САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет
Кафедра параллельных алгоритмов

Н. А. Лебединская, Д. М. Лебединский

Операционная система Linux

САНКТ-ПЕТЕРБУРГ
2010

С О Д Е Р Ж А Н И Е

ВВЕДЕНИЕ

Данный набор лабораторных работ представляет собой краткое введение в операционную систему Linux для пользователя. Он предназначен для проведения практических занятий со студентами, изучающими операционную систему Linux в рамках курса «Операционные системы, среды и оболочки», читаемого одним из авторов на факультете международных отношений Санкт-Петербургского государственного университета.

Лабораторные работы разделены согласно обсуждаемым в них темам на 4 главы. Каждая лабораторная работа содержит изложение теоретического материала, набор примеров, которые могут быть введены и выполнены, а также определенное количество заданий для самостоятельной работы.

Лабораторные работы ориентированы на использование дистрибутива Fedora операционной системы Linux (версия 10), однако можно использовать и другие дистрибутивы, содержащие используемые в работах пакеты. Все необходимое для работ программное обеспечение является свободно распространяемым и может быть получено бесплатно по адресу http://fedoraproject.org/

Имеется также список литературы, содержащий основные учебники по операционной системе Linux.

1. Работа в командном интерпретаторе BASH

1.1. Ввод команд в командной строке

Важной частью операционной системы Linux, как и любой операционной системы из семейства UNIX-подобных систем, является командный интерпретатор. Это программа, позволяющая вводить с клавиатуры команды и выполнять их. Она имеет текстовый интерфейс, однако идея текстовой команды лежит в основе запуска любой программы, в том числе и в системах с оконным интерфейсом. С любой кнопкой на любой панели или любым пунктом любого меню оконного интерфейса, активизация которых приводит к запуску некоторой программы, связана текстовая команда, запускающая указанную программу. Кроме того, возможности командного интерпретатора, оснащенного необходимым набором команд, например для работы с файлами, далеко превосходят возможности инструментов с оконным интерфейсом для той же цели, например, файловых менеджеров.

В Linux существует несколько командных интерпретаторов. Они похожи друг на друга, однако отличаются в деталях. По умолчанию дистрибутив Fedora использует один из наиболее мощных командных интерпретаторов, называющийся BASH.

Находясь в графическом режиме, для непосредственного ввода команд нужно запустить приложение терминала. Обычно это делается путем выбора пункта «Терминал» в подменю «Системные» меню «Приложения», кнопка вызова которого располагается в левом верхнем углу (первая кнопка верхней панели; на ней находится логотип Fedora и текст «Приложения»). Вообще говоря, это верно для дистрибутива Fedora; другие дистрибутивы также, как правило, имеют в своем составе приложение терминала, но расположение и состав меню и кнопок могут отличаться от описанного здесь.

Если запустить приложение терминала, то появляется окно терминала, в которое выводится приглашение командного интерпретатора, означающее, что он запущен и ожидает ввод очередной команды. Как правило, приглашение состоит из имени компьютера, имени пользователя и названия текущей папки, т. е. той, в которой ищутся файлы, для которых не указан путь.

Пример.

Запустив приложение терминала, введите date в ответ на приглашение командного интерпретатора. Что делает эта команда? Что будет, если ввести Date?

Командный интерпретатор предоставляет весьма богатый выбор возможностей для редактирования командной строки. Разумеется, работают клавиши со стрелками для перемещения курсора по тексту команды и клавиша Backspace для удаления символа перед курсором. Удалить символ под курсором можно комбинацией клавиш C-D (здесь и далее «C-» означает клавишу Ctrl). Однако, если нажать C-D на пустой строке, приложение терминала закрывается (по традиции, C-D использовалось для завершения сеанса работы в операционной системе UNIX). Также работают клавиши Home и End. Отменить набранную, но еще не запущенную команду можно, нажав C-C.

Командный интерпретатор BASH запоминает историю, т. е. последовательность ввода команд. Можно извлечь предыдущую команду, нажав стрелку вверх. Нажав стрелку вверх несколько раз, можно извлечь команду, которая была введена пред-, предпредпоследней и т. д. Если в командной строке отображается такая команда, нажатие стрелки вниз меняет эту команду на ту, которая была введена после нее. Имеется также весьма удобная возможность декрементного поиска по истории команд. Для этого нужно нажать C-R, и на экране появляется специальное приглашение; можно набирать символы, и по мере их набора будет отображаться последняя команда, содержащая в своем тексте набранную строку. Если нажимать C-R далее, будет отображаться предыдущая команда, содержащая набранную строку, и т. д. Как только нужная команда найдена, можно нажать Enter для ее запуска или Esc, чтобы извлечь найденную команду из истории в командную строку для редактирования. Команда, извлеченная из истории, может быть отредактирована обычным образом и запущена на выполнение.

Пример.

Введите и запустите последовательно команды

```
> file.txt
> file2.txt
ls
ls -l
ls -a
```

Что делают эти команды? Какая разница между действиями последних трех команд? Извлеките из истории предыдущую команду и вторую с конца, запустите ее. Извлеките из истории последнюю команду, содержащую символ >. Отредактируйте ее, заменив 2 на 3, и запустите. Как убедиться в том, что эта команда сработала правильно?

Следующая, очень полезная возможность командного интерпретатора — автодополнение. Например, если Вы знаете, как начинается ваша команда, Вы можете ввести в командную строку часть имени команды, и нажать клавишу Tab. Если существует единственная команда, имеющая указанное Вами начало, введенный текст будет автоматически дополнен до имени этой команды. Если же таких команд несколько, компьютер подаст звуковой сигнал. Повторное нажатие Tab приведет к выводу на экран имен всех таких команд, если их не очень много. Если же их много, будет выведено их количество, и компьютер спросит, хотите ли Вы видеть их все — ответ «y» приведет к выводу их имен на экран.

Пример.

Наберите la в командной строке и нажмите Tab. Сколько всего имеется команд, имена которых начинаются с la? А сколько есть команд, имена которых начинаются с l? Из всех команд, начинающихся с la, выберите команду с самым длинным именем и введите ту начальную часть имени, которой достаточно для однозначного определения имени команды. Нажмите Tab и убедитесь, что автодополнение работает.

Ускорить набор команд можно также при помощи мыши. Если выделить мышью часть текста, и затем нажать среднюю кнопку мыши, выделенный текст вставляется в позицию курсора. Одно из отличий оконной системы Gnome, используемой в Linux, от MS Windows состоит в том, что в большинстве случаев нет необходимости явно использовать операцию copy; при выделении текста мышью эта операция выполняется автоматически. Также, средняя кнопка мыши обычно означает вставку выделенного текста. При этом текст может вставляться в позицию текстового курсора, как, например, в приложении терминала, или в позицию курсора мыши, как во многих текстовых редакторах.

Пример.

Запустите команду ls. Найдите в терминальном окне имя файла file2.txt и выделите его мышью. Наберите в командной строке текст rm и затем пробел, после чего скопируйте туда выделенное имя файла, нажав среднюю кнопку мыши. Запустите полученную команду, и затем снова команду ls. Что делает команда rm?

Всякая вводимая с клавиатуры команда имеет определенную структуру. Прежде всего скажем, что введенная с клавиатуры команда рассматривается командным интерпретатором как набор отдельных слов. Чаще всего слова отделяются друг от друга пробелами, однако бывают случаи, когда пробелы не являются разделителями слов — такие случаи будут рассмотрены позже. Например, команда ls -l file.txt состоит из трех слов: «ls», «-l» и «file.txt». Первое слово команды чаще всего является ее именем (исключения возможны, если перед именем идут присваивания — эта ситуация будет обсуждаться в разделе про переменные). В данном случае именем команды является слово «ls». Все остальные слова команды называются ее параметрами. В данном случае параметра два: «-l» и «file.txt».

Параметры команды предназначены для того, чтобы управлять ее поведением. Среди всех возможных параметров выделяется некий специальный тип параметров, называемый опциями. В Windows опции начинаются обычно с символа «/», тогда как в Linux — с «-». Обычно опции содержат один символ (кроме «-») и либо указывают на выбор одного из нескольких возможных вариантов поведения команды, либо устанавливают значение какого-либо параметра команды. В последнем случае у опции обычно имеется аргумент; в зависимости от конкретной команды и конкретной опции текст нужного значения этого аргумента может писаться слитно с текстом самой опции или раздельно в следующем аргументе (параметре) команды. Например, у компиляторов семейства gcc имеется опция без параметров -c, которая означает «остановиться на этапе компиляции и не производить компоновку». Также имеется опция -W с параметром, пишущимся слитно, означающая выбор ситуаций, в которых должно выдаваться предупреждение (warning). Например, опция -Wall означает, что предупреждение должно выдаваться во всех подозрительных случаях. Имеется и опция -o с параметром, пишущимся отдельно, который задает имя выходного файла. Так, если в командной строке присутствует пара параметров (подряд) «-o program», то выходной файл будет называться program.

В большинстве случаев, хотя и не всегда, односимвольные опции без параметров можно объединять для экономии числа набираемых символов. Например, команда «ls -al» делает то же самое, что и «ls -a -l».

Кроме того, федерация бесплатного программного обеспечения (источник большого количества программ под Linux) ввела длинные аналоги односимвольных опций. Такие аналоги начинаются с двух минусов, за которыми следует слово, описывающее действие опции, или несколько слов, разделенных минусами (разумеется, без пробелов в тексте опции). Такие опции почти никогда не пишутся слитно со своими параметрами.

Кроме собственно текста команды (имени и аргументов), командная строка может содержать некоторые специальные конструкции, также управляющие поведением программы. К ним относятся конструкции переадресации ввода-вывода и организации программных конвейеров и управляющие конструкции.

Например, можно продолжить команду на следующую строку, поставив в самом ее конце символ «\». Таким образом одна команда может занимать несколько строк, если в конце каждой из них, кроме последней, стоит символ «\». При этом, для последующих строк команды (кроме первой) выдается специальное приглашение, обычно оно состоит просто из символа «>».

Пример.

В ответ на приглашение командного интерпретатора введите строку «echo Hello, \». В ответ на специальное приглашение введите строку «world!». Что делает команда echo?

Оказывается, можно не только размещать одну команду на нескольких строках, но и наоборот, размещать несколько команд в одной командной строке. Для этого команды нужно разделять символом «;».

Пример.

Наберите две команды «echo Hello,; echo world!» и посмотрите, что они делают.

Иногда необходимо отменять специальный смысл некоторых специальных символов. Например, в том случае, когда нужно вывести на экран символ «;» при помощи команды echo или сделать так, чтобы один параметр команды, скажем, имя файла, содержал в себе символы пробела. Для этого есть несколько способов. Если нужно отменить

специальный смысл небольшого числа символов, перед каждым из них нужно поставить символ «\». Это работает в том числе и для пробелов, т. е. конструкция из «\» и следующего за ним пробела воспринимается как символ пробела, являющийся частью аргумента команды, а не разделяющий аргументы. Конструкция «\\» воспринимается как один символ «\».

Пример.

Введите команду «echo \\\\\\» и запустите ее. Как объяснить ее результат? Введите и запустите команду «> a\ file.txt». Как убедиться в том, что эта команда работает правильно?

Если нужно лишить специального смысла большое количество символов, предварять каждый из них символом «\» может быть утомительно. В этом случае можно заключить весь аргумент команды, содержащий специальные символы, в двойные или одинарные кавычки. Разница между ними только в том, какой набор специальных символов будет лишаться своего специального смысла. Одинарные кавычки в этом смысле гораздо мощнее, т. е. внутри одинарных кавычек теряют свой специальный смысл гораздо больше специальных символов.

Пример.

Вторую команду из предыдущего примера можно записать и так: «> "a file.txt"». Запустите ее и убедитесь, что она работает.

1.2. Получение справочной информации по командам и не только

Одной из сильных сторон операционной системы UNIX и многочисленных ее клонов является возможность получения информации о ней самой прямо в процессе работы. Для этого существует набор команд.

Первая и древнейшая команда для этой цели называется man. В простейшем случае она вызывается следующим образом: man слово. В качестве слова выступает название той сущности, справку по которой Вы хотите получить. Это может быть команда, процедура языка C, имя конфигурационного или специального, т. е. представляющего некоторое устройство, файла и т. д.

Если в системе имеется запрошенная Вами страница руководства, она будет выведена на экран. Обычно для этого требуется наличие форматирующей тексты программы из семейства troff и фильтра less. Если все это у Вас установлено, на экране появляется начало файла из справочного руководства, посвященного запрошенной Вами теме. Листать этот файл можно вперед и назад стрелками вверх/вниз (построчно) и клавишами PgUp/PgDn (постранично). Также реализована возможность поиска интересующей Вас строки. Для этого нужно нажать /, затем ввести искомую строку и нажать Enter. Если эта строка встречается несколько раз, перейти к следующему вхождению можно, нажав n. Вернуться в самое начало файла можно, нажав Home, перейти в конец — End. Для выхода из просмотра справочной информации достаточно нажать q.

Страницы руководства, доступные по команде man, представляют собой более-менее обычные текстовые файлы, однако существуют соглашения о структуре этих файлов, благодаря чему поиск необходимой информации упрощается. В частности, почти во всех таких файлах присутствуют определенные разделы. Например, все такие файлы начинаются с синтаксиса обсуждаемой в них команды, а также почти все имеют разделы FILES (файлы), SEE ALSO (смотри также), BUGS (ошибки), AUTHOR (автор).

Поскольку команда man может выдавать справку не только по командам, весь набор справочных файлов, которые называются страницами справки (хотя отдельные файлы могут быть очень длинными и содержать сотни страниц в обычном понимании этого слова), делится на несколько разделов. Разделы обычно обозначаются цифрой, хотя в последнее время появились разделы, обозначаемые буквами. Например, раздел 1 содержит информацию о командах, 2 — о системных вызовах, т. е. процедурах, реализованных в операционной системе и позволяющих, например, работать с файлами и папками, 3 — о функциях языка C, 4 — о специальных файлах, 5 — о форматах конфигурационных файлов. Если при вызове команды man раздел не указан, информация ищется во всех разделах и выдается первый найденный файл. Однако бывает так, что одно и то же слово может означать разные вещи, и тогда указывать раздел необходимо. Например, если Вы хотите получить справку по функции printf и даете команду man printf, в результате Вы получите справку по команде printf. Исправить ситуацию можно, явно указав, что нас интересует третий раздел руководства; команда будет выглядеть так: man -S 3 printf

Пример.

Выясните, что делает команда ls. Как используется и что означает опция -d этой команды? Выполните команду mkdir dir1, затем команду > dir1/afile.txt. Что будет делать команда ls dir1? А команда ls -d dir1? А команда ls -ld dir1? Что делает команда rm? А rmdir? А rm -r?

Гораздо более полную информацию о командах и не только можно получить по команде info. Запустив ее, Вы попадаете на главную страницу весьма обширной коллекции гипертекстовых (т. е. текстовых, но с возможностью перемещаться по ссылкам) документов, описывающих почти все команды, реализованные Free Software Foundation под аббревиатурой GNU. Эта главная страница начинается с описания важнейших команд, с помощью которых Вы можете просматривать документацию или перемещаться по ссылкам. Ссылки бывают двух видов: меню и перекрестные ссылки; смысл у них похожий, но команды перемещения различаются. После краткого описания команд главная страница содержит огромное меню, включающее в себя все темы, доступные в этом руководстве. Если Вы знаете, как называется интересующая Вас тема, Вы можете указать ее при вызове команды info также, как в команде man: info тема. Тогда Вы сразу попадете на корневую страницу указанной темы.

Пример.

Запустите info. Как выйти из info? Как получить полный список команд, доступных в info? Что делает команда bison?

Кроме man и info, в Linux имеется еще ряд команд информационно-справочного характера. Одна из важнейших команд такого рода называется file. Она позволяет узнать тип файла, исходя из его содержимого, а не расширения имени, как обычно определяется тип файла в Windows. Эта команда бывает полезна, если по каким-то причинам имя файла оказалось повреждено или утеряно, если расширение его имени Вам неизвестно или неинформативно (например, .bin или .dat). Также, в отличие от Windows, исполняемые файлы (программы) в Linux обычно не имеют вообще никакого расширения. Используется эта команда так: file имя_файла.

Пример.

Что представляет собой файл /bin/cp? А файл /lib/libc-2.7.so?

Также полезной является команда which. Она выдает полный путь для исполняемого файла указанной команды и вызывается так: which команда.

Пример.

Где находится исполняемый файл команды man? А команды rm? А команды ls?

1.3. Работа с файлами

Как и в операционных системах семейства Windows, в Linux информация в компьютере хранится в виде набора файлов. Однако организация этого набора несколько отличается. В Linux (как и в любом другом клоне UNIX) нет деления всего пространства для размещения файлов на отдельные диски, каждый из которых адресуется отдельно от остальных с использованием своей буквы. Вместо этого в Linux поддерживается единое пространство, называемое корневой файловой системой, к которому присоединяются («монтируются» в терминологии Linux) вместо некоторых конкретных папок диски и прочие вместилища файлов. Например, пусть в этом пространстве имеется пустая папка afolder. Если я хочу использовать какой-либо диск, я могу «смонтировать», т. е. подключить его к этой папке. После этого, любое обращение к этой папке будет означать доступ к корневой папке данного диска. В ней могут быть подпапки, и они будут видны пользователю как подпапки папки afolder. После того, как работа с данным диском завершена, его нужно «размонтировать», после чего папка afolder опять окажется пустой. Монтировать диски можно на любые пустые папки.

Разделы жестких дисков обычно монтируются при загрузке Linux и размонтируются при подготовке к перезагрузке или выключению питания, так что для них подобный стиль работы особых проблем не вызывает. Однако для съемных дисков проблемы могут возникать, поскольку при извлечении носителя без размонтирования файловая система на нем может быть серьезно попорчена. Предпринимаются усилия по исправлению ситуации. Во-первых, это автоматические монтировщики, которые монтируют диски при установке носителя или обращении к нему и размонтируют, если диск определенное время не используется. Для дискет можно использовать специальный пакет mtools, который содержит аналоги команд MS-DOS и позволяет работать с дискетами, не монтируя их.

При указании конкретной папки во всем пространстве для хранения файлов в Linux используется набор правил, похожий на принятый в Windows. В Linux корневая папка всей файловой системы обозначается символом «/». Далее следуют имена папок, разделенные тем же символом, причем каждая следующая папка содержится в предыдущей.

Файловая система Linux имеет определенную структуру, которой обычно придерживаются производители дистрибутивов и на которую рассчитывают авторы программ под Linux. Рассмотрим эту структуру кратко, указав несколько стандартных папок и смысл их содержимого.

Папка /bin содержит исполняемые файлы наиболее важных команд; исполняемые файлы всех остальных команд помещаются (по мере убывания важности) в папки /usr/bin и /usr/local/bin.

Папка /boot содержит файлы, необходимые для загрузки Linux — ядро операционной системы (т. е. собственно ту программу, которая обеспечивает работу операционной системы Linux) и некоторые вспомогательные файлы. Если используется загрузчик grub, то его файлы помещаются в папку /boot/grub. В этой папке пользователя интересует в основном конфигурационный файл этого загрузчика, содержащий список установленных на компьютере операционных систем с указанием того, как они должны быть загружаемы.

Папка /dev содержит файлы, связанные с устройствами компьютера. Каждое устройство представлено здесь одним или несколькими файлами, запись и чтение которых означает общение с данным устройством. В частности, здесь имеются файлы, соответствующие жестким дискам и их разделам. Достаточно новые дистрибутивы подключают жесткие, оптические диски и другие накопители, такие как флэш-память, через эмуляцию SCSI (т. е. даже не SCSI-устройства, например, IDE или SATA диски, выглядят как SCSI). Это означает, что первому диску будет соответствовать файл /dev/sda, второму — /dev/sdb и т. д. Однако оптические дисководы имеют собственное именование /dev/scd0, /dev/scd1 и т. д. Имена файлов, соответствующих разделам на дисках, получаются из имени файла, соответствующего всему диску, приписыванием номера раздела. Например, /dev/sda2 — второй раздел первого диска. Разделы нумеруются с 1, и вспомогательные первичные разделы, содержащие несколько логических разделов, также имеют номера. Первичный раздел — раздел, информация о котором присутствует в таблице разделов на диске, обычно в главной загрузочной записи. Поскольку такая таблица разделов может описывать не более четырех разделов, а на больших дисках их требуется гораздо больше, некоторые первичные разделы затем подразделяются на несколько так называемых логических разделов.

Папка /etc содержит обычно конфигурационные файлы, относящиеся к системе в целом.

Папка /home содержит домашние папки пользователей. Каждый пользователь имеет личную папку, имя которой

обычно совпадает с регистрационным именем пользователя, в которой он является полноправным хозяином. Домашняя папка содержит личные настройки пользователя и его рабочие файлы.

Папка /lib содержит наиболее важные системные библиотеки; остальные библиотеки помещаются (по мере убывания важности) в папки /usr/lib и usr/local/lib.

Папка /media содержит папки, обычно создаваемые автоматическим монтировщиком, куда подключаются не упомянутые в файле fstab разделы жестких дисков, flash-накопители, оптические диски и т. д.

К папке /proc подключается обычно некоторая специальная файловая система, не имеющая своего диска и целиком содержащаяся в памяти; содержимое ее файлов и папок определяет работу ядра операционной системы, позволяя получать информацию об этой работе и устанавливать ее параметры.

Папка /root является домашней папкой суперпользователя и хранит его настройки.

Папка /usr содержит ряд других папок для разных целей. Часть из них была уже упомянута. Из остальных упомянем папку /usr/share/doc, содержащую документацию по программам, установленным в системе.

Когда запускается командный интерпретатор, пользователь попадает обычно в свою домашнюю папку, которая становится текущей. Текущая папка используется для поиска файлов, у которых не указан путь. Обычно имя текущей папки отображается как часть приглашения командного интерпретатора. Посмотреть полный путь от корневой папки корневой файловой системы до текущей папки можно при помощи команды pwd. Изменить при необходимости текущую папку можно командой cd. Последняя команда имеет два варианта работы: если ее запустить без параметров, текущей становится домашняя папка работающего в данный момент пользователя; если указать параметр, он определяет ту папку, которая станет текущей. При этом, как и в любой другой команде, путь можно указывать тремя способами: абсолютный путь (начинается с корневой папки корневой файловой системы, первым символом такого пути является «/», затем идет имя подпапки корневой папки корневой файловой системы и т. д.); относительный путь (начинается с текущей папки, сначала идет имя ее подпапки, затем «/», имя следующей подпапки, и т. д.); и, наконец, путь, начинающийся с домашней папки (первым символом такого пути является знак «~», за которым следует регистрационное имя пользователя, затем «/», имя подпапки соответствующего домашнего директория, и т. д.; если путь должен начинаться с домашней папки работающего в данный момент пользователя, его регистрационное имя можно не писать).

Кроме имен папок, в пути можно использовать специальную конструкцию — две точки подряд. Эта конструкция означает перемещение по папкам на один уровень вверх. Обычно используется в относительных путях.

Пример.

Определите полный путь своей домашней папки, запустив команду pwd. Измените текущую папку на /usr/local/lib и убедитесь, что текущей папкой стала именно она. Поднимитесь на два уровня вверх одной командой. Вернитесь в свою домашнюю папку. Попробуйте перейти в домашнюю папку пользователя root.

Для работы с файлами в Linux используются команды cp (копирование файлов), mv (перемещение файлов), rm (удаление файлов). Команды cp и mv могут иметь сколько угодно параметров и работают так: если последний аргумент — имя папки, файлы, имена которых указаны в предыдущих параметрах, копируются (cp) или перемещаются (mv) в указанную папку. Иначе может быть всего два аргумента, и файл, указанный первым из них, копируется или переименовывается в файл, указанный вторым параметром.

Команда rm удаляет файлы. Параметров также может быть сколько угодно, и все указанные ими файлы удаляются.

Еще одной полезной командой является cat. У нее может быть сколько угодно параметров, и она последовательно выводит на экран содержимое всех файлов, указанных параметрами.

Пример.

Создайте пустой файл file.txt, выполнив команду > file.txt . Убедитесь, что файл создан и действительно пуст. Скопируйте этот файл в файл file2.txt . Убедитесь в наличии обоих файлов. Переименуйте file.txt в file.html . Удалите file2.txt и убедитесь, что все сделано правильно.

Иногда требуется сохранить вывод той или иной программы в файле. Для этого в BASH предусмотрены конструкции, называемые переадресацией вывода. Чтобы просто записать вывод той или иной команды в файл, к тексту команды достаточно приписать текст > имя_файла . Например, чтобы сохранить список файлов из текущей папки в файле list.txt , достаточно выполнить команду ls > list.txt . При этом предыдущее содержимое файла теряется, т. е. еще до начала работы команды файл усекается до нулевой длины. Если же нужно дописать вывод команды в конец уже существующего файла, сохранив его содержимое, нужно воспользоваться конструкцией >>, так что предыдущая команда в этом варианте (дописывающая список файлов из текущей папки в конец файла list.txt) будет выглядеть так: ls >> list.txt .

Сказанное в последнем абзаце можно использовать, например, для того, чтобы объединить содержимое нескольких файлов в один при помощи команды cat. Например, команда, которая записывает подряд содержимое файлов file.txt и file1.txt в файл file2.txt , будет выглядеть так: cat file.txt file1.txt > file2.txt . Однако, команда cat file.txt file1.txt > file.txt будет делать совсем не то, что предполагалось. Вместо того, чтобы приписывать содержимое file1.txt к файлу file.txt , она просто скопирует file1.txt в file.txt , поскольку сначала (еще до начала работы cat) file.txt будет сброшен до нулевой длины. Правильная команда, приписывающая file1.txt к file.txt , выглядит так: cat file1.txt >> file.txt .

Пример.

Запишите в файл today.txt текущую дату. Проверьте содержимое этого файла. Допишите в конец этого файла вывод команды who и выдайте новое содержимое этого файла на экран. Что делает команда who?

Вообще говоря, далеко не всякий вывод любой команды можно перенаправить в файл указанным выше способом. Дело в том, что возможность такого перенаправления вывода определяется тем, как этот вывод осуществлялся в запускаемой программе. Например, в языке C имеются два стандартных потока для вывода информации: stdout

и stderr. Для того, чтобы их различать, второй из них называется стандартным потоком для вывода сообщений об ошибках. Описанный выше способ перенаправляет в указанный файл только тот вывод, который был в программе послан в поток stdout. Для перенаправления в файл того вывода, который посылался в stderr, используются следующие конструкции: 2> имя_файла для перезаписи указанного файла и 2>> имя_файла для дописывания вывода программы в конец.

Пример.

Выполните команду cp xxx xxx1 . Попробуйте перенаправить вывод этой команды в файл report следующим образом: cp xxx xxx1 > report . Объясните полученный результат. Как нужно перенаправить вывод этой команды в файл report? Сделайте это и выдайте на экран содержимое файла report, чтобы убедиться в правильности работы BASH.

Аналогично можно перенаправить и ввод информации в программу, чтобы вводимая программой информация бралась не с клавиатуры, а из файла или из самой командной строки. Для этого существуют три конструкции.

Первая конструкция выглядит как < имя_файла . При ее использовании все выглядит так, как если бы содержимое указанного файла было бы набрано на клавиатуре.

Вторая конструкция выглядит как << конец . В этом случае пользователь набирает вводимые данные, и признаком конца ввода служит строка, содержащая только тот текст, который был обозначен словом конец. Например, если ввод выглядел как

```
cat << special
apple
orange
pear
special
```

то на вход команды cat поступит текст

```
apple
orange
pear
```

На первый взгляд, разница между таким вводом и обычным вводом с клавиатуры несущественна. Однако BASH умеет выполнять команды, записанные в файле. И тогда разница становится существенной, поскольку без переадресации ввода запущенная даже из файла команда будет брать данные с клавиатуры, а при использовании такой переадресации дополнительные строки, содержащие данные для программы и признак конца, будут браться из того же файла, в котором был записан текст самой команды.

Наконец, последний вариант выглядит как <<< данные . В этом случае данные, подаваемые на вход программы, содержатся непосредственно в командной строке.

Команда cat без параметров читает свой стандартный ввод и переписывает его в стандартный вывод. Такое поведение команды cat может быть использовано для создания небольших текстовых файлов с определенным содержимым (как?). Чтобы закончить ввод текста при помощи cat, нужно нажать C-D на пустой строке.

Пример.

Создайте текстовый файл file.txt с данными

```
first line
second line
third line
```

Выдайте содержимое этого файла на экран. Что будет делать команда cat < file.txt ? Запустите команду cat << last-line . Чем ее поведение отличается от поведения просто команды cat ? Наконец, что будет делать команда cat <<< first-line ? Как надо изменить предыдущую команду, чтобы вместо «-» между first и line выдавался пробел?

Возможность переадресации ввода и вывода команд наводит на мысль о создании программного конвейера, когда вывод предыдущей команды используется как ввод следующей. Оказывается, такая возможность также предусмотрена и для такой организации совместной работы команд используется символ «|» между командами. Например, команда wc подсчитывает количество символов перевода строки, слов и байтов в тексте, поданном ей на стандартный ввод. Для того, чтобы воспользоваться этим фактом для подсчета числа файлов в текущей папке, достаточно объединить ls и wc в один программный конвейер: ls | wc . Если мы хотим получить только число файлов в текущей папке, подавив прочий вывод wc, нужно использовать ее опцию -l, вот так: ls | wc -l .

Пример.

Выдайте на экран число пользователей, в данный момент работающих на компьютере. Какие команды для этого надо объединить в конвейер?

В операционной системе Linux имеется достаточно много команд, специально разработанных для работы в составе конвейеров. Подробно важнейшие из них будут рассмотрены в главе «Текстовые базы данных». Сейчас упомянем лишь некоторые:

- less — команда, позволяющая удобно просматривать вывод предыдущей команды конвейера, даже если он не умещается на одной странице. Клавиши, управляющие таким просмотром, обсуждались в предыдущем разделе в связи с командой man, поскольку для вывода информации на экран man использует less.

- sort — команда, сортирующая строки своего стандартного ввода и выдающая их в правильном порядке в стандартный вывод.
- grep — команда, выдающая в стандартный вывод те строки своего стандартного ввода, которые содержат текст, соответствующий заданному (как параметр) регулярному выражению.
- sed — потоковый текстовый редактор, применяющий заданную как параметр программу для редактирования к строкам своего стандартного ввода, и выдающий результаты на стандартный вывод.
- awk — тоже программа для преобразования проходящих через нее текстовых файлов. Гораздо мощнее и удобнее в использовании, чем sed, но и значительно медленнее работает.
- head — программа, выделяющая из проходящего через нее файла начало. Может быть выделено указанное число байтов (опции -c число) или строк (-n число).
- tail — программа, выделяющая из проходящего через нее файла конец. Может быть выделено указанное число байтов (опции -c число) или строк (-n число).

Большинство команд из вышеперечисленного списка могут встречаться не только в середине или в конце, но и в начале конвейера. Для этого после всех опций команды можно указать набор имен входных файлов. Так, команда head -n 100 file.txt выдаст на экран первые 100 строк файла file.txt .

Пример.

Выдайте на экран последние 50 строк файла справки по команде mv. Запишите весь текст справки по mv в файл mv.txt и выдайте на экран его строки с 100-й по 200-ю.

Есть еще одна команда, позволяющая разрезать файлы на части. Она называется split. Ее параметрами являются имя разрезаемого файла и префикс имени для результирующих файлов, т. е. имена результирующих файлов начинаются с указанного префикса, к которому приписываются окончания aa, ab, ..., az, ba, ..., zz (по умолчанию длина окончаний 2, но опция -a число позволяет задать это число самому). Кроме того, размер всех частей, кроме последней, определяется опциями (до параметров): -b число определяет размер в байтах, а -l число определяет размер в строках (задавать и то и другое одновременно нельзя).

В Linux имеются также две команды для сравнения файлов. Для текстовых файлов предусмотрена команда diff (ее параметрами являются имена сравниваемых файлов). Для совпадающих файлов она не выдает ничего, а для различающихся — какие преобразования претерпели строки первого файла, чтобы получился второй, т. е. какие строки из первого файла надо убрать, какие изменить и какие добавить. Эта команда достаточно умна для того, чтобы заметить, например, что если первый файл отличается от второго добавлением одной строки, скажем, в начале, то разница состоит именно в этом, а не во всех строках, поскольку из-за сдвига строк все строки первого файла отличаются от строк второго файла с теми же номерами. Для сравнения двоичных файлов предусмотрена команда cmp; смысл ее параметров такой же, как у diff. Для совпадающих файлов она, как и diff, не выдает ничего, но для различающихся выдает только номер байта и строки первого отличия. Если мы хотим получить все отличия, нужно использовать опцию -l; однако, в этом случае выдаются адреса и значения несовпадающих байтов, и эта команда, в отличие от diff, сравнивает только байты с одинаковыми номерами и неспособна заметить, например, что некоторый байт был вставлен или удален.

Пример.

Разрежьте файл mv.txt из предыдущего примера на части по 1024 байта. Сколько получилось частей? Соберите их вместе в файл mv1.txt и сравните с оригиналом. Создайте текстовый файл file1.txt с данными

```
first line
second line
third line
fourth line
fifth line
sixth line
```

Запишите в файл file2.txt те же данные, но без первой строки. Сравните file1.txt и file2.txt при помощи diff, cmp и cmp -l .

Для правильной работы с файлами и для понимания причин многих ошибок полезно также ознакомиться с традиционной моделью безопасности операционной системы UNIX, почти без изменений сохраняющейся в ее многочисленных клонах, одним из которых является Linux. Опишем здесь вкратце эту модель.

Каждый пользователь компьютера, на котором установлена операционная система UNIX, имеет свое регистрационное имя, под которым он известен системе. Кроме того, имеется ряд групп пользователей, каждая из которых также имеет зарегистрированное в системе название. Каждый файл (а папки тоже рассматриваются как разновидность файлов) имеет хозяина — пользователя, которому он принадлежит (обычно это создавший его пользователь), а также группу пользователей, к которой он относится. Вместе с файлом хранятся также разрешения на его использование, причем первая часть разрешений касается хозяина файла, вторая — пользователей из группы и третья — всех остальных пользователей компьютера. В каждой из этих частей три базовых флага, т. е. логических значения: первый показывает, можно ли читать файл, второй — писать в файл, и третий — выполнять файл как программу (для папок последний флаг означает, можно ли сделать эту папку или какую-либо ее подпапку текущей).

Когда пользователь запускает на выполнение какую-либо команду (программу), в операционной системе UNIX возникает новый процесс, т. е. объект, хранящий всю внутреннюю информацию, связанную с выполнением конкретной программы. Каждый процесс имеет пользователя, от имени которого он запущен, и также группу. Кроме этого, каждый процесс имеет пользователя, правами которого он обладает, и также группу (последние пользователь и группа называются эффективными, поскольку именно они определяют, разрешается ли процессу доступ к конкретному файлу или нет). Совпадают ли эти пары (пользователь, группа) или нет, зависит от специальных флагов файла, в котором содержалась запущенная программа. Если у этой программы специальный флаг пользователя установлен, образованный ею процесс будет обладать правами хозяина этой программы; если же он сброшен, процесс будет обладать правами того пользователя, который его запустил, т. е. эффективный пользователь этого процесса будет тот же, что и обычный. То же верно и в отношении группы.

Кроме (большого числа) обычных пользователей, имеется один суперпользователь (в терминологии Windows он называется администратор), которому можно все. Обычно его регистрационное имя — root, однако редко когда суперпользователь входит в систему под своим регистрационным именем (по соображениям безопасности). Чаще всего у него есть еще другие регистрационные имена с обычными правами, а когда ему нужно получить права администратора, он пользуется командой su, которая спрашивает пароль администратора и, если он введен верно, предоставляет такие права.

Менять права доступа к файлу может только его хозяин. Для этого существует команда chmod. Первый ее параметр означает желаемое изменение прав, второй — имя файла, права доступа к которому нужно изменить. Первый параметр строится по следующим правилам: сначала идут буквы из набора u, g, o, a, означающие пользователей, чьи права меняются (u — владелец файла, g — пользователи из группы, o — все остальные пользователи, a — все пользователи, сокращение для ugo), затем один из символов +, -, = (+ — добавить права, указанные следующими за этим символом буквами, - — отменить указанные права, = — установить в точности указанные права) и наконец буквы, указывающие требуемые права, из набора r, w, x, s (r — право на чтение, w — на запись, x — на выполнение, s — специальные биты, для пользователя или группы, в зависимости от того, чьи права меняются).

Узнать права доступа к файлу, а заодно и регистрационное имя хозяина с группой, можно при помощи опции -l команды ls. Первый столбец вывода такой команды содержит тип файла (наиболее часто используются значения «-» — обычный файл и «d» — папка), следующие 3 столбца — разрешения хозяина (первая позиция: «r» — можно читать, «-» — нельзя читать; вторая позиция: «w» — можно писать, «-» — нельзя писать; третья позиция: «x» — можно выполнять, но специальный флаг сброшен, т. е. программа выполняется с правами запустившего ее, «-» — нельзя выполнять и «s» — можно выполнять и специальный флаг установлен, т. е. программа выполняется с правами хозяина файла, в котором содержится), аналогично следующие 3 столбца — разрешения группы и, наконец, следующие 3 столбца — разрешения всех остальных (здесь буква «s» не используется). Далее в каждой строчке идет некоторое целое число (что оно означает, скажем позже), за ним имя хозяина файла и название соответствующей группы.

Пример.

Создайте пустой файл sample.txt . Выясните, какими правами доступа он обладает. Запишите в этот файл строку first line. Отмените право на чтение для влядельца этого файла. Попробуйте вывести его содержимое на экран. Верните отмененное право. Запишите в этот файл строки

```
#!/bin/bash
cat > result.txt
```

Установите право владельца файла на его исполнение. Запустите этот файл командой ./sample.txt . Введите строки

```
apple
pear
grape
```

Убедитесь, что введенные Вами строки попали в правильный файл (в какой?). Сотрите файл sample.txt и создайте его заново, записав туда строки

```
#!/bin/bash
cat << end-of-input > result.txt
table
chair
sofa
end-of-input
```

Снова установите право владельца на выполнение этого файла, запустите его и убедитесь, что содержимое файла result.txt изменилось (как?).

Операционная система Linux позволяет заводить так называемые ссылки на файлы, причем ссылки бывают двух сортов: жесткие и символические. Жесткие ссылки — альтернативные имена того же самого файла. Любые изменения файла, произведенные с указанием одного из таких имен, будут видны под любым другим именем того же файла. При удалении одного из таких имен файл реально удаляется, только если удаляемое имя было последним. Число имен файла показывает команда ls -l сразу после прав доступа. Символические ссылки действуют похожим образом (их чтение и запись означают чтение и запись того файла, на который они ссылаются), но привязываются не к содержимому файла, а к его имени, и при удалении того файла, на который ссылается символическая ссылка,

ссылка становится неработоспособной (как говорят, повисает), и ее можно только удалить. Команда ls -l после имени символической ссылки выдает символы «->» и путь с именем того файла, на который она ссылается.

Для организации ссылок используется команда ln. Сначала, если необходимо, используется опция -s (символическая ссылка; если нет этой опции, ссылка будет жесткой). Далее указываются исходные файлы, на которые нужно ссылаться. После этого указывается имя ссылки (если был один исходный файл), или имя папки, куда помещаются ссылки на исходные файлы; в этом случае их имена совпадают с именами исходных файлов.

Пример.

Создайте пустой файл file.txt . Создайте ссылку на него с именем file2.txt . Сколько имен у файла file.txt ? Запишите в файл file.txt список файлов из текущей папки. Выдайте на экран содержимое file2.txt. Убедитесь, что содержимое file2.txt совпадает с file.txt . Припишите к file2.txt список всех пользователей компьютера, работающих в данный момент. Убедитесь еще раз, что содержимое file.txt и file2.txt опять одинаковое. Удалите file.txt . Выведите на экран содержимое file2.txt . Сколько теперь имен у файла file2.txt ? Создайте символическую ссылку file.txt на file2.txt . Убедитесь, что символическая ссылка создана правильно. Выдайте на экран содержимое file.txt . Припишите к file.txt текущую дату. Убедитесь, что по-прежнему содержимое file.txt совпадает с file2.txt . Удалите file2.txt . Попробуйте выдать на экран содержимое file.txt . Удалите file.txt .

При работе с файлами часто возникает необходимость указывать несколько файлов, имена которых похожи друг на друга. Для этой цели в BASH имеется механизм шаблонов. В шаблоне символ «*» означает любую строку, «?» означает любой, но ровно один символ, и конструкция [...] означает любой символ из числа заключенных между квадратными скобками. Например, шаблону a* соответствуют все имена файлов, начинающиеся на букву a (в том числе с любыми расширениями). Шаблону [abc]x будут соответствовать имена файлов ax, bx, cx. Если в каком-либо из слов команды встречается хотя бы одна из таких конструкций, BASH автоматически заменяет это слово на список соответствующих ему как шаблону имен файлов. При этом шаблоны могут содержать не только имена файлов, но также пути. Например, слово a*/b* будет заменено на список всех имеющихся путей, соответствующих данному шаблону, т. е. строк, состоящих из имени подпапки текущей папки, начинающегося на букву a, затем символа «/», затем имени файла или папки, лежащих в этой подпапке, имя которых начинается с b. Однако, область действия символа * в шаблоне ограничена именем одной папки или файла, т. е. * не может соответствовать строке, содержащей разделитель имен папок «/». Шаблоны такого рода можно указывать в качестве параметров любой команды, понимающей несколько имен файлов в том месте командной строки, где стоит шаблон. Например, ls *x выдаст список имен файлов, кончающихся на x.

Пример.

Заведите пустые файлы с именами apple.txt , orange.txt , pear.txt , bear.txt , wear.txt , alpha.txt , clock.txt. Выдайте на экран имена всех файлов, содержащие букву a. Далее, выдайте на экран список файлов, имена которых содержат не менее двух букв a; имена которых содержат любой символ (один) и за ним ear.txt. Наконец, выдайте на экран имена файлов pear.txt и bear.txt, использовав всего один параметр команды ls . Попытайтесь переименовать файлы с расширением .txt , заменив расширение на .html , командой mv *.txt *.html . Объясните полученный результат.

Кроме того, имеется возможность явно указать набор строк, которые должны быть использованы в определенном месте слова. Для этого используются фигурные скобки, в которых эти строки перечислены через запятую. Например, слово a{b1,c2,d3}x превратится в ab1x ac2x ad3x. Отличие этой конструкции от предыдущих состоит в том, что полученные в результате такой замены слова не обязаны быть именами существующих файлов. Также, эта конструкция может быть вложенной сама в себя, например слово a{b{c,d},e}x превращается в abcx abdx aex.

Из всего вышесказанного видно, что до выполнения команды BASH подвергает командную строку весьма существенным преобразованиям. Может возникнуть вопрос: как узнать, во что превращается то или иное слово командной строки? Для этого проще всего воспользоваться специальной командой echo. Действие этой команды состоит только в том, чтобы выводить ее параметры на экран. Поэтому, если результат преобразования того или иного слова командной строки не ясен, можно дать команду echo слово, и на экран будет выдан результат его преобразования.

Пример.

