Знакомство с Linux

3.1 Введение

Новые пользователи UNIX и Linux могут быть ошеломлены размерами и очевидной сложностью системы, которая предстала перед ними. Существует много хороших книг по использованию UNIX для всех уровней подготовки: от новичка до эксперта. Но ни одна из этих книг не обсуждает особенности Linux. Хотя 95% всего связанного с использованием Linux абсолютно аналогично другим UNIX-системам, наиболее прямой путь освоения этой системы - это по учебнику, написанному применительно к Linux. Вот эта книга и есть такой учебник.

Эта глава не заводит в дебри деталей и не обсуждает наиболее сложные прим. переводчика: они (и уже у нас тоже) говорят - продвинутые аспекты Linux. Вместо этого делается попытка поставить новичка крепко на ноги, чтобы он мог в дальнейшем читать и более общие книги по UNIX, понимая базовые различия других UNIX-систем и Linux.

Здесь не предполагается каких-то предварительных знаний, за исключением первоначального знакомства с персональным компьютером и MS-DOS. Но даже если вы не успели побывать пользователем MS-DOS, вы все равно все здесь поймете. На первый взгляд UNIX очень похож на MS-DOS (в конце концов фрагменты MS-DOS были спроектированы с оглядкой на операционную систему CP/M, которая, в свою очередь, проектировалась с оглядкой на UNIX). Но только при очень уж поверхностном взгляде можно говорить о похожести UNIX и MS-DOS. Если вы абсолютный новичок в мире персональных компьютеров, этот учебник вам поможет.

И прежде, чем начать, призываем: *не бойтесь экспериментировать*. Система вас не укусит. Работая на ней вы ничего не сможете сломать. UNIX имеет встроенные средства защиты, чтобы не дать "нормальным" пользователям (это теперь и вы) возможность испортить важные для системы файлы. Самое плохое, что вы можете натворить - это уничтожить все свои файлы, а тогда, может быть придется и переинсталлировать заново систему прим. переводчика: как правило, чтобы довести систему до переинсталляции, надо иметь прав больше, чем у "нормального" пользователя.

3.2 Базовые концепции UNIX

UNIX это многозадачная, многопользовательская операционная система. Это означает, что много людей может одновременно использовать один компьютер, выполняя много различных задач. (Это существенное отличие от MS-DOS, где только один человек может использовать в данный момент операционную систему). В UNIX пользователи должны себя идентифицировать при входе, что состоит из двух шагов: **ввода имени** (имя,

по которому вас идентифицирует система) и **входной пароль**, который является вашим секретным словом для открытия вашего счета (регистрации в системе). Поскольку только вы знаете пароль, никто не может войти в систему под вашим именем.

В традиционных UNIX-системах системный администратор присвоит вам имя и начальный пароль при вашей регистрации в системе (при заведении в системе нового пользователя). Но поскольку на своем персональном компьютере вы и системный администратор, вы должны себя (как пользователя) зарегистрировать в системе, прежде чем в нее войдете (смотрите Раздел 3.2.1 ниже). Для дальнейших разговоров возьмем условное имя ``larry''.

Кроме прочего, каждая система UNIX имеет приписанное ей **hostname** (хозяйское имя). Это хозяйское имя добавляет машине характера и очарования. Hostname используется для идентификации отдельных машин в сети, но даже если ваша машина не в сети, она все равно должна иметь hostname. В Разделе 4.10.2 мы подробно расскажем об установке hostname на вашей машине. Например, имя машины, обсуждаемой ниже - ``mousehouse'' (мышиная норка).

3.2.1 Регистрация в системе (открытие счета)

Прежде, чем вы сможете использовать систему, вы должны зарегистрировать себя в системе. Это необходимо потому, что неразумно использовать имя суперпользователя (root) для обычных нужд. Пользователь root нужен для выполнения привилегированных команд и сопровождения системы, как это описывается в Разделе 4.1.

Для того, чтобы зарегистрировать себя, вам необходимо зайти в систему под именем root и использовать команду useradd или adduser. Об этой процедуре смотрите подробней в Разделе 4.4.

3.2.2 Вход в систему

При входе вы увидите на экране подсказку, например, такого вида:

```
mousehouse login:
```

Введите свое имя и нажмите клавишу Return. Наш герой larry напечатает следующее:

```
mousehouse login:  larry
Password:
```

Теперь введите ваш пароль (password). При вводе пароль не будет отображаться на экране, так что набирайте внимательнее. Если вы неправильно набрали пароль, то увидите на экране сообщение

```
Login incorrect
```

и вам следует попытаться еще раз.

Когда вы наконец правильно введете имя пользователя и пароль, вы официально будете допущены в систему и можете в ней свободно путешествовать.

3.2.3 Виртуальные консоли

Системная **консоль** - это монитор и клавиатура, связанные непосредственно с системой. (Поскольку UNIX многопользовательская система, вы можете иметь дополнительные терминалы, связанные через последовательные порты с вашей системой, но они не будут консолями). Linux, как и некоторые другие версии UNIX, обеспечивает доступ к **виртуальным консолям** (или VC), которые позволяют войти в систему под несколькими именами в одно время.

Для демонстрации этого войдите в систему (как было показано ранее). Теперь нажмите alt-F2. Вы должны снова увидеть подсказку login: , то есть перед вами вторая виртуальная консоль, а вы вошли через первую. Чтобы переключиться обратно на первую VC, нажмите alt-F1. *Оп-ля!* Вы снова на первой консоли.

Свежеинсталлированный Linux возможно позволит вам работать с четырьмя первыми VC, используя от alt-F1 до alt-F4. Но возможно обеспечить работу с 12-ю VC - по одной на каждую функциональную клавишу. Как видите, использование VC может быть очень эффективным - вы можете работать на нескольких VC одновременно.

В то время, как использование виртуальных консолей ограничено (кроме прочего, в каждый момент времени вы можете видеть только одну виртуальную консоль) оно дает вам представление о многопользовательских возможностях UNIX. Пока вы работаете на VC #1, вы можете переключиться на VC #2 и начать работу над чем-то другим.

3.2.4 Shells и команды

В большинстве ваших исследований мира UNIX вы будете общаться с ним через оболочку **shell**. Shell - это просто программа, которая воспринимает введенное пользователем, (т.е. команды, которые вы напечатаете) и транслирует это в команды системе. Это можно сравнить с программой

COMMAND.COM под MS-DOS, которая делает нечто похожее. Shell - это лишь один из интерфейсов UNIX. Существует много различных интерфейсов, таких как X Window System, которая позволяет выполнять команды используя мышь и клавиатуру в сочетании.

Как только вы вошли, система запускает shell и вы можете вводить для него команды. Вот короткий пример. Как раз Larry вошел в систему и система вновь выдала **подсказку**:

```
mousehouse login:  larry
Password:  larry's password
Welcome to Mousehouse!
/home/larry#
```

``/home/larry#'' это подсказка shell, показывающая, что он готов принимать команды. (Подробнее про подсказку позже). Давайте попросим систему сделать что-нибудь интересьненькое:

```
/home/larry# make love
make:  *** No way to make target `love'.  Stop.
/home/larry#
```

Хм, как оказалось, "make" - это имя существующей в системе программы и shell пытался выполнить эту команду. (Жаль, но система отнеслась к просьбе недружественно).

Это подводит нас к жгучему вопросу: Что такое команды? Что происходит, когда вы вводите ``make love''? Первое слово командной строки ``make'' это имя команды, которую предполагается выполнить. Все остальное в командной строке воспринимается как аргументы команды.

Примеры:

```
/home/larry# cp foo bar
```

Здесь имя команды ``cp'', а аргументы ``foo'' и ``bar''.

Когда вы вводите команду, shell делает несколько вещей. Во-первых, смотрит на то, что может (должно) быть именем команды и является ли это внутренней для shell командой. (Внутренняя, это команда, которую shell знает как выполнять. Существует ряд таких команд, мы о них поговорим позже). Shell также проверяет, не является ли команда синонимом другой или требуется подстановка имени. Если этого не надо делать, shell ищет соответствующую этому имени программу на диске. Если shell находит

такую программу, он ее выполняет, передавая ей аргументы из командной строки.

В нашем примере shell ищет программу по имени make и пытается выполнить ее с аргументом love. make - это программа, которая часто используется при компиляции больших программ, она берет в качестве аргумента имя "целевого" файла компиляции. В случае ``make love'' мы приказали команде make откомпилировать love. Поскольку make не смог найти файла с таким именем, он сообщил (несколько забавным образом) о невозможности выполнить команду и вернулся в подсказку.

Что случится, если мы введем команду, а shell не сможет найти программу с этой командой? Давайте попробуем:

```
/home/larry# eat dirt
eat:  command not found
/home/larry#
```

Все очень просто, если shell не может найти программу с именем данным в командной строке (здесь ``eat''), он выдает сообщение об ошибке, которое объясняет причину невыполнения команды. Вы часто будете видеть это сообщение, если будете вводить имена команд с ошибками. (например, напечатаете ``make love'' вместо ``make love'').

3.2.5 Выход из системы

Прежде, чем идти дальше, мы расскажем, как выйти из системы. При наличии подсказки shell используйте команду

```
/home/larry# exit
```

для выхода. Есть другие способы выхода, но этот самый безопасный.

3.2.6 Смена пароля

Вы также должны представлять, как можно менять пароль. Команда "passwd" прим. переводчика: именно с пропущенными буквами она и пишется спросит вас про старый пароль и про новый. Она попросит дважды ввести новый пароль для надежности. Внимание! Не забывайте свой пароль, иначе вам придется просить системного администратора уничтожить его и установить новый (Если вы и есть системный администратор, смотрите Раздел 4.4).

3.2.7 Файлы и каталоги

Во многих операционных системах (включая UNIX) существует концепция **файла**, по которой его можно рассматривать просто, как набор информации,

которому дано имя. Примерами файлов будут: программа, которая может выполняться, письмо, полученное по электронной почте, написанная вами статья. Существенно то, что все, что хранится на диске, хранится в отдельных файлах.

Файлы идентифицируются по именам. Например, файл, содержащий вашу статью может быть сохранен под именем my-paper. Эти имена обычно каким-то образом отражают содержание. Не существует стандартного формата имен файлов, как в MS-DOS и других операционных системах; в общем случае имена файлов могут содержать любые символы (кроме / - смотрите ниже обсуждение формирования "путей") и ограничены 256 символами по длине.

Одновременно с концепцией файла рассмотрим и концепцию каталога. Каталог - это совокупность файлов. Его можно рассматривать как "папку", содержащую множество различных файлов. Каталоги сами по себе также получают имена, по которым вы их различаете. Каталоги организованы в древовидную структуру, т.е. каталоги могут содержать другие каталоги.

К файлу можно обращаться по пути (pathname), формируемой из имени файла, которому предшествует имя каталога, содержащего файл Например, скажем, Larry имеет каталог, названный papers, который содержит три файла: history-final, english-lit, и masters-thesis. (Каждый из этих трех файлов содержит информацию о проводимых Larry работах). Для того, чтобы обратиться к файлу english-lit, Larry может указать маршрут:

```
papers/english-lit
```

Как вы видите, имена каталогов и файлов разделяются единичным слэшем (/). Поэтому имена файлов сами по себе не могут содержать этот символ. Пользователи MS-DOS увидят в этом что-то знакомое, поскольку в MS-DOS для этого используется бэкслэш (\).

Как уже говорилось, каталоги могут быть вставлены друг в друга. Например, пусть Larry в каталоге papers имеет другой каталог с названием notes. Этот каталог содержит файлы с именами math-notes и cheat-sheet. Путь файла cheat-sheet будет

```
papers/notes/cheat-sheet
```

Поэтому путь - это маршрут, который надо проделать, чтобы добраться до конкретного файла. Каталог выше данного (под)каталога называется **родительским каталогом**. Здесь каталог papers является родительским для каталога notes.

3.2.8 Дерево каталогов

Большинство систем UNIX имеет стандартную структуру каталогов, что облегчает конкретную установку системы. Структура представляет из себя дерево каталогов, начинающееся с каталога ``/", известного под названием "корневой каталог". Каталоги ниже / относятся к числу важнейших подкаталогов: среди них /bin, /etc, /dev, и /usr. Эти каталоги в свою очередь содержат другие каталоги, которые содержат системные конфигурационные файлы, программы и т.д.

В частности, каждый пользователь имеет **домашний каталог**, который выделяется пользователю для хранения его файлов. В вышеприведенном примере все файлы Larry (такие как cheat-sheet и history-final) содержались в домашнем каталоге Larry. Обычно пользовательский домашний каталог находится под каталогом /home и называется именем пользователя. Так домашний каталог Larry будет /home/larry.

На Рис. 3.2.8 представлено простое дерево каталогов. Оно даст вам некоторое представление о том, как организуется дерево каталогов в вашей системе.

