры, до параметра, содержащего опцию *–g*. Она считается нераспознанной. Поэтому на экран выдаётся сообщение об ошибке (так как никаких указаний по его устранению не было) и в переменную *OPTION* помещается символ “?”.

Цикл продолжается до тех пор, пока не будут просмотрены все параметры.

4.18. Вход в систему и первоначальные настройки

Часто при входе в систему, выходе из неё или при простом запуске командной оболочки[35] пользователю требуется выполнить определённый набор действий[36]. Для этого при запуске и при завершении командная оболочка выполняет определённый набор скриптов, располагаемых в домашнем каталоге пользователя (таблица 8).

Таблица 8. Спецаильные файлы командной оболочки BASH

Файл	Назначение
.bash_history	Содержит историю всех введенных пользователем команд.
.bash_profile	Исполняется оболочкой при входе пользователя в систему.
.bash_logout	Исполняется оболочкой при выходе пользователя из системы.
.bashrc	Выполняется при простом запуске пользователем оболочки.

Обратите внимание, что все эти файлы имеют имена, начинающимся с .bash

Кроме этого, в правах на файлы *~/.bash** не указана возможность их исполнения. Это потому, что командой, исполняющей эти файлы, всегда является сама командная оболочка.

Таким образом, если необходимо, чтобы какая-то команда выполнялась каждый раз при входе в систему, то эту команду надо поместить в файл *.bash_profile*. Например, для вывода текущей даты и времени при входе в систему в файл *.bash_profile,* надо добавить команду **date**.

[35] Такой запуск также называется «интерактивным».

[36] Например, пользователю необходимо выводить информацию имеется ли в его электронном почтовом ящике письма или записать в журнал время и дату, когда он вошел в систему и т.п.

Практическое задание N 1. Знакомство с операционной системой LINUX. Способы хранения информации.

1. Поменять пароль.
2. Используя команду **mount,** описать как построена файловая система на вашей машине.
3. Используя команды оболочки создать в домашнем каталоге дерево каталогов согласно схеме, приведенной ниже: в домашнем каталоге - cat1, содержит cat2 и cat3. Каталог cat1/cat2 содержит каталог cat3. Каталог cat1/cat3 содержит каталог cat4. Каталог cat1/cat2 содержит каталог cat5. Каталог cat1/cat2/cat3 содержит cat6 и cat7. Каталог cat1/cat8 содержит символическую ссылку на каталог cat1/cat2/cat3/cat6. Каталог cat1 содержит каталог cat8.
4. Нарисовать граф, соответствующий созданной файловой системе.
5. Удалить каталоги с дублирующимися именами.
6. Удалить неразрешенную ссылку cat6.
7. Написать маски файлов для списков приведенных ниже:

   file1, file5, file6, file8
   file, fail, from, fax
   asd, dfg, qwe, dsa, fkl, jss
   adks, aeks, awks, alks.
8. Скопировать файлы из домашнего каталога, начинающиеся с символов .b в каталог cat1/cat8.
9. Поменять права на скопированные файлы так чтобы любой пользователь системы мог прочитать их содержимое, используя символьное представление прав доступа. Используя восьмеричное представление прав доступа, изменить права каталога cat5 так, чтобы к каталогу имели доступ только пользователи группы.
10. Вывести содержимое всех каталогов начиная с самого верхнего для задания (использовать только одну команду и один раз).
11. Определить тип командной оболочки используемой вами.

Контрольные вопросы

1. Как проходит процедура идентификации?
2. Что такое регистрационное имя?
3. Что такое командная оболочка? Как можно определить её тип?
4. Файл */etc/passwd*. Зачем он нужен? Какова его структура?
5. Что такое файл?
6. Что такое файловая система? Как её можно описать? Команда mount.
7. Что такое каталог?
8. Что такое путь файла? Абсолютный и относительный путь?
9. Типы файлов, используемых в ОС Linux. Что такое метаданные?
10. Символьные и блочные устройства. Отличия и примеры?
11. Связь. Типы связей. Команда создания связи.
12. Файловая система UNIX. Назначение основных каталогов.

13. Команда определения текущего каталога.
14. Команда изменения текущего каталога.
15. Команда вывода содержимого каталога.
16. Электронный справочник man.
17. Генерация имен файлов. Символы "*", "?", [], цитирование.
18. Команда копирования файлов.
19. Команды удаления файлов и каталогов.
20. Команды создания и удаления каталогов.
21. Владельцы, группы и права.
22. Команды изменения прав. Символьные и восьмеричные представления прав доступа.
23. Изменение владельца и группы.

Практическое задание № 2. Командная оболочка bash

1. Определить тип используемой командной оболочки.
2. Вывести на экран значения всех переменных среды окружения. Проанализировать полученные результаты и объяснить значения известных вам переменных окружения.
3. Используя процедуру экспортирования изменить приглашение командной строки так, чтобы в основном приглашении имя машины выводилось красным цветом, а в приглашении для второй строки выводился номер команды и символ ">".
4. Используя системную переменную HOME и список команд выполнить следующие действия: одной командой -- перейти в домашний каталог, в случае удачного перехода выдать содержимое файла /etc/passwd.
5. Используя команды printf и read, выведете приглашение пользователю ввести команду. Если пользователь нажал <ENTER> без ввода команды, сообщить ему об ошибке, в противном случае выполнить то, что он ввел.
6. Оформите предыдущий пункт как скрипт и выполните его.
7. Вывести значения всех переменных среды окружения в файл с именем \~/envs.
8. Используя системную переменную HOME, список, каналы и перенаправление вывода, выполнить следующие действия одной командой - перейти в домашний каталог, выдать содержимое файла /etc/passwd, отсортированное по имени пользователя в файл passwd.orig. Подсказка - команда сортировки – sort.
9. Используя перенаправление ввода с разделителем и перенаправление вывода, добавить в файл passwd.orig информацию о себе согласно формату записи файла /etc/passwd (все поля должны быть обязательно заполнены).
10. Описать содержимое файла ~/.bash_profile и всех файлов, которые он использует.
11. Написать скрипт, выполняющий следующие действия: выводит меню, содержащее все файлы с расширением .c текущего каталога. После выбора пользователем файла, компилирует его.

12. Написать скрипт, анализирующий параметры командной строки с использованием специальных переменных. Все параметры должны быть выданы на экран.
13. Написать скрипт, анализирующий параметры командной строки. Параметры должны быть следующие - -d каталог, -f файл, -c, -r. При анализе опций должны быть установлены переменные: DIR, FILE, COMPIL, RUN. Поле анализа опций выполнить следующие действия: если определена переменная DIR и такой каталог существует, то выдать его содержимое. Если определена переменная FILE и такой файл существует, то выдать его содержимое на экран. Если переменная не определена, то в качестве имени файла использовать .bashrc. Если определена переменная COMPIL и определена переменная FILE, то откомпилировать указанный файл. Если результат компиляции положительный, то, если определена переменная RU, исполнить откомпилированный файл.
14. Используя цикл for объединить все файлы с расширением txt в текущем каталоге в файл \~/textx.txt. Для объединения использовать перенаправление потоков ввода-вывода.

Контрольные вопросы

1. Что такое командная оболочка?
2. Что такое команда? Формат команды?
3. Что означает символ "\textbackslash" введенный перед нажатием <ENTER>?
4. Что такое скрипт--файл?
5. Что такое среда окружения? Зачем она нужна?
6. Как задать значение переменной окружения и как вывести его на экран?
7. Переменная оболочки. Отличие от переменной окружения.
8. Как задать формат командной строки? Отличие PS1 и PS2.
9. Списки команд. Логические операции над командами.
10. Подстановка команд, переменных и арифметических выражений.
11. Команда read.
12. Зачем нужны файлы .bash*. Почему они не имеют прав на исполнение?
13. Условный оператор if-fi. Команда test.
14. Блок case-esac.
15. Специальные переменные.
16. Цикл select.
17. Циклические конструкции.
18. Функции.
19. Команда getopts.
20. Понятие процесса. Типы процессов. Атрибуты процесса. Фоновое выполнение команд. Команды jobs, fg,bg, ps, pstree.
21. Каналы ввода-вывода. Перенаправление каналов.