Московский государственный университет
физический факультет
кафедра квантовой теории и физики высоких энергий

В.А.Ильина П.К.Силаев

Краткий справочник по работе с UNIX

Москва 2012

Содержание

1 Введение

Вы должны понимать, что это не описание UNIX, и не введение в UNIX. Это совокупность произвольно отобранных сведений, которые, по нашему мнению, совершенно необходимы пользователю, который намерен писать и отлаживать те или иные счетные задачи в среде UNIX. К пониманию внутренних шестеренок UNIX или особенностей администрирования UNIX эти сведения почти никакого отношения не имеют.

Типичная ситуация при решении счетных задач сейчас выглядит таким образом: Вы где-то у себя дома пишете и отлаживаете счетную программу. Потом Вы соединяетесь с тем или иным вычислительным сервером (кафедральный вычислительный кластер, "суперкомпьютер" "Скиф", и т.п.), который находится где-то далеко, копируете туда отлаженную программу, компилируете ее, и запускаете на счет (ну или ставите в очередь для запуска на счет).

Нравится Вам или не нравится, но на удаленном вычислительном сервере с вероятностью 95% будет стоять UNIX. Версии UNIX могут быть разные — это быть BSD, может быть Solaris (если речь идет о компьютерах Sun), может быть та или иная версия Linux[1] — может быть Slackware, или Debian, или SUSE, или Red Hat, или Fedora, или Ubuntu, и т.д. — всего более двух десятков версий, некоторые из которых коммерческие, некоторые free, некоторые open source.

Важно понимать, что с точки зрения пользователя все это почти одно и то же, и Вам не придется переучиваться. Все, что Вам нужно — это умение работать с файлами и директориями, умение редактировать Ваш файл, умение скомпилировать его и умение запустить на счет.

Забавно, что все перечисленное — это, в сущности, почти все, что должна уметь делать любая операционная система. Беда в том, что интерфейсы операционных систем бывают разные: они бывают графические или текстовые. Понятно, почему по миру широко шагает графический интерфейс — в нем (как правило) можно не помнить ни одной команды, все делается через мышку и меню. Однако, как ни странно, если Вы хорошо знаете команды текстового интерфейса, то Ваша производительность будет гораздо выше, чем в графическом (поверить в это трудно, но это экспериментальный факт). Осколок текстового интерфейса в Windows — программа "`cmd`": появляется окошко с командной строкой, куда можно вбивать команды, позволяющие копировать, стирать, переименовывать файлы, редактировать файлы в текстовом редакторе, запускать программы, и т.п. Нравится Вам или не нравится, но доступ к удаленному вычислительному серверу с вероятностью 97.5% будет реализован через текстовый интерфейс.

Далее мы будем (очень неаккуратно) называть то место, где рисуется командная строка, терминалом. Если Вы сидите за UNIX'овой машиной, то обыкновенно в Вашем распоряжении имеется несколько штук (6 или побольше) текстовых терминалов. Может оказаться, что пущен еще и графический интерфейс (обыкновенно это X Windows). Более того, можно пустить не один, а парочку графических интерфейсов (т.е. парочку X-серверов). Между текстовыми терминалами и X-серверами можно переключаться, нажимая[2] <Ctrl-Alt-F1>, <Ctrl-Alt-F2>, ... <Ctrl-Alt-F12>. Нередко 1–6 — текстовые, а 7–8 — графические. Но бывает и наоборот: 1 — графический, остальные — текстовые.

Внутри графического интерфейса можно запустить очень много окошек с терминалами, каждый из которых ничем не лучше и не хуже, чем другие текстовые терминалы. Единственная разница заключается в разделении пользователей. В разные текстовые терминалы могут входить разные пользователи, поэтому текстовые терминалы требуют ввода login и password.

[1] Кстати, изначально предполагалось, что все двоичные исполняемые файлы под Linux совместимы, в отличие от других вариантов UNIX. К сожалению, сейчас слово "совместимость" представляется излишне оптимистичным. Вообще, структура двоичных исполняемых файлов в разных вариантах UNIX — наука пугающая и захватывающая.

[2] Это не закон природы, но обыкновенно используют именно эти клавиши.

Чтобы войти в графический интерфейс, тоже надо вколотить login и password, но ровно один раз. После этого, пуская очередное окошко с терминалом, вы сразу входите в систему.