При помощи команды echo выясните, какие параметры получает команда mv *.txt *.html из предыдущего примера. Создайте пять файлов с именами от f1.txt до f5.txt, каждый из которых содержит свой номер (одну цифру). Объедините содержимое файлов f5.txt, ..., f1.txt (именно в таком порядке) и выдайте его на экран, использовав всего один параметр команды cat .

Операционная система Linux позволяет работать не только с файлами, но и с папками. Создать папку можно при помощи команды mkdir имя_папки. В отличие от Windows, имя этой команды нельзя сократить до md. Удалить пустую папку можно командой rmdir имя_папки. Чтобы удалить непустую папку со всем ее содержимым, нужно пользоваться командой rm -r имя_папки.

Пример.

Создайте в своей домашней папке подпапки с именами alpha , beta , gamma , delta . В папке alpha создайте пустые файлы horse , donkey, bear . В папке beta создайте пустые файлы old , new, model . В папке gamma создайте пустые файлы snow , water . В папке delta создайте пустые файлы oak , leaf, soup . При помощи команды echo с одним параметром выдайте на экран все строки вида папка/файл, где имя папки содержит букву «l», а имя файла — букву «o». В домашней папке выведите на экран все, соответствующее шаблону *e*e*. Почему в полученный результат не вошли beta/model или delta/leaf ?

Рассмотрим теперь архивацию файлов и папок в операционной системе Linux. В Linux существуют программы zip для архивации и unzip для распаковки файлов и папок, однако обычно в Linux для этих целей пользуются другими

программами. Традиционная архивация в Linux происходит в два этапа. Сначала, все подлежащие архивации файлы и папки объединяются в один большой файл без всякого сжатия. Для этого существует несколько программ, одной из наиболее популярных является программа tar. Типичный вызов этой программы для архивации выглядит как tar cf имя_архива имена_архивируемых_файлов_и_папок.

После того, как один большой файл получен, он подвергается сжатию. Для этой цели также имеется несколько программ, наибольшей популярностью из которых пользуются gzip и bzip2 (последняя сжимает файлы несколько лучше). Простейший вызов команды bzip2 выглядит так: bzip2 имя_файла. Кроме того, перед именем файла можно указать степень сжатия (-9 — максимальная; чем выше степень сжатия, тем медленнее работает программа и тем больше памяти она требует).

Распаковка производится в обратном порядке: сначала сжатый файл восстанавливается (командами gunzip или bunzip2) и затем из восстановленного файла извлекаются архивированные в нем файлы и папки (команда tar xf имя_несжатого_архива).

Такой двухступенчатый процесс имеет свои достоинства и недостатки. С одной стороны, достигается более высокая степень сжатия, поскольку обычно чем больше файл, тем лучше он сжимается. С другой стороны, из такого архива нельзя извлечь или заменить в нем один файл — необходимо восстанавливать весь архив.

Пример.

При помощи команды tar заархивируйте папки alpha , beta , gamma , delta из предыдущего примера в один архив с именем dirs.tar . Скопируйте этот архив под именем dirs2.tar . Сожмите одну копию полученного архива при помощи gzip, а другую — bzip2. Какая программа сжимает лучше и насколько? Восстановите одну из сжатых копий. Создайте в домашней папке подпапку arch. Поместите туда восстановленный архив и распакуйте его там. Убедитесь, что все заархивированные файлы и папки действительно извлечены из архива (сравнив домашнюю папку с подпапкой arch при помощи команды diff с опцией -r).

1.4. Управление процессами

Как уже говорилось в предыдущем разделе, при запуске программы в операционной системе Linux появляется новый процесс, представляющий собой специальный объект операционной системы, хранящий всю необходимую информацию о выполняющейся программе. Операционная система Linux является вполне многозадачной, и это значит, что одновременно могут выполняться несколько процессов. Для удобства управления процессами каждый из них имеет свой уникальный номер (обычно целое число порядка нескольких тысяч), называемое PID (process identifier, или по-русски идентификационный номер процесса).

Каждую программу, запускаемую из командного интерпретатора, можно запустить в двух разных режимах: интерактивном и фоновом. Если команда запускается в интерактивном режиме, она захватывает терминал и продолжает общение с пользователем до конца своей работы; при этом приглашение к вводу дальнейших команд не появляется до тех пор, пока интерактивная программа не завершится. Если программа запущена в фоновом режиме, командный интерпретатор запускает ее, но не ждет окончания ее работы, а сразу выдает на экран ее номер (среди всех таким образом запущенных программ) в квадратных скобках и PID соответствующего процесса, и выдает приглашение, ожидая ввода дальнейших команд. При этом обычно в фоновом режиме запускаются программы, не выдающие ничего на экран и не читающие ничего с терминала — в противном случае начнется хаос. Если Вы хотите запустить в фоновом режиме команду, пользующуюся терминальным вводом и выводом, ввод и вывод такой команды должны быть переадресованы (см. предыдущий раздел). В оконных системах обычно в фоновом режиме запускаются программы с оконным интерфейсом, не имеющие консольной составляющей, т. е. не пишущие на терминал и не читающие с него.

Для запуска команды в фоновом режиме нужно поставить в конец командной строки символ «&».

Пример.

Команда sleep с единственным параметром ждет указанное число секунд и затем завершается. Запустите команду ожидания 5 минут в фоновом режиме. Каков будет номер запущенной работы? А идентификационный номер процесса?

Интерактивную программу как правило можно остановить нажатием комбинации клавиш C-C. Однако есть программы, использующие эту комбинацию клавиш для своих внутренних целей. Так же, как правило, интерактивную программу можно приостановить комбинацией клавиш C-Z. Выполнение приостановленной таким образом программы может быть продолжено в интерактивном (командой fg %номер) , в фоновом — bg %номер режимах.

Пример.

Запустите команду ожидания 10 минут в интерактивном режиме. Приостановите выполнение этой программы. Возобновите его в фоновом режиме.

Для того, чтобы прекратить выполнение фоновой программы, существует команда kill. В качестве параметров могут выступать PIDы прекращаемых процессов или номера прекращаемых работ, перед которыми стоит символ «%». Увы, не все процессы могут быть прекращены таким способом. Если простая команда kill не работает, можно попытаться дать опцию -9 , означающую более сильный вариант. Это работает в гораздо большем числе случаев, однако программа, ожидающая медленного ввода-вывода на некотором устройстве, не убивается и так. В этом случае можно пытаться взаимодействовать с самим устройством. Например, команда монтирования гибкого диска может подвиснуть, если заданы неправильные параметры диска (например, число цилиндров большее, чем может обеспечить дисковод). Тогда единственный способ прекратить безуспешные попытки смонтировать этот диск — нажать на кнопку извлечения диска на дисководе.

Пример.

Запустите команду ожидания 10 минут в фоновом режиме. Прекратите выполнение образованного ею процесса досрочно (командой kill).

Для вывода списка всех запущенных в системе процессов используется команда ps. Она выводит по одной строке на каждый процесс, содержащей следующую информацию: PID, с какого терминала запущен процесс, сколько времени он работает (считается только время активной работы данного процесса, но не время простоя в ожидании, например, завершения работы некоторых других процессов), какой командой этот процесс запущен (обычно выдается только имя команды, без параметров командной строки). По умолчанию (без дополнительных опций) выводятся только процессы, относящиеся к данной сессии. Чтобы вывести информацию о всех процессах, нужно дать опцию -e.

Список запущенных работ можно получить по команде jobs.

Пример.

Запустите три команды ожидания по 10 минут. Убедитесь (двумя разными способами), что все эти процессы действительно запущены. Получите полный список выполняющихся на компьютере процессов; как узнать их количество? Как Вы думаете, почему количество запущенных процессов много меньше типичных значений PID?

Каждый выполняющийся в операционной системе Linux процесс может сам порождать новые процессы. Если один процесс порождает другой, то породивший называется родительским, а порожденный — дочерним. Отношение порождения накладывает некоторую структуру на множество процессов, увидеть которую позволяет команда pstree. Она выводит информацию о процессах в виде дерева, в котором каждый процесс имеет в качестве сыновей те процессы, которые он породил. Для экономии места в выводе команды pstree одинаковые поддеревья братьев, т. е. сыновей одного процесса, склеиваются с указанием кратности.

Пример.

Запустите команду pstree. Какой процесс породил процесс, обозначенный как mingetty? Сколько всего таких процессов?

Каждый запущенный процесс в системе имеет такую важную характеристику, как приоритет. В Linux это целое число от -20 (наивысший приоритет) до 19 (наинизший). Чем выше приоритет, тем больше шансов у обладающего им процесса получить процессорное время для своего выполнения. Механизм приоритетов позволяет иметь в фоновом режиме большие вычислительные задачи, не замедляющие работу интерактивных программ, поскольку они выполняются только в том случае, если у системы нет другой работы. Для управления приоритетами существует команда nice, позволяющая указывать приоритет для вновь запускаемой программы. Если эта команда запускается без параметров, она выдает текущий уровень приоритета, на моей системе это 0. Если нужно запустить некоторую команду с другим уровнем приоритета, нужно дать опцию -n сдвиг, и затем текст команды. Сдвиг — это целое число, добавляемое к текущему уровню приоритета, чтобы получить тот уровень, на котором запускается команда-параметр. Например, команда nice -n 5 gcc *.cpp означает «запустить команду gcc *.cpp с уровнем приоритета, равным 5 + текущий уровень приоритета». Более того, Linux позволяет менять приоритет уже запущенных программ при помощи команды renice приоритет список_PID. Например, команда renice 1 2945 устанавливает в 1 приоритет процесса, PID которого равен 2945. Обычные пользователи (не суперпользователь) могут менять приоритеты только своих процессов, причем только в худшую сторону.

Для управления запущенными процессами в окне терминала существует также программа top. Она позволяет отображать на экране список запущенных процессов (обновляемый периодически через определенный интервал времени), показывая их важнейшие характеристики, а также некоторые системные параметры, такие, как использование процессора, оперативной памяти, разделов подкачки и т. д. Кроме того, она позволяет останавливать процессы и менять их приоритет. Также вид этой программы может быть настроен в весьма широких пределах. Для того, чтобы войти в эту программу, достаточно дать команду top. Самыми полезными клавишами для взаимодействия с этой программой являются h — help (помощь, т. е. справка по клавишам) и q — quit (выход).

Пример.

Запустите top из терминального окна. Какова загрузка процессора, сколько израсходовано оперативной памяти, используется ли пространство подкачки?

Кроме того, для той же цели используется программа «Системный монитор» (меню «Приложения» — подменю «Системные»). По функциям эта программа очень напоминает «Диспетчер задач» Windows. Основное окно этой программы имеет четыре вкладки: «Система» — основная информация о системе, «Процессы» — список процессов с указанием количества потребляемых каждым процессом ресурсов, позволяет завершить тот или иной процесс, «Ресурсы» — общее потребление ресурсов (процессорное время, память, сеть), «Файловые системы» — список подключенных (смонтированных) файловых систем с указанием параметров (устройство — файл в папке /dev, обычно дисковый раздел; точка монтирования — папка в едином пространстве файлов и папок, к которой подключена корневая папка на указанном устройстве; тип файловой системы, определяющий структуру специальных данных о расположении файлов, например, для разделов Windows может быть FAT32 или NTFS; общий объем; свободное место; процент заполнения).

Пример.

Запустите «Системный монитор». Как называется эта операционная система, и какова версия используемого ядра? Запущен ли процесс nautilus, что он делает, сколько процессорного времени и памяти требует? Сильно ли загружен процессор, используется ли пространство подкачки, активна ли сеть? На каком устройстве находится корневая файловая система, сколько на нем свободного места, насколько оно заполнено?

1.5. Разные полезные команды

В операционной системе Linux имеется большое количество полезных команд. Рассмотрим важнейшие из них по порядку.

Первая из обсуждаемых здесь команд предназначена для вычисления арифметических выражений. Она называется expr, ее параметрами являются элементарные части вычисляемого выражения, такие как числа и знаки операций, скобки. Например, команда expr 3 + 5 вычисляет сумму 3 и 5, причем опустить разделяющие их пробелы нельзя. Еще одной особенностью использования этой команды является необходимость отменять специальный смысл некоторых символов. Например, скобки и звездочка являются специальными символами в BASH, и если нужно использовать их в выражении, то перед ними нужно ставить \. Например, команда expr 2 * 7 скорее всего не будет работать; чтобы она делала то, что нам нужно, придется написать expr 2 * 7. То же верно и в отношении скобок; они тоже должны представлять собой отдельные параметры. Допустимо также использовать операции сравнения, такие как = или <=.

Пример.

Вычислите выражение 2(3+4).

Следующая полезная команда называется find. Она предназначена для поиска файлов. Типичная командная строка выглядит как find папки выражение . Папки — список папок через пробел. Поиск файлов осуществляется путем вычисления выражения на каждом файле, расположенном в подпапках любой глубины в указанных в командной строке папках. Начало выражения распознается как параметр, равный «!» или «(» или начинающийся с «-».

Выражение состоит из проверок и действий, соединенных операциями. Типичные проверки задаются, например, так: -name шаблон или -path шаблон . Шаблон здесь строится по обычным правилам, используемым для указания файлов; однако, правила соотвествия отличаются от принятых в BASH: * может соответствовать строке, содержащей символ разделения папок «/».

Первая из указанных в предыдущем абзаце проверок определяет имя файла; ее значение — истина, если имя файла соответствует указанному шаблону, и ложь иначе. Вторая проверка аналогичным образом исследует весь путь от папки, указанной в командной строке до имени файла, включая и то, и другое. Путь всегда начинается с одной из явно указанных папок и никогда не преобразуется в абсолютный.

Типичными действиями являются печать имени файла (-print), выполнение команды (-execdir команда ;) или удаление файлов/папок (-delete). Первое из этих действий всегда истинно, второе истинно, если команда завершилась успешно, третье истинно, если удаление удалось. В тексте команды (второе из рассмотренных действий) может встречаться конструкция {}, которая заменяется на имя обрабатываемого файла; разумеется, она (как и завершающая эту конструкцию точка с запятой) должна быть в кавычках, чтобы она не обрабатывалась самим BASH, поскольку фигурные скобки являются специальными символами BASH. Команда запускается в том директории, откуда была запущена команда find.

Выражение для find может содержать несколько проверок или действий, соединенных операциями. Знаки операций должны быть отдельными параметрами; допустимо использовать скобки, ! (логическое не), -a (логическое и; если первый параметр имеет значение ложь, второй не вычисляется), -o (логическое или; если первый параметр имеет значение истина, второй не вычисляется), «,» (список; всегда вычисляются оба аргумента, значением является значение последнего).

Например, команда find . -name a* -a -print ищет и печатает имена всех файлов, расположенных в текущей папке и всех ее подпапках.

Пример.

Заведите в своей домашней папке подпапки books с файлами math, music, drawing; food с файлами fish, meat, potatoes; work с файлами text, table, pen. Найдите при помощи find в подпапках домашней папки все файлы, имена которых содержат 1) букву t 2) две буквы t.

Следующая полезная команда — полноэкранный текстовый редактор nano. В командной строке можно указать имя редактируемого файла, а перед ним — «+строка,столбец» для установки курсора в требуемую позицию.

Все обычные клавиши вставляют символы в текст. Управляющими считаются комбинации клавиш с Ctrl, Meta (Alt- одновременно или сначала Esc, потом требуемая клавиша). Дважды нажав Esc и затем три цифры от 000 до 255, можно вставить символ с указанным кодом.

Пример.

При помощи nano, наберите следующую программу на C++ (файл prog.cpp).

```
 #include <iostream>
using namespace std; int
main() { cout<<"Hello, world!"
; return EXIT_SUCCESS; }
```

Выйдя из nano, убедитесь, что программа набрана правильно.

Следующая полезная команда называется indent. Параметром ее является имя файла; она автоматически расставляет символы перевода строки и пробелы в начале строк, чтобы программа выглядела красиво.

Пример.

Примените программу indent к только что набранной программе и посмотрите результат.

Очередная полезная команда в Linux называется uname. Она выдает информацию о системе, в которой выполняется. Она имеет много различных опций, определяющих, какую информацию выдавать. Вся доступная информация

выдается при помощи опции -a. Она включает название операционной системы, имя и домен компьютера, версию ядра операционной системы, дату его компиляции, тип процессора, платформу и некоторую другую информацию.

Пример.

Выясните при помощи uname, какая версия ядра Linux используется в Вашей системе. Совпадает ли это с тем, что выдает «Системный монитор» из раздела «Управление процессами»? Как определяется в Linux тип процессора?

Есть еще одна команда, выдающая полезную информацию о системе. Это команда df, печатающая сводку об использовании и свободном пространстве смонтированных файловых систем. Можно, конечно, узнать об этом и из «системного монитора», но df выдает эту информацию в виде обычного текста.

Пример.

Узнайте с помощью df, сколько файловых систем подключено в данный момент. Где больше всего свободного места?

В Linux имеются также две весьма удобные утилиты для вычисления контрольной суммы: md5sum и sha1sum. Эти контрольные суммы используются, например, для проверки того, что некоторый длинный файл передан по сети без ошибок. Отличительной особенностью этих контрольных сумм от обычных циклических является то обстоятельство, что их невозможно подделать, т. е. невозможно сознательно изменить файл так, чтобы при этом такая контрольная сумма сохранила свое прежнее значение.

Пример.

Посчитайте md5-сумму программы одного из предыдущих примеров prog.cpp. Измените при помощи nano один из символов этой программы и посчитайте md5-сумму снова. Насколько она изменилась?

Очередная полезная команда называется pr. Она позволяет печатать файлы в несколько столбцов, их число определяется опцией -число, например -2 означает два столбца. Она используется, например, в команде ls для того, чтобы список файлов занимал как можно меньше строк за счет того, что имена файлов печатаются в несколько столбцов. Обычно, она также подготавливает файлы к печати, разбивая их на страницы и добавляя на каждую страницу несколько строк в начале и в конце; это поведение может быть подавлено опцией -T.

Пример.

Выведите список файлов в текущей папке на экран командой ls. Сколько в выводе столбцов? Выведите тот же список файлов в 2, 3 и 4 столбца.

Следующая полезная команда, обсуждаемая здесь, — sync. Она записывает все системные буфера на диск, поэтому если система повиснет и будет принудительно выключена, записанные на диск данные не пропадут.

Для отсылки и просмотра электронной почты существует команда mail. Она имеет консольный и весьма запутанный интерфейс, так что для обычного использования электронной почты она почти непригодна, однако она незаменима для автоматической отсылки почты из разных программ. Синтаксис ее для этой цели таков: mail -s тема адрес источник. Здесь адрес — обычный адрес электронной почты, но если почта отсылается пользователю того же компьютера, то достаточно указать его регистрационное имя. Под источником здесь понимается один из способов переадресации ввода. Например, команда mail -s test root < message пошлет суперпользователю электронное письмо с темой test, текст которого будет взят из файла message.

Пример.

Пошлите сами себе электронное письмо с темой «about mail» и текстом «Here was I» одной командой без дополнительных файлов.

Как говорилось раньше, команда mail умеет еще и читать почту, для чего она должна быть запущена без параметров. Она выводит приглашение, и выполняет вводимые команды. Пустая команда означает «читать очередное сообщение», d — удалить, q — выйти.

Пример.

Убедитесь в том, что сообщение из предыдущего примера дошло правильно, и удалите его.

Можно также обмениваться мгновенными сообщениями при помощи команды write. Ее синтаксис следующий: write пользователь терминал. Сам текст сообщения читается из стандартного ввода, но ввод может быть переадресован.

Пример.

Запустите еще одно приложение терминала, при помощи команды who выясните названия запущенных терминалов. С одного из них пошлите сообщение на другой и вызвав его окно, убедитесь, что сообщение дошло.

В BASH имеется также удобное средство для измерения времени работы некоторой команды. Чтобы выяснить, сколько времени работала команда, нужно в начале командной строки поставить слово time, естественно отделив его пробелами от текста команды. После завершения работы команды система выдаст на экран затраченное на работу команды время, измеренное с точностью до миллисекунд. При этом, обычно выдается три разных значения времени: real — астрономическое время, прошедшее от момента запуска программы до момента ее завершения; user — общее процессорное время, затраченное на выполнение собственно программы пользователя; sys — общее процессорное время, потраченное программой на системные вызовы.

Такой способ определения времени работы программы подходит и для программных конвейеров; если же требуется установить время работы более сложной составной команды, ее текст нужно заключить в скобки.

Пример.

Убедитесь в том, что команда sleep работает правильно, несколько раз задавая различные небольшие интервалы времени. Что можно заметить, смотря на вывод time для sleep? Чем это объясняется?

1.6. Переменные в BASH

Очень часто в процессе работы требуется запоминать промежуточные результаты. Для этого почти во всех языках программирования используется механизм переменных. В командных интерпретаторах он также используется. В BASH он также используется для управления поведением запущенных программ.

В некоторых языках программирования, таких как C или Pascal, переменные нужно объявлять заранее, указывая их тип, т. е. разновидность хранящейся в них информации. В BASH переменные имеют только один тип — строки, и поэтому объявлять их заранее не нужно, они создаются при первом присваивании, т. е. первой попытке записать в переменную какое-либо значение.

Присваивание значения переменной выглядит следующим образом: имя_переменной=значение . Важно, чтобы с точки зрения BASH весь текст присваивания представлял собой одно слово. В частности, это означает, что нельзя писать пробелы ни вокруг знака равенства, ни в самом значении. Если же необходимо включить символы пробела в состав присваиваемого значения, все значение должно быть заключено в кавычки. Например, команда A=B запишет в переменную A текст «B». Но команда A=B C не будет работать ожидаемым образом (на самом деле, такая команда означает «запустить команду C, на время ее работы присвоив переменной A текст «B»). Чтобы присвоить переменной A текст «B C», надо написать A="B C". Когда переменная больше не нужна, можно освободить занимаемую ею память командой unset имя_переменной .

Механизм переменных BASH (как и почти всех командных языков) отличается от традиционного еще и тем, что для извлечения значения переменной перед ее именем нужно поставить знак «$». Например, чтобы посмотреть значение переменной A, нужно дать команду echo $A . Конструкция извлечения значения из переменной может встречаться не только в тексте команды, но и в правой части операции присваивания. Если мы хотим использовать эту конструкцию многократно или в сочетании с явно указанным текстом, необходимо помнить следующее:

а) Если конструкция извлечения значения переменной стоит вплотную к некоторому тексту, начинающемуся с буквы, так что непонятно, где кончается имя переменной и начинается последующий текст — имя переменной нужно заключать в фигурные скобки; так, если нужно извлечь значение переменной XYZ вплотную перед текстом UVW, нужно написать ${XYZ}UVW .

б) Если извлеченное значение переменной должно быть частью параметра команды или присваиваемого значения, содержащего пробелы, конструкция извлечения значения переменной вполне может встречаться внутри кавычек. Например, если переменная A имеет значение Hello , то присваивание B="$A world!" запишет в переменную B текст «Hello world!».

Пример.

Присвойте переменной A значение D. Убедитесь в том, что переменной действительно присвоено правильное значение. Теперь проделайте аналогичные действия, но уже с текстом «We want reboot» в качестве значения. Присвойте переменной B значение переменной A, выведите значение B на экран. Присвойте переменной C значение A, за которым следует значение переменной B (сначала вплотную, а затем, второй раз, — через пробел). Наконец, присвойте переменной D значение переменной A, за которым вплотную следует текст «now». В каждом из последних трех заданий убедитесь, что нужным переменным действительно присвоены нужные значения. Освободите память, занимаемую переменной A. После этого, попытайтесь вывести ее значение на экран.

Оказывается, кроме простейшего извлечения значения переменной, обсуждавшегося выше, в BASH существуют и другие конструкции для этой цели, применяющие к извлекаемому значению переменной определенные преобразования. Рассмотрим важнейшие из них.

а) Косвенное обращение. Конструкция ${!имя} извлекает значение указанной переменной, рассматривает его как имя другой переменной, и извлекает ее значение, которое и считается значением этой конструкции.

Пример.

Присвойте переменным A и B значения one и two соответственно. Присвойте переменной D значение A. Выведите на экран значение косвенного обращения к переменной D. Повторите эту команду, предварительно присвоив D значение B.

б) Подстановка значений по умолчанию. Конструкция ${имя:-слово} извлекает значение из указанной переменной, если такая переменная существует и ее значение не является пустой строкой. В противном случае подставляется указанное слово.

Пример.

Напишите конструкцию, извлекающую значение переменной C с значением по умолчанию UNKNOWN и протестируйте ее в трех случаях: переменная C не определена, переменная C имеет пустое значение и переменная C имеет значение ABC.

в) Присваивание значений по умолчанию осуществляется при помощи конструкции ${имя:=слово}, которая по смыслу похожа на предыдущую, но если указанная переменная не определена или имеет пустое значение, указанное слово становится ее новым значением.

Пример.

Переделайте предыдущий пример с использованием присваивания значений по умолчанию вместо подстановки. В каждом из трех вариантов тестирования проверяйте не только значение рабочей конструкции, но и значение переменной C после его извлечения таким образом.

г) Проверка наличия переменной. Конструкция ${имя:?слово} извлекает значение из указанной переменной, если такая переменная существует и ее значение не является пустой строкой. В противном случае указанное слово выдается

в стандартный поток вывода сообщений об ошибках, и, если командный интерпретатор не был интерактивным, т. е. выполнял команды из файла, а не с клавиатуры, он завершается.

Пример.

Переделайте предыдущий пример с использованием проверки наличия переменной вместо присваивания значений по умолчанию.

д) Извлечение подстроки. В самом общем виде эта конструкция выглядит как ${имя:начало:длина} . Значением ее является подстрока значения переменной, начинающаяся с позиции, заданной выражением «начало» (символы нумеруются с нуля) и имеющая указанную длину. Если :длина опущено, берется подстрока до конца строки. Если длина указана, она должна быть неотрицательной. Начало может быть отрицательным; в этом случае оно отсчитывается от конца значения переменной.

Пример.

Напишите конструкцию, извлекающую 1) первые пять символов 2) символы с 10-го по 17-й 3) последние 10 символов 4) символы с 20-го по 15-й с конца из значения переменной C. Протестируйте написанные конструкции.

е) Длина значения переменной как строки вычисляется при помощи конструкции ${#имя} .

Пример.

Протестируйте эту конструкцию на переменной C со значениями «one», «two and three», «привет».

ж) Извлечение значения переменной без самого короткого начала, соответствующего заданному шаблону, производится при помощи конструкции ${имя#шаблон} . Шаблон строится по тем же правилам, что и при указании нескольких файлов одним словом. Такая конструкция, равно как и родственные ей конструкции, описанные ниже, в основном используется для манипуляции с именами файлов. Например, пусть в переменной A содержится путь; данная конструкция позволяет извлечь из переменной A этот путь без первой входящей в его состав папки таким образом: ${A#*/} . Имеются еще три аналогичные конструкции: ${имя##шаблон} — извлечь значение указанной переменной без самого длинного начала, соответствующего шаблону; ${имя%шаблон} — извлечь значение указанной переменной без самого короткого конца, соответствующего шаблону; ${имя%%шаблон} — извлечь значение указанной переменной без самого длинного конца, соответствующего шаблону.

Пример.

Пусть в переменной C содержится имя файла с путем. Написать конструкции, извлекающие из C 1) имя файла без пути, 2) имя файла с путем, но без расширения. Протестировать написанные конструкции.

з) Замена образца на некоторую строку при извлечении значения переменной выполняется при помощи конструкции ${имя/образец/строка} . Образец строится по тем же правилам, что и при указании файлов; заменяется самая длинная подстрока, соответствующая образцу. Однако, если он начинается с символа «/», то заменяются все вхождения текста, соответствующего образцу (в противном случае заменяется только первое вхождение). Также, если образец начинается с «#», соответствующий ему текст должен быть в начале обрабатываемой строки, т. е. значения переменной; если с «%» — в конце. Также, строка замены вместе с предшествующим ей символом «/» может быть опущена — в этом случае текст, соответствующий образцу, просто удаляется.

Пример.

Написать и протестировать конструкцию, заменяющую в имени файла с путем 1) расширение .txt на .html; 2) любое расширение на .html; 3) любое имя на file (для имен файлов без пути; расширение сохраняется).

Даже в командном интерпретаторе иногда возникает необходимость вычислять арифметические выражения. Можно делать это при помощи команды expr, описанной в предыдущем разделе; однако, в BASH имеются и другие средства для этого. Рассмотрим теперь арифметическую подстановку.

Синтаксис арифметической подстановки таков: $((выражение)). Под выражением здесь понимается обычное арифметическое выражение, составленное из целых чисел, переменных (перед их именами здесь нет необходимости ставить знак «$») и операций, как в языке C. Значением этой конструкции является результат выражения, который может быть использован в правой части присваивания, в параметрах команд и в строках, заключенных в кавычки. При этом, естественно, требуется, чтобы значения переменных, входящих в это выражение, представляли собой целые числа, а не произвольные строки. Например, команда echo $((3+5)) вычислит указанное выражение и выведет его результат на экран, а команда I=$((I+1)) увеличит значение переменной I на единицу.

Пример.

Напишите и протестируйте команду, увеличивающую значение переменной C в три раза.

Часто также возникает необходимость присвоить переменной или использовать в параметре команды результат другой команды. Для этого существует так называемая командная подстановка. Эта конструкция имеет (на выбор пользователя) два разных синтаксиса: ‘команда‘ (здесь использована обратная одинарная кавычка, расположенная в левом верхнем углу клавиатуры, под тильдой) или $(команда) (скобки одинарные, в отличие от арифметической подстановки, где они двойные). Значением командной подстановки будет результат работы указанной команды, т. е. тот текст, который команда выводит в свой стандартный вывод.

Пример.

Создайте файл f.txt и запишите в него строку «Hello world!». Присвойте переменной A содержимое этого файла; убедитесь, что предложенная Вами конструкция работает. Присвойте переменной J значение 10. Используйте командную подстановку для присваивания переменной I результата вычисления выражения 3+2*J при помощи команды expr. Убедитесь в правильности результата.

Обычно, переменные, создаваемые в BASH, не видны в тех программах, которые запускаются из BASH. Для того, чтобы они стали видны, необходимо применить к ним команду export следующим образом: export имена_переменных_через_

Если какая-то переменная нужна только для выполнения одной команды, можно поместить присваивание ей значения непосредственно в командную строку перед именем команды.

Пример.

Присвойте переменной A значение val. Запустите вложенную копию BASH командой bash. Видна ли там переменная A? Выйдите из вложенной копии BASH (C-D на пустой строке) и запустите ее снова, поставив присваивание значения переменной A непосредственно перед командой bash в ту же командную строку. Видна ли теперь во вложенной копии BASH переменная A? Наконец, выйдите снова из вложенной копии BASH и экспортируйте переменную A при помощи команды export. Запустив вложенную копию BASH еще раз, проверьте, видна ли из нее переменная A.

Существует несколько (довольно много) предопределенных переменных, управляющих работой BASH и запускаемых из него программ. Рассмотрим вкратце важнейшие из них.

Одной из самых важных переменных является переменная PATH. Она содержит список папок, в которых ищутся исполняемые файлы команд через двоеточие. Для программы, лежащей в папке, не указанной в PATH, всегда нужно указывать путь; даже если программа лежит в текущей папке и она не прописана в PATH, для ее запуска нужно указать перед именем исполняемого файла ./ (где . означает текущую папку).

Пример.

Выведите на экран содержимое переменной PATH. Сохраните ее значение в переменной A и присвойте ей пустую строку. Попробуйте после этого запустить команду cp. Что происходит? Верните старое значение PATH.

Также важной переменной является TERM. Она досталась в наследство от времен, когда пользователи работали с терминалами, подключенными к большому компьютеру, и описывает тип текущего терминала, который показывает, какие управляющие последовательности символов понимает этот терминал. Правильное значение этой переменной важно для программ, использующих нетривиальные возможности терминала, например, для полноэкранных тектстовых редакторов. В большинстве случаев значение этой переменной устанавливается автоматически, однако бывают ситуации, например, при прямом дозвоне до какого-либо компьютера при помощи терминально-модемной программы вроде minicom, когда автоматическое значение этой переменной неправильно и его нужно устанавливать вручную.

Пример.

Выведите на экран значение переменной TERM и сохраните его в переменной A. Присвойте переменной TERM значение DUMB (самый «тупой» терминал, который может только выводить обычные печатаемые символы на экран). Попробуйте запустить полноэкранный текстовый редактор nano (команда nano). Что происходит? Верните старое значение TERM и попытайтесь запустить nano теперь (для выхода из nano нужно нажать C-X).

Следующей мы рассмотрим переменную LANG. Ее содержимое определяет используемые региональные стандарты, такие как используемый язык, конкретизированный страной, кодировку символов, формат представления чисел, дат, денежных сумм и т. п. Она может быть использована, например, в том случае, если выбранный по умолчанию региональный стандарт неверен или приводит к неправильной работе программ, например, если русские буквы отображаются неправильно. В этом случае можно присвоить ей значение C (американский стандарт) — все будет по-английски, но зато, скорее всего, не будет никаких проблем с неправильным отображением букв на экране.

Пример.

Выведите на экран значение переменной LANG и сохраните его в переменной A. Запустите команду man cp. Присвойте переменной LANG значение C. Попробуйте теперь запустить man cp. В чем разница? Верните прежнее значение LANG.

Переменные PS1–PS4 управляют видом различных приглашений. PS1 содержит текст, использующийся для описания обычного приглашения к вводу очередной команды, PS2 аналогичным образом описывает вторичное приглашение, выдаваемое тогда, когда ввод команды требует дополнительных строк. Это бывает, если команда не завершена синтаксически, например, открыта и не закрыта кавычка, или когда команда принудительно продолжается на следующую строку при помощи символа «\» перед самым концом строки.

Помимо обычного текста, являющегося частью приглашения, значения этих переменных могут содержать довольно много разных специальных конструкций, обычно выглядящих как «\буква» и означающих некоторые параметры текущей сессии, такие как имя пользователя, имя компьютера, путь или имя текущей папки и т. д.

Пример.

Выведите на экран значение переменной PS1 и сохраните его в переменной A. Что означают специальные конструкции в значении переменой PS1? Присвойте PS1 значение «\w>». Что изменилось? Перейдите в другую папку. Что означает конструкция «\w» в значении PS1? Верните прежнее значение PS1.

1.7. Управляющие структуры

Как и в любом языке программирования, в командном интерпретаторе BASH имеются управляющие структуры, позволяющие менять порядок выполнения команд, выполняя команды условно или повторяя выполнение одного и того же набора команд несколько раз.

Начнем с условных конструкций, позволяющих выполнять те или иные команды в зависимости от истинности или ложности некоторых условий. Прежде, чем начать обсуждение условных конструкций, обсудим вообще понятие условия в BASH.

В BASH в качестве условия может выступать абсолютно любая команда. Такое условие считается истинным, если команда завершилась успешно, и ложным, если неуспешно.

Всякая программа, запускаемая из командного интерпретатора, обязана вернуть операционной системе так называемый код возврата — небольшое целое число. Именно поэтому функция main в любой программе на языке C должна вырабатывать целый результат, которым и служит этот самый код возврата. По коду возврата операционная система понимает, успешно ли завершилась программа или нет. В языке C для этого имеются две константы. Если функция main возвращает EXIT_SUCCESS, такое завершение рассматривается как успешное; если же EXIT_FAILURE — завершение с ошибками. Вообще говоря, некоторые программы подходят к этой проблеме дифференцированно, в случае разных ошибок возвращая разные коды; в этом случае по коду возврата можно судить о том, какая именно ошибка имела место.

Вышеописанный подход к условиям позволяет иметь очень широкий спектр условий, однако на практике, как правило, чаще всего встречаются достаточно простые условия типа «существует ли такой-то файл» или «является ли некоторая строка именем папки» и т. д. Для проверки таких условий, как правило связанных с файлами, используется команда test. Ее параметры описывают некоторое условие такого рода, и она завершается успешно, если это условие истинно, и с ошибками, если оно ложно.

Типичное условие выглядит как предикат имя_файла, например условие -f file.txt проверяет, что в текущей папке существует обычный файл с именем file.txt . Аналогично, условие -d dir проверяет, что в текущей папке существует папка с именем dir . Условия можно объединять при помощи операций «логическое и» (-a) и «логическое или» (-o). Например, условие -f abc -o -d abc означает, что в текущей папке существует файл или папка с именем abc . Также, можно строить отрицание условия, ставя перед ним восклицательный знак.

Простейшие условные конструкции имеют следующий синтаксис: команда && команда (операция «логическое и») и команда || команда (операция «логическое или»). Первая из них выполняется следующим образом: выполняется первая команда; если она завершилась с ошибками, на этом выполнение всей конструкции заканчивается, и вся конструкция рассматривается как завершившаяся неудачно; иначе (если первая команда завершилась успешно), выполняется вторая команда, и ее результат (успешно или неуспешно она завершилась) определяет результат всей конструкции. Вторая конструкция выполняется с точностью до наоборот: если первая команда завершилась успешно, вся конструкция завершается успешно, не выполняя второй команды; если первая команда завершилась с ошибками, выполняется вторая команда и определяет результат всей конструкции.

Пример.

Выполните конструкцию test -f file && echo это файл. Что она делает? Напишите конструкцию, выводящую текст «папка», если в текущей папке существует подпапка с именем dir1 . Напишите конструкцию, удаляющую файл f.txt, если он существует. Напишите конструкцию, выводящую «да», если файл testfile существует, и «нет», если он не существует; протестируйте написанную конструкцию, т. е. выполните ее как в том случае, когда файл существует, так и в том, когда он не существует. Напишите конструкцию, выводящую слово «папка», если abc — это имя папки, «файл», если abc — файл, и «нет», если ни то, ни другое.

Кроме условий, связанных с файлами, в BASH возможна проверка арифметических условий, для чего существует арифметическая команда. Ее синтаксис таков: ((выражение)) . Синтаксис похож на арифметическую подстановку, но без знака «$» спереди. В отличие от арифметической подстановки, это — полноценная команда; она вычисляет свое выражение и завершается успешно, если его результат отличен от нуля и неуспешно, если результат нулевой.

Пример.

Напишите и протестируйте конструкцию, проверяющую значение переменной I следующим образом: если оно меньше 100, выводится текст small, если от 100 до 1000, то текст average, иначе текст large.

Кроме вышеописанных простейших условных конструкций, имеется также аналог условного оператора из традиционных языков программирования. Его синтаксис таков: if условие1; then команды1; elif условие2; then команды2; else команды3; fi . Сначала выполняется условие1; если оно истинно, выполняются команды1, иначе проверяется условие2; если оно истинно, выполняются команды2, иначе команды3. При этом частей elif ... then ... может быть сколько угодно; часть else ... может отсутствовать.

Пример.

Написать условный оператор, проверяющий, в какой из папок из набора /bin, /usr/bin, /usr/local/bin лежит файл, имя которого содержится в переменной F, и печатающий имя первой такой папки (папки проверяются в указанном порядке), если в одной из них действительно лежит указанный файл, и «no file», если такого файла нет ни в одной из указанных папок. Протестировать этот условный оператор для следующих значений переменной F: cp, ar, lib.

В BASH имеется также аналог оператора выбора (switch в C или case в Pascal). Его синтаксис следующий: case слово in образец | образец | ...) команды ;; ... esac . Слово проверяется на соответствие образцам, и если соответствие найдено, выполняются соответствующие команды, на чем выполнение этого оператора и заканчивается. Если соответствие так и не найдено, ничего не происходит. Например, следующая конструкция выводит словесный эквивалент оценки, содержащейся в переменной MARK: case $MARK in 2) echo неудовлетворительно ;; 3) echo удовлетворительно ;; 4) echo хорошо ;; 5) echo отлично ;; *) echo неизвестно ;; esac .

Пример.

Написать и протестировать конструкцию, решающую обратную задачу: по словесной оценке в переменной LEVEL выдать ее числовой эквивалент.

BASH имеет также аналоги операторов цикла, позволяющие выполнять одни и те же команды многократно в зависимости от значений переменных-счетчиков или истинности некоторых условий.

Простейшие операторы цикла, управляемые условием, выглядят так: while условие; do команды; done и until условие; do команды; done . Первый из них выполняет условие, и если оно ложно, прекращает работу; если же оно

истинно, выполняет команды и затем снова проверяет условие, и т. д. Второй вариант отличается от первого только трактовкой условия: он выполняет команды, пока условие ложно.

Операторы цикла позволяют программировать задачи, требующие зависящее от параметров число действий. Например, следующая конструкция вычисляет сумму натуральных чисел от 1 до содержащегося в переменной N числа и записывает результат в переменную SUM: SUM=0; I=0; while ((I<=N)); do SUM=$((SUM+I)); I=$((I+1)); done .

Пример.

Под факториалом натурального числа в математике понимают произведение всех натуральных чисел от 1 до этого числа включительно. Напишите и протестируйте конструкцию, вычисляющую факториал числа, содержащегося в переменной M.

Имеются также еще две разновидности операторов цикла. Первая из них позаимствована из языка C и выглядит как for ((выражение1 ; выражение2 ; выражение3)); do команды; done . Сначала вычисляется выражение1. Затем вычисляется выражение2, и если оно дает нулевой результат, выполнение цикла прекращается. Если же результат ненулевой, выполняются команды и вычисляется выражение3. Затем опять вычисляется выражение2, и т. д.

Пример.

Перепишите команду, вычисляющую факториал числа, с использованием цикла for ... . Напишите и протестируйте команду (с использованием цикла for...), создающую 20 пустых файлов file1.txt ... file20.txt.

Вторая разновидность цикла for ... предназначена для перебора всех слов в некоторой строке; чаще всего она используется для выполнения определенных действий над всеми файлами в текущей папке, имена которых соответствуют набору шаблонов. Синтаксис у этой разновидности оператора цикла такой: for имя in строка ; do команды ; done . Строка обычно представляет собой набор шаблонов, каждый из которых заменяется на набор соответствующих ему имен файлов, после чего для каждого из них переменная с указанным именем принимает значение данного имени и выполняются команды. Например, следующая конструкция удаляет все файлы, имена которых начинаются с буквы a, из текущей папки: for A in a*; do rm $A; done . Конечно, данная конкретная задача может быть решена много проще (как?), но бывают случаи, когда использование такой формы цикла for необходимо (см. следующий пример).

Пример.

Напишите команду, переименовывающую все файлы из текущей папки с расширением .cpp, заменяя расширение на .txt (имя должно сохраняться).

В BASH есть также некоторая управляющая конструкция, не имеющая аналогов в традиционных языках программирования и представляющая собой некое сочетание операторов выбора и цикла. Она имеет синтаксис select имя in строка; do команды; done . При выполнении этой конструкции строка изменяется, как и в цикле for, и все составляющие результат слова выводятся на экран с указанием их номеров. Затем у пользователя спрашивают номер выбранного им слова, это слово присваивается переменной с указанным номером и выполняются команды, после чего опять у пользователя спрашивают номер, и т. д. Пользователь может прекратить этот процесс, ответив на очередной вопрос комбинацией клавиш C-D. Программно этот процесс прекращается выполнением команды break в теле select (т. е. среди команд между do и done, относящихся к данному оператору select...). Например, следующая команда выводит на экран имена всех файлов в текущей папке, начинающиеся на букву a, и позволяет пользователю удалить некоторые из них, введя номера удаляемых файлов: select A in a*; do rm $A; done .

Пример.

Модифицируйте предыдущую конструкцию так, чтобы удалялось не более трех файлов, т. е. чтобы после удаления трех файлов эта конструкция автоматически завершалась.

Наконец, в BASH есть аналоги составных операторов из традиционных языков программирования. Один из них — последовательность команд в круглых скобках рассматривается как одна команда, выполняемая в своей копии BASH. Если вместо круглых скобок использовать фигурные (они должны отделяться от команд пробелами, и после последней команды должна стоять точка с запятой) — результат будет почти тем же, но такая последовательность команд будет выполнена в той же копии BASH, которая выполняет объемлющую команду, т. е. новая копия BASH не будет запущена. Эта конструкция может быть использована, например, для того, чтобы запустить последовательность команд в фоновом режиме. Команда (sleep 600; echo now it is time to sleep) & через 10 минут выведет на терминал соответствующее сообщение; это можно использовать для организации напоминаний самому себе. Однако, терминал может быть настроен так, чтобы останавливать фоновые процессы, пытающиеся выводить нечто на терминал. Если это так, то, чтобы указанный пример работал, нужно дать команду stty -tostop .

Пример.

Изменить пример из предыдущего абзаца так, чтобы вместо echo использовалось write. Запустить его и убедиться, что он работает.

1.8. Функции и скрипты

В большинстве языков программирования используется понятие подпрограммы (процедуры, функции). Существует такое понятие и в BASH. Описывается процедура в BASH следующим образом: имя () составная_команда. Вызывается такая процедура так же, как и любая команда. Например, описание p () (echo Some procedure) описывает процедуру p, вызов которой приводит к печати текста «Some procedure».

Пример.

Описать процедуру f, вызов которой приводит к печати сообщения «time over» после 20 секунд ожидания. Вызвать эту процедуру и убедиться, что она работает.

Иногда процедурам для работы требуется некоторая дополнительная информация. Такая информация обычно предоставляется через механизм параметров. Параметры передаются процедурам по тем же правилам, что и обычным командам. Для использования переданных параметров в теле процедуры используются конструкции $номер_параметра. Параметры нумеруются, начиная с 1. Например, следующая функция вычисляет произведение своих двух параметров: p () (expr $1 * $2) . Для извлечения значения параметров можно использовать все те способы, которые были описаны в связи с извлечением значений переменных. Простейший способ, описанный выше, годится только для параметров с номерами от 1 до 9. Так, конструкция $12 означает первый параметр, за которым сразу стоит цифра 2; если нужен 12-й параметр, нужно писать ${12}. Также, описанные в разделе про переменные способы извлечения значений могут пригодиться при использовании значений по умолчанию; например, функция p () (expr ${1:-1} * ${2:-1}) тоже вычисляет произведение своих параметров, однако она выдает разумный ответ даже в случае одного или вообще без параметров.

Пример.

Напишите и протестируйте процедуру, возвращающую сумму целых чисел от 1 до значения параметра (без циклов). Измените ее так, чтобы она работала и без параметров (в этом случае она должна выдавать 0).

Кроме конструкций, извлекающих значения индивидуальных параметров, существует набор способов получить некоторую информацию обо всех параметрах сразу:

а) Конструкция $* означает совокупность всех параметров, как правило, разделенных пробелами.

б) Конструкция $# означает число параметров.

Пример.

Напишите функцию, вычисляющую сумму своих параметров (предполагается, что все ее параметры — целые числа).

Существует также полезная команда shift с неотрицательным целым параметром. Она сдвигает все параметры на указанное число позиций влево; если число-параметр не указано, сдвиг выполняется на 1 позицию. Например, после команды shift 3 четвертый параметр становится первым, пятый — вторым, и т. д.

Пример.

Перепишите функцию из предыдущего примера, используя команду shift.

Большинство языков программирования наряду с понятием процедуры имеют также понятие локальных переменных. Локальные для некоторой процедуры переменные видны только в этой процедуре и существуют, лишь пока эта процедура выполняется. Удобство локальных переменных в том, что можно иметь в разных процедурах одноименные переменные, никак не связанные между собой. Еще одно удобство локальных переменных связано с понятием рекурсии. Рекурсивная функция — функция, которая вызывает саму себя. Каждый вызов такой функции имеет свои собственные копии локальных переменных, и переменные из разных вызовов никак не связаны между собой.

BASH поддерживает рекурсию, т. е. процедуры BASH могут вызывать самих себя, как непосредственно, так и через другие функции.

По умолчанию все переменные в BASH — глобальные, т. е. видимые всем процедурам. Если нужно иметь некоторые локальные переменные, их можно объявить при помощи команды local, перечислив в качестве параметров их имена.

Пример.

Напишите функцию, вычисляющую факториал неотрицательного целого числа, с использованием рекурсии (без циклов).

BASH позволяет не только вводить команды с клавиатуры, но и записывать их в файл, и затем запускать этот файл на выполнение так же, как и программы на машинном языке. Такие командные файлы называются скриптами.

Каждый скрипт (не только из команд BASH, но и почти для любого интерпретатора) начинается со следующей строки: #!имя_интерпретатора_с_путем . В частности, скрипты BASH начинаются со строки #!/bin /bash . После этого следуют команды, выполнение которых должно происходить при запуске скрипта.

Скрипты BASH могут быть запущены двумя способами:

а) Как и всякая другая программа, при помощи указания имени скрипта (возможно, с путем) и его параметров в командной строке. В этом случае для обработки команд скрипта запускается отдельная копия BASH.

б) При помощи специальной команды «.» (ее параметрами являются имя скрипта и его параметры). В этом случае скрипт запускается в той же копии BASH, которая обрабатывает команду его запуска. Для такого запуска скрипта первая строка с указанием BASH в качестве интерпретатора не нужна.

Разница между этими способами запуска скрипта в том, что в первом случае изменения состояния командного интерпретатора, вызванные выполнением скрипта, по завершении этого выполнения теряются. Во втором случае они остаются в силе.

Пример.

Напишите скрипт, меняющий текущую папку на корневую. Запустите этот скрипт двумя способами, описанными выше. В чем разница?

Механизм передачи параметров скрипту практически совпадает с используемым для функций. Разница состоит в том, что конструкция $0 означает имя скрипта (в случае скриптов), а для функций ее значение не меняется на имя вызванной функции.

2. Оболочка mc

2.1. Файловые операции mc

Работа с файлами из командной строки страдает недостаточной наглядностью; исправить это положение призвана оболочка mc. Ее полное название — Midnight Commander, ее интерфейс сильно напоминает Far Manager из мира Windows, однако имеются и отличия.

Запускается mc из командной строки при помощи команды mc. Сразу появляются панели, как в Far, под ними — строка с приглашением командного интерпретатора и в самой нижней строке — подсказки с назначением функциональных клавиш F1–F10. Здесь нужно заметить, что функциональные клавиши не всегда работают в mc. Например, клавиши F1 или F10 чаще всего воспринимаются приложением терминала как команды себе и не доходят до mc. В mc есть удобная замена для функциональных клавиш — нажать Esc, отпустить ее и затем нажать соответствующую цифру. Кроме того, в руководстве по mc встречаются функциональные клавиши с номерами от 11 до 20. Чтобы нажать функциональную клавишу с номером 10+n, нужно нажать одновременно Shift и Fn.

Как и в Far, одна из панелей является активной. В ней располагается подсвеченный файл, и положение подсветки можно менять при помощи клавиш со стрелками. Клавиша Ins отмечает подсвеченный файл и перемещает подсветку на один файл вперед. Файл можно подсветить, щелкнув на нем левой кнопкой мыши. В отличие от Windows, отмечать файлы правой кнопкой мыши нельзя. Как и в Far, переключиться на другую панель можно клавишей Tab . Поменять панели местами можно комбинацией клавиш C-u . Последнее бывает удобно, если брать команды из истории команд, поскольку после этого происходит автоматическое переключение на левую панель (наверно, это ошибка).

Если в папке лежит много файлов и нужно быстро подсветить файл с заданным именем, можно воспользоваться комбинацией клавиш M-s для инкрементного поиска файла по имени (Alt-буква в Far). После нажатия M-s появляется окно, и по мере набора текста в нем на панели подсвечивается имя первого файла, которое начинается с введенного текста. Если нужно найти второе, третье и т. д. имя, нужно еще несколько раз нажать M-s. Для выхода из инкрементного поиска нужно два раза нажать Esc.

Как и в Far, панели можно отключить комбинацией клавиш C-o; повторное нажатие этой комбинации клавиш включает панели обратно.

mc позволяет выполнять основные операции работы с файлами — копирование (F5), перемещение/переименование (F6), удаление (F8). Если некоторые файлы отмечены — операция выполняется над ними, иначе — над подсвеченным файлом. По умолчанию файлы копируются и перемещаются в папку соседней панели, но в появляющемся диалоге можно указать другую папку. Кроме того, при операциях копирования или перемещения файлов можно задавать шаблоны как для исходных, так и для результирующих имен файлов. Например, если нужно переименовать все файлы с расширением .cpp в файлы с тем же именем, но расширением .txt, можно воспользоваться следующей последовательностью команд:

1) «+» на дополнительной числовой клавиатуре (серый «+») вызовет диалоговое окно «Отметить группу». Введя в нем *.cpp и нажав Enter, отмечаем все файлы с расширением .cpp .

2) F6 (переместить) и вводим исходный шаблон *.cpp, шаблон результата *.txt и нажимаем Enter.

Пример.

Запустите mc. Создайте (из командной строки) три пустых файла f1.txt, f2.txt, f3.txt. Скопируйте их, добавив к имени спереди букву a. Удалите сначала файл f1.txt, подсветив его, затем разом f2.txt и f3.txt, отметив их.

mc позволяет также работать с папками. Копировать, перемещать и удалять папки можно таким же образом, как и файлы. Для создания новой папки предусмотрена отдельная команда (F7). Чтобы зайти в подпапку, необходимо подсветить ее и нажать <Enter>. Для перехода в объемлющую папку нужно подсветить элемент «вышестоящая папка» («..») и нажать на <Enter>.

Пример.

Создайте в своей домашней папке подпапку test. Зайдите в нее. Создайте там три пустых файла f, g, h. Выйдите из этой папки и скопируйте ее под именем test2. Зайдите туда и убедитесь в наличии там нужных файлов.

Благодаря используемому в mc понятию «виртуальная файловая система», аналогичным образом можно заходить в архивы, на удаленные ftp-серверы и т. д., причем для виртуальных файловых систем предусмотрены способы указания путей, во многом напоминающие пути для обычных папок, так что заходить в такие папки можно при помощи обычной команды cd. Так, например, чтобы зайти в папку files внутри архива arch.tar, расположенного в папке /home/user1, можно воспользоваться командой cd /home/user1/arch.tar#utar/files . Однако, находясь на виртуальной файловой системе, например внутри архива, нельзя выполнять команды BASH; кроме того, многие из обычных файловых операций для некоторых типов виртуальных файловых систем еще не реализованы.

Пример.

Создайте в своем домашнем директории папку new1, зайдите в нее и создайте в ней три пустых файла. Выйдите из этой папки и заархивируйте ее. Войдите внутрь полученного архива и скопируйте один из пустых файлов в другую панель.

Кроме архивов, существуют также следующие типы виртуальных файловых систем:

- FTP (на панели будет показано содержимое папки на удаленном компьютере; операции копирования будут приводить к пересылке файлов по сети). Для этого используется команда cd со следующим путем: /#ftp:!имя:пароль@адрес:п
. Восклицательный знак необязателен, он означает необходимость использования proxy-сервера. Имя пользователя и пароль также необязательны; если пароль не указан, он будет запрошен; если нет также имени, будет использовано имя anonymous и стандартный для таких случаев пароль (обычно email адрес текущего пользователя). Порт также необязателен; если он не указан, используется стандартный порт для FTP.

- Дискеты (если установлен пакет mtools). Можно пользоваться командой cd a: для перехода к корневой папке дискеты. Недостатки такого подхода — медленно работает и не понимает русских букв в именах файлов.

Существует ряд способов ускорить смену папки.

- Справочник папок. mc позволяет запомнить несколько самых часто посещаемых папок в так называмом справочнике папок, присвоив каждой из них некоторое название, и затем выбирать одну из этих папок из списка. Справочник папок вызывается комбинацией клавиш C-\ .

- M-c (быстрая смена папки). Открывает окно, в котором надо ввести путь к нужной папке. Очень удобно, когда командная строка содержит уже набранную часть команды, потому что в этом случае клавиша Enter означает не вход в подсвеченную папку, а запуск команды из командной строки.

- Переменная CDPATH. Эта переменная содержит набор абсолютных путей некоторых папок через «:». Если папка, в которую нужно перейти, не является подпапкой текущей папки, то она ищется во всех папках, пути которых указаны в CDPATH.

- Переход к предыдущей папке осуществляется командой cd − . Для этой же цели можно щелкнуть левой кнопкой мыши на значке < с левой стороны заголовка соответствующей панели. Также (с правой стороны заголовка панели) имеются значки ^ (история папок) и > (следующая папка).

Пример.

Внесите в справочник папок папку new1 из предыдущего примера. Перейдите в корневую папку всей файловой системы и оттуда — опять в папку new1, вызвав ее из справочника папок. При помощи щелчков мыши по значкам < и > в заголовке панели, перейдите в предыдущую папку и затем обратно.

mc позволяет удобно искать файлы как по имени, так и по содержимому. По сравнению с командой find его возможности кажутся ограниченными, но в большинстве случаев их вполне достаточно. Соответствующий диалог вызывается комбинацией клавиш M-? (можно также воспользоваться клавишей F9 — вызов служебного меню, затем пунктом «Поиск файла» в меню «Команда»). В указанном диалоге можно задать папку, в подпапках которой будет производиться поиск, шаблон имени файла (как в BASH) и/или текст, который нужный файл должен содержать. В качестве результата выдается другой диалог, содержащий список найденных файлов. Нажатие Enter на некотором файле из списка открывает содержащую его папку и подсвечивает этот файл.

Пример.

Заведите в своей домашней папке подпапки f1, f2, f3, вложенные одна в другую. В папке f3 создайте файл a с текстом qwerty и b с текстом asdfgh. Вернитесь в свою домашнюю папку и найдите файлы, имена которых содержат букву b. Затем найдите файлы, содержащие строку wer в своем содержимом.

Также имеется удобный диалог для изменения прав доступа к файлам. Для его вызова нужно воспользоваться комбинацией клавиш C-x c или пунктом «Права доступа» меню «Файл». В этом диалоге права доступа представлены как набор позиций, каждую из которых можно включить или выключить (для изменения ее состояния можно щелкнуть на ней левой кнопкой мыши или, подогнав к ней фокус при помощи клавиши Tab, нажать пробел). Если отмеченных файлов нет, устанавливаются права подсвеченного файла. Если есть подсвеченные файлы, можно работать с каждым отдельно (нажатие кнопки «установить» устанавливает права одного файла и переходит к следующему) или установить права всех отмеченных файлов одновременно при помощи соответствующей кнопки.

Пример.

Создайте в своей домашней папке файл prog с текстом

```
#!/bin/bash
echo Hello, world!
```

Сделайте этот файл исполняемым. Что изменилось на панели? Запустите этот файл (проще всего — подсветить его и нажать Enter).

Содержимое панелей и способ его отображения во многом могут быть настроены. Для этого используются меню «Левая панель» и «Правая панель»; по возможностям они совпадают, но относятся к разным панелям, как следует из названия.

Первым пунктом этого меню идет «Формат списка...». Здесь можно управлять способом отображения списка файлов на панели. Имеется три стандартных способа отображения и также возможность для самостоятельной его

настройки. Из всех этих способов один отображает файлы в два столбца; все остальные (в том числе и пользовательский) отображают по одному файлу на строке. Для тонкой настройки отображения списка файлов применяется строка, описывающая нужный способ и состоящая из слов, указывающих элементы информации о файле, которые должны быть показаны, а также разделителей и специальных слов через пробел. Под специальными словами здесь понимаются half, указывающее на то, что панель занимает половину терминального окна, или full (все окно — удобно для полного отображения очень длинных имен файлов). Из элементов информации о файле можно упомянуть name (имя файла), size (размер файла), perm (права доступа), type (один символ, описывающий тип файла, такой как «папка», «исполняемая программа», «символическая ссылка» и т. д.). В качестве разделителя используется | в тех местах, где он нужен. Также аналогичным образом настраивается нижняя строка, содержащая информацию о подсвеченном файле.

Пример.

Какой пункт в списке форматов отображения панели располагает файлы в два столбца? Установите формат отображения файлов так, чтобы левая панель отображалась на весь экран и содержала только имя файла. Добавьте еще разделитель и права доступа. Верните старый способ отображения файлов на левой панели.

Следующие пункты меню «Левая нанель» устанавливают режимы работы левой панели как панели быстрого просмотра подсвеченного файла на правой панели, информации об этом файле или дерева папок. Далее следуют пункты, первый из которых открывает диалог выбора порядка сортировки, и второй — ввода маски фильтра, позволяющий отображать только те файлы, имена которых соответствуют шаблону (как в BASH).

Пример.

Отсортируйте список файлов на панели по имени, затем по расширению, затем по времени доступа. Установите фильтр, отображающий только те файлы, имена которых начинаются с буквы f и убедитесь, что фильтр работает, создав два пустых файла, из которых один удовлетворяет условию фильтра, а другой — нет. Верните обратно отображение всех файлов.

Далее следуют пункты, позволяющие переходить на некоторые виртуальные файловые системы (FTP, SMB, ...), и, наконец, пункт обновления содержимого панели (что быстрее делается комбинацией клавиш C-r).

Важной и полезной возможностью mc также является так называемый критерий панелизации (далеко идущее обобщение фильтра панели). Это команда, вывод которой отображается на панели в качестве списка имен файлов. Задействовать такую команду можно комбинацией клавиш C-x ! или пунктом «Критерий панелизации» меню «Команда». Например, если есть файл list, содержащий список имен файлов, то отобразить файлы из списка на панели можно при помощи команды cat list, использованной в качестве критерия панелизации. Если Вы уйдете из полученной псевдопапки (назовем это так), она пропадет. Если Вы хотите часто пользоваться каким-то критерием панелизации, разумно запомнить его. Для этого после ввода команды в соответствующем диалоге нужно нажать кнопку «Добавить» и далее ввести имя нового критерия панелизации.

Пример.

Отобразите на панели только 5 первых файлов из домашней папки.

Также полезными командами являются вычисление размеров папок (соответствующий пункт меню «Команда») и сравнение папок правой и левой панелей (C-x d).

Пример.

Перейдите в корневую папку всей файловой системы и выясните, какая из ее подпапок содержит больше всего информации, а какая — меньше всего. Создайте в домашней папке две подпапки cmp1 с файлами a, b, c и cmp2 с файлами a, b, d. Сравните содержимое этих папок.

mc предоставляет весьма богатые возможности по работе с файлами, однако если и их не хватает, всегда остается возможность набрать нужную команду в командной строке. Однако, команда cd реализована в самом mc. С одной стороны, это позволяет использовать ее для перехода на виртуальные файловые системы вроде содержимого архивов, но с другой накладывает ряд ограничений. Например, встроенная в mc команда cd не позволяет использовать в путях символы-заменители фрагментов текста, такие как ? и *.

Для ускорения ввода командной строки имеется ряд средств.

Во-первых, можно автоматически поместить в командную строку имя подсвеченного файла (M-Enter), список отмеченных файлов (C-x t для активной панели, C-x C-t для пассивной) или путь папки, содержимое которой отображается на панели (C-x p для активной панели, C-x C-p для пассивной).

Во-вторых, существует большое количество специальных конструкций вида %буква, которые можно использовать в командной строке; перед выполнением команды они будут заменены на их значения. Их достаточно много; упомянем лишь самые главные.

%m Имя текущего файла меню.

%f Имя подсвеченного файла.

%x Расширение имени подсвеченного файла

%b Имя подсвеченного файла без расширения.

%d Полный абсолютный путь папки, отображаемой на активной панели.

%t Список имен отмеченных файлов через пробел (для активной панели).

%u Подобно %t, но после использования этой конструкции отметка с файлов снимается.

%s Подобно %t, если отмеченные файлы есть; подобно %f, если их нет.

%% Знак %.

Для конструкций, относящихся к активной панели, имеются аналоги, относящиеся к пассивной. Они отличаются тем, что в их записи используется большая буква вместо маленькой. Например, %F означает имя подсвеченного файла пассивной панели.

Пример.

Заведите в папке активной панели текстовый файл f.txt, содержащий qwertyuiop. Скопируйте его на пассивную панель. Наберите команду, сравнивающую содержимое этого файла из папок разных панелей, сначала при помощи комбинаций клавиш, затем при помощи специальных конструкций с %. Наберите команду, записывающую в файл t.txt список отмеченных файлов активной панели, сначала так, чтобы отметка после выполнения этой команды оставалась, затем так, чтобы она сбрасывалась.

2.2. Другие полезные возможности mc

Кроме описанных в предыдущем параграфе, mc имеет ряд и других полезных возможностей. Начнем с описания встроенной программы для просмотра файлов. Обычно она вызывается для подсвеченного файла клавишей F3 .

Встроенная программа для просмотра содержимого файлов имеет несколько режимов работы, которые можно выбирать нажатием функциональных клавиш. При выходе из этой программы последний режим запоминается и восстанавливается при следующем ее запуске.

Клавиша F1 вызывает окно с текстом руководства.

Клавиша F2 управляет режимом отображения длинных строк. Существуют два таких режима: перенос длинных строк на следующие экранные строки и отображение частей длинных строк с возможностью перемещения видимого окна вправо и влево.

Клавиша F4 переключает режим отображения файла из текстового (обычного) в двоичный и наоборот. В двоичном режиме содержимое файла отображается побайтно, причем значение в каждом байте выводится в виде двузначного шестнадцатеричного числа. Для удобства данные выводятся фиксированными порциями по 16 байтов на строку, слева от двоичных данных выводится адрес первого байта каждой строки, а справа — текстовый вид соответствующей порции байтов.

Клавиша F5 позволяет перейти к строке с определенным номером, F6 и F7 обеспечивают поиск информации (F6 — поиск текста, соответствующего регулярному выражению). При поиске в двоичном режиме в строке поиска указываются значения байтов, а если нужно искать строку, заданную последовательностью символов, ее нужно заключить в кавычки.

Клавиша F8 включает и выключает режим интерпретации содержимого файла. Если этот режим выключен, содержимое файла показывается так, как оно записано на диске; если этот режим включен, содержимое файла пропускается через специальный фильтр, назначенный для данного типа файлов (для каждого типа файлов может быть назначен отдельный фильтр). Например, для просмотра html-файлов запускается lynx — текстовый броузер, т. е. программа, предназначенная для отображения web-страниц на текстовых терминалах. Бывает такая ситуация, что необходимый фильтр отсутствует в системе. Тогда встроенная программа просмотра не запускается, выдавая сообщение об ошибке. Если все же нужно просмотреть этот файл, хотя бы в непосредственном виде, то нужно открыть в этой программе другой файл (на котором программа запустится), отключить там режим интерпретации, закрыть этот файл и после этого пытаться посмотреть тот файл, который был нужен изначально.

Клавиша F9 управляет режимом форматирования. Если он включен, некоторые последовательности символов рассматриваются как указания на выделение фрагментов текста жирным шрифтом или подчеркиванием (на современных терминалах это как правило показывается сменой цвета).

Клавиши F3 или F10 закрывают программу просмотра файлов и возвращают mc в режим показа панелей.

Пример.

Создайте в своей домашней папке текстовый файл sample.txt с текстом «qwertyuiop». Просмотрите его содержимое в обычном (текстовом) и двоичном режимах. Создайте еще один текстовый файл, содержащий две длинных строки (например, по 150 символов каждая). Откройте этот файл для просмотра; нажмите F2 и посмотрите, что изменится. Создайте файл f.html с содержимым

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html><head>
<meta content="text/html; charset=UTF-8"
http-equiv="content-type">
<body>Привет!</body></html>
```

Нажмите F8 и посмотрите, что изменится.

Встроенная программа просмотра файлов может быть также использована для отображения вывода некоторой команды (обычно такой, которая ничего не вводит с клавиатуры). Для этого существует комбинация клавиш M-! , вызывающая окно для ввода команды.

Пример.

Запустите в просмотрщике команду man cp. Нажмите F9 и посмотрите, что изменится. Найдите в этом тексте описание опции -a . Перейдите к 150-й строке.

В mc также существует встроенный редактор. Он вызывается для подсвеченного файла нажатием F4. В нем, как и в программе просмотра файлов, меняется смысл большинства функциональных клавиш.

Клавиша F1, как всегда, вызывает окно с текстом руководства. Клавиша F2 служит для записи файла на диск. Клавиша F3 начинает/заканчивает выделение текста. Клавиша F4 служит для поиска и замены фрагментов текста. F5 копирует выделенный текст в позицию курсора, F6 перемещает выделенный текст. F7 — поиск фрагментов текста, F8 — удаление блока (выделенной порции текста). F9 вызывает строку меню в верхней строке экрана, F10 — выход из редактора.

Кроме вышеперечисленных команд, встроенный редактор имеет ряд других, тоже весьма полезных. Так, C-u отменяет последнюю операцию редактирования; C-f записывает выделенный текст в файл; F15 (т. е. Shift-F5) вставляет содержимое другого текстового файла в позицию курсора. Shift-F3 начинает выделение прямоугольного блока символов, Ins переключает режим вставки/перезаписи символов, C-PgUp и C-PgDn позволяют быстро переместиться в начало и конец файла соответственно, M-l переходит к строке с определенным номером, M-b переходит к парной скобке.

Пример.

Откройте для редактирования файл f.html из первого примера этого раздела. Выделите слово «Привет» и сохраните его в файл block.txt . Затем удалите выделенный текст и сохраните файл. Выйдите из редактора и посмотрите полученный файл. Откройте его снова для редактирования, вставьте на прежнее место содержимое файла block.txt и перейдите к 3-й строке, затем в начало и в конец файла. Введите после слова «Привет» выражение (2+3*(3+2)-1)*7+2. Поставьте курсор на первую открывающую скобку и одной командой перейдите к парной скобке. Отмените несколько последних команд так, чтобы выражение исчезло. Перейдите к началу файла и найдите в тексте <body> . Замените угловые скобки на соответствующие круглые и выйдите из редактора без сохранения файла.

mc позволяет настраивать стандартные действия, выполняемые при попытке открыть или просмотреть файл, в зависимости от типа этого файла. Причем для выбора таких действий можно использовать не только расширение имени файла, но и тип его содержимого, определяемый командой file.

Правила соответствия действий типам файлов записаны в файле ~/.mc/bindings. Это обычный текстовый файл, пустые строки которого, равно как и строки, начинающиеся с #, пропускаются. Все остальные строки делятся на секции; каждая секция соответствует одному типу файла и описывает соответствующие ему действия. Начинается секция со строки, имеющей в первом столбце непробельный символ. Все остальные строки секции начинаются с пробела или символа табуляции и следуют за первой до следующей строки, не начинающейся с пробела.

Первая строка секции определяет тип файла, описываемый данной секцией. Она имеет следующий вид: слово/описание . Слово описывает способ определения типа файла и определяет трактовку описания. Оно может быть одним из следующего списка:

1) shell В этом случае для описания возможно два способа трактовки. Если оно не начинается с точки, оно означает полное имя файла; в этом случае секция описывает действия для файла с конкретным именем. Если же описание начинается с точки, оно трактуется как расширение имени файла, и такая секция описывает действия для всех файлов с указанным расширением.

2) regex В этом случае описание рассматривается как регулярное выражение, и имя файла должно содержать текст, соответствующий данному регулярному выражению.

3) type Как и в предыдущем случае, описание рассматривается как регулярное выражение, но на этот раз соответствующий ему текст ищется в выводе команды file для подсвеченного файла.

4) directory означает папку, имя которой содержит текст, соответствующий описанию, рассматриваемому как регулярное выражение.

5) include означает общую секцию; в этом случае описание является именем такой секции; она может быть включена в любую другую секцию.

6) default соответствует любому файлу; в этом случае описание может быть любым.

Внутри секции строки имеют вид действие=команда . Если в качестве действия используется Include , то команда трактуется как имя общей секции, которую надо включить (она должна быть объявлена ранее с помощью слова include). В остальных случаях команда представляет собой однострочную команду, запускаемую для реализации указанного действия, которое может принимать значения Open (соответствует нажатию клавиши Enter на подсвеченном файле), View (клавиша F3) и Edit (F4).

В используемых командах возможно использование специальных конструкций, как и в командной строке (см. предыдущий раздел). Однако, кроме описанных там, можно использовать и ряд других, о которых будет сказано здесь. %view — запуск встроенной программы просмотра, %{текст подсказки} — выдается окно с указанным текстом подсказки и вся конструкция заменяется на ответ пользователя, %var{имя:значение} — извлекает значение переменной с указанным именем, а если такой нет — заменяется на указанное значение.

Пример.

Посмотрите, какая команда открывает файлы с расширением .pdf и какая — с расширением .tgz . Откройте файл с текстом лабораторных работ. Какая программа используется для просмотра таких файлов? Отредактируйте файл, содержащий соответствия программ типам файлов так, чтобы для просмотра .pdf-файлов по клавише F3 использовалась команда acroread. Сохраните файл и попробуйте теперь просмотреть содержимое файла с текстом лабораторных работ. Какая программа используется сейчас? Какая программа используется для открытия .html-файлов? Посмотрите, как она работает, открыв файл f.html из предыдущего примера. Отредактируйте еще раз файл соответствия программ типам файлов, чтобы для открытия файлов .html использовалась программа firefox. Убедитесь, что все работает.

Для часто используемых операций mc позволяет создавать меню пользователя. Существует общесистемный файл меню, файл меню для конкретного пользователя и файл меню для конкретной папки (используется последний существующий из указанного списка). Такое меню вызывается нажатием F2; редактировать эти файлы можно, воспользовавшись пунктом «Файл меню» подменю «Команда» из верхней строки экрана.

Формат файла меню также очень прост. Как и для файла расширений, строки, начинающиеся с символа #, трактуются как комментарии. Также, файл делится на секции, каждая из которых соответствует одному пункту меню. Секция состоит из заголовка и набора команд, описывающих действие, выполняемое при выборе данного пункта. Заголовок состоит из строк, начинающихся в первом столбце (т. е. имеющих там непробельный символ), тогда как команды начинаются с пробела или символа табуляции. В тексте команд допустимо пользоваться специальными конструкциями, начинающимися с %.

Заголовок секции может содержать строку, начинающуюся с «+» и содержащую условие включения данного пункта в меню. Если такая строка есть, данный пункт появится в меню только в том случае, если условие истинно. Если такой строки нет, пункт будет в меню всегда. Аналогично, заголовок секции может содержать строку, начинающуюся с «=». Она также содержит условие, которое определяет, будет ли пункт меню выполняться. Если нужно задать оба этих условия одинаковыми, можно написать одну строку условия, начав ее с «+=». Наконец, обязательная строка заголовка определяет текст данного пункта. Если она начинается с буквы или цифры, этот символ становится «горячей клавишей» для данного пункта.

Условие представляет собой цепочку элементарных условий, соединенных логическими операциями | (или) и & (и). Такая цепочка вычисляется слева направо, т. е. приоритеты логических операций одинаковы.

Элементарные условия имеют вид буква параметр. Кроме того, можно добавить спереди «!», что будет означать отрицание этого элементарного условия.

Буква может принимать одно из следующих значений:

1) f В этом случае параметр трактуется как образец, и условие истинно, если имя файла соответствует образцу, причем могут использоваться как образцы в стиле BASH, так и регулярные выражения, в зависимости от значения опции shell_patterns (ее значение 0 или 1 может быть задано в начале файла меню).

2) d Аналогично предыдущему пункту, но проверяется имя текущей папки.

3) t В этом случае параметр — набор букв, каждая из которых соответствует определенному условию на подсвеченный файл, и проверяется, что хотя бы одно из условий выполнено. Буквы в данном условии могут быть следующие: а) n — не папка, б) r — обычный файл, в) d — папка, г) l — символическая ссылка, д) c — байт-ориентированное устройство, е) b — блок-ориентированное устройство, ж) f — именованный канал, з) s — сокет, и) x — исполняемый файл, й) t — файл помечен.

4) x имя означает проверку того, что файл с указанным именем исполняем.

5) для пунктов 1–3 есть аналоги для текущего файла на пассивной панели (вместо маленьких букв используются большие).

Пример.

Добавьте первым пунктом Вашего пользовательского меню уничтожение выбранных файлов, а вторым — копирование отмеченных файлов с добавлением буквы r к их имени спереди. Убедитесь, что все работает. Сделайте так, чтобы эти пункты меню были а) показаны б) работоспособны, только если подсвеченный файл имеет расширение .html . Опять убедитесь, что все работает.

3. Работа с текстовыми базами данных

3.1. Поиск

Для поиска имеется утилита grep. Она ищет во входном файле строки, содержащие текст, соответствующий указанному регулярному выражению, и выводит их в стандартный поток вывода. Регулярное выражение — это строка, написанная в соответствии с определенными правилами, позволяющими описывать некоторое (возможно, даже бесконечное) количество фрагментов текста при помощи одной строки. Эти правила будут описаны ниже.

Регулярные выражения используются в очень многих программах для операционной системы Linux. В основном правила их составления очень похожи, однако какие символы понимаются как обычные, а какие — как управляющие, может сильно зависеть от конкретной программы. Поэтому в описании основных конструкций регулярных выражений в большинстве случаев предполагается, что все служебные символы, их образующие, трактуются как специальные. Если в какой-то конкретной программе это не так, в большинстве случаев проблема решается, если перед соответствующим символом поставить обратную косую черту. Например, ниже описывается операция + (одно или более повторений регулярного выражения). Однако, в программе grep символ + понимается как обычный, и если нужно использовать операцию +, то нужно написать \+ .

Кроме того, многие символы из состава регулярных выражений понимаются как управляющие командным интерпретатором, и потому вместо описания регулярного выражения непредсказуемо меняют команду. Чтобы этого избежать, в BASH проще всего заключать регулярные выражения в апострофы, в подавляющем большинстве случаев этого достаточно. Также уместно здесь напомнить, что до запуска команды grep можно проверить, что будет подано ей в качестве командной строки, при помощи команды echo .

Командная строка при вызове grep выглядит как

```
grep опции регулярное-выражение имена-файлов
```

Имена входных файлов могут отсутствовать. Если это так, то входная информация читается со стандартного ввода, что позволяет использовать grep как фильтр. Выходная информация всегда поступает на стандартный вывод, так что если нужно сохранить ее в файле, нужно перенаправить стандартный вывод в файл.

Наиболее важные опции:

-E означает использование расширенного языка регулярных выражений. Расширенный язык регулярных выражений отличается от обычного (на Linux) только тем, какие символы воспринимаются как управляющие, и какие — как обычные. В любом случае поменять статус символа (превратить управляющий символ в обычный и наоборот) можно, поставив перед ним \.

-F означает использование в качестве регулярных выражений только строк из обычных символов (подавляет все управляющие символы). Это позволяет иметь очень много строк поиска и работать очень быстро.

-P означает использование регулярных выражений языка Perl.

-f имя-файла позволяет читать регулярные выражения из указанного файла, по одному на строку. В этом случае ищутся строки, содержащие текст, соответствующий любому из содержащихся в файле регулярных выражений.

-e регулярное-выражение позволяет задавать регулярные выражения, начинающиеся с «-», или иметь несколько регулярных выражений, заданных в командной строке.

-A число означает вывод указанного числа строк после совпадения. Выводит также строку, содержащую «--» между группами последовательных совпадений.

-B число аналогично предыдущей опции, но выводит строки до совпадения.

-C число — выводит указанное число строк контекста (аналогично двум предыдущим).

-v изменить смысл поиска на противоположный: выводить несоответствующие строки.

-w искать только те соответствия, которые образуют целые слова.

-x аналогично предыдущей, только не слова, а целые строки.

Пример.

Создайте текстовый файл f.txt , содержащий следующий текст:

```
apple
pear
grape
plum
orange
carrot
```

```
cucumber
cabbage
```

Что выведет команда grep p f.txt ? Запустите ее и проверьте правильность своей догадки. А команда grep r f.txt ? Команда grep -v r f.txt?

Наиболее важные конструкции языка регулярных выражений:

а) Обычный (не управляющий) символ соответствует одной строке, состоящей из этого символа. Например, регулярное выражение «x» соответствует тексту «x» и никакому другому.

б) Конкатенация регулярных выражений соответствует конкатенации строк (т. е. конкатенации регулярных выражений соответствует строка, являющаяся конкатенацией строк, соответствующих каждому из регулярных выражений). В частности, согласно этому правилу, строка из обычных символов соответствует только себе самой.

Пример.

Что выведет команда grep ra f.txt ? Почему? Убедитесь в правильности своей догадки. А команда grep -A 1 l f.txt ? Здесь используется файл f.txt из предыдущего примера.

в) Конструкции, соответствующие пустой строке в определенном месте. ^ — начало строки, $ — конец строки, \< и \> — начало и конец слова соответственно.

Пример.

Для того же файла f.txt, что выдаст команда grep '^c' f.txt ? А команда grep 'e$' f.txt ? В чем отличие этих команд от grep c f.txt и grep e f.txt ? Зачем в этих командах регулярное выражение заключается в апострофы?

г) Конструкции, соответствующие символу из некоторого набора. В самом простейшем варианте — просто набор символов, заключенных в квадратные скобки (например, [abcdef] соответствует a, b, c, d, e или f). Допустимо указывать диапазоны, но результат зависит от выбранного языка (локали, поскольку от этого зависит порядок символов). Т. е. предыдущий пример можно сократить до [a-f]. Также, поставив в начало такого набора символ ^, смысл набора меняется на противоположный (т. е. такая конструкция соответствует всем символам, не присутствующим внутри квадратных скобок). Например, выражение [^A-Za-z] соответствует любому символу, не являющемуся латинской буквой. Имеются также удобные конструкции для сокращения такого рода регулярных выражений, например «.» соответствует любому одинарному символу (кроме символа перевода строки). Также, имеются обозначения для наборов символов из наиболее часто встречающихся классов, таких, как буквы, цифры, печатные символы и т. д. Например, конструкция [[:alnum:]] соответствует любому буквенно-цифровому символу. В этой конструкции внешняя пара [] указывает на то, что это регулярное выражение соответствует символу из набора, а [:alnum:] — на конкретный набор символов.

Пример.

Для того же файла f.txt, что выдаст команда grep '[ar]' f.txt ? Напишите команду, выдающую те строки из этого файла, которые содержат по крайней мере одну букву из набора b, l, m. Напишите еще одну команду, которая выдает те строки из f.txt, в которых после r стоит a или o.

д) Различные конструкции, определяющие количество повторений регулярного выражения. Если есть некоторое регулярное выражение, из него можно построить новое, которому будут соответствовать строки, состоящие из определенного числа частей, каждая из которых соответствует исходному регулярному выражению. Например, если r — это регулярное выражение, то регулярному выражению r? будет соответствовать пустая строка (0 раз r) или строка, соответствующая r (один раз r). Аналогично, операция $*$ означает ноль или более повторений своего параметра-регулярного выражения; + — одно и более повторений. Можно даже явно указывать диапазон требуемого числа повторений при помощи операции $\{m,n\}$, где m и n — границы числа повторений. При использовании этой конструкции вполне возможно, что разным повторениям регулярного выражения будут соответствовать разные строки. Например, строка abc соответствует регулярному выражению [abc]+ (первому повторению регулярного выражения [abc] соответствует a, второму — b и третьему — c). Приоритет операций, задающих число повторений той или иной конструкции, выше, чем у конкатенации, так что регулярное выражение ab+ соответствует строкам ab, abb, abbb и т. д., а не ab, abab, ababab и т. д.

Пример.

Создайте текстовый файл g.txt , содержащий следующий текст:

```
xza
xyz
xyyz
xyyyza
xyyyyza
tyyyyza
xyyyyta
zxdfa
```

Что выведет команда grep 'xy*z' g.txt ? А команда grep 'xy\+z' g.txt ? А команда grep -x 'xy*z' g.txt ? Почему? Что выведет команда grep 'xy\{2,3\}z' g.txt ?

е) Конструкция выбора альтернативы |. Регулярное выражение, составленное из других при помощи этой конструкции, соответствует строкам, соответствующим любой из альтернатив. Например, a|b|c — это то же самое, что и

[abc]. Приоритет у этой операции ниже, чем у конкатенации, так что регулярное выражение begin|end соответствует строкам begin и end, а не beginnd и begiend.

Пример.

Создайте текстовый файл h.txt , содержащий следующий текст:

```
one tree
two trees
three apples
four plums
five rabbits
six wires
seven dolls
eight balls
```

Напишите команду, выдающую строки, содержащие текст tree или ball, тремя способами. Подсказка: можно использовать конструкцию выбора альтернативы, два регулярных выражения в одной командной строке или чтение регулярных выражений из файла. Увы, в языке регулярных выражений, понимаемом большинством программ для Linux, нет операции «и», т. е. нельзя потребовать, чтобы текст соответствовал нескольким регулярным выражениям сразу. Обойдя эту трудность, напишите команду, выдающую на экран те строки файла h.txt, которые содержат как букву a, так и букву s, в любом месте строки и в любом порядке. Подсказка: можно воспользоваться операцией выбора альтернативы, а можно — программным конвейером.

ж) Операция группирования. Часто при составлении регулярных выражений возникает необходимость использовать операции не в том порядке, в котором указывают их приоритеты. Для решения этой проблемы можно использовать операцию группирования (круглые скобки). Например, регулярное выражение (begin)+ соответствует строкам из нескольких идущих подряд друг за другом слов begin. Также операция группирования используется для ограничения области действия конструкции выбора альтернативы. Например, abc(de|fg)hij соответствует abcdehij или abcfghij. Еще операция группирования используется для конструкции обратных ссылок, чему посвящен следующий пункт.

Пример.

Создайте текстовый файл i.txt , содержащий следующий текст:

```
beginbeginbegin
beginendbegin
endendendend
a line end
begin++++++end
begin of the text
```

Напишите команду, выдающую на экран строки, состоящие только из повторений слова begin . Также, выдайте на экран строки, состоящие из повторений слова begin или слова end (смесь тех и других не допускается). Наконец, выдайте на экран строки, состоящие из нескольких повторений слова begin или end, причем в одной строке допустимо чередовать и то, и другое слово в любых сочетаниях.

з) Так называемая обратная ссылка. Конструкция, состоящая из символа обратной косой черты, за которой идет цифра от 1 до 9, означает фрагмент текста, соответствующий группе с указанным номером. Группы — это фрагменты регулярного выражения, ограниченные при помощи операции группирования. Они нумеруются подряд в порядке следования открывающих скобок. Например, регулярное выражение '(.)\1' соответствует паре одинаковых символов. Эта конструкция удобна, но выводит за рамки традиционного понимания регулярных выражений, т. е. выражений, описывающих языки, которые распознаются конечными автоматами.

Пример.

Для файла f.txt из предыдущих примеров напишите команду, выдающую на экран строки, содержащие два раза подряд одну и ту же пару символов, и убедитесь в ее работоспособности. Также, напишите команду, содержащую подряд пару символов и затем ту же пару символов, но в обратном порядке. Проверьте.

3.2. Сортировка

Сортировка, или упорядочение строк текстового файла в соответствии с некоторым порядком, имеет важное значение в базах данных. Чаще всего используются лексикографический порядок (как слова в словаре) для текста или обычный для чисел.

Для сортировки текстовых файлов используется программа sort. Она имеет следующую командную строку:

```
sort опции имена-файлов
```

Такая команда сортирует строки входных файлов, если их нет — своего стандартного ввода, и по порядку выдает их на стандартный вывод.

Обычно сортировка производится по всей длине строки в алфавитном порядке, но есть опции, позволяющие производить сортировку по определенным полям (поля разделяются переходом от непробельных символов к пробельным, но можно задать разделитель явно), а также сортировать по численному значению полей. Также существуют опции для удаления строк-дубликатов.

Опции программы sort делятся на два класса: общие опции и опции сортировки.

Перечислим основные общие опции программы:

-o имя-файла задает выходной файл (вместо стандартного потока вывода).

-s делает сортировку устойчивой, т. е. порядок строк, все поля которых совпадают, сохраняется неизменным. Может показаться бессмысленной, но сортировка может производится не по полному набору полей.

-t символ-разделитель использует указанный символ для разделения полей.

-u выводить только одну из набора совпадающих строк.

-k позиция1,позиция2 указывает на часть поля (или полей), по которой происходит сортировка. Позиция в данной опции выглядит как номер-поля.номер-символа-внутри-поля. Кроме того, к позиции можно добавлять буквы из набора Mbdfinr, определяющие способ сортировки (см. опции сравнения).

К основным опциям сравнения относятся:

-b — убрать ведущие пробелы.

-d — при сравнении строк обращать внимание только на буквы, цифры и пробелы.

-f — не различать большие и маленькие буквы.

-g — превращать начало каждой строки в вещественное число и сортировать по полученным значениям.

-n — численная сортировка (начало каждой строки рассматривается как число, однако превращения в вещественное число не происходит, из-за чего работает гораздо быстрее предыдущей опции).

-i — не обращать внимание на непечатные символы.

-M — сортировать названия месяцев по порядку месяцев (зависит от выбранного языка); нераспознанные значения считаются меньше любого месяца.

-r — сортировка в обратном порядке (т. е. по убыванию).

Пример.

Отсортируйте файл /etc/passwd , содержащий список известных системе пользователей, по имени пользователя. Отсортируйте вывод команды ls по возрастанию (убыванию) длин файлов. Отсортируйте вывод команды df (информация о свободном пространстве на подключенных файловых системах) по объему файловой системы (подсказка: потребуется еще одна команда в конвейере, чтобы убрать заголовок).

3.3. Соединение нескольких файлов

Важной операцией над базами данных является операция соединения двух файлов. Эта операция важна, поскольку хранение одной базы данных в нескольких связанных по смыслу файлах (в терминологии баз данных, таблицах) позволяет в большинстве случаев избавиться от избыточной, т. е. присутствующей в базе неоднократно, информации. Хранение избыточной информации опасно, поскольку оно провоцирует ошибки, в первую очередь противоречивость данных — например, если обновляется только одна запись из многих, хранящих одинаковую информацию.

Поэтому для избежания избыточности набор файлов базы данных нормализуется, т. е. последовательно приводится к очередной нормальной форме. Всего их пять, и чем больше номер, тем меньше избыточность; пятая нормальная форма используется очень редко, поскольку опирается на весьма нетривиальные зависимости между данными.

Однако, при работе с базами данных иногда возникает необходимость собрать одну таблицу из нескольких файлов, в совокупности содержащих требуемую информацию. Для этого в Linux предусмотрена специальная команда join, соединяющая две таблицы в одну.

Команда join имеет следующую командную строку:

join опции имя_файла1 имя_файла2

Действие этой команды состоит в том, что на стандартный вывод выдаются все строки, которые получаются объединением одной строки первого файла и одной строки второго; строки можно объединить, если значение фиксированного поля одной строки совпадает со значением фиксированного поля другой строки. Номера сравниваемых полей в обоих файлах задаются опциями. Для правильной работы этой команды оба входных файла должны быть отсортированы по значениям этих полей.

Рассмотрим теперь более подробно наиболее важные опции этой команды:

-i Не обращать внимания на различие между большими и маленькими буквами.

-1 номер Задать номер сравниваемого поля в первом файле, по умолчанию 1 (поля нумеруются с 1).

-2 номер То же самое для второго файла.

-j номер То же, что обе предыдущие опции с указанным номером.

-t символ Установить разделитель полей (по умолчанию пробел). Используется в обоих входных файлах и для разделения полей результата.

-o формат Использовать указанный формат для построения выходных строк. Формат — это список спецификаций вроде номер-файла.номер-поля через пробел или запятую. По умолчанию выводится значение сравниваемого поля, затем остальных полей из соответствующей строки первого файла, затем — второго.

Пример.

Создайте текстовый файл colors.txt , содержащий следующий текст:

```
apple red
pear green
grape green
plum yellow
orange orange
carrot red
cucumber green
cabbage white
```

и файл price.txt , содержащий следующий текст:

```
apple 0.25$
pear 0.20$
grape 0.45$
plum 0.05$
orange 0.50$
carrot 0.10$
cucumber 0.15$
cabbage 0.8$
```

Отсортируйте оба этих файла по первому полю, и соедините обе таблицы в одну при помощи команды join.

3.4. Манипуляции — sed

Утилита sed — это так называемый потоковый редактор; он позволяет редактировать проходящий через него поток информации при помощи определенного набора команд в пакетном (т. е. не интерактивном) режиме. Он имеет два буфера для данных, которые называются пространствами шаблонов и задержки соответственно. Принцип его работы состоит в следующем: очередная строка копируется в пространство шаблонов, после чего выполняются команды редактирования. Каждая команда редактирования может иметь определенное условие на входную строку; выполняются только те команды, условия которых истинны. После этого (если нет опции -n) содержимое пространства шаблонов записывается в стандартный поток вывода. Наконец, пространство шаблонов очищается, и начинается следующий цикл. При этом та информация, которую необходимо сохранить между циклами, должна быть помещаема в пространство задержек, так как его содержимое не очищается в момент начала нового цикла.

Командная строка при вызове sed должна содержать опции, скрипт с командами редактирования, а также имена входных файлов (если информация читается из файлов, а не из стандартного ввода). Рассмотрим сначала наиболее употребительные опции программы sed:

-n Запрещает автоматическую передачу содержимого пространства шаблонов в стандартный вывод в конце цикла редактирования очередной строки (т. е. такая передача, если она необходима, должна осуществляться явно, например при помощи команды p).

-l число Задает максимальную длину строки для разрезания длинных строк. По умолчанию используется значение 70. Значение 0 означает «никогда не разрезать строки».

-r Указывает на использование расширенных регулярных выражений (по умолчанию используются базовые).

-e командное-выражение Требует добавить соответствующие команды из выражения в набор команд редактирования. Полезно, если команды начинаются с -. Также позволяет иметь несколько команд в одной строке. Другой способ использовать скрипты из нескольких команд — воспользоваться апострофами и разделять команды переходом на следующую строку.

-f имя-файла — то же, что и предыдущая опция, но команды берутся из файла с указанным именем. Еще один способ иметь скрипт редактирования из нескольких команд.

Как уже говорилось выше, каждая команда может иметь условия, при которых она выполняется. Эти условия в зависимости от команды могут быть трех типов:

- безадресные, т. е. безусловные, команды;
- одноадресные команды. Такой команде может предшествовать так называемый адрес или условие одного из следующих типов: число — истинно только для строки с указанным номером, первая шаг — истинно для всех строк, начиная с первой через шаг (например, конструкция 1~2 соответствует строкам с нечетными номерами, и команда 1~2p будет печатать строки с нечетными номерами), $ — истинно для последней строки ($p напечатает последнюю строку), /регулярное-выражение/ — истинно для строк, имеющих подстроки, соответствующие регулярному выражению (конструкция /z/p будет печатать строки, содержащие букву z), \cрегулярное-выражениес, где с можно заменить на любой одиночный символ, — то же, но позволяет использовать символ / в регулярном выражении;

- двухадресные команды, которые понимают диапазон из двух адресов (через «,»). Например, конструкция 10,$p напечатает строки с 10-й и до конца.

Кроме вышеперечисленных конструкций, между адресами и командой можно поставить восклицательный знак, что меняет смысл адреса на противоположный: в этом случае команда будет выполняться для строк, не соответствующих адресу. Теперь рассмотрим наиболее употребительные команды редактирования:

- # начинает комментарий, продолжающийся до конца строки.
- q — одноадресная команда; завершает работу sed (но если нет опции -n, то содержимое пространства шаблонов печатается).
- d очищает пространство шаблонов и немедленно начинает новый цикл (т. е. редактирование очередной строки).
- p выводит содержимое пространства шаблонов. Например, командное выражение p (без опции -n) удваивает каждую строку файла.
- s/регулярное-выражение/замена/флаги заменяет подстроки в пространстве шаблонов, соответствующие регулярному выражению, на текст, указанный в поле замена, в соответствии со значениями флагов. При этом вместо символа / (во всех трех вхождениях) можно использовать любой другой. Текст замены может содержать конструкцию обратной ссылки (см. раздел про grep), а также символ &, обозначающий весь текст, соответствующий регулярному выражению. Флаги могут быть такими:
  - g — применить замену ко всем подстрокам, соответствующим регулярному выражению (по умолчанию замена производится только для первой такой подстроки);
  - число — заменить подстроку с указанным номером;
  - командные флаги — одна из команд p, w, e, y. В этом случае указанная команда выполняется для данной строки, если замена была произведена успешно;
  - i включает режим безразличия к регистру символов;
  - m включает специальное значение символов ^ и $ в регулярном выражении.

  Команда s является основным инструментом при программировании на sed.
- астроки — строки, следующие за командой a (каждая, кроме последней, должна кончаться обратной косой чертой) помещаются в очередь и выводятся после обработки текущей строки, но до обработки следующей.
- істроки — то же, что и предыдущая команда, но строки выводятся немедленно.
- сстроки удаляет строки, указанные адресами, и выводит строки-параметр вместо последней из них.
- = печатает номер текущей строки и символ перевода строки. Может быть использовано для порождения листингов, т. е. добавления к каждой строке спереди ее номера, однако после использования этой команды требуются дополнительные усилия для склеивания пар подряд идущих строк.
- r имя-файла выводит содержимое указанного файла между текущим и следующим циклами. Увы, вставить содержимое файла до текущей строки невозможно; в частности, это означает, что нельзя вставить содержимое одного файла в самое начало другого. Если это необходимо, нужно добавить лишнюю первую строку, затем выполнить для нее команду r и, наконец, убрать добавленную первую строку.
- w имя-файла пишет содержимое пространства шаблонов в указанный файл. Например, программа

  ```
  1~2w odd.txt
  2~2w even.txt
  ```

  записывает строки с четными номерами в файл even.txt , а с нечетными — в odd.txt .
- e — командная подстановка (содержимое пространства шаблонов трактуется как команда для командного интерпретатора и заменяется результатом этой команды). Очень полезно для программирования, например, арифметических действий. Например, если в пространстве шаблонов записано целое число и надо добавить к нему, например, единицу, можно воспользоваться командой s, чтобы заменить это число на вызов команды expr, добавляющий к этому числу 1, и затем воспользоваться командой e;
- y/исходные-символы/символы-замены/ заменяет символы первой из указанных строк на символы второй с таким же номером (транслитерация). К сожалению, указывать диапазоны символов нельзя. Например, команда y/ABCDEF/abcdef/ заменит все вхождения первых шести букв английского алфавита в верхнем регистре на такие же буквы в нижнем. Если нужно подвергнуть такому преобразованию все буквы, придется полностью выписать весь алфавит два раза (большие и маленькие буквы).

- D удаляет текст из пространства шаблонов до первого символа перевода строки. Если что-то осталось, запускается новый цикл без дополнительного чтения входной строки, иначе новый цикл запускается обычным образом.
- N добавляет к содержимому пространства шаблонов символ перевода строки и затем очередную строку, прочитанную из ввода (стандартного ввода или указанных в командной строке файлов). Полезно, например, для склеивания подряд стоящих строк — см. комментарий к команде = .
- P печатает часть текста из пространства шаблонов до первого символа перевода строки.
- h переписать пространство шаблонов в пространство задержки (предыдущее содержимое пространства задержки теряется).
- H дописать в пространство задержки символ перевода строки и затем содержимое пространства шаблонов.
- g и G — аналогично двум предыдущим командам, но в другую сторону (из пространства задержки в пространство шаблонов).
- x меняет местами содержимое пространств задержки и шаблонов.
- : метка задает расположение метки в наборе команд (безадресная команда).
- b метка означает безусловный переход к метке (если метка опущена, начинается новый цикл).
- t метка означает условный переход к метке. Переход происходит, если после загрузки последней строки или предыдущего условного перехода была успешная замена по команде s.
- Последовательность команд может быть заключена в фигурные скобки. Это полезно, например, для использования одного и того же адреса у многих команд.

Примеры.

Напишите команду, печатающую а) первые 15 строк б) каждую третью строку в) строки, начиная с 25-й. Напишите команду, заменяющую все вхождения слова tree на bush: а) по всему файлу б) в строках с 50-й по 100-ю в) в строках, содержащих слово cut . Напишите команду, заменяющую букву а в начале каждого слова на большую. Напишите команду, принимающую на стандартный ввод набор чисел (по одному на строке) и выдающую набор чисел, полученных из исходных умножением каждого на три. Напишите команду, сортирующую вывод команды df по возрастанию размера свободного пространства на файловых системах, с правильной обработкой заголовка (подсказка: заголовок можно сохранить в файле, пока sort сортирует все остальное). Напишите программу, добавляющую к каждой строке спереди ее номер. Напишите команду, выдающую строки исходного текста (файла или результатов работы предыдущей команды), выровненные по вхождению слова begin, т. е. выводятся только строки, содержащие begin, не ранее 11-го столбца, причем до этого слова должно быть выведено ровно 10 символов (все предыдущие символы, если такие были, выбрасываются). Напишите команду, исправляющую десятичную запятую во входном тексте на десятичную точку (можно считать, что числа во входном тексте представляют собой отдельные слова). Последний пример носит сугубо практический характер: именно по причине нестандартной записи десятичных дробей система печати для Linux отказывалась печатать файлы, создаваемые детским графическим редактором TuxPaint.

Из приведенных примеров ясно, что sed — мощное средство, однако, поскольку его использование основано на небольшом количестве фиксированных команд, и набор управляющих структур крайне беден (только безусловный и условный переход по метке), пользоваться sed для решения сложных задач затруднительно.

3.5. Более сложные манипуляции — awk

awk представляет собой обобщение sed; сохраняя ту же цель, он представляет собой полномасштабный язык программирования.

awk изначально ориентирован на обработку текстовых баз данных. Это означает, что в нем присутствуют средства для обработки записей (в терминологии баз данных) и составляющих их полей. Обычно, в большинстве текстовых баз данных, записями являются строки, которые делятся на поля либо группами подряд идущих пробельных символов, либо некоторым специальным символом. Однако, для удобства программирования, awk поддерживает режим, в котором полями являются строки, а записи (группы полей) отделяются друг от друга пустой строкой.

Командная строка awk выглядит так:

```
awk опции программа входные-файлы
```

Среди опций можно отметить:

-F разделитель-полей определяет символ, используемый для разделения полей в пределах одной записи.

-f файл означает, что программа должна быть взята из указанного файла, а не из соответствующего аргумента командной строки.

-v имя=значение выполнить указанное присваивание до начала работы программы.

Программа выглядит как набор элементов, разделенных переводом строки. Элементы бывают двух типов — определения пользовательских функций и пары образец действие . Образец определяет, когда необходимо выполнить соответствующее действие, и может иметь следующий вид:

BEGIN соответствующее действие выполняется в самом начале работы программы, до обработки входных файлов. Обычно используется для инициализации процесса обработки, например для присваивания начальных значений переменным. Например, если нам нужно вычислить сумму значений некоторого поля, здесь уместно присвоить той переменной, в которой накапливается сумма, нулевое значение.

END действие выполняется в конце работы программы, после обработки входных файлов. Обычно используется для завершения процесса обработки, например для вывода результатов, содержащих обобщающую информацию по всей совокупности входных данных. Например, если нам нужно вычислить сумму значений некоторого поля, здесь уместно вывести ее окончательное значение.

Остальные образцы служат для определения тех записей, для которых выполняются действия, и могут иметь вид:

пусто означает, что действие выполняется для любой записи.

/регулярное-выражение/ означает, что действие выполняется для тех записей, которые содержат текст, удовлетворяющий указанному регулярному выражению.

выражение может быть любым допустимым в awk выражением, и действие выполняется для тех записей, которые удовлетворяют указанному выражению (т. е. на которых оно истинно). Например, выражение NR==20 означает 20-ю запись (забегая вперед, NR — встроенная переменная, которая означает номер текущей записи).

образец1, образец2 означает, что действие выполняется для диапазона записей, определяемого образцами, включительно. Лучше, однако, пользоваться и для этой цели средством, описанным в предыдущем пункте.

Язык, на котором пишутся действия, отдаленно напоминает C, однако, если синтаксис похож, то семантика (по крайней мере выражений) резко отличается.

В awk имеются следующие типы данных:

а) простые переменные, которые могут содержать числа (в awk все числа рассматриваются как вещественные, целые числа — частный случай, отдельный тип для них не предусмотрен) или строки, причем один тип автоматически преобразуется в другой по мере необходимости. Например, если к строкам применяется операция сложения, они автоматически преобразуются в числа, и наоборот, если к числам применяется операция конкатенации строк, они автоматически преобразуются в строки.

б) ассоциативные массивы, доступ к элементам которых осуществляется через [] (как в C, но индексами могут быть не только числа, но и строки). Возможны многомерные массивы, если в [] через ≪,≫ указаны несколько индексов.

В awk нет необходимости объявлять переменные, они создаются при первом присваивании. В отличие от BASH, для извлечения значения переменной достаточно указать ее имя; также, присваивание выглядит как в C:

```
имя = значение
```

Также имеется несколько предопределенных переменных, например, NF — число полей в текущей записи или NR — число считанных к данному моменту записей.

В качестве операндов в выражениях, кроме переменных и констант, допустимо использовать значения полей текущей записи. Для ссылки на них предусмотрена конструкция $номер, где номер — номер поля (поля нумеруются с 1) или 0, что означает всю текущую запись целиком. При этом номер в этой конструкции не обязан быть константой — он может быть результатом любого выражения, возвращающего целое число, и в этом выражении вполне могут участвовать значения других полей текущей записи.

Набор операций содержит обычные арифметические операции, возведение в степень ** и конкатенацию строк (эта операция не имеет значка и вызывается, если подряд идут два операнда). Деление двух целых чисел дает вещественный результат. Также, операция взятия остатка от деления % определяется для вещественных чисел (a%b=a-int(a/b)*b). Здесь int(...) — функция, превращающая число в целое путем отбрасывания того, что стоит после ≪.≫.

Кроме того, имеются операции сравнения (если хотя бы один их операнд представляет собой строку, которую нельзя превратить в число, сравнение будет строковым, иначе — числовым). Также имеются операции ~ и !~, проверяющие соответствие строки (левый операнд) регулярному выражению (правый операнд; должен быть константой типа регулярное выражение, т. е. строка вида /регулярное выражение/). Наконец, имеется операция in, проверяющая наличие индекса в ассоциативном массиве.

Набор операторов напоминает C, но есть и отличия. ≪;≫ разделяет операторы; также можно разделять операторы переходом на новую строку. Для вывода информации используется оператор print, в котором через ≪,≫ указываются выводимые строки. При выводе они автоматически разделяются символом-разделителем полей. Можно также перенаправить вывод при помощи стандартных средств >, >> в файл и | — в канал некоторой команде. Имеется также оператор printf, параметрами которого являются строка формата (смысл — как в C) и список выводимых значений. В отличие от print, оператор printf не выводит ничего лишнего автоматически, так что даже символ перевода строки в конце, если он нужен, должен быть указан явно.

Составной, условный оператор и операторы цикла напоминают язык C. Как и в C, условием может быть любое выражение. Если это число, то условие считается истинным, если оно не 0. Если это строка, то проверяется, что она не пустая, в том числе строковая константа "0"считается истинным условием.

Кроме того, есть еще одна форма оператора for для перебора всех элементов массива:

```
for (i in массив) тело
```

Также, в версии gawk (GNU awk), скомпилированной с поддержкой switch, имеется этот оператор. Отличия от C состоят в возможности указания в качестве параметра case не только чисел, но и строк и даже регулярных выражений.

Кроме операций, в gawk имеется большое количество встроенных функций для работы с числами, строками и файлами, а также возможность определять собственные функции.

Из встроенных функций уместно упомянуть следующие:

Числовые — int (отбрасывает цифры после «.»), rand — случайные числа, а также полный набор элементарных функций (как в C).

Строковые — index (ищет индекс первого вхождения строки-второго параметра в первый, индексы начинаются с 1), length, match(строка, регулярное-выражение, массив) — регулярное выражение может быть константой типа регулярное выражение или строкой, в массив (если он задан) записываются по индексу 0 — весь соответствующий текст, n — текст, соответствующий n-ой группе, split(строка, массив, разделитель-полей) — делит строку на части и записывает их в массив по индексам 1,..., sub(регулярное-выражение, замена, строка) — заменяет текст в строке (один раз), gsub — то же, но все неперекрывающиеся вхождения, gensub — то же, но добавляется третий числовой аргумент (перед строкой), указывающий номер заменяемого вхождения, substr(строка, начало, длина).

Еще одна важная функция — system (как в C), она позволяет выполнить команду, текст которой содержится в строке-параметре.

Синтаксис определения пользовательских функций следующий:

```
function имя(список-параметров) { тело }
```

Возврат значения из функции, как в C, производится оператором return выражение.

В качестве примера рассмотрим следующую задачу: имеется файл, в каждой строке которого записан URL некоторого файла и через « » размер этого файла (в Мб, но без суффикса «M»). Нужно скачать все эти файлы, однако их общий размер превышает дисковую квоту (для примера 500Мб). Таким образом, нужно разбить список на несколько частей, в каждой из которых суммарный объем файлов не превышает квоту.

Пример.

Что делает команда

```
awk '{ print $3, $2, $1 }' f.txt
```

Напишите команду, которая выдает информацию об имеющихся в текущей папке файлах (по одному на строку) в следующем виде: имя-файла длина пользователь-хозяин-файла. Напишите программу, выдающую суммарную длину всех имеющихся в текущей папке файлов. Напишите программу, получающую на вход набор строк, каждая из которых содержит три числа, и выдающую номера тех строк, числа в которых упорядочены по возрастанию. Под квадратной матрицей будем понимать набор из некоторого числа строк, каждая из которых содержит одинаковое число чисел, разделенных пробелами, причем количество чисел в каждой строке равно числу строк.Напишите программу, получающую на вход квадратную матрицу, и выдающую ее след, т. е. сумму диагональных элементов (диагональными считаются те элементы, у которых номер строки, в которой они встречаются, равен их номеру в этой строке). Напишите программу, которая записывает каждую входную строку в свой файл, имя которого образуется из слова file, за которым следует номер строки, и затем расширение .txt . Напишите программу, которая перемножает числа в каждой входной строке и все полученные произведения складывает. Можно считать, что строки, подаваемые на вход, содержат только числа, разделенные пробелами, но ни количество входных строк, ни количество чисел в каждой строке, а оно может быть разным у разных строк, заранее неизвестны.

4. Текстовый редактор vi

4.1. Получение информации о vi из него самого

Для запуска системы встроенной справки необходимо нажать F1 или набрать строку «:h», завершив ее Enter. Если vi запущен в эмуляторе терминала gnome terminal, то нажатие F1 воспринимается терминалом как команда себе, и приводит к появлению справки по терминалу.

Запустив систему справки, Вы оказываетесь в главном файле справки на первой странице; ее содержимое описывает, как перемещаться по файлам справки. Листать файл справки можно, как обычно, клавишами «PageUp» и «PageDown», для перемещения курсора работают клавиши со стрелками. Кроме того, файлы справки объединены по принципу гипертекста; ссылки называются ярлыками (tag) и выделяются с двух сторон символами | или цветом. Но даже если символы | скрыты, они становятся видны, когда курсор попадает на то место, где они должны быть. Для перехода по ссылке необходимо поставить на нее курсор и нажать C-] (здесь и далее для обозначения комбинаций клавиш используется традиция emacs, в которой C- означает control-, M- означает meta- (alt-), и для обозначения остальных модификаторов (shift- или русский регистр) используется сам символ).

Можно перемещаться по ссылкам и при помощи мыши (два щелчка левой кнопкой на ярлыке), для чего необходимо дать команду «:set mouse=a» Enter. Для возврата обратно нужно нажать C-T или C-O (если vi запущен из-под mc, то mc перехватывает C-O).

Можно также попытаться нажать C-], когда курсор стоит и на обычном слове (не ярлыке), и vi попытается найти информацию о нем. Если попытка неуспешна, в нижней строке экрана отображается сообщение об ошибке.

Кроме того, можно получить справку по интересующей теме, задав аргумент у команды «:h». Например, справка по команде «i» в командном режиме выдается командой «:h i», а справка по команде C-w (также в командном режиме) — командой «:h CTRL-W». Если команда состоит из нескольких комбинаций клавиш, нажимаемых последовательно, они разделяются символом «_». Также, допустимы несколько префиксов, указывающих на тот режим, к которому относится интересующая нас команда. Например, префикс «v_» означает визуальный режим, а префикс «i_» — режим вставки (полный перечень префиксов — на второй странице главного файла справки, доступного по команде «:h»). Некоторые клавиши, входящие в состав команд, обозначаются как < ... >, например, <Enter>. Полный список таких обозначений можно найти в файле «intro.txt.gz» (команда «:h intro»), перейдя на ярлык «notation» (пункт 4 на первой странице). В команде «:h» можно использовать и другой текст, который необходимо искать в файлах справки. В том числе, таким образом можно найти развернутое описание ошибок по их кодам.

Весь справочный материал разбит на следующие разделы:

1) Основы — здесь есть файлы quickref (краткая справка по наиболее употребительным командам) и tutor (получасовой курс для начинающих).

2) Руководство пользователя — описание того, как выполнить те или иные задачи редактирования.

3) Справочное руководство — подробное описание всех деталей vi.

4.2. Режимы и общие замечания о командах

Редактор vi имеет ряд режимов, в зависимости от которых меняется трактовка вводимых символов. При обычном входе в редактор пользователь попадает в командный режим, в котором все вводимые символы трактуются как команды, а не как символы, вставляемые в текст. При этом различаются обычные команды (такие, как 'x' для удаления символа, на котором стоит курсор) и команды, начинающиеся с символа ':'. При нажатии ':' курсор перемещается в последнюю строку экрана; здесь можно ввести имя команды и ее параметры (если необходимо), и завершить команду нажатием 'Enter'. С одной из таких команд (h) мы уже встречались в предыдущем параграфе.

Подавляющее большинство команд могут иметь численный аргумент. Для обычных команд это число набирается непосредственно перед командой, для команд, начинающихся с «:» — между «:» и командой. Обычно, аргумент означает число повторений команды, но существуют команды, для которых присутствие аргумента радикально меняет смысл команды.

Кроме командного режима, существует также режим вставки текста. В этом режиме обычные символы воспринимаются как текст, который вставляется в редактируемый файл. Для удобства пользователей, в режиме вставки в последней строке экрана отображается текст, указывающий на то, что редактор находится в этом режиме.

Для перехода из обычного режима в режим вставки существует целый набор команд. Например, команда 'i' начинает вставку текста перед тем символом, на котором находится курсор, а команда 'a' — после.

Для того, чтобы из режима вставки вернуться в командный режим, нужно нажать Esc.

Существует еще визуальный режим — режим, предназначенный для выделения части текста. Vim умеет выделять последовательные куски текста, а также прямоугольники. Основные команды визуального режима такие же, как и у

командного, но есть и отличия; они прежде всего касаются операций с выделенным куском текста. Выделение текста мышью автоматически переводит vim в визуальный режим.

4.3. Открытие, запись и закрытие файла в vi

В графическом интерфейсе (gvim) для открытия файла предусмотрена специальная кнопка на панели инструментов. Нажатие ее приводит к появлению диалога открытия файла, в котором обычным способом можно выбрать директорий и имя файла.

Также можно указать имя файла в командной строке. Еще один способ — воспользоваться командой :edit имя-файла. В последнем случае, если до этой команды редактировался другой файл, он закрывается. Если же в нем были несохраненные изменения, команда не работает, однако можно воспользоваться командой :edit! имя-файла. Как и во многих других случаях, ! означает принудительное выполнение команды. Также можно временно спрятать редактируемый файл, открыв новый командой :hide edit имя-файла.

Vim позволяет иметь список из нескольких файлов для редактирования. Простейший способ сделать это — задать несколько имен файлов в командной строке. При этом отображается на экране первый файл, и для перехода к другим существуют команды :next, :previous, :first, :last с очевидным смыслом. Они работают также, как и :edit. Можно изменить список редактируемых файлов, воспользовавшись командой :args новый-список. Если эта команда вызвана без параметров, она показывает текущий список, в котором текущий файл выделен «[]».

После работы с файлом записать его можно командой «:w». Эта команда допускает аргумент — имя файла, куда записывать текст. В отличие от большинства других редакторов, где подобная операция меняет имя файла, в vi имя файла остается старым. Закрыть файл можно командой «:q». Сделать то и другое одновременно можно командой «:wq», а закрыть файл без записи изменений — командой «:q!».

4.4. Окна

Vim позволяет также редактировать несколько файлов одновременно, разделяя окно терминала вертикально или горизонтально на несколько окон редактирования. Одновременно активным может быть лишь одно такое окно, и оно получает вводимые символы или команды. Также некоторые файлы могут быть спрятаны, т. е. их буфера со всеми изменениями находятся в памяти, однако они не показаны на экране (нет соответствующих окон).

Для открытия дополнительных окон имеются команды :split имя-файла и :vsplit имя-файла. Они делят текущее окно на два (одно под другим или одно справа, другое слева соответственно) и открывают в новом окне указанный файл. Если имя файла не указано, в новом окне открывается тот же файл, который был в старом, позволяя иметь несколько окон для редактирования одного и того же файла.

Можно открыть новое окно с пустым, безымянным файлом командами «:new» и «:vnew»; при первой записи такого файла необходимо указать его имя.

Численный аргумент этих команд означает размер нового окна (число строк или столбцов в зависимости от способа разбиения текущего окна).

Закрыть одно из окон (и убрать соответствующий файл из памяти) можно, как обычно, командой «:q» (см. пред. раздел), но можно и не убирать файл из памяти, а только «спрятать» его командой «:hid». Все спрятанные окна можно вернуть обратно командой «:unh». К сожалению, нельзя возвращать файлы по-одному, указывая их имена. Также, вид окон после этого может отличаться от того, который был до прятания файлов; но можно указать максимальное количество окон, которое может быть открыто, в численном аргументе.

Закрыть все окна, кроме текущего, можно командой «:on».

Для более тонкой работы с окнами имеется целая группа обычных команд, начинающихся с C-w.

Например, команда «C-w +» увеличивает размер окна на 1 (строку или столбец в зависимости от стиля разбиения), а «C-w −» — уменьшает; «C-w _» увеличивает размер по максимуму, а с численным аргументом — устанавливает его равным аргументу.

«C-w ↑» перемещает пользователя в окно, находящееся над данным; аналогично, «C-w ↓» — в окно под данным, и т. д.

Vi позволяет также перемещать окна. «C-w K» перемещает окно на самый верх (работает даже в том случае, если применяется вертикальное разделение окон; тогда окно помещается выше всех остальных и занимает всю ширину), «C-w J» — в самый низ, «C-w H» делает окно самым левым, «C-w L» — самым правым. Выбор букв, используемых в этих командах, объясняется тем, что в vi соответствующие маленькие буквы можно использовать для перемещения курсора в командном режиме так же, как и клавиши со стрелками. Vi — не единственная программа в UNIX, использующая такое соглашение — в качестве другого примера можно назвать терминальную версию NetHack.

4.5. Перемещения в vi

В vi существует огромное количество команд для перемещений по файлу. Кроме стандартных клавиш со стрелками, «Home», «End», «PageUp», «PageDown» и несколько менее стандартных «C-←» и «C-→» (слово назад и вперед), а «C-Home», «C-PageUp», ... не работают ожидаемым образом, существуют следующие команды перемещения.

Некоторые из упомянутых команд можно предварить префиксом g. Тогда они будут относиться не к строкам файла, а к строкам экрана — это существенно, когда одна строка файла занимает несколько строк экрана. Например, «↑» — переход в позицию с тем же номером (или ближайшую к ней) в предыдущей строке файла, а «g↑» — то же, но в предыдущей строке экрана.

Команда ^ ставит курсор на первый непробельный символ строки, а команда | — на первый символ строки. Отличие ее от «Home» в том, что можно задать номер столбца в численном аргументе.

Команда «f» требует после себя еще один символ, и ставит курсор на n-ое вхождение указанного символа вправо от текущей позиции (n — значение численного аргумента). Команда «F» — аналогично влево. Команда «;» повторяет последнюю команду такого сорта, а «,» — повторяет в противоположном направлении.

Команды gg и G с численным аргументом переходят на строку с указанным номером; разница между ними в том, что без аргумента первая команда переходит на первую строку, а вторая — на последнюю. Обе команды оставляют курсор на первом непробельном символе строки. Команда % с численным аргументом вызывает переход на строку, находящуюся в файле на указанном проценте его длины (в числе строк, а не символов). Т. е. если в файле 10 строк, то независимо от их длин «20%» вызывает переход на вторую строку файла.

Команды «)» и «(» позволяют перемещаться на указанное в аргументе число предложений вперед и назад, соответственно. Предложение распознается как текст, кончающийся на «.», «!» или «?», за которым, возможно, следуют закрывающие конструкции (скобки или кавычки), и затем пробел. Можно также перемещаться по абзацам (разделенным пустой строкой или соответствующими макросами troff) при помощи команд «}» и «{».

Имеются также специальные команды для редактирования программ на C/C++ и Java: перемещение на указанное число не закрытых «(» (команда «[(», «)» («])»), «{» («[{»), «}» («]}»), #if или #else («[#»), #else или #endif («]#»). Также имеются команды перемещения к началу или концу метода (для Java) вперед или назад.

Имеются команды поиска образца: «/образец <Enter> » — искать образец вперед по тексту, то же, но с ? вместо / — искать назад, то же без образца — повторить поиск в направлении, определяемом символом («/» — вперед, «?» — назад), «n» — повторить последний поиск, «N» — повторить последний поиск в обратном направлении. Здесь образец — регулярное выражение, определяющее искомый текст. Однако, правила его составления существенно отличаются от тех, которые используются в grep и awk. Исчерпывающее их изложение может быть найдено в «:h pattern». Здесь отметим только весьма удобную конструкцию, отсутствующую в grep, а именно операцию «и» для регулярных выражений, означающую, что искомый текст обязан соответствовать обоим регулярным выражениям.

«%» без числового аргумента ищет в данной строке следующую конструкцию, у которой может быть пара (например, скобку или начало/конец комментария) и переходит к этой паре.

Наконец, vi позволяет запоминать позиции в файле и впоследствии к ним возвращаться, при помощи так называемых меток. Именем метки может быть одна латинская буква, прописная или строчная. Поставить метку можно командой «m», за которой следует имя метки. Перейти к позиции, сохраненной в метке, можно командой «‘» или «’», за которой следует имя метки. Разница между этими командами состоит в том, что первая из них ставит курсор в точности на позицию метки, а вторая — на первый непробельный символ соответствующей строки.

4.6. Поиск и замена

О поиске речь шла в предыдущем параграфе. Для совмещения поиска и замены существует команда «:диапазонs/образец/замена/флаги число». Образец указывается таким же образом, как и в команде поиска. В строке замены, кроме прочих управляющих конструкций, можно ссылаться на весь найденный текст (\0) или на его часть, отвечающую d-й группе (\d), где d — цифра от 1 до 9.

Разнообразие возможных флагов весьма велико, укажем лишь важнейшие из них:

c — запрашивать подтверждение для каждого акта замены. Среди возможных вариантов ответа упомянем следующие: y — заменить данное вхождение, l — заменить данное вхождение и на этом закончить, n — пропустить данное вхождение, <Esc> — закончить, a — заменить данное и все последующие вхождения.

g — заменить все вхождения в каждой обрабатываемой строке (по умолчанию в каждой строке заменяется только первое вхождение).

i — не различать регистр букв при поиске.

n — выдать только число замен, не меняя при этом текст.

Вместо символа «/», ограничивающего образец и строку замены, можно использовать любой другой, кроме букв, цифр и «\», «"» и «|». Эта возможность используется, если «/» нужно включить в образец или строку замены.

Также, возможно вместо строки замены использовать выражения. Для этого строка замены должна начинаться с «\ =». Это позволяет вычислять строку замены при помощи аппарата выражений vi. При этом, символ-ограничитель образца и строки замены не должен встречаться в выражении. Более подробно выражения будут описаны в параграфе про программирование в vi.

4.7. Блоки

Как и любой другой текстовый редактор, vi предоставляет возможность выделять часть текста (блок) и затем копировать его в другое место или уничтожать. Для выделения блока в vi существует специальный режим, называемый визуальным.

Чтобы выделить часть текста, нужно поставить курсор на начало блока и перейти в визуальный режим при помощи одной из следующих команд: «v» начинает выделять последовательность символов, «V» начинает выделять последовательность строк и «C-v» — прямоугольный блок.

Далее, нужно поставить курсор на конец блока и дать одну из следующих команд: «y» — запомнить содержимое выделенного блока, «d» — удалить блок (содержимое блока при этом запоминается, так что его можно впоследствии вставить в другое место). Обе эти команды прекращают визуальный режим.

Чтобы вставить так запомненное содержимое блока в другое место, нужно поставить курсор в точку, куда должна производиться вставка, и воспользоваться командой «p» или «P». Разница между этими командами в том, что первая вставляет текст после символа, на котором стоит курсор, а вторая — перед.

Редактор vi позволяет также иметь несколько запомненных фрагментов текста при помощи так называемых регистров. Для записи выделенного фрагмента в регистр «f» необходимо дать команду «"fy» в визуальном режиме (сразу после «"» идет имя регистра — символ); для вставки содержимого регистра в текст можно использовать команду «"fp». Также можно записать в регистр удаляемый текст, воспользовавшись командой «d» вместо «y».

Наконец, отдельные текстовые объекты (такие, как слова или предложения) можно выделять, запоминать и помещать в регистры без перехода в визуальный режим. Например, если нужно удалить или просто запомнить слово, можно дать команду «daw» или «yaw» соответственно (не в визуальном режиме!). А если нужно поместить это слово в регистр, то перед вышеуказанной командой нужно использовать префикс из «"» и имени регистра.

4.8. Программирование в vi

Редактор vi позволяет создавать новые команды, комбинируя уже имеющиеся. Данный параграф содержит краткое описание таких возможностей vi.

Во-первых, следует упомянуть о макросах, состоящих из команд vi. Для записи такого макроса нужно дать команду «q», за которой непосредственно следует имя регистра (должно быть маленькой латинской буквой). Введя требуемую последовательность команд, для окончания записи просто нажать «q». Запустить так записанную последовательность команд на выполнение можно при помощи команды «@», за которой сразу же следует имя регистра.

Однако записанную последовательность команд можно не только запустить, но и отредактировать. Это возможно, поскольку записанные в регистр команды хранятся в нем «открытым текстом». Для такого редактирования проще всего воспользоваться стандартными операциями вставки содержимого регистра в файл и записи из файла в регистр.

Кроме записи последовательности команд в регистр, vi допускает и написание скриптов. Для запуска скриптов существует команда «:source имя».

Скрипты vi могут содержать переменные, выражения и управляющие структуры, реализованные в виде «:»-команд, как, впрочем, и любые команды vi.

Переменные бывают двух сортов: обычные (глобальные) и квалифицированные, имя которых выглядит как буква:имя. Буква может принимать значения «s» — переменная локальна для скрипта (в частности, она невидима и для вызываемых скриптов), «g» — глобальная переменная, «v» — переменная, определенная самим Vim, «b» — переменная, локальная по отношению к буферу, «w» — к окну. Имеются и другие квалификаторы.

Для присваивания значений переменным существует команда «:let имя = выражение», для уничтожения переменной — команда «:unlet имя». Простые переменные могут хранить числа и строки, что определяется последним присваиванием. Как и в awk, эти типы автоматически преобразуются по мере необходимости в зависимости от применяемых к ним операций.

Кроме констант (чисел и строк) и переменных, в качестве операндов могут выступать регистры («@имя»), переменные окружения («$имя») и результаты функций (например, exists("a") проверяет наличие переменной a).

Множество операций содержит стандартный для C набор арифметических операций и операций сравнения, а также условную операцию. Конкатенация строк обозначается операцией «.».

Для вывода результатов используется команда «:echo».

Из управляющих конструкций отметим лишь условный оператор «:if» — «:elseif» — «:else» — «:endif» и цикл «:while» — «:endwhile». Как и в awk, условием может быть любое выражение, однако его результат всегда преобразуется в число, так что любая строка, не начинающаяся с цифры, превращается в 0 и, следовательно, трактуется как ложное условие.

Удобным средством запуска команд являются команды «:execute», выполняющая результат выражения, рассматриваемый как «:»—команда, и «:normal», выполняющая команды командного режима (параметром последней команды будет не выражение, а непосредственно символы, составляющие команды для выполнения).

Наконец, рассмотрим создание новых команд Vim. Для этого имеется целая серия команд «:map что как», связывающая команду «что» с последовательностью команд «как». После такой команды ввод «что» приводит к выполнению последовательности команд «как». В зависимости от режимов, в которых нужно определить новую команду, меняется название команды «:map». Собственно «:map» определяет команду для командного, визуального режимов и режима ожидания команды (этот режим означает, что введена часть команды, например введен «y» и ожидается команда перемещения или определения текстового объекта); однако есть разновидность

«:nmap», определяющая команду только для командного режима. Допустимо также, чтобы команда начиналась в одном режиме (в том, для которого она определена) и заканчивалась в другом.

ЛИТЕРАТУРА

1. Колисниченко Д. Н. Fedora 8. Руководство пользователя. М. и др.:Диалектика, 2008.
2. Колисниченко Д. Н. Самоучитель. Linux. 2-е изд. СПб.: БХВ-Петербург, 2008.
3. Кофлер М. Весь Linux. Установка, конфигурирование, использование.7-е изд. М.: Бином: Бином-Пресс, 2007.
4. Маслаков В. Г. Linux. М. и др.: Питер, 2009.
5. Орлов А. А. Самоучитель Linux. СПб.: БХВ-Петербург, 2007.
6. Граннеман С. Linux : карманный справочник: необходимый код и команды. М. и др.: Вильямс, 2007.
7. Стахнов А. Linux. 3-е изд. СПб.: БХВ-Петербург, 2009.