3.2.9 Текущий рабочий каталог

Команды, которые вы даете shell, выдаются из вашего **текущего каталога**. Вы можете думать о вашем рабочем каталоге, как о каталоге в котором вы находитесь. При начальном входе в систему вашим рабочим каталогом автоматически становится домашний каталог (в нашем случае /home/larry). При обращении к файлу вы можете обращаться к нему с учетом вашего местоположения, вместо того, чтобы указывать полный путь.

```
/_____bin
  |_dev
  |_etc
  |_home_____larry
  |       |_sam
  |_lib
  |_proc
  |_tmp
  |_usr__X386
     |_bin
     |_emacs
     |_etc
     |_g++-include
     |_include
     |_lib
     |_local_____bin
     |        |_emacs
```

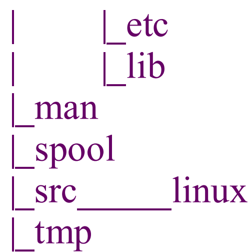


Рис 3.1: Типичное (урезанное) дерево каталогов Unix

Вот например, у Larry есть каталог papers, а papers содержит файл history-final. Если Larry хочет посмотреть этот файл, он может использовать команду

```
/home/larry# more /home/larry/papers/history-final
```

Команда more просто показывает файл на экране порциями. Поскольку текущий рабочий каталог Larry /home/larry, он вместо этого может обратиться к файлу с учетом своего текущего местоположения. Команда будет

```
/home/larry# more papers/history-final
```

Так что вы можете начинать имя файла (такого как papers/final) с символа, отличного от ``/'', система предполагает, что вы обращаетесь к файлу с учетом вашего текущего рабочего каталога. Это называют **относительным именем** (а полный маршрут - **полное (абсолютное) имя** - т.е. путь от корня до данного имени).

3.2.10 Обращение к домашнему каталогу

Оболочки (shell), т.е. программы, которые читают и выполняют команды пользователя, могут использоваться (одновременно в одной системе) разные. В большинстве систем Linux используются tcsh или bash при начальной регистрации в системе. В tcsh и bash вы можете обратиться к своему домашнему каталогу, используя тильду (``~''). Например, команда

```
/home/larry# more ~/papers/history-final
```

эквивалентна

```
/home/larry# more /home/larry/papers/history-final
```

Символ ``~'' просто заменяет имя вашего домашнего каталога.

Вы также можете обратиться к домашнему каталогу другого пользователя с помощью тильды. Имя ``~karl/letters" будет интерпретировано shell, как ``/home/karl/letters" (если /home/karl - домашний каталог для пользователя karl). Использование тильды упрощает обращение; не существует каталога с именем ``~" - так что это просто "синтаксический сахар", который имеется в распоряжении shell.

3.3 Первые шаги в UNIX

Перед тем, как начать, важно заметить, что все имена файлов и команд чувствительны к большим и малым буквам (чего нет в системах типа MS-DOS). Например, команда make очень отличается от Make или MAKE. То же относится и к именам каталогов.

3.3.1 Первая прогулка

Теперь мы можем войти в систему и узнать, как обращаться к файлам и менять местоположение в файловой системе, чтобы упрощать свою жизнь в ней. Команда для перемещения по дереву каталогов - cd, (``change directory"). Вы скоро обратите внимание, что многие часто используемые команды Unix состоят из двух-трех букв. Формат команды cd:

```
cd <directory>
```

где <directory> - имя каталога, в который вы желаете перейти. Как мы уже говорили, когда вы входите в систему, вы автоматически оказываетесь в своем домашнем каталоге. Если Larry желает двинуться ниже по дереву, он должен использовать команду

```
/home/larry# cd papers
/home/larry/papers#
```

Как видите, изменилась подсказка, отразив изменение местоположения (новый рабочий каталог). Теперь он в каталоге papers и может посмотреть на свой файл history-final с помощью команды

```
/home/larry/papers# more history-final
```

Чтобы вернуться назад из подкаталога papers, надо использовать команду

```
/home/larry/papers# cd ..
/home/larry#
```

(Обратите внимание на пробел между ``cd" и ``.."). Каждый каталог содержит имя ``.." , которое отсылает к родительскому (для данного каталога) каталогу. Также каждый каталог имеет имя ``.", которое ссылается на него самого. Поэтому команда

```
/home/larry/papers# cd .
```

никуда не переведет.

В команде cd вы можете использовать маршруты. Чтобы перейти в домашний каталог Карла, вы можете воспользоваться командой

```
/home/larry/papers# cd /home/karl
/home/karl#
```

Используя команду cd без аргументов вы из любого места дерева вернетесь в свой домашний каталог.

```
/home/karl# cd
/home/larry#
```

3.3.2 Разглядывание содержимого каталогов

Теперь вы знаете, как ходить-бродить по каталогам, но вероятно возникает вопрос: "Ну и что с того?" Само по себе хождение по каталогам бесполезно, давайте познакомимся с новой командой ls. ls (LiSt) выдает на экран перечень файлов и каталогов (по умолчанию из текущего каталога). Например,

```
/home/larry# ls
Mail
letters
papers
/home/larry#
```

Здесь мы видим, что у Larry три "единицы хранения" в его текущем каталоге: Mail, letters и papers. Но это мало, что говорит: каталоги это или файлы? Можно использовать опцию (прим. переводчика: часто в документации по UNIX используют в этом контексте слово "флаг") -F в команде ls, чтобы получить больше информации.

```
/home/larry# ls -F
Mail/
```

```
letters/
papers/
/home/larry#
```

Приписанные справа к именам файлов / говорят о том, что это (под)каталоги.

Использование ls -F (обратите внимание "-F" пишется без пробела) может дать также ``*" в конце некоторых имен файлов. Это будет говорить о том, что это **выполняемые** файлы или программы. Если, при вызове ls -F, ничего справа не приписано к имени, то это "нормальный" файл, т.е. не каталог и не выполняемый файл.

В общем, каждая команда UNIX может иметь несколько опций в дополнение к другим аргументам. Эти опции обычно записываются со знаком ``-", как это было показано на примере ls -F. Опция -F сообщает команде ls, что необходимо выдать дополнительную информацию о типе файлов.

Если вы напишете в команде ls имя каталога, то она выдаст содержимое указанного каталога.

```
/home/larry# ls -F papers
english-lit
history-final
masters-thesis
notes/
/home/larry#
```

Или, чтобы было интереснее, давайте посмотрим, что имеется в системном каталоге /etc/.

```
/home/larry# ls /etc

Images      ftpusers    lpc         rc.new      shells
adm         getty       magic       rc0.d       startcons
bcheckrc    gettydefs   motd        rc1.d       swapoff
brc         group       mount       rc2.d       swapon
brc~        inet        mtab        rc3.d       syslog.conf
csh.cshrc   init        mtools      rc4.d       syslog.pid
csh.login   init.d      pac         rc5.d       syslogd.reload
default     initrunlvl  passwd      rmt         termcap
disktab     inittab     printcap    rpc         umount
fdprm       inittab.old profile     rpcinfo     update
fstab       issue       psdatabase  securetty   utmp
ftpaccess   lilo        rc          services    wtmp
/home/larry#
```

(Для вышедших из MS-DOS пользователей полезно обратить внимание, что имена файлов могут быть длиннее 8 символов и содержать точку на любой позиции. Можно даже использовать несколько точек в одном имени).

Давайте поднимемся вверх по дереву (прим. переводчика: так уж сложилось, что в UNIX начальной вершиной дерева является "корень (root)") , используя команду ``cd ..", а затем спустимся в другой каталог (/usr/bin).

```
/home/larry# cd ..
/home# cd ..
/# cd usr
/usr# cd bin
/usr/bin#
```

Вы, разумеется, можете передвигаться по каталогам большими шагами, например, сразу выполнить cd /usr/bin.

Постарайтесь погулять по каталогам, используя команды ls и cd. В некоторых случаях вы можете напороться на раздражающее сообщение ``Permission denied"(обращение запрещено). Это всего лишь сработала система защиты UNIX, чтобы выполнять команды в тех или иных каталогах вы должны иметь на это разрешение. Подробнее об этом поговорим в Разделе 3.9.

3.3.3 Создание новых каталогов

Пора познакомиться с тем, как создавать каталоги. Это связано с использованием команды mkdir. Попробуйте следующее:

```
/home/larry# mkdir foo
/home/larry# ls -F
Mail/
foo/
letters/
papers/
/home/larry# cd foo
/home/larry/foo# ls
/home/larry/foo#
```

Наши вам поздравления! Вы только что создали новый каталог и зашли в него. Поскольку пока нет файлов в этом новом каталоге, давайте познакомимся с тем, как копировать файлы.

3.3.4 Копирование файлов

Копирование файлов осуществляется командой cp (CoPy):

```
/home/larry/foo# cp /etc/termcap  .
/home/larry/foo# cp /etc/shells  .
/home/larry/foo# ls -F
shells     termcap
/home/larry/foo# cp shells bells
/home/larry/foo# ls -F
bells     shells     termcap
/home/larry/foo#
```

Команда cp копирует файлы, перечисленные в командной строке, в файл или каталог, указанный последним аргументом. (прим. переводчика: несколько файлов одной командой cp можно скопировать только в каталог; в файл можно скопировать только один файл). Обратите внимание на то, как мы используем каталог ``."для ссылки на текущий каталог.

3.3.5 Перемещение файлов

Новая команда с именем mv (MoVe) перемещает файлы вместо их копирования. Синтаксис команды очевиден.

```
/home/larry/foo# mv termcap sells
/home/larry/foo# ls -F
bells     sells     shells
/home/larry/foo#
```

Обратите внимание, что теперь termcap уже не существует, а на его месте файл sells. Это можно использовать для переименования файлов, что мы сейчас и сделали. Но можно и переносить файлы в совсем другие каталоги.

Внимание! Команды mv и cp уничтожат содержимое файла в который они пишут (если он существовал), не спрашивая вашего разрешения. Будьте внимательны, когда вы переносите файл в другой каталог: там уже может существовать файл с таким именем и вы его затрете.

3.3.6 Удаление файлов и каталогов

Мы тут с вами "нарифмовали" ненужных файлов, изучая работу команды ls. Для удаления файлов используется команда rm (ReMove).

```
/home/larry/foo# rm bells sells
/home/larry/foo# ls -F
shells
```

```
/home/larry/foo#
```

У нас ничего не осталось, кроме shells, но не будем переживать. Обратите внимание, что команда rm не будет вас переспрашивать перед удалением, так что будьте осторожны.

Родственная rm команда rmdir. Эта команда удаляет каталоги, но только пустые каталоги. Если в каталоге есть хоть какие-нибудь файлы или подкаталоги, она распишется в бессилии.

3.3.7 Рассматривание файлов

Команды more и cat используются для просмотра содержимого файлов. more выдает файл на дисплей "поэкранно", в то время, как cat выдает весь файл разом. (прим. переводчика: если файл многострочный, то, при использовании команды cat файл промелькнет и на экране останутся последние строки).

Чтобы посмотреть файл shells, используем команду

```
/home/larry/foo# more shells
```

При использовании команды more нажимайте клавишу пробел для перехода к следующей странице и b для возврата к предыдущей. Нажав q, вы выйдете из more.

А теперь попробуйте команду cat etc/termcap/. Текст промелькнет слишком быстро, чтобы успеть его прочитать. На самом деле команда ``cat'' (conCATenate) в основном используется для других целей, для той же конкатенации нескольких файлов. Это в дальнейшем будет обсуждаться.

3.3.8 Получение оперативной помощи

Практически каждый UNIX имеет то, что называется "Руководство" - man (``manual pages''). Эта команда man содержит документацию на различные команды системы, ресурсы, конфигурационные файлы. Например, если вы хотите найти информацию о других опциях команды ls, введите

```
/home/larry# man ls
```

и вам на экран будут выведены страницы Руководства.

К сожалению, большинство страниц руководства написаны с ориентацией на пользователей, имеющих некоторые представления о работе соответствующих команд. Поэтому страницы Руководства обычно содержат справочные данные по командам, а не учебный материал.

Но Руководство неоценимо для освежения памяти, если вы забыли синтаксис команды. Руководство может также много рассказать вам о командах, которые мы даже не упомянем в этой книге.

Я предлагаю вам посмотреть в Руководстве те команды, которые мы уже обсуждали и все, с которыми мы будем встречаться. Вы обнаружите, что не на все команды есть Руководство. Тому несколько причин. Одна - некоторые страницы Руководства еще просто не написаны (*the Linux Documentation Project* - программа подготовки документации для Linux как бы отвечает за решение этой проблемы. Мы уже собрали большую часть документации). Во-вторых, команда может быть внутренней командой shell или синонимом (alias), что обсуждалось в Разделе 3.2.4, в каждом из этих случаев для них нет собственных страниц. Возьмем для примера cd, которая является внутренней командой shell. Shell выполняет эту команду, но она не имеет своей отдельной программы.

3.4 Краткая информация о базовых командах

Этот раздел представляет некоторые наиболее полезные базовые команды UNIX, включая те, о которых говорили в предыдущем разделе.

Обратите внимание, что опции обычно начинаются с ``-'' и во многих случаях несколько однобуквенных опций могут следовать за одним минусом, записанные слитно. Например, вместо использования ls -l -F, можно использовать ls -lF.

Вместо перечисления всех возможных опций каждой команды, мы будем говорить только о тех, которые полезны или важны в данное время. Действительно, большинство из этих команд имеет большое число опций (большинство из которых никогда не используется). Вы можете для каждой команды с помощью man посмотреть все возможные опции.

Обратите также внимание на то, что многие из команд берут список файлов или каталогов, как аргументы, обозначенные как ``<file1> ... <fileN>''. Например, команда cp берет в качестве аргументов список файлов, которые надо копировать, за которыми следует имя целевого файла или каталога. При копировании нескольких файлов в качестве целевого может выступать только каталог.

cd

Изменяет текущий рабочий каталог.
Синтаксис: cd <directory>;
<directory> - каталог, в который перейти (``.'' ссылается на текущий каталог, ``..'' - на родительский каталог).
Пример: cd ../foo переводит из текущего каталога в ../foo.

ls

Выдает информацию о файлах в каталоге.
Синтаксис: ls <file1> ... <fileN>
Где <file1> ... <fileN> имена файлов или каталогов, информацию про которые надо выдать.
Опции: Здесь больше опций, чем вы думаете. Наиболее часто используемые: -F (для представления информации о типах файлов), и -l (выдает в длинном (``long'') формате информацию о размерах файлов, владельцах, правах доступа и т.д. В деталях это будет обсуждаться далее).
Пример: ls -lF /home/larry выдаст содержимое каталога /home/larry.

cp

Копирует файл(ы) в файл или каталог.
Синтаксис: cp <file1> ... <fileN> <destination>
Где <file1> ... <fileN> имена копируемых файлов, а <destination> файл или каталог, в который копируют.
Пример: cp ../frog joe копирует файл ../frog в файл или каталог joe.

mv

Перемещает файл(ы) в другой файл или каталог. Эта команда не эквивалентна копированию с последующим уничтожением оригинала. Она может быть использована для переименования файлов, как команда RENAME из MS-DOS.
Синтаксис: mv <file1> ... <fileN> <destination>
Где <file1> ... <fileN> имена перемещаемых файлов, а <destination> имя файла или каталога, в который перемещают.
Пример: mv ../frog joe перемещает файл ../frog в файл или каталог joe.

rm

Удаляет файлы. Имейте в виду, когда в UNIX удаляются файлы, они невосстановимы (не как в MS-DOS, где вы можете "разудалить" файл).
Синтаксис: rm <file1> ... <fileN>
Где <file1> ... <fileN> имена удаляемых файлов.
Опции: -i потребует вашего подтверждения перед удалением файла.
Пример: rm -i /home/larry/joe /home/larry/frog удаляет файлы joe и frog в каталоге /home/larry.

mkdir

Создает новые каталоги.
Синтаксис: mkdir <dir1> ... <dirN>
Где <dir1> ... <dirN> создаваемые каталоги.
Пример: mkdir /home/larry/test создает каталог test в каталоге /home/larry.

rmdir

Эта команда удаляет пустые каталоги. При использовании rmdir ваш текущий рабочий каталог должен находиться вне удаляемого каталога.
Синтаксис: rmdir <dir1> ... <dirN>
Где <dir1> ... <dirN> удаляемые каталоги.
Пример: rmdir /home/larry/papers удаляет каталог /home/larry/papers, если он пустой.

man

Выдает страницу Руководства по данной команде или ресурсу. (здесь "ресурс" - это любая системная утилита, которая не является командой, например библиотечная функция).
Синтаксис: man <command>
Где <command> имя команды или ресурса, о котором запрашивается информация.
Пример: man ls - дает помощь по команде ls.

more

Выдает содержимое названных файлов поэкранно.
Синтаксис: more <file1> ... <fileN>
Где <file1> ... <fileN> отображаемые файлы.
Пример: more papers/history-final представляет файл papers/history-final.

cat

Используется для конкатенации файлов. cat используется также для выдачи полного содержания файла разом
Синтаксис: cat <file1> ... <fileN>
Где <file1> ... <fileN> выдаваемые файлы.
Пример: cat letters/from-mdw выдает на дисплей файл letters/from-mdw.

echo

Просто повторяет аргументы.
Синтаксис: echo <arg1> ... <argN>
Где <arg1> ... <argN> "повторяемые" аргументы.
Пример: echo "Hello world" выдает на экран ``Hello world".

grep

выдает все строки в названном файле(лах), которые содержат заданный образец.
Синтаксис: grep <pattern> <file1> ... <fileN>
Где <pattern> - образец (представленный регулярным выражением) и <file1> ... <fileN> - файлы, в которых производится поиск.
Пример: grep loomer /etc/hosts выдаст все строки, в которых файл /etc/hosts, содержит образец ``loomer".

3.5 Исследование файловой системы

Файловая система есть собрание файлов и иерархия каталогов. Я обещал поводить вас по файловой системе - и время настало. У вас достаточно интеллекта и знаний извлечь пользу из того, что я говорю и у вас есть карта дорог. (Смотрите Рис. 3.2.8).

Перво-наперво вернемся в корневой каталог (cd /) и сделаем ls -F. Вы, очевидно, увидите каталоги: bin, dev, etc, home, install, lib, mnt, proc, root, tmp, user, usr и var. (Можете увидеть и несколько отличный вариант - не волнуйтесь, различные версии Linux могут иметь отличия).

Присмотримся к каждому каталогу.

/bin

bin - это сокращенно от ``binaries" (т.е. двоичные или выполняемые файлы). Здесь находится много важных системных программ. Используйте команду ``ls -F/bin" чтобы посмотреть имеющийся здесь список файлов. Вы можете обнаружить здесь уже знакомые вам команды, вроде cp, ls и mv. Это и есть программы соответствующих команд. Когда, например, вы используете команду cp, вы выполняете программу /bin/cp.

Используя ls -F, вы увидите, что большинство (если не все) файлов в /bin имеют справа от имени звездочку (``*"). Это говорит о том, что файлы выполняемые, как описано в Разделе 3.3.2.

/dev

Следующая остановка на нашем пути - dev. Вновь посмотрите на содержимое с помощью ls -F.

"Файлы" в /dev известны как **драйверы устройств** - они используются для доступа к устройствам и ресурсам системы, таким как диски, модемы, память и т.д. Например, как вы можете читать данные из файла, точно также вы можете читать входные сигналы от мыши, имея доступ к /dev/mouse. Имена файлов, начинающиеся на fd - это дисководы гибких дисков. fd0 - первый дисковод, fd1 - второй. Теперь самые шустрые из вас заметят, что здесь имеется больше дисководов, чем те два, которые мною упоминались: они представляют специфические типы дисководов. Например, fd1H1440 представляет доступ к high-density, 3.5" дискетам на дисководе 1.

Вот перечень некоторых из наиболее используемых файлов устройств.

- /dev/console/ относится к системной консоли, т.е. к монитору, напрямую связанному с системой.

- Различные /dev/ttyS и /dev/cua устройства используются для доступа к последовательным портам. Например, /dev/ttyS0 относится к ``COM1" под MS-DOS. Устройства /dev/cua относятся к "звонящим" (``callout") устройствам, которые используются совместно с модемами.

- Устройства, имена которых начинаются с hd, имеют доступ к жестким дискам. /dev/hda относится ко *всему* первому жесткому диску, а hda1 только к *первому разделу* /dev/hda.

- Устройства с именами, начинающимися на sd - SCSI-драйверы. Если у вас SCSI жесткий диск, вместо доступа к нему через /dev/hda, вы будете обращаться к /dev/sda. SCSI ленты доступны через устройства st, a SCSI CD-ROM через sr.

- Устройства lp обеспечивают доступ к параллельным портам. /dev/lp0 относится к ``LPT1" в MS-DOS.

- /dev/null используется как "черная дыра" - любые данные, посланные сюда, канут в Лету. Если вы хотите подавить вывод команды на экран, вы можете перенаправить этот вывод в /dev/null. Мы об этом позже еще поговорим.

- Устройства с именами /dev/tty относятся к "виртуальным консолям" вашей системы (доступ путем нажатия alt-F1, alt-F2 и т.д.). /dev/tty1 соответствует первой VC, /dev/tty2 соответствует второй и т.д.

- Устройства, чьи имена начинаются на /dev/pty, это "псевдотерминалы". Они используются для входа с удаленных "терминалов". Например, если ваша машина в сети, вход к вам по telnet будет использовать одно из устройств /dev/pty.

/etc

/etc содержит множество всевозможных системных файлов конфигурации. Они включают /etc/passwd (файл паролей), /etc/rc (командный файл инициализации) и т.д.

/sbin

/sbin используется для хранения важных системных двоичных файлов, используемых системным администратором.

/home

home содержит домашние каталоги пользователей. Например, /home/larry - домашний каталог пользователя ``larry". На вновь инсталлированной системе этот каталог может быть пуст в связи с временным отсутствием зарегистрированных пользователей.

/lib

/lib содержит образы **разделяемых библиотек** (**shared library images**). Эти файлы содержат код, который могут использовать многие программы. Вместо того, чтобы каждая программа имела свою собственную копию этих выполняемых файлов, они хранятся в одном общедоступном месте - в /lib. Это позволяет сделать выполняемые файлы меньше и сэкономит место в системе.

/proc

/proc - это "виртуальная файловая система", в которой файлы хранятся в памяти, а не на диске. Они связаны с различными **процессами**, происходящими в системе, и позволяют получить информацию о том, что делают программы и процессы в указанное время. Более детально мы рассмотрим это в Разделе 3.11.1.

/tmp

Многие программы нуждаются в создании рабочих файлов, которые нужны короткое время. Каноническое место для этих файлов в /tmp (там обычно чаще проводится уборка мусора).

/usr

/usr - это очень важный каталог. Он состоит из ряда подкаталогов, которые в свою очередь содержат наиболее важные и полезные программы и файлы конфигурации, используемые системой.

Различные каталоги, описанные выше, необходимы для нормального функционирования системы, но большинство вещей, содержащихся в /usr необязательны для системы. Но это такие необязательные вещи, которые делают систему полезной и интересной. Без /usr вы бы имели достаточно занудную систему, содержащую только программы, вроде cp и ls. /usr содержит много больших программных пакетов и конфигурационных файлов, которые их сопровождают.

/usr/X386

/usr/X386 содержит The X Window System, если вы ее инсталлировали. The X Window System - это мощная графическая среда, которая содержит большое количество графических утилит и программ, отображающих "окна" на вашем экране. Если вы знакомы с Microsoft Windows или Macintosh environments, то X Windows будет выглядеть весьма похоже. Каталог /usr/X386 содержит все выполняемые и конфигурационные файлы X Window, а также файлы поддержки. Более детально это будет обсуждаться в Разделе 5.1.

/usr/bin

/usr/bin настоящее хранилище для различных программ UNIX. Он содержит большинство выполняемых программ, которых нет ни в каких других местах, например, в том же /bin их нет.

/usr/etc

Точно также, как и /etc, содержит всевозможные системные программы и конфигурационные файлы. /usr/etc содержит даже больше утилит и файлов. В общем, файлы, находящиеся в /usr/etc несущественны для системы, в отличие от тех, которые находятся в /etc, и очень существенны.

/usr/include

/usr/include содержит **include-файлы** для компилятора Си. Эти файлы (большинство имен которых заканчивается на .h (от слова ``header'') объявляют имена структур данных, подпрограмм и констант, используемых при написании программ на Си. Те файлы, которые находятся в /usr/include/sys в общем случае используются при программировании на системном уровне UNIX. Если вы знакомы с языком программирования Си, здесь вы найдете такие хэдеры (фрагменты программ, вставляемые обычно в начало программы), stdio.h, которые описывают такие функции, как printf().

/usr/g++-include

/usr/g++-include содержит include-файлы для компилятора Си++ (очень похожие на /usr/include).

/usr/lib

/usr/lib содержит библиотеки-"заглушки" и "статические" библиотеки, эквивалентные файлам из /lib. При компиляции программа "связывается" с библиотеками, находящимися в /usr/lib, которые в свою очередь направляют программы обращаться в /lib, если им нужен актуальный код. Кроме того, многие другие программы хранят в /usr/lib свои конфигурационные файлы.

/usr/local

/usr/local в большой степени похож на /usr - он содержит различные программы и файлы, несущественные для системы, но превращающие ее в удовольствие и восторг. В общем, эти программы, находящиеся в /usr/local специализируются на специфике вашей системы, т.е. /usr/local сильно отличается в различных UNIX. Здесь вы найдете такие большие программные пакеты, как TeX (система форматирования документов) и Emacs (большой и мощный редактор), если вы их установите.

/usr/man

Этот каталог содержит страницы Руководства. Здесь два подкаталога для каждого "раздела" Руководства. (С помощью команды "man man" вы можете получить более подробную информацию). Например, /usr/man/man1 содержит исходные тексты (неотформатированный оригинал) страниц Руководства в разделе 1 и /usr/man/cat1 содержит отформатированные страницы для раздела 1.

/usr/src

/usr/src содержит исходные коды (неоткомпилированные программы) для различных программ вашей системы. Наиболее важная вещь здесь, это /usr/src/linux, содержащий исходные коды ядра Linux.

/var

/var содержит каталоги, которые часто меняются в размере или имеют тенденцию быстро расти. Многие из этих каталогов "квартировались" в /usr, но поскольку мы стремимся сделать его достаточно стабильным, каталоги, которые часто меняются были перенесены в /var. К числу таких каталогов относятся:

/var/adm

/var/adm содержит различные файлы, интересные системному администратору, специфические системные файлы, фиксирующие ошибки и проблемы, возникающие в системе. Другие файлы фиксируют входы в систему, как и неудачные попытки войти. Это будет обсуждаться в Главе 4.

/var/spool

/var/spool содержит файлы, которые предварительно формируются для других программ. Например, если ваша машина подключена к сети, входная почта будет помещаться в /var/spool/mail до тех пор, пока вы не прочитаете ее или не удалите. Входящие и исходящие новости помещаются в /var/spool/news и т.д.

3.6 Типы оболочек

Как я уже много раз говорил, UNIX - это многозадачная, многопользовательская операционная система. Многозадачность *очень* полезна - однажды привыкнув к ней, вы будете всегда ее использовать. Прежде всего, вы сможете выполнять задачи в фоновом режиме, переключать задачи и объединять их в конвейер, достигая сложных результатов простыми средствами.

Многие из возможностей, которые мы будем обсуждать в этом разделе, обеспечиваются самой оболочкой (shell). Будьте внимательны, не путайте UNIX (фактическую операционную систему) с оболочкой - оболочка, это лишь интерфейс с находящейся за ней системой. Оболочка обеспечивает выполнение громадного числа функций помимо собственно UNIX.

Оболочка - это не только интерпретатор интерактивных команд, которые вы можете ввести, получив от оболочки подсказку (готовности принимать команды). Это также мощный командный язык, который позволяет писать программы (**shell-scripts**), объединяющие несколько команд в **командный файл**. Пользователи MS-DOS почувствуют здесь нечто схожее с ``batch-файлами''. Использование программ на языке оболочки (shell) - это очень мощное средство, которое позволяет автоматизировать и и существенно повысить эффективность использования UNIX. Смотрите дополнительно в Разделе 3.13.1.

Существует несколько типов оболочек в мире UNIX. Две главные - это ``Bourne shell''(shell Баурна) и ``C shell'. Shell Баурна (или просто shell) использует командный синтаксис, похожий на первоначально для UNIX придуманный (вроде UNIX System III). В большинстве UNIX-систем shell Баурна имеет имя /bin/sh (где sh сокращение от ``shell''). C shell использует иной синтаксис, чем-то напоминающий синтаксис языка программирования Си. В большинстве UNIX-систем он имеет имя /bin/csh.

В Linux есть несколько вариаций этих оболочек. Две наиболее часто используемые, это Новый Shell Баурна (Bourne Again Shell) или ``Bash'' (/bin/bash) и Tcsh (/bin/tcsh). Bash - это развитие прежнего shell с добавлением многих полезных возможностей, частично содержащихся в C shell. Поскольку Bash можно рассматривать как надмножество синтаксиса прежнего shell, любая программа, написанная на добром старом shell Баурна должна работать и в Bash. Для тех, кто предпочитает использовать синтаксис C shell, Linux поддерживает Tcsh, который является расширенной версией C shell.

Тип оболочки, которую вы решили использовать - это почти как выбор религии. Некоторые предпочитают синтаксис shell Баурна с дополнительными возможностями, предоставляемыми Bash, а некоторые - более структурированный синтаксис C shell. Для "нормальных" команд, таких как cp и ls, тип используемого вами shell никакой роли не играет. Только когда вы начнете писать командные файлы или использовать

некоторые новые свойства оболочек, различия между ними становятся существенными.

При обсуждении далее некоторых свойств оболочек мы будем обращать внимание на различие между Баурновским shell и C shell. (Если вам это действительно очень интересно, почитайте Руководство по поводу bash и tcsh).

3.7 "Уайлдкард" - "дикая карта"

Ключевое свойство большинства оболочек Unix - это способность ссылаться сразу более, чем на один файл, используя для этого специальные символы. Эти, так называемые "дикие карты" (**wildcards**), позволяют ссылаться, скажем, на все файлы, содержащие символ "n". (прим. переводчика: Мне не известен хороший перевод этой идиомы (wildcards), наиболее часто у нас встречается "генераторы" и "расширители" символов - но это тяжеловато. Чтобы далее не испытывать мучений - буду использовать слово "уайлдкард". Кстати, и оболочку удобнее далее именовать как shell, так легче воспринимается то, что это язык программирования).

Уайлдкард ``*'' относится к любому символу или строке символов в имени файла. Например, когда вы используете символ ``*'' в имени файла shell заменяет ее всеми возможными именами файлов из каталога, на который вы ссылаетесь. Вот простенький пример. Предположим, что Larry имеет файлы frog, joe и stuff в своем текущем каталоге:

```
/home/larry# ls
frog    joe    stuff
/home/larry#
```

Для обращения сразу ко всем файлам с буквой ``o'' в имени, мы можем использовать команду

```
/home/larry# ls *o*
frog    joe
/home/larry#
```

Как видите, ``*'' уайлдкард была заменена всеми возможными именами файлов из имевшихся в текущем каталоге.

Использование просто ``*''даст совпадение со всеми именами, поскольку все символы совпадают с уайлдкард.

```
/home/larry# ls *
```

```
frog    joe    stuff
/home/larry#
```

Вот еще несколько примеров.

```
/home/larry# ls f*
frog
/home/larry# ls *ff
stuff
/home/larry# ls *f*
frog    stuff
/home/larry# ls s*f
stuff
/home/larry#
```

Процесс замены ``*" на имена файлов называется расширением уайлдкард и выполняется shell. Это важно: конкретные команды, вроде ls, никогда не видят ``*" в своем списке параметров. Shell, расширяя уайлдкард, включает в список параметров все имена, прошедшие сравнение с шаблоном. Так что команда

```
/home/larry# ls *o*
```

расширяется shell до фактической

```
/home/larry# ls frog joe
```

Одно важное замечание относительно ``*" уайлдкард. Использование этой уайлдкард не даст совпадения с именами файлов, которые начинаются с точки (``."). Эти файлы воспринимаются как "спрятанные", хотя на самом деле их никуда не прятали. Они не показываются в списке, выдаваемом нормальной командой ls и не выбираются при использовании ``*" уайлдкард.

Вот пример. Мы уже упоминали, что каждый каталог имеет два специальных файла: ``." - указание на текущий каталог и ``.." - указание на родительский каталог. Однако, если вы используете команду ls, эти два файла не будут отображены.

```
/home/larry# ls
frog    joe    stuff
/home/larry#
```

Если вы используете опцию -a в команде ls, то вы сможете отобразить имена, начинающиеся на ``.":

```
/home/larry# ls -a
.    ..    .bash_profile    .bashrc    frog    joe
stuff
/home/larry#
```

Как видим, два специальных файла ``." и ``..", также, как два других "спрятанных" файла - .bash_profile и .bashrc. Эти два файла используются при входе larry в систему. Более подробно о них в Разделе 3.13.3.

Обратите внимание, что когда мы используем ``*" уайлдкард, ни один из файлов, с именами, начинающимися на ``." не отображается.

```
/home/larry# ls *
frog    joe    stuff
/home/larry#
```

Это мера предосторожности: если ``*" уайлдкард выбирала бы имена файлов, начинающиеся на ``.", она бы также выбрала имена ``." и ``..". Но это может быть опасно при выполнении ряда команд.

Другой уайлдкард является ``**?**". ``?" уайлдкард позволяет подставить строго один символ. Так ``ls ?" выдаст на только имена файлов, состоящие из одного символа, а ``ls termca?" выдаст ``termcap", но не выдаст на экран ``termcap.backup". Вот еще один пример:

```
/home/larry# ls j?e
joe
/home/larry# ls f??g
frog
/home/larry# ls ????f
stuff
/home/larry#
```

Как видите, уайлдкард позволяет описывать много файлов за один раз. При обзоре простейших команд в Разделе 3.4 мы говорили, что команды cp и mv могут копировать или перемещать множества файлов за один раз. Например,

```
/home/larry# cp /etc/s* /home/larry
```

скопирует все файлы в /etc , начиная с ``s"в каталог /home/larry. Формат команды cp на самом деле

```
cp <file1> ... <fileN> <destination>
```

где <file1> ... <fileN> - список копируемых файлов, а <destination> это файл или каталог, в который производится копирование. mv имеет аналогичный синтаксис.

Обратите внимание, что если производится копирование или перемещение более, чем одного файла, <destination> должен быть каталогом. В файл скопировать или переместить можно только один файл.

3.8 Трубопроводы UNIX

3.8.1 Стандартный вход и стандартный выход

Многие команды UNIX получают информацию с так называемого **стандартного входа** и посылают информацию на (опять же) так называемый **стандартный выход**. (Для них часто используются сокращения ``**stdin**'' и ``**stdout**'' соответственно). Ваш shell организует дело так, что стандартным входом служит клавиатура, а стандартным выходом - экран.

Вот пример использования команды cat. Нормально cat читает данные из файлов, чьи имена даны в командной строке и посылает эти данные прямехонько на stdout. Поэтому при выполнении команды

```
/home/larry/papers# cat history-final masters-thesis
```

на экран пойдет файл history-final, а за ним следом masters-thesis.

Но если команде cat не даны имена файлов в качестве параметров, она читает данные с stdin и опять же посылает на stdout. Вот пример.

```
/home/larry/papers# cat
Hello there.
Hello there.
Bye.
Bye.
[ctrl-D]
/home/larry/papers#
```

Как видите, каждая строка, которую напечатал пользователь, немедленно выдается командой cat на экран. При вводе со стандартного входа команда знает, что ввод закончен тогда, когда она получит в каком-то виде сигнал EOT (End-Of-Text). Обычно он обеспечивается нажатием ctrl-D.

Вот другой пример. Команда сортировки sort читает построчно текст (здесь опять с stdin, поскольку имена файлов в параметрах не указаны, и посылает отсортированный результат на stdout. Попробуйте так.

```
/home/larry/papers# sort
bananas
carrots
apples
[ctrl-D]
apples
bananas
carrots
/home/larry/papers#
```

Теперь мы можем упорядочить наш список продуктов, подлежащих закупке, в лексикографическом порядке... ну разве UNIX не полезная вещь?

3.8.2 Перенаправление входа и выхода

Теперь, предположим, что мы хотим послать результат сортировки в файл, чтобы где-то сохранить список планируемых покупок. Shell дает нам возможность **перенаправлять** стандартный выход в файл, используя символ ``>''. Вот как это работает.

```
/home/larry/papers# sort > shopping-list
bananas
carrots
apples
[ctrl-D]
/home/larry/papers#
```

Как вы можете видеть, результат работы команды sort не отображается на экране, вместо этого он сохраняется в файле shopping-list (список покупок). Давайте посмотрим на этот файл.

```
/home/larry/papers# cat shopping-list
apples
bananas
carrots
/home/larry/papers#
```

Теперь мы можем не только сортировать (упорядочивать) список планируемых покупок, но и сохранять его! Но предположим, что мы хранили наш неотсортированный исходный закупочный список в файле под именем

items. Один из способов сортировки и сохранения его, это отсортировать файл с данным именем, вместо получения файла со стандартного входа, и перенаправить стандартный выход в файл. Например так

```
/home/larry/papers# sort items > shopping-list
/home/larry/papers# cat shopping-list
apples
bananas
carrots
/home/larry/papers#
```

Но это можно сделать и по-другому. Перенаправлен может быть не только стандартный выход, но также и стандартный *вход*, используя символ ``<''.

```
/home/larry/papers# sort < items
apples
bananas
carrots
/home/larry/papers#
```

Технически, sort < items эквивалентно sort items, но последний вариант позволяет нам продемонстрировать сказанное: sort < items ведет себя так, словно данные файла items были напечатаны на клавиатуре. shell обслуживает перенаправление. sort не было дано имя файла (items) и команда читала со стандартного входа, как будто шел ввод с клавиатуры.

Это иллюстрирует концепцию **фильтра**. Фильтр, это программа, которая получает данные со стандартного входа, обрабатывает их каким-то образом и посылает результат обработки на стандартный выход. С помощью перенаправления стандартные вход и выход могут быть переведены на файлы. sort - простейший фильтр: она сортирует входные данные и посылает результат на стандартный выход. cat - даже еще проще: она ничего не делает со входными данными, а только выдает все, что не поступит, на выход.

3.8.3 Использование конвейера

Мы уже показали, как использовать команду sort в качестве фильтра. Но эти примеры предполагали, что вы откуда-то получили данные в файл, или ввели данные с клавиатуры своими собственными руками. А что, если данные, которые вы хотите отсортировать, являются выходными данными другой программы, например, такой как ls? Если вы используете при сортировке опцию -r, данные будут расположены в порядке, обратном лексикографическому. Если вы хотите получить перечень файлов вашего каталога в обратном порядке, один из способов сделать это будет:

```
/home/larry/papers# ls
english-list
history-final
masters-thesis
notes
/home/larry/papers# ls > file-list
/home/larry/papers# sort -r file-list
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Здесь мы сохранили результат работы команды ls в файле, а затем выполнили sort -r над этим файлом. Но это очень коряво выглядит и требует создания временного файла для хранения результата работы ls.

Выход из положения дает трубопровод (**pipeline**) (прим. переводчика: в нашей литературе принят термин "**конвейер**", так далее и будем переводить "pipeline"). Конвейер - это еще одно замечательное свойство shell, которое позволяет связывать последовательность команд в конвейер, где stdout первой команды посылается прямо на stdin второй команды и так далее. Здесь мы хотим послать stdout команды ls на stdin команды sort. Символ ``|" олицетворяет конвейер:

```
/home/larry/papers# ls | sort -r
notes
masters-thesis
history-final
english-list
/home/larry/papers#
```

Эта команда намного короче и, очевидно, проще набирается. Другой полезный пример. Команда

```
/home/larry/papers# ls /usr/bin
```

выдает на дисплей длинный список имен файлов, большинство из которых слишком быстро промелькнет на экране, чтобы вы успели прочитать их. Давайте подключим к просмотру перечня имен файлов каталога /usr/bin команду more.

```
/home/larry/papers# ls /usr/bin | more
```

Теперь вы можете постранично листать файл в свое удовольствие.

Но чудеса на этом не кончаются! Мы можем связать в конвейер более, чем две команды. Команда head представляет из себя фильтр, который отображает первые строки входного потока (здесь, пришедшего по конвейеру). Если мы хотим отобразить последнее имя текущего каталога, упорядоченного по алфавиту, мы можем написать:

```
/home/larry/papers# ls | sort -r | head -1
notes
/home/larry/papers#
```

где head -1 просто выдает первую строку получаемого входного потока (в данном случае это отсортированный в обратном порядке перечень имен файлов текущего каталога, выданных командой ls).

3.8.4 Перенаправление с добавлением

Использование ``>'' для перенаправления выхода смертельно для файла, в который происходит перенаправление (если было, что уничтожать), другими словами

```
/home/larry/papers# ls > file-list
```

уничтожает прежнее содержимое файла file-list. Если вместо этого использовать символ перенаправления ``>>'', выход будет добавлен к содержимому названного файла (вместо того, чтобы быть записанным на место старого).

```
/home/larry/papers# ls >> file-list
```

добавит выходную информацию команды ls в файл file-list.

Имейте в виду, что перенаправления и конвейер, это средства, предоставляемые оболочкой shell, это синтаксис shell и символы ``>'', ``>>'' и ``|'' не имеют никакого отношения к командам, как таковым.

3.9 Права доступа к файлам

3.9.1 Концепция прав доступа

Поскольку UNIX - многопользовательская система, чтобы защитить файлы каждого пользователя от дурного влияния других пользователей, UNIX поддерживает механизм, известный, как **система прав доступа к файлам**. Этот механизм позволяет каждому файлу приписать конкретного владельца.

Как пример, поскольку Larry создал файлы в своем домашнем каталоге, именно Larry владелец этих файлов и имеет к ним доступ.

UNIX позволяет также совместно использовать файлы нескольким пользователям и группам пользователей. Если Larry так пожелает, он может закрыть доступ к своим файлам так, что никто другой не сможет к ним подступиться. Однако в большинстве систем по умолчанию другим пользователям разрешается читать ваши файлы, но запрещается изменять или удалять.

Как объяснялось выше, каждый файл имеет конкретного владельца. Но, кроме того файлами, также владеют конкретные **группы** пользователей, которые определяются при регистрации пользователей в системе. Каждый пользователь становится членом как минимум одной группы пользователей. Системный администратор может даровать пользователю доступ более, чем к одной группе.

Группы обычно определяются типами пользователей данной машины. Например, в университетском UNIX пользователи могут быть разбиты на группы студент, преподаватель, руководство, гость. (прим. переводчика: осмелюсь предположить, что в отечественной книге перечисление примеров групп было бы начато с группы "руководство"...).

Есть также несколько системно-зависимых групп (вроде bin и admin), которые используются самой системой для управления доступом к ресурсам. Очень редко обычный пользователь принадлежит к этим группам.

Права доступа подразделяются на три типа: *чтение* (*read*), *запись* (*write*) и *выполнение* (*execute*). Эти типы прав доступа могут быть предоставлены трем классам пользователей: владельцу файла, группе, в которую входит владелец, и всем (прочим) пользователям.

Разрешение на чтение позволяет пользователю читать содержимое файлов, а в случае каталогов - просматривать перечень имен файлов в каталоге (используя, например, ls). Разрешение на запись позволяет пользователю писать в файл и изменять его. Для каталогов это дает право создавать в каталоге новые файлы и каталоги, или удалять файлы в этом каталоге. Наконец, разрешение на выполнение позволяет пользователю выполнять файлы (как бинарные программы, так и командные файлы). Разрешение на выполнение применительно к каталогам означает возможность выполнять команды вроде cd.

3.9.2 Интерпретация прав доступа

Давайте рассмотрим пример, демонстрирующий работу с правами доступа. Используя команду ls с опцией -l можно получить на экране перечень файлов

данного каталога в "длинном" формате, включающем информацию о правах доступа.

```
/home/larry/foo# ls -l stuff

-rw-r--r--   1 larry    users        505 Mar 13 19:05 stuff

/home/larry/foo#
```

Первое поле в выведенной строке представляет права доступа. Третье поле - владельца файла (larry) и четвертое - группу (users). Очевидно, что последнее поле есть имя файла (stuff), а остальные поля мы обсудим позже.

Этим файлом владеет larry, и он принадлежит группе users. Давайте посмотрим на права доступа. В строке -rw-r--r-- по порядку указаны права владельца, группы и всех прочих.

Первый символ этой строки прав доступа (``-'') представляет тип файла. Символ ``-'' означает, что это обычный файл (в противоположность каталогу или специальному файлу какого-то устройства). Следующие три позиции (``rw-'') представляют права доступа, которые имеет владелец файла larry. Символ ``r''означает ``read''(читать), ``w'' - ``write''(писать). Таким образом larry может читать файл stuff и писать в него.

Как мы уже упоминали, кроме разрешений на чтение и запись существует разрешение на выполнение ``execute'' - представляемое символом ``x''. Но в данном случае на этой позиции ``-'', так что у Larry нет прав на выполнение этого файла. И это чудесно, файл stuff совсем даже не является программой. Разумеется, поскольку Larry владеет файлом, он может дать сам себе разрешение на выполнение этого файла, если захочет. Мы эту процедуру скоро обсудим.

Следующие три символа r-- представляют права доступа группы для этого файла. Эта группа имеет имя users. Поскольку тут есть только ``r'', любой пользователь этой группы может только читать файл.

Последние три символа представляют ту же комбинацию r--, то есть для всех прочих доступно чтение этого файла и запрещены запись и выполнение.

Вот еще несколько примеров на права доступа.

-rwxr-xr-x
: Владелец файла может читать, писать и выполнять файл. Члены группы и все прочие пользователи могут читать и выполнять файл.

-rw-------

Владелец файла может читать и писать в файл. Всем остальным доступ к файлу закрыт.

-rwxrwxrwx

Все могут читать писать и выполнять файл.

3.9.3 Зависимости

Важно заметить, что права доступа, которые имеет файл зависят также от прав доступа к каталогу, в котором этот файл находится. Например, даже если файл имеет -rwxrwxrwx, другие пользователи не смогут до него добраться, если у них не будет прав на чтение и выполнение каталога, в котором находится файл. Например, если Larry захочет ограничить доступ ко всем своим файлам, он может просто изменить права доступа своего домашнего каталога /home/larry на drwx------. Таким образом, никто другой не будет иметь доступ в его каталог, а следовательно посторонним будут недоступны и все файлы. Так что Larry может не заботиться об индивидуальной защите своих файлов.

Другими словами, чтобы иметь доступ к файлу, вы должны иметь доступ ко всем каталогам, лежащим на пути от корня к этому файлу, а также разрешение на доступ собственно к этому файлу.

Обычно пользователи UNIX весьма открыты всеми своими файлами. Обычно файлам устанавливается защита -rw-r--r--, которая позволяет другим пользователям читать файлы, но ни коим образом их не менять. Каталогам обычно устанавливаются права доступа drwxr-xr-x, что позволяет другим пользователям ходить с правами экскурсантов по вашим каталогам. Но ничего в них не трогать и не записывать.

Но многие пользователи хотят держать других пользователей подальше от своих файлов. Установив права доступа файла, -rw------- вы никому не покажете этот файл и не дадите записать в него. Также хорошо закрывает от всех файлы защита соответствующего каталога drwx------ .

3.9.4 Изменение прав доступа

Команда chmod используется для установки (изменения) прав доступа файла. Только владелец файла может менять права доступа к нему.

Синтаксис команды имеет вид:

```
chmod {a,u,g,o}{+,-}{r,w,x} <filenames>
```

Кратко, вы выбираете из **a**ll (все), **u**ser (пользователь), **g**roup (группа) или **o**ther (другие). Далее указываете, либо вы добавляете права (+), либо лишаете

прав (-). И наконец, вы указываете один или несколько режимов: **r**ead, **w**rite или **e**xecute. Несколько примеров допустимых команд:

chmod a+r stuff
: Дает всем пользователям право читать файл stuff.

chmod +r stuff
: То же самое, что и ранее (a - по умолчанию).

chmod og-x stuff
: Лишает права на выполнение всех, кроме владельца.

chmod u+rwx stuff
: Разрешает владельцу все (read, write и execute).

chmod o-rwx stuff
: Запрещает все (read, write и execute) пользователям категории другие (other).

3.10 Управление связями файлов

Связи позволяют давать одному физическому файлу много имен. Системой файлы распознаются по **индексам файлов**, которые являются уникальными идентификаторами в рамках системы. Команда ls -i выдаст вам индексы файлов. На самом деле каталог - это перечень индексов файлов с соответствующими этим индексам номерами. Каждое имя файла в каталоге привязано к конкретному индексу.

3.10.1 Жесткие связи

Команда ln используется для создания множества связей для одного файла. Например, скажем, что у вас есть файл foo. Используя ls -i можно посмотреть индекс этого файла.

```
# ls -i foo
22192 foo
#
```

Здесь файл foo имеет в файловой системе индекс 22192. Мы можем создать новую связь для этого файла под именем bar:

```
# ln foo bar
```

С помощью ls -i можно убедиться, что оба файла имеют один и тот же индекс.

```
# ls -i foo bar
22192 bar   22192 foo
```

```
#
```

Теперь, обращаясь к foo или bar мы фактически обратимся к одном у тому же файлу. Поэтому, если мы меняем что-то в файле foo, эти же самые изменения произойдут в файле bar.

Эти связи известны, как *жесткие связи* (*hard links*), поскольку они реализуются прямой ссылкой на индекс файла. Обратите внимание, что в рамках одной файловой системы вы можете организовать только жесткие связи; символические связи (смотрите ниже) не имеют этого ограничения.

Когда вы удаляете файл командой rm, на самом деле вы удаляете только одну ссылку на файл. Если вы введете команду

```
# rm foo
```

Удаляется только связь, имеющая имя foo; bar будет как и прежде существовать. Файл только тогда действительно удаляется, когда на него больше нет связей. Обычно файлы имеют только одну связь, так что команда rm действительно приведет к удалению файла. Однако, если файл имеет много ссылок, применение rm приведет только к удалению одной связи; для того, чтобы удалить файл, вы должны удалить все связи на этот файл.

Команда ls -l покажет число ссылок на файл (кроме прочей информации)

```
# ls -l foo bar
-rw-r--r--   2 root     root          12 Aug  5 16:51 bar
-rw-r--r--   2 root     root          12 Aug  5 16:50 foo
#
```

Вторая колонка с цифрой ``2'' показывает число связей файла.

Самом деле оказывается, что каталоги представляют из себя справочник типа "имена-индексы". Кроме прочего, каждый каталог имеет минимум две жесткие ссылки: ``.'' (ссылка, указывающая на самого себя) и ``..'' (ссылка, указывающая на родительский каталог). В корневом каталоге (/) ссылка ``..'' указывает на сам же каталог /.

3.10.2 Символические связи

Символические связи, это другой тип связей, отличающийся от жестких связей. Символические связи позволяют давать новые имена файлам, но при этом не ссылаются на индекс файла.

Команда ln -s создаст символическую ссылку на указанный файл. Например, если мы воспользуемся командой

```
# ln -s foo bar
```

мы создадим символьную ссылку bar, указывающую на файл foo. Если теперь используем команду ls -i, то увидим, что два файла имеют различные индексы.

```
# ls -i foo bar
22195 bar   22192 foo
#
```

Однако, используя ls -l, мы видим, что файл bar имеет символический указатель на foo.

```
# ls -l foo bar
lrwxrwxrwx   1 root     root           3 Aug  5 16:51  bar -> foo
-rw-r--r--   1 root     root          12 Aug  5 16:50 foo
#
```

При символической ссылке не используются биты прав доступа (они всегда отображаются, как rwxrwxrwx). Вместо этого, права доступа к файлу, полученному символической ссылкой, определяются правами доступа к файлу, на который он ссылается (в нашем примере определяется правами файла foo).

Функционально, жесткие ссылки и символические ссылки похожи, но есть некоторые различия. Например, вы можете создать символическую ссылку на файл, который не существует; так нельзя сделать применительно к жесткой ссылке. Символические ссылки обрабатываются ядром иным образом, чем жесткие. Это скорее техническое отличие, но иногда важное. Символические ссылки полезны, поскольку они позволяют идентифицировать файл, на который они указывают; для жестких ссылок нет простого способа определить, какие файлы привязаны к одному и тому же индексу.

Ссылки используются во многих местах системы Linux. Символические ссылки особенно важны для образов разделяемых библиотек в /lib. Смотри дополнительную информацию в Разделе 4.7.2.

3.11 Управление работами

3.11.1 Работы и процессы

Управление работами (**job control**) это возможность, которую предоставляют многие оболочки, включая (Bash и Tcsh). Управление

работами (прим. переводчика: job - работа в добрые старые времена страшноватых IBM/360 переводилось как "задание", но лучше это не тащить в сегодня) позволяет управлять множеством команд или **работ** одновременно. Прежде, чем вы закопаетесь значительно глубже, следует поговорить о **процессах**.

Каждый раз, когда вы выполняете программу, вы начинаете то, что известно, как *процесс*. Процесс - это название для выполняемой программы. Команда ps выдает перечень имеющих место в данный момент процессов. Вот пример:

```
/home/larry# ps

  PID TT STAT  TIME COMMAND
   24  3 S     0:03 (bash)
  161  3 R     0:00 ps

/home/larry#
```

PID (**Process IDentificator**), перечисленные в первой колонке, это неповторяющиеся числа приписанные всем идущим процессам.

Последний столбец (COMMAND) дает имя выполняемой команды. Здесь мы видим только процессы, которые инициировал Larry. (В системе выполняется и много других процессов. Команда ``ps -aux" может выдать перечень всех происходящих в данный момент процессов).

В выведенном перечне указаны bash (это оболочка, используемая Larry) и сама команда ps. Как вы видите, bash выполняется параллельно с командой ps. bash выполнит ps, когда Larry введет команду. После окончания ps (после того, как выдана таблица процессов), управление возвращается к процессу bash, который выдает на экран подсказок готовности к приему новых команд.

Выполняемый процесс известен shell как *работа*. Термины *процесс* и *работа* взаимозаменяемы. Однако процесс обычно воспринимается, как "работа", когда речь идет об **управлении работами** (**job control**)- свойстве shell, позволяющем уделять внимание нескольким независимым работам.

В большинстве случаев пользователи выполняют в каждый момент времени одну работу, ту которая соответствует последней переданной shell команде. Однако, используя управление работами, вы можете одновременно выполнять несколько работ, по необходимости переключаясь с одной на другую. Какая от этого польза? Давайте предположим, что вы редактируете текстовый файл и неожиданно хотите прерваться и сделать что-то другое. С помощью управления работами вы можете отложить редактирование и, вернувшись к подсказке shell, начать какую-то другую работу. После этого вы можете вернуться к редактированию, именно к тому месту, где вы

прервали редактирование. Это всего один пример. Управление работами очень полезно на практике.

3.11.2 Выполнение работ на переднем плане и в фоне

Работы могут выполняться как на **переднем плане**, так и в **фоне**. На переднем плане в каждый момент может быть только одна работа. Работа переднего плана, это работа, с которой вы взаимодействуете, она получает информацию с клавиатуры и посылает результаты на ваш экран. (Кроме, разумеется, случаев, когда вы сами перенаправляете вход или выход, как описывалось в Разделе 3.8). С другой стороны, фоновые работы не получают информации с терминала, в общем случае они тихо (в смысле - мирно) выполняются, не испытывая потребности в общении с пользователем.

Некоторые работы требуют очень большого времени для своего завершения и не свершают ничего внешне интересного в процессе этой работы. Компиляция программ - одна из таких работ, как и компрессия больших файлов. Нет вразумительных причин, почему вы должны при этом сидеть рядом и мучительно ждать, когда эти работы закончатся. Вы можете просто запустить их в фоне. Пока они там выполняются, вы можете заняться другими программами.

Работы могут быть также **отложены**. Отложенная работа - это работа, которая в данный момент не выполняется и временно остановлена. После того, как вы остановили работу, в дальнейшем вы можете ее продолжить как на переднем плане, так и в фоне. Возобновление приостановленной работы не изменит ее состояния - при возобновлении она начнется с того места, на котором была приостановлена.

Имейте в виду, что приостановка работы, это не *прерывание* работы. Когда вы прерываете идущий процесс (нажимая клавиши прерывания, обычно это ctrl-C), то убиваете процесс насовсем. (Клавиши прерывания можно переустанавливать командой stty. По умолчанию прерывание находится под ctrl-C, но мы не можем это гарантировать для всех систем). Если работа убита, то уж убита, и нет другого способа возобновить ее, как вновь запустить сначала, используя прежнюю команду. Заметим также, что некоторые программы могут перехватывать прерывания, тогда нажатие ctrl-C не приведет к немедленному прекращению работы. Это позволит программе выполнить необходимые операции аккуратного завершения. Некоторые программы вообще не позволят вам их прервать.

3.11.3 Работа в фоне и ликвидация работ

Давайте начнем с простого примера. Команда yes - вроде бы бесполезная команда, посылающая бесконечный поток "y" на стандартный выход. (Но это очень полезно. Если вы направите через конвейер эти "y" на вход другой

команды, которая требует ответов yes и "no" на вопросы, поток "y" даст подтверждение на все вопросы). Попробуйте.

```
/home/larry# yes
y
y
y
y
y
```

Это закончится в *бесконечности*. Вы можете убить процесс, нажав клавиши прерывания; обычно это ctrl-C. Чтобы нас больше не раздражал поток нескончаемых "y", перенаправим его в /dev/null. Как вы помните, /dev/null выступает в качестве "черной дыры" для данных. В ней исчезают бесследно любые данные.

```
/home/larry# yes > /dev/null
```

Ох, теперь намного лучше. Ничего не печатается, но и подсказка shell не появляется. Это потому, что программа продолжает работать, посылать "y" в /dev/null. Снова нажмите клавиши прерывания, чтобы прекратить это.

Давайте предположим, что мы хотим, чтобы команда yes продолжала работать, но также хотим получить обратно подсказку shell, чтобы выполнять другие работы. Мы можем перевести команду yes в фоновый режим, что позволит ей выполняться, но без выхода на взаимодействие с пользователем.

Чтобы переместить процесс в фоновый режим, необходимо после команды символ ``&''.

```
/home/larry# yes > /dev/null &
[1] 164
/home/larry#
```

Вы видите, что мы вновь получили подсказку. Но что значит ``1 164''? И выполняется ли команда yes на самом деле?

``1'' представляет **номер работы** для программы yes. Shell приписывает номер каждой выполняемой работе. Поскольку "yes" - одна единственная работа, которая в данный момент выполняется, ей присвоен номер 1. ``164'' - идентификатор процесса (PID); это номер, присвоенный системой работе. Любой из этих номеров можно использовать при обращении к работе, как это будет показано в дальнейшем.

Теперь мы имеем выполняемый процесс yes в фоновом режиме, непрерывно посылающий поток "y"-ков в /dev/null. Чтобы проверить состояние этого процесса, используйте внутреннюю команду shell - jobs.

```
/home/larry# jobs
[1]+  Running                yes >/dev/null  &
/home/larry#
```

Ясно, что она выполняется. Вы можете также воспользоваться командой ps, показанной ранее, для проверки статуса работ.

Для завершения работы используйте команду kill. Эта команда может брать в качестве аргумента как номер работы, так и идентификатор процесса. Это была работа номер 1, так что используя команду

```
/home/larry# kill %1
```

мы ликвидируем работу. При идентификации работы по номеру необходимо впереди ставить символ процента (``%'').

Теперь, после ликвидации, мы можем снова использовать jobs для проверки:

```
/home/larry# jobs

[1]+  Terminated             yes >/dev/null

/home/larry#
```

Работа действительно мертва, и если мы снова воспользуемся командой jobs, ничего не будет выведено на экран.

Вы можете также убить работу, используя номер идентификатора процесса (PID), который выводится наряду с работой, когда вы начинаете работу (в фоновом режиме). В нашем пример PID равен 164, так что команда

```
/home/larry# kill 164
```

эквивалентна

```
/home/larry# kill %1
```

Вам не надо использовать ``%'', когда вы обращаетесь к работе по номеру идентификатора процесса.

3.11.4 Остановка и возобновление работы

Есть другой способ перевести работу в фоновый режим. Вы можете начать работу нормально (в режиме переднего плана), **остановить** работу и продолжить в фоновом режиме.

Сначала начнем работу "нормально":

```
/home/larry# yes > /dev/null
```

Поскольку опять работа выполняется на переднем плане, вы не получите обратно на экран подсказку shell.

Теперь, вместо того, чтобы прерывать работу с помощью ctrl-C, мы остановим работу. *Приостановка* работы не убивает ее. Чтобы осуществить приостановку работы, надо нажать соответствующие клавиши, обычно это ctrl-Z.

```
/home/larry# yes > /dev/null
[ctrl-Z]
[1]+  Stopped                 yes >/dev/null
/home/larry#
```

Пока работа остановлена, она просто не выполняется. На нее не тратится время процессора. Но вы всегда можете возобновить работу, и она продолжится как ни в чем не бывало.

Для возобновления работы в режиме переднего плана используйте команду fg (``foreground" - передний план).

```
/home/larry# fg
yes >/dev/null
```

Shell снова выдаст на экран имя команды, чтобы вы могли проконтролировать, какую работу вы активизировали в режиме переднего плана. Вновь остановите работу с помощью ctrl-Z. В этот раз используйте команду bg (``background" - задний план, фоновый режим), чтобы перевести работу в фоновый режим. Эффект будет аналогичен тому, как если бы вы набрали после команды ``&".

```
/home/larry# bg
[1]+ yes >/dev/null &
/home/larry#
```

И мы получили назад подсказку. Команда jobs сообщит, что команда yes действительно выполняется, и мы можем снова ее убить с помощью команды kill, как мы это уже делали.

Как теперь остановить работу? Использование ctrl-Z не поможет, поскольку работа находится в фоновом режиме. Ответ - переместить работу на передний план, а затем остановить. Вы можете использовать fg как для остановленных работ, так и для работ, находящихся в фоне.

Существует большая разница между фоновой работой и остановленной. Остановленная работа не выполняется и не использует время процессора, да и никакой работы, честно говоря, в этот момент не делает (но занимает память, хотя по воле своппинга может оказаться на диске). Работа в фоновом режиме и выполняется, и занимает память. Она может даже выводить что-то на экран, хотя это может раздражать вас, когда вы работаете над чем-то другим. Например, если вы использовали команду:

```
/home/larry# yes &
```

без перенаправления stdout в /dev/null, поток "y" будет выводиться на экран без возможности прервать это (вы не сможете использовать ctrl-C для прерывания работ фонового режима). Чтобы остановить эту бесконечную выдачу, вам следует использовать команду fg для перевода работы в режим переднего плана, а затем использовать ctrl-C, чтобы ее убить.

Еще одно замечание. Команды fg и bg обычно переводят на передний план или в фоновый режим работы, которые были остановлены последними (что определяется символом ``+'' после номера работы, это когда вы используете команду jobs). Если вы выполняете много работ одновременно, вы можете перевести на передний план или, наоборот, в фоновый режим конкретную работу заданием идентификатора работы в качестве аргумента команд fg или bg, как в

```
/home/larry# fg %2
```

(перевод на передний план работы номер 2) или

```
/home/larry# bg %3
```

(перевод в фон работы номер 3).

Для этих команд нельзя использовать идентификаторы процессов. Кроме того, использование только номеров работ, как в

/home/larry# %2

эквивалентно

/home/larry# fg %2

Помните, что управление работами, это свойство shell. Команды fg, bg и jobs - внутренние команды shell. Если по какой-то причине вы используете shell, который не поддерживает управление работами, там вы не найдете этих команд.

В дополнение к этому, есть некоторые аспекты управления работами, которые различаются в Bash и Tcsh. Некоторые оболочки не имеют управления работами, хотя большинство оболочек Linux имеют такую возможность.

3.12 Использование редактора vi

Текстовый редактор, это программа, используемая для редактирования файлов, которые содержат текст, например письма, C-программы или системные конфигурационные файлы. Хотя в Linux много всяких разных редакторов, единственный редактор, который вы с гарантией найдете в любом UNIX - это vi (``visual editor''). vi - это не самый простой в использовании редактор. Но поскольку он так распространен в мире UNIX и в любой момент может вам потребоваться, он заслуживает хоть какого-то описания здесь.

Выбор редактора, это дело персонального вкуса и стиля. Многие пользователи предпочитают витиеватый и мощный *Emacs* - редактор с самым большим набором возможностей, по сравнению со всеми другими редакторами в мире UNIX. Например, Emacs имеет свой собственный встроенный диалект языка программирования LISP и множество расширений (одно из которых ``Eliza''- в некотором роде программа искусственного интеллекта). Однако, поскольку Emacs со всеми поддерживающими его файлами сравнительно велик, его нет на многих системах. vi, наоборот, маленький и удаленький, но, увы, более сложный в использовании. Но когда вы с ним освоитесь, вы поймете, что он очень простой. Правда осваивать его сложно.

Этот раздел - вразумительное введение в vi. Мы не будем обсуждать все его свойства, а только те, которые вы должны знать, чтобы начать работать. Если вы пожелаете знать больше деталей, обратитесь к страницам Руководства.

3.12.1 Концепции

При использовании vi в любое время вы можете находиться в одном из трех режимов работы. Эти режимы известны как командный режим, режим вставки и режим последней строки.

Когда вы начинаете работать с vi - вы в командном режиме. Этот режим позволяет использовать определенные команды для редактирования файлов или перехода в другие режимы. Например, напечатав ``x'' при нахождении в командном режиме, удаляете символ, находящийся перед курсором. Стрелки передвигают курсор по редактируемому файлу. Большинство команд, используемых в командном режиме, состоит из одного или двух символов.

Вставку или редактирование текста вы осуществляете в режиме вставки. При использовании vi вы, возможно, большую часть времени находитесь именно в этом режиме. Вы переходите в режим вставки с помощью команды ``i'' (``insert'' - вставка) из командного режима. В режиме вставки вы вставляете текст в документ на место, указываемое курсором. Для завершения режима вставки и возврата в командный режим следует нажать esc.

Режим последней строки - это специальный режим, используемый для расширения возможностей командного режима. При вводе таких команд они появляются в последней строке экрана. Например, если вы напечатаете ``:'' в командном режиме, вы перейдете в режим последней строки и сможете использовать такие команды, как ``wq'' (записать (write) файл и выйти (quit) из vi), или ``q!'' (выйти из vi без сохранения изменений). Режим последней строки в общем случае используется для команд vi, которые длиннее одного символа. В режиме последней строки вы вводите однострочные команды и нажимаете enter для их выполнения.

3.12.2 Начала vi

Лучший способ освоить эту концепцию, это вызвать vi и отредактировать файл. В примере ``screens'', приводимом ниже, мы собираемся только показать несколько строк текста, будто бы экран состоит всего из шести строк (вместо двадцати четырех).

Вызов vi

vi <filename>

где <filename> - имя редактируемого файла.

Ну так вызовите vi, напечатав

```
/home/larry# vi test
```

для редактирования файла test. Вы увидите нечто вроде

```
____________________________________________________________________
__
|~_                                          |
|~                                          |
|~                                          |
|~                                          |
|~                                          |
|"test"_[New_file]_________________________________________________|
```

Столбец символов ``~" говорит о том, что вы стоите на конце файла.

3.12.3 Вставка текста

Вы находитесь в командном режиме; для того, чтобы вставлять текст в файл, нажмите i (что переведет вас в режим вставки) и начинайте печатать.

```
____________________________________________________________________
__
|Now is the time for all good men to come to the aid of the party._ |
|~                                          |
|~                                          |
|~                                          |
|~                                          |
|~__________________________________________________________________
_|
```

При вставке текста вы можете напечатать столько строк, сколько пожелаете (нажимая return после каждой строки), и можете корректировать ошибки используя клавишу возврата (backspace).

Для завершения режима вставки и возврата в командный режим нажмите esc.

В командном режиме вы можете использовать клавиши со стрелками для перемещения по файлу. Здесь, поскольку мы имеем только одну строку текста, попытки использовать стрелки "вверх" и "вниз" приведут лишь к тому, что vi на вас загудит.

Есть несколько способов вставки текста, отличных от использования команды i. Например, команда a вставляет в текст, начиная *после* текущего положения курсора, вместо текущей позиции курсора. Используйте левую стрелку для перемещения курсора между словами ``good" и ``men".

```
_______________________________________________________________________
__
|Now is the time for all good_men to come to the aid of the party.  |
|~                                     |
|~                                     |
|~                                     |
|~                                     |
|~_____________________________________________________________________
_|
```

Нажмите a, для начала режима вставки, напечатайте ``wo'', а затем нажмите esc для возврата в командный режим.

```
_______________________________________________________________________
__
|Now is the time for all good women to come to the aid of the party.|
|~                                     |
|~                                     |
|~                                     |
|~                                     |
|~_____________________________________________________________________
_|
```

Для того, чтобы начать вставку текста в строку ниже текущей, используйте команду ``o''. Например, нажмите o и напечатайте строчку или две

```
_______________________________________________________________________
__
|Now is the time for all good women to come to the aid of the party.|
|Afterwards, we'll go out for pizza and beer._            |
|~                                     |
|~                                     |
|~                                     |
|~_____________________________________________________________________
_|
```

Но помните, что в любое время вы находитесь либо в командном режиме (где команды, такие как i, a или o могут применяться) или в режиме вставки (где вы вставляете текст, а затем с помощью esc возвращаетесь в командный режим) или в режим последней строки (в котором вы расширяете расширяемые команды, как это обсуждается ниже).

3.12.4 Удаление текста

В командном режиме команда "x" удаляет символ перед курсором. Если вы нажмете x пять раз, вы закончите в ситуации:

```
____________________________________________________________________
__
|Now is the time for all good women to come to the aid of the party.|
|Afterwards, we'll go out for pizza and _                    |
|~                                          |
|~                                          |
|~                                          |
|~__________________________________________________________________
_|
```

Теперь нажмите a, вставьте некоторый текст, а затем нажмите esc:

```
____________________________________________________________________
__
|Now is the time for all good women to come to the aid of the party.|
|Afterwards, we'll go out for pizza and Diet Coke._          |
|~                                          |
|~                                          |
|~                                          |
|~__________________________________________________________________
_|
```

Вы можете удалять целые строки, набирая команду dd (т.е. нажимая d дважды). Если ваш курсор на второй строке, и вы напечатали dd,

```
____________________________________________________________________
__
|Now is the time for all good women to come to the aid of the party.|
|~                                          |
|~                                          |
|~                                          |
|~                                          |
|~__________________________________________________________________
_|
```

Чтобы удалить слово, на котором находится курсор, используйте команду dw. Поместите курсор на слово ``good'' и напечатайте dw.

```
____________________________________________________________
__
|Now is the time for all women to come to the aid of the party.     |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~__________________________________________________________________
_|
```

3.12.5 Изменение текста

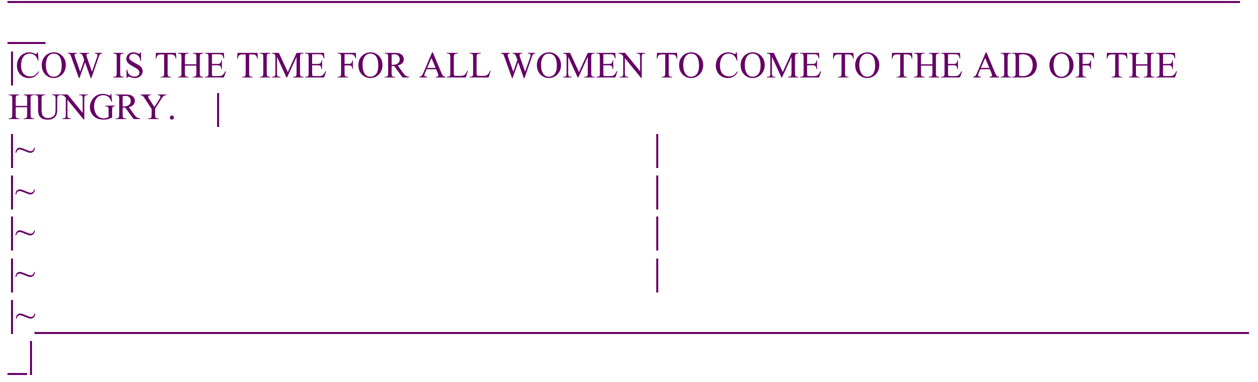
Вы можете заменить фрагменты текста, используя команду R. Поместите курсор на первую букву слова ``party'', нажмите R и напечатайте слово ``hungry''.

```
____________________________________________________________
__
|Now is the time for all women to come to the aid of the hungry._   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~__________________________________________________________________
_|
```

Использование R для редактирования текста очень походит на на команды i и a, но R заменяет прежний текст вместо вставки в него. Команда r заменяет один символ, отмеченный курсором. Например, переместите курсор на начало слова ``Now'' и напечатайте r, а следом C, то вы получите:

```
____________________________________________________________
__
|Cow is the time for all women to come to the aid of the hungry._   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
|~                                                                   |
```

Команда ``&tilde" изменяет размер буквы, отмеченной курсором: большую делает маленькой и наоборот. Например, если вы поместите курсор на ``o" в ``Cow" и затем последовательно будете нажимать ~, вы в конечном итоге получите:



3.12.6 Команды перемещения

Вы уже знаете, как использовать стрелки для перемещений по документу. Вы также можете использовать команды h, j, k, и l для перемещения курсора влево, вниз, вверх и вправо соответственно. Это удобно, если (по каким-то причинам) ваши клавиши со стрелками не работают как надо.

Команда w перемещает курсор на начало следующего слова; b - перемещает на начало предыдущего слова.

Команда 0 (это ноль) передвигает курсор на начало текущей строки, а команда $ перемещает на конец строки.

При редактировании больших файлов вы хотите перемещаться вперед и назад сразу на размер экрана. Нажатием ctrl-F курсор перемещается на экран вперед, с помощью ctrl-B - на экран назад.

Для того, чтобы переместить курсор в конец файла, напечатайте G. Можно переместиться также на любую строку, напечатав команду 10G вы переместите курсор на десятую строку файла. Для того, чтобы встать на начало (на первую строку), используйте 1G.

Вы можете сочетать команды перемещения с другими командами, такими как удаление. Например, команда d$ удалить от местоположения курсора до конца строки; dG удалит все от курсора до конца файла и т.д.

3.12.7 Сохранение файлов и выход из vi

Для выхода из vi без внесения изменений в ранее существовавший файл используйте команду :q!.

Когда вы напечатаете ``:'', курсор переместится на последнюю строку экрана, поскольку вы перейдете в режим последней строки.

```
________________________________________________________________________
__
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE
HUNGRY.    |
|~                                                      |
|~                                                      |
|~                                                      |
|~                                                      |
|~                                                      |
|:______________________________________________________________________
_|
```

В режиме последней строки могут выполняться некоторые расширенные команды. Одна из них - q!, которая позволяет выйти из vi без записи. Команда :wq сохраняет (записывает) файл, а затем выходит из vi. Команда ZZ (в режиме команд, без `:'') эквивалентна :wq. Помните, что вы должны нажать enter после набора команды в режиме последней строки. Если хотите записать файл без выхода из /vi, используйте просто :w.

3.12.8 Редактирование еще одного файла

Для того, чтобы отредактировать другой файл, используйте команду :e. Например, чтобы прекратить редактирование файла test и перейти к редактированию файла foo, используйте команду

```
________________________________________________________________________
__
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE
HUNGRY.    |
|~                                                      |
|~                                                      |
|~                                                      |
|~                                                      |
|~                                                      |
|:e foo_________________________________________________________________|
```

Если вы используете :e без предварительного сохранения файла, то сначала вы получите сообщение об ошибке.

```
________________________________________________________________________
__
 |No_write_since_last_change_(":edit!"_overrides)____________________|
```

которое просто означает, что vi не желает редактировать другой файл, пока не будет сохранен первый. В этот момент вы можете использовать :w, чтобы сохранить исходный файл, а затем использовать :e или использовать команду

```
________________________________________________________________________
__
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE
HUNGRY.    |
|~                                                  |
|~                                                  |
|~                                                  |
|~                                                  |
|~                                                  |
|:e!
foo____________________________________________________________|
```

``!" говорит vi, что вы на самом деле имеете в виду - редактировать новый файл без сохранения изменений, которые делались в первом.

3.12.9 Включение других файлов

Если вы используете команду :r, вы можете включить содержимое другого файла в текущий файл. Например, команда

```
:r foo.txt
```

вставит содержимое файла foo.txt в данное место текста.

3.12.10 Выполнение команд Shell

Вы можете также выполнять команды прямо из vi. Команда :r! работает как :r, но вместо чтения файла она вставляет выход данной команды в буфер, в место, где находится курсор. Например, если вы используете команду

```
:r!  ls -F
```

вы получите в результате

```
________________________________________________________________________
__
|COW IS THE TIME FOR ALL WOMEN TO COME TO THE AID OF THE
HUNGRY.    |
|letters/                                        |
|misc/                                           |
|papers/_                                        |
|~                                               |
|~______________________________________________________________________
_|
```

Вы можете выполнить команду а, находясь в редакторе vi и вернуться в редактор после ее завершения Например, если вы используете команду

```
:!  ls -F
```

будет выполнена команда ls -F, а результат выдан на экран, а не вставлен в редактируемый файл. Если вы используете команду

```
:shell
```

vi запустит shell, который позволит временно "отложить" vi и выполнить команды. После выхода из shell (используя команду exit) вы вернетесь в vi.

3.12.11 Получение помощи

vi не слишком силен в интерактивной помощи (да и большинство UNIX-ов также), но вы всегда можете посмотреть страницы Руководства для vi. vi - это "визуальная составляющая" редактора ex; это ex делает многое для поддержания режима последней строки и командного режима в vi. Так что в дополнение к чтению Руководства по vi посмотрите также Руководство по ex.

3.13 Установка среды

Shell обеспечивает различные механизмы настройки вашей рабочей среды. Мы уже упоминали ранее, что shell больше, чем команда интерпретации - это также мощный язык программирования. Но обсуждение программирования на shell увело бы нас далеко в сторону, а мы бы хотели познакомить вас с некоторыми способами упрощения вашей работы в UNIX за счет использования некоторых дополнительных полезных свойств shell.

Как мы упоминали ранее, различные оболочки используют различный синтаксис для написания своих программ. Например, Tcsh использует синтаксис, похожий на язык Си, в то время как shell Баурна имеет другой синтаксис. В этом разделе мы не будем заниматься их различиями, а рассмотрим примеры, используя синтаксис shell Баурна (прим. переводчика: как все обычно и делают).

3.13.1 Сценарии shell

прим. переводчика: применительно к программам этого типа в англоязычной литературе последнее время преимущественно используют слово script - "сценарий", хотя то, что под этим имеется в виду во многих книгах на русском называется "традиционно" как "программа на shell" или "командный файл"

Предположим, что вы часто используете серию команд и хотели бы сократить объем постоянной печати за счет группировки их в одну команду. Например, команды

```
/home/larry# cat chapter1 chapter2 chapter3 > book
/home/larry# wc -l book
/home/larry# lp book
```

об'единяют файлы, содержащие главы книги: chapter1, chapter2, chapter3 и помещают результат в файл book. Затем подсчитывается число строк в книге (в файле book) и отображается на дисплее и, наконец, печатается командой lp.

Вместо введения каждый раз этих команд, вы можете собрать их в один сценарий (командный файл). Сценарии shell мы кратко опишем в Разделе 3.13.1. А сценарий, который выполнит вышеприведенные команды, будет выглядеть следующим образом

```
#!/bin/sh
# A shell script to create and print the book

cat chapter1 chapter2 chapter3 > book
wc -l book
lp book
```

Если этот сценарий будет помещен в файл makebook, то вы можете просто использовать далее команду

```
/home/larry# makebook
```

которая выполнит все команды сценария. Сценарии shell - это обычные текстовые файлы, которые вы можете создавать с помощью редактора вроде emacs или vi. (vi обсуждался в Разделе 3.12)

Давайте посмотрим на этот сценарий. Первая строка ``#!bin/sh/'' говорит о том, что этот файл есть сценарий и сообщает shell, как выполнить сценарий. В данном случае необходимо передать сценарий для выполнения команде bin/sh/, где bin/sh/ - сама программа shell. Почему это важно? В большинстве систем UNIX bin/sh/ - это shell Баурновского типа, например bash. Инициируя работу сценария shell выполняется, используя bin/sh/, при этом мы гарантируем, что сценарий будет выполняться именно под shell Баурновского типа (а не, скажем, под C shell). Этот сценарий будет выполняться под shell Баурна, даже если вы используете Tcsh (или какой-то другой C shell) как свою рабочую оболочку.

Вторая строка представляет из себя комментарий. Комментарии начинаются символом ``#'' и могут продолжаться до конца строки - они игнорируются shell и могут использоваться программистом для пояснений.

Остальные строки сценария - обычные команды в том виде, в каком бы вы их вводили прямо на выполнение. Shell читает каждую строку сценария и выполняет эту строку, как будто вы ввели эту строку в ответ на подсказку shell.

Права доступа важны для сценариев. Если вы создали сценарий, вы должны убедиться, что вы имеете права на его выполнение. Если вы создавали сценарий в редакторе, то он (обычно) не получает автоматически прав на выполнение. Можно использовать команду

```
/home/larry# chmod u+x makebook
```

чтобы дать самому себе разрешение на выполнение shell-сценария makebook.

3.13.2 Перемещение shell и среда

Shell позволяет определять **переменные**, как и большинство языков программирования. Переменная - это порция данных, которой дано имя. (прим. переводчика: В языке shell переменные не определяются (в традиционном смысле), так как все они одного типа - "строкового", речь может идти только об их инициировании: присваивании начальных значений).

ВНИМАНИЕ! Имейте в виду, что Tcsh, также, как и C shell, используют различные механизмы определения переменных, отличающиеся от используемых здесь. Здесь обсуждается shell Баурна. Когда вы присвоите значение переменной (используя оператор ``=''), вы сможете получить это

значение, добавив перед именем переменной символ ``$'', как это показано ниже

```
/home/larry# foo=``hello there''
```

Переменной foo присвоено значение ``hello there''. Теперь вы можете обратиться к этой переменной, добавив перед именем символ ``$''. Команда

```
/home/larry# echo $foo
hello there
/home/larry#
```

дает тот же самый результат, что и

```
/home/larry# echo ``hello there''
hello there
/home/larry#
```

Эти переменные являются внутренними для shell. Это означает, что только shell имеет доступ к этим переменным. Это может быть полезно для сценариев; если вам надо сохранить информацию о имени файла, вы, например, можете поместить его в переменную. Команда set может показать вам перечень всех определенных переменных shell.

Shell позволяет **экспортировать** переменные в среду. **Среда -** это множество переменных, к которым могут иметь доступ все выполняемые команды. Определив однажды переменную внутри shell (прим. переводчика: определить - здесь означает "присвоить значение"), командой export вы можете передать ее среде.

ВНИМАНИЕ! Здесь вновь есть отличие между Bash и Tcsh. При использовании Tcsh используется другой синтаксис для помещения переменных в среду (используется команда setenv). Дополнительную информацию можно найти в Руководстве по Tcsh.

Среда очень важна в системах UNIX. Она позволяет конфигурировать некоторые команды за счет установки переменных, о которых знают команды.

Вот небольшой пример. Переменная среды PAGER используется командой man. Она указывает команду, которая используется в свою очередь командой man для просмотра Руководства на экране. Если вы установите в качестве значения PAGER имя другой команды, то эта команда вместо будет обеспечивать просмотр вместо more (которая применялась по умолчанию).

Присвойте PAGER значение ``cat''. Выдача на экран руководства будет вся разом, а не поэкранно, как это делала команда more.

```
/home/larry# PAGER=cat
```

Теперь экспортируйте PAGER в среду.

```
/home/larry# export PAGER
```

Попробуйте команду man ls. Руководство промелькнет по вашему экрану без (желательных) задержек.

Теперь, если присвоить PAGER значение ``more'', то для выдачи на экран будет использоваться команда more.

```
/home/larry# PAGER=more
```

Обратим внимание на то, что нам не надо заново использовать команду export после изменения значения PAGER. Необходимо только раз экспортировать переменную; любые изменения, которые будут происходить после этого, будут отражаться в среде.

Страницы Руководства для конкретных команд содержат информацию о том, использует ли команда какие-то переменные среды. Например, Руководство по команде man говорит о том, что для определения режима выдачи страницы руководства на экран используется переменная PAGER. Некоторые команды совместно используют переменные среды, например, многие команды используют переменную среды EDITOR для указания используемого редактора.

Переменные среды используются также для сохранения важной информации о процедуре входа. Например переменная HOME содержит имя вашего домашнего каталога.

```
/home/larry/papers# echo $HOME
/home/larry
```

Другая интересная переменная среды - PS1, которая определяет главную подсказки shell. Например,

```
/home/larry# PS1=``Your command, please:  ''
Your command, please:
```

Для переустановки подсказки обратно в нормальное состояние (когда она показывает текущий рабочий каталог, после которого следует значек ``#"), выполните следующее:

```
Your command, please:  PS1=``\w# "
/home/larry#
```

В Руководстве bash есть подробное описание синтаксиса, используемого при установке подсказки.

3.13.2.1 Переменная среды PATH

Когда вы используете команду ls, как shell находит соответствующий выполняемый файл (программу) для ls? На самом деле в большинстве систем ls находится в /bin/ls. shell использует переменную среды PATH ("ТРОПА") для указания возможного местоположения выполняемых файлов соответствующих команд.

Например, ваша переменнаяPATH может иметь значение

```
/bin:/usr/bin:/usr/local/bin:.
```

Это список каталогов (в которых shell будет искать команду), отделяемых друг от друга двоеточием ``:". Когда вы используете команду ls, shell прежде всего проверяет наличие /bin/ls, затем /usr/bin/ls и т.д.

Обратите внимание на то, что переменная PATH не помогает находить обычные файлы. Например, если вы используете команду

```
/home/larry# cp foo bar
```

shell не использует PATH для нахождения местопребывания файлов foo и bar - предполагается, что эти имена однозначно определяют место. shell использует PATH только для нахождения команды cp.

Это экономит вам массу времени; это означает, что вы не обязаны помнить, где находятся выполняемые файлы команд. Во многих системах выполняемые файлы разбросаны во многих местах, таких как /usr/bin, /bin или /usr/local/bin. Вместо того, чтобы писать полное имя команды (вроде /usr/bin/cp), вы просто указываете в PATH перечень каталогов, которые бы вы хотели, чтобы shell автоматически просматривал.

Обратите внимание, что PATH содержит ``.", что означает "текущий рабочий каталог". Это позволяет вам создавать shell-сценарии или программы и выполнять их как команды из текущего каталога, без необходимости

указывать это прямо (как в случае ./makebook). Если каталог не указан в PATH, то shell не будет его просматривать в поиске команд, это касается и текущего каталога.

3.13.3 Shell-Сценарии инициализации

В дополнение к shell-сценариям, которые создаете вы, существует множество сценариев, которые использует сам shell для своих целей. Наиболее важными среди них являются **сценарии инициализации**, которые автоматически выполняются shell при вашем входе в систему.

Сценарии инициализации сами по себе - это обычные сценарии, как это описывалось выше. Но они очень полезны при установке вышей среды путем автоматического выполнения набора команд при вашем входе в систему. Например, если вы всегда используете команду mail для проверки своей почты в момент входа в систему, вы можете поместить эту команду в свой сценарий инициализации и она будет выполнена автоматически.

Как Bash, так и Tcsh делают различие между начальным shell (вызываемым при входе в систему) и прочими вызовами shell. Начальный shell вызывается в момент входа пользователя в систему; часто это единственный экземпляр shell, который вы используете. Но если вы вызываете shell из другой программы, такой какvi, вы тем самым запускаете новый (экземпляр) shell. Кроме того, когда вы запускаете на выполнение shell-сценарии, вы автоматически инициируете новый экземпляр shell.

Файлы инициализации, используемые в Bash: /etc/profile (устанавливается системным администратором, выполняется всеми экземплярами начальных пользовательских bash, вызванными при входе пользователей в систему), $HOME/.bash_profile (выполняется при входе пользователя) и $HOME/.bashrc (выполняемый всеми прочими не начальными экземплярами bash). Если .bash_profile отсутствует, вместо него используется .profile.

Tcsh использует следующие сценарии инициализации: /etc/csh.login (выполняется всеми пользовательскими tcsh в момент входа в систему), $HOME/.tcshrc (выполняется во время входа в систему и всеми новыми экземплярами tcsh) и $HOME/.login (выполняется во время входа после .tcshrc). Если .tcshrc отсутствует, вместо него используется .cshrc.

Для того, чтобы лучше понять функции этих файлов, вам следует больше узнать о shell. Программирование на shell сложный вопрос, далеко выходящий за рамки этой книги. Дополнительную информацию можно получить из Руководства на bash и tcsh.