Еще одна вещь, которую полезно понимать. Программы, обеспечивающие Вам интерфейс командной строки ("оболочка", "shell"), бывают разные. Вы можете встретить "`bash`", "`zsh`", "`csh`", "`tcsh`", или даже просто "`sh`". Все это приблизительно одно и то же, но с некоторыми различиями в синтаксисе. Например, может существенно отличаться синтаксис работы с environment variables (неважно, что это). Вообще-то синтаксис там достаточно разветвленный — с условными операторами, циклами, возможностью заводить переменные, и т.п. Если Вы не собраетесь писать настоящие сложные программы ("скрипты") для исполнения оболочкой, то Вы, скорее всего, не почувствуете никакой разницы. Если вдруг разница будет, то можно попробовать пустить Вашу любимую оболочку поверх той, которая пускается по умолчанию. Для этого достаточно в командной строке набрать "`bash`" или "`csh`", ну а уж пустится ли она — это как повезет. Чтобы выйти из любой оболочки, наберите в командной строке команду

```
exit
```

и нажмите <Enter>. Для окошка с терминалом это приводит к закрытию окошка, для текстового терминала это означает `logout`, для удаленного терминала это означает `logout` и прекращение коннекции, для оболочки, пущенной поверх исходной оболочки это означает возврат в исходную оболочку.

Нынешние оболочки несколько дружелюбнее к пользователю, чем раньше. Например, нажимая стрелку вверх, можно одну за другой вызвать в командную строку те команды, которые вводились ранее, подредактировать их и снова запустить. (Количество запоминаемых последних команд — "длина истории" — зависит от текущих настроек системы). Кроме того, можно не листать все команды подряд, а набрать символ <!> (восклицательный знак) и начало команды:

```
!gcc
```

после чего нажать <Enter>. При этом запустится последняя из команд компиляции. Возможно, это будет не в точности то, что Вам нужно, но она станет последним событием истории, и Вы сможете получить ее для редактирования однократным нажатием стрелки вверх.

Бывает так, что Вы что-то такое запустили, и вдруг сообразили, что это не то, что Вам нужно. (Например, Ваша собственная программа печатает на экране что-то, из чего сразу видно, что она работает совсем не так, как Вы рассчитывали). Прервать исполнение команды/программы можно нажатием клавиши <Ctrl-c>.

Поскольку UNIX является network transparent, то, разумеется, есть возможность пустить shell и на удаленной машине. Когда-то, когда хакеров было еще не так много, можно было пользоваться "`rsh`" (remote shell), "`telnet`", а для перекачки файлов — "`ftp`". Более того, можно было разрешать беспарольный доступ: если Вы зашли на одну из машин, то другая Вас пускает (под тем же именем) без введения пароля. Сейчас такая политика — это грубое нарушение сетевой безопасности, поскольку в перечисленных программах содержимое пакетов, бегающих по сети, не шифруется. Если Вы делаете login на удаленную машину, то Ваш password шлется по сетке буковка за буковкой, как он есть — мечта любого хакера.

Поэтому сейчас для открытия терминала на удаленной машине используется программа "`ssh`", а для перекачки файлов — "`scp`" (здесь буква "s" означает "secure"). Синтаксис команды `ssh` в простейшем случае необыкновенно прост:

```
ssh oryx7 -l studfve35
```

(здесь "`l`" — буква <*l*> нижнего регистра, от слова login). Имя пользователя "studfve35" особых вариаций не допускает, а вот имя машины можно вводить по-разному, в зависимости от настроек. Если имя домена "bog.msu.ru" автоматически дописывается к имени машины, то

можно ограничиться “oryx7”, если нет, то надо писать полностью: “oryx7.bog.msu.ru”. Можно написать и явный IP-адрес: “213.131.20.147”. Еще один вариант — приклеить login пользователя к hostname машины:

```
ssh studfve35@oryx7.bog.msu.ru
```

В любом случае результат будет один и тот же — у Вас запросят password и, если Вы введете его правильно, пустят на удаленную машину. При нынешней скорости сети (в отсутствие сбоев) заметить, что вы работаете не на локальном, а на удаленном терминале, практически невозможно — это та же самая, обыкновенная command line, только живет она там, а не тут. Выход из удаленного терминала реализуется так же, как и для обычного (“локального”) терминала — с помощью команды “`exit`”.

Кстати, совершенно необязательно, чтобы на машине, с которой Вы подключаетесь к удаленному вычислительному серверу, жил UNIX. Разумеется, UNIX тут гораздо естественней, но и с Windows-машины легко открыть терминал. Соответствующих программ довольно много, но, пожалуй, самой удобной является программа putty (она бесплатная). В комплект putty входит и программа pscp, которая позволяет копировать файлы между Вашим посадочным местом и удаленным сервером. Впрочем, если Вы привыкли к графическому интерфейсу, то копировать файлы Вам, скорее всего, будет удобнее с помощью программы WinSCP.

2 Историческая справка, или “почему UNIX” и “почему текстовый интерфейс” (рекомендуется не читать)

Почему на машинах, предназначенных для работы, стоит UNIX, а не Windows, довольно легко понять. Во-первых, UNIX гораздо бережнее относится к ресурсам машины. Если два процесса одновременно обращаются к диску, то процесс замедляется в 2 раза, а не в 4. Если на 4-х ядерной машине пущены 4 счетные задачи, то использование CPU составляет 4x100%, а не 4x80%, и т.д. Объяснение этих различий в эффективности довольно простое — то, что в UNIX было заложено изначально (многозадачность, многопользовательность, и т.п.) в Windows является позднейшей надстройкой. Дело в том, что Windows является в определенном смысле “внучкой” UNIX. Когда появились первые персоналки, выяснилось, что UNIX на них не влезает[3]. Тогда UNIX урезали и чуть переделали (выкинули пользователей, администрирование, многозадачность, изменили файловую систему и набор команд) — так появился DOS. В сущности, DOS — это ровно один терминал UNIX. Командная строка, в которой можно вводить системные команды, работать с файловой системой, пустить на исполнение ровно одну программу (редактор, компилятор или свою собственную счетную).[4]

Между тем к этому времени уже был разработан графический интерфейс, который называется X Windows. Этот стандарт не привязан к определенной операционной системе (точно так же, как язык “C” может существовать и под DOS, и под VMS, и под UNIX, и под MacOS, и т.д.). Он был реализован на тогдашних “больших” ЭВМ и рабочих станциях (машинах, существенно более мощных и дорогих, чем тогдашние персоналки). Надо сказать, что чтение описания стандарта X Windows параллельно с чтением описания функций из библиотеки Windows — довольно забавное занятие. Так же как и разбирательство между Apple и Microsoft — кто у кого украл “Окна”.

[3]Первые персоналки вообще представляли собой захватывающее зрелище — представьте себе небольшой чемоданчик, напоминающий осциллограф. Экран осциллографа — это дисплей (монохроматический, размером с экран смартфона), а слева и справа от него располагаются 2 дисковода — 5-ти дюймовых, single-sided, каждый на 180 Kb. И никакого винчестера (жесткого диска). Операционная система — DOS 1.0.

[4]Впрочем, можно было еще пустить “резидентную” программу (она должна была быть очень маленькой, например, вирусом), которая в определенном смысле работала параллельно с другими программами.

Было понятно, что для массового пользователя графический интерфейс удобнее. И хотелось вернуть утраченное — хотя бы многозадачность. Первая версия Microsoft Windows, с которой нам довелось поработать, была программой, которая пускалась поверх DOS. А дальше она позволяла уже из себя пускать одновременно несколько программ (оконных или текстовых терминальных). Иными словами, она (в несколько ущербном виде) возвращала многозадачность. Тем самым пошла обратная эволюция — к системе пытались прикрутить то, что было выкинуто из UNIX при создании DOS. Получившаяся операционная система (теперь должно быть понятно, почему ее можно считать "внучкой" UNIX) несколько напоминает велосипед, к которому прикрутили крылья и весла. При практической работе со счетными программами главным неудобством, пожалуй, является то, что нет возможности иметь несколько текстовых терминалов (аналогов cmd) в отсутствие графического интерфейса. В UNIX графический интерфейс (X Windows) можно запускать, а можно не запускать.[5] В любом случае у Вас есть несколько текстовых терминалов. Кроме того, как уже было сказано, вся система является network transparent, так что Вы можете пустить текстовый терминал с другой машины. В действительности, X Windows тоже network transparent, так что, если Вам очень хочется, то можно пускать и X-овые приложения на удаленном графическом дисплее.[6] Однако, как правило, этой возможностью не пользуются. По очень простой причине: это понапрасну расходует вычислительные ресурсы сервера, его память, а также сетевые ресурсы системы. Как нетрудно догадаться, текстовый терминал в этом отношении гораздо экономичнее.

Теперь должно быть понятно, почему "удаленный рабочий стол" Windows в реальной работе не используется. Он поедает еще больше системных ресурсов, чем одно X-овое приложение. Разумеется, если бы Вы были единственным пользователем как сервера, так и всей сети, то расточительное отношение к ресурсам не играло бы никакой роли, но в реальной жизни так не бывает.

3 Файловая система

Файловая система устроена так: существует корневая директория[7] ("корень") с именем "/", в ней могут находиться как файлы, так и другие директории. В свою очередь, в каждой из этих директорий тоже могут находиться как файлы, так и другие директории, и т.д. Все это принято называть деревом (ветки - директории, листья - файлы). Полное имя файла собирается как путь от корня до этого файла:

```
/dirname1/dirname2/filename
```

(в корне находится директория "dirname1", в директории "dirname1" находится "dirname2", а в "dirname2" находится файл "filename").

Работая с системой, пользователь может лазить по этому дереву, т.е. делать ту или иную директорию "текущей". Это директория, которая служит своего рода точкой отсчета для всех команд по работе с файлами, запуска программ и т.п.

Существуют общепринятые имена директорий: пользователям для работы отведены директории

```
/home/username
```

где "username" — имя пользователя (login name).

[5] Кроме того, поверх X Windows можно запускать, а можно не запускать тот или иной Window Manager — есть еще и такая вещь.

[6] Схема работы очень простая: Вы сидите за UNIX'овым посадочным местом, на котором пущены X Windows. Затем Вы пускаете X-овое приложение на удаленном вычислительном сервере, но указываете, что рисовать окошко следует не на дисплее сервера (кстати, Вам, скорее всего, и не позволено там что-либо рисовать), а на дисплее Вашего посадочного места.

[7] Директория — это как бы "папка".

В директории “/etc” содержатся файлы с системной информацией; в директориях “/usr” и “/usr/local” живут установленные программы, например в “/usr/lib” находятся статические и динамические библиотеки, в “/usr/bin” находятся двоичные исполняемые файлы, а в “/usr/include” — инклудовские файлы “ *.h”; в директории “/var” помещаются временные файлы, в директории “/tmp” помещаются совсем временные файлы, и т.д.

Существуют специальные имена директорий:

- Директория “ . ” (точка) — это текущая директория;
- Директория “ . . ” (две точки подряд) — это директория на один уровень вложения вверх от текущей;
- Директория “ ~ ” (тильда) — это “домашняя” директория пользователя /home/username.

4 Команды для работы с файлами

В принципе программа “mc” (MidnightCommander) для того и написана, чтобы не знать команд работы с файлами. Эта программа до некоторой степени имитирует графический интерфейс в текстовом режиме. Чтобы ее пустить, в командной строке надо набрать команду

```
mc
```

и нажать <Enter>. Чтобы посмотреть Help по mc, надо нажать клавишу <F1>. (Выход из Help — клавиша <F10>, или двукратное нажатие <Esc>). Выход из mc — клавиша <F10>, при этом Вас, как обычно, переспросят (“правда выйти?”).

Интерфейс mc нередко поддерживает мышку, что удобно, хотя и несколько замедляет работу (клавиатура быстрее). Вы должны понимать, насколько это нетривиальный и необязательный факт: ведь мышка живет на локальном терминале, который должен правильно передавать мышиные события на удаленный терминал. Поэтому следует быть готовым к тому, что мышка работать не будет или будет работать неправильно. В этом случае надо внимательно рассматривать самую нижнюю строку терминала — там содержатся подсказки по основным ключевым клавишам. Еще одна вещь, которую надо твердо помнить — если перед Вами появилось несколько полй, куда надо вбивать насколько параметров или ставить галочки, то переход между ними реализуется клавишей <Tab>. (Иногда можно не ходить по полям по порядку, с предыдущего на следующее, а прямо скакнуть на нужное поле — это если в его имени содержится так или иначе подсвеченная буква — тут можно смело жать на соответствующую клавишу).

К сожалению, иногда оказывается, что на счетном сервере mc не установлен. В этом случае знание хотя бы самых элементарных команд перемещения по дереву файловой системы, удаления, переименования, копирования, архивации и разархивации файлов окажется необходимым.

4.1 Подсказка по системным командам

Команда “man” позволяет получить подсказку по командам командной строки и даже по некоторым стандартным функциям языка “C”.

Например, если Вам интересен полный список флагов и синтаксических возможностей команды “ls”, следует набрать команду

```
man ls
```

это даст достаточно полное описание команды. К сожалению, для некоторых команд понимание этого описания требует хорошего знания контекста, так что чтение описаний “man” для всех команд подряд не есть способ освоить систему. Никакого дерева подсказок, оглавления

и перекрестных ссылок тут нет, только в конце приводится перечень команд, связанных с описанной командой. Выход из просмотра описания — нажатие клавиши <q>.

Нередко (хотя и не всегда) в системе бывает развернута более интерактивная система подсказок "`info`". (Впрочем, бывает и наоборот — `man` по какой-то теме отсутствует, а `info` — есть). Команда

```
info ls
```

высвечивает подсказку (нередко дословно совпадающую с выдачей команды "`man`"), которая является частью достаточно большого и достаточно систематизированного описания всех команд системы. Этот файл можно просто читать подряд, поскольку описания однотипных команд сгруппированы в главы, кроме того он снабжен оглавлением, попасть в которое из любого места можно нажатием клавиши "<" ("меньше"). Соответственно, из оглавления можно попасть в любое место, просто поставив курсор на соответствующий пункт оглавления и нажав <Enter>. Поиск ключевого слова — нажатие клавиши </> (знак деления), после чего следует набрать слово и нажать <Enter>. При повторном поиске предыдущее слово помнится, так что заново набирать его не надо. Выход из "`info`" — нажатие клавиши <q>.

4.2 Перемещение по дереву

В `mc` есть две равноправные панели — левая и правая. Одна из них активна — на ней живет полоска, которую можно двигать стрелочками. Переключиться на другую панель (сделать активной ее) можно с помощью клавиши <Tab>. Чтобы залезть в директорию, надо поставить полоску на имя директории и нажать <Enter>. Чтобы вылезти из директории на один уровень вверх, надо поставить полоску на имя директории " `..` " (это самый верх панели), и нажать <Enter>.

Из командной строки текущую директорию меняет команда "`cd`". При этом возможности синтаксиса достаточно разнообразны.

Простейший вариант:

```
cd dirname
```

где директория "`dirname`" находится в текущей директории. Соответственно,

```
cd ..
```

выводит нас на один уровень вверх по сравнению с текущей директорией, а

```
cd ~
```

возвращает нас домой — в директорию `/home/username`.

Можно указать и полное имя директории:

```
cd /home/username/mydir3
```

или еще более витиевато:

```
cd ../../mydir3/mydir4
```

Это значит, что из текущей директории мы перемещаемся на 2 уровня вверх, оттуда в "`mydir3`" (предполагается, что она там есть), и из "`mydir3`", в свою очередь, в "`mydir4`" (тоже предполагается, что она там есть).

Необходимо помнить, что можно сильно упростить себе жизнь, воспользовавшись метасимволами (metacharacters). Символ "`*`" — это любое количество (в том числе ноль) любых букв, а "`?`" — ровно одна любая буква. Так что вместо

```
cd /home/username/mydir3
```

можно набрать

```
cd /ho*/use?nam?/m*3
```

Впрочем, если у Вас несколько директорий, вписывающихся в шаблон "`m*3`", то Вы попадете в неизвестно какую из них. При умелом пользовании метасимволами можно сильно сократить длину команды.

4.3 Удаление директорий

В `mc` все очень просто — ставите полоску на директорию и жмете ключевую клавишу <F8> (запоминать ничего не надо — нижняя строка `mc` содержит подсказку "8 Delete"). Вас переспросят, причем если директория не пуста, то переспросят с особым накалом. Но если Вы ответите "да", то `mc` немедленно удалит директорию, причем вместе с содержимым, и удалит безвозвратно — никаких корзин.[8]

Кроме того, в `mc` можно удалить сразу несколько директорий, если пометить их клавишей <Ins> (меченые директории так или иначе (в зависимости от типа терминала) подсвечиваются), и нажать <F8>.

Из командной строки можно дать команду

```
rmdir dirname
```

которая удалит директорию "dirname" из текущей директории, если "dirname" пустая. Если она не пустая, то система будет ругаться, и это правильно. Команда признает метасимволы, т.е.

```
rmdir mydir?
```

удалит все пустые директории, вписывающиеся в шаблон mydir?. При этом с командой

```
rmdir *
```

надо быть поосторожнее.

Команда допускает и полные имена директорий:

```
rmdir /home/username/mydir3
```

4.4 Создание директорий

В `mc` все очень просто — жмете ключевую клавишу <F7> (подсказка "7 Mkdir"). Появится поле, куда надо вколотить имя директории. **Предупреждение**: существуют диалекты UNIX (например, большинство вариантов Linux), допускающие имена директорий и файлов с пробелами, с кириллицей, и т.п. **Никогда** не пользуйтесь этой возможностью. Рано или поздно вы столкнетесь с ситуацией, когда Вам придется лихорадочно все переименовывать, потому что файловая система какого-либо вычислительного сервера или какого-либо носителя файлов этих изысков не понимает.

В командной строке можно дать команду

```
mkdir dirname
```

которая создаст директорию "dirname" в текущей директории.

[8] Корзины предусмотрены, если Вы работаете в графическом интерфейсе X Windows и используете File Manager (что-то вроде "проводника"). Однако, как уже было сказано, при удаленном доступе графический интерфейс обычно недоступен.

4.5 Копирование файлов

В `mc` все очень просто — на одной панели (неважно какой) Вы должны попасть в директорию с файлом-источником, на другой — в директорию, куда Вы хотите файл скопировать. (Если Вы хотите получить копию исходного файла с другим именем, то на обеих панелях может быть и одна и та же директория). После этого Вы жмете ключевую клавишу <F5> (подсказка "5 Copy"). Появится поле, куда можно вколотить новое имя файла. Если имя менять не хочется, то просто жмите <Enter>. Опять-таки можно скопировать несколько файлов, если пометить их нажатием клавиши <Ins>, а потом нажать <F5> (это будет просто копирование, без переименования). Более того, `mc` позволяет копировать и директории — целиком, вместе со всем содержимым.

В командной строке можно дать команду:

```
cp filename1 filename2
```

Это создаст копию файла "filename1" с именем "filename2".

Если Вы внимательно прочли предыдущий текст, для Вас должно быть очевидно, что команда

```
cp ./filename1 ./filename2
```

делает ровно то же самое, что и предыдущая; команда

```
cp  /home/username/mydir1/fil3  /home/username/mydir4/fil7
```

не только создает копию с новым именем, но и кладет ее в другую директорию. Если имя копии совпадает с исходным, то его можно опустить, указав только имя директории. Поэтому команда

```
cp  /home/username/mydir1/fil3  /home/username/mydir4
```

есть синоним команды

```
cp  /home/username/mydir1/fil3  /home/username/mydir4/fil3
```

Смысл команд

```
cp fil3  ../mydir4
```

```
cp  *  /home/username/mydir4
```

тоже должен быть очевиден. Кстати, при исполнении команды вида "`cp * ...`", если в текущей директории кроме файлов есть еще и директории, то система аккуратно предупредит Вас, что директории она не копирует (в отличие от `mc`).

4.6 Перемещение (оно же переименование) файлов

Все в точности совпадает с копированием — только ключевая клавиша <F6> (подсказка "6 RenMov"), а команда "`mv`" вместо "`cp`". Разница в том, что исходный файл-источник не сохраняется, а уничтожается.

4.7 Удаление файлов

Все в точности совпадает с копированием — только ключевая клавиша <F8> (подсказка "8 Delete"), а команда "`rm`" вместо "`cp`". Разница в том, что файл (или несколько файлов) уничтожаются, без всякого копирования. **Предупреждение**: будьте очень осторожны с командой типа

```
rm *.c
```

4.8 Просмотр содержимого директории

В `mc` все очень просто — на двух панелях выведены два списка файлов и директорий. На активной панели (где живет полоска) — список файлов и директорий, живущих в текущей директории. На другой — список файлов и директорий, живущих в еще какой-то директории (может быть, той же самой, т.е. текущей). Строк в списке может быть больше, чем строк на экране, но полоска может по этому списку бегать (пользуйтесь стрелками, клавишами <PgUp>, <PgDn>, <Home>, <End>).

В командной строке существует команда "`ls`" — она печатает полный список файлов и директорий, находящихся в текущей директории.

Как обычно, существует синтаксическое разнообразие. Например, допускаются флаги. Команда

```
ls -l
```

это печать более подробой информации о файлах и директориях. Кроме достаточно очевидной информации о дате и времени последнего изменения файла, о длине файла в байтах (это полезно только для файлов, для директорий тут печатается вовсе не суммарный вес ее содержимого), о "количестве единиц хранения" (это полезно только для директорий, для файлов тут будет единичка), о том, кто является собственником (владельцем) файла, о том, какая группа пользователей является владельцем файла, появляется еще и информация об атрибутах файла.

Эта информация может иметь вид:

```
drwxr--r--
```

либо

```
-rw-r--r--
```

либо

```
-rwxr-xr-x
```

и т.п. Смысл довольно простой: первая буква показывает, директория это (`d`) или файл (`-`). Далее идут 3 группы символов по 3 буквы. Первая группа (первые 3 буквы) — это что владелец файла может делать с файлом, вторая — что группа владельцев[9] файла может делать с файлом, третья — что все остальные могут делать с файлом. А делать с файлом можно 3 вещи: читать (`r`) (для директории это означает получать список файлов, находящихся в этой директории на первом уровне вложения), писать в него (`w`) (для директорий это означает возможность создавать и стирать файлы), и запускать его на исполнение (`x`, буква <x> от слова "eXecute") (для директорий это означает получать доступ к файлам, находящимся в директории). Так что буквы

```
-rwxrwxrwx
```

означают, что все могут делать с этим файлом все, что угодно, буквы

```
-rwx------
```

означают, что только Вы можете делать с этим файлом все, что угодно, а буквы

```
-rw-r--r--
```

означают, что Вы можете писать и читать, а все остальные — только читать.

[9]Обыкновенно группа владельцев Вашего файла — это попросту пользователи, которые принадлежат к той же группе, что и Вы. Скорее всего, это будет попросту группа "Users".

Кстати, из этого набора атрибутов вообще говоря не следует, что этот файл по своей структуре не является исполняемым. Команда "`chmod`" позволяет произвольно менять атрибуты файлов.[10] Вы вполне можете скомпилировать Вашу счетную программу, получить двоичный исполняемый файл "myprog", а потом командой `chmod` убрать у него атрибут исполняемости:

```
chmod -x myprog
```

(совершенно бессмысленное действие). И наоборот, Вы можете сделать исполняемым текстовый файл. Если Вы сделаете исполняемым файл с программой на языке "C", и попробуете его запустить, то система будет очень ругаться, потому что она ждет команд типа "`ls`", "`cp`", "`rm`", и т.п. Но Вы можете поступить разумнее: соорудить текстовый файл "myscript" с последовательностью команд системы и сделаеть его исполняемым:

```
chmod +x myscript
```

В этом случае говорят, что Вы написали "скрипт". Это чрезвычайно удобный инструмент. Например, можно написать скрипт, запустить его, и он сам будет запускать несколько Ваших счетных программ последовательно, одну за другой, а в промежутках между запусками архивировать и сжимать их выдачу. Вообще-то, с этой задачей справится и программа на языке "C", но скрипт удобнее.

Команда

```
ls -a
```

печатает список всех файлов текущей директории, в том числе тех, которые являются служебными конфигурационными файлами (их имена начинаются с точки: ".cshrc", ".profile", ".bashrc", и т.п.). Команда

```
ls -al
```

есть печать подробной информации про все файлы, включая служебные.

Кроме того, можно печатать не все файлы подряд, а только вписывающиеся в шаблон. Так, команда

```
ls *.c
```

ограничивает вывод только файлами с именами, кончающимися на ".c" (обыкновенно это файлы с программами на языке "C"). Команда

```
ls myfil??
```

ограничивает вывод только файлами с именами, вписывающимися в шаблон "myfil??". Команда

```
ls /etc
```

выводит содержимое директории "`/etc`". Команда

```
ls /usr/lib/*.a
```

выводит список файлов, чьи полные имена вписываются в шаблон "`/usr/lib/*.a`" (обыкновенно это список файлов со стандартными статическими библиотеками. Статическая библиотека — это файл с набором откомпилированных стандартных функций, которые дописываются в Ваш исполняемый файл на этапе линкования). Команда

```
ls /usr/lib/*.so
```

[10]Если, конечно, у Вас есть на это право. Как правило, если Вы владелец файла, то Вы можете менять его атрибуты. К сожалению, на практикуме поэкспериментировать с "`chmod`" Вам не удастся.

работает совершенно аналогично (обыкновенно это список файлов со стандартными динамическими библиотеками. Динамическая библиотека — это файл с набором откомпилированных стандартных функций, которые подгружаются при запуске Вашего исполняемого файла).

Лирическое отступление: перенаправление выдачи

Очень часто оказывается удобно перенаправлять выдачу команд. Ее можно перенаправить в файл или на вход другой команды.

Если Вы хотите перенаправить выдачу команды "`ls`" в файл, достаточно набрать команду:

```
ls /etc/*.hihi >myfil2
```

Если файла "myfil2" не было, то он будет создан. Если файл уже существовал, его содержимое будет затерто и заменено на выдачу "`ls`". Если Вы хотите дописать выдачу команды "`ls`" в хвост уже существующего файла "myfil1", то надо набрать команду:

```
ls /etc/*.hihi >>myfil1
```

Если Вы хотите перенаправить выдачу команды на вход другой команды, например на вход команды "`wc`" (обычный пример программистского хулиганства: "wc" — это Word Counter), то следует набрать команду:

```
ls /etc/*.hihi | wc
```

Команда "`wc`" занимается довольно бессмысленным делом — она считает количество букв, слов и строк в выдаче. Тем не менее это способ быстро узнать количество, например, ps-файлов в текущей директории:

```
ls *.ps | wc
```

Более полезный пример — команда "`grep`". Эта команда сохраняет в выдаче только те строки, в которых встречается указанное ключевое слово. Поэтому команды

```
ls -l *hahaha*
```

и

```
ls -l | grep hahaha
```

приведут к более-менее одному результату.

Еще один полезный пример — команда "`more`". Как правило, возможностей прокрутки терминала вполне хватает, чтобы рассмотреть всю выдачу команды "`ls`". Если все же не хватает, то надо перенаправить выдачу:

```
ls | more
```

Выдача будет происходить порциями, при этом следующую порцию Вы получите, нажав на пробел. Если Вам надоело, то прервать вывод (совсем прервать) можно, нажав клавишу <q>.

4.9 Просмотр содержимого файла

В "`mc`" все очень просто - достаточно поставить полоску на файл и нажать <F3>. Выход из просмотра — клавиша <F10> (см. подсказку на нижней строке экрана "10 Quit"), либо два раза подряд нажать клавишу <Esc>.

Здесь могут встретиться два подводных камня.

Во-первых, `mc` знает, что некоторые файлы имеют особый смысл. Например, файлы "*.ps" — это PostScript файлы с картинкой. Поэтому при нажатии <F3> для ps-файла он попытается пустить ту или иную смотрелку графических файлов. Но при работе с удаленного терминала ему это не удастся, так что Вы ничего не увидите. Выход очень простой — открыть редактор (клавиша <F4>) — в нем будет видно содержимое (текстовое) ps-файла.

Во-вторых, при попытке посмотреть содержимое двоичного файла (например, содержимое двоичного исполняемого файла, полученного в результате компиляции Вашей программы на языке "C"), на экране появится двоичный мусор. Часть символов из этого мусора терминал воспринимает как команды (форматирования текста, раскраски текста, и т.п.). В результате терминал несколько утратит здравый смысл, и прочитать, что на нем пишется, будет затруднительно (даже после выхода из просмотра файла с помощью <F10> или двукратного нажатия <Esc>). Иногда помогает команда

```
reset
```

вслепую набранная в командной строке. Если это не поможет, придется перезапустить терминал.

При работе с командной строки есть несколько возможностей.

- Команда

  ```
  cat myfile
  ```

  печатает содержимое файла от начала до конца. Если файл слишком длинный, и возможностей прокрутки терминала не хватает, то можно перенаправить вывод `cat` в программу `more`:

  ```
  cat myfile | more
  ```

  либо попросту

  ```
  more myfile
  ```

  Опять-таки, можно перенаправить вывод "`cat`" в программу "`grep`", которая оставит только строки, содержащие ключевое слово. При этом целесообразно добавить флаг "`-n`", при этом будут печататься еще и номера строк, содержащих ключевое слово:

  ```
  cat myfile | grep -n hihihi
  ```

  Забавно, что слово "`cat`" здесь на самом деле излишне — команда "`grep`" согласна искать ключевые слова не только на своем входе, но и в файлах, причем не в одном. Так, команда

  ```
  grep -n printf *.c
  ```

  напечатает все строки, которые содержат слово "printf", содержащиеся во всех c-файлах текущей директории. При этом будут печататься еще и имена файлов и номера строк.

- Более интерактивный способ просмотра содержимого файла — это команда "`less`" (опять-таки программистское хулиганство: раз есть "more", должен быть и "less"):

  ```
  less myfile
  ```

  выведет на экран содержимое файла с возможностью постраничного листания и построковой прокрутки. В сущности, возможности "`less`" почти совпадают с возможностями

смотрелки <F3> из `mc`. (Дополнительные возможности `mc` в обычной жизни не слишком полезны). Поиск ключевого слова при работе "`less`" делается так же, как в "`info`" — нажимаем клавишу </> (знак деления), набираем слово, и нажимаем <Enter>. При повторном поиске предыдущее слово помнится, так что заново набирать его не надо. Выход из просмотра — нажатие клавиши <q>.

- Еще одна возможность посмотреть кусочек файла — это команда "`tail`".

  ```
  tail myfile
  ```

  выводит на экран несколько последних строк файла "myfile" (обычно 10 штук). Если Вы хотите получить определенное количество строк (например, 13), наберите

  ```
  tail -n 13 myfile
  ```

  А если Вы хотите увидеть хвост файла, начиная с определенной строки, например с 63-й, наберите

  ```
  tail -n +63 myfile
  ```

  Еще один флаг команды "`tail`" нужен, если Вы хотите посмотреть выдачу Вашей счетной программы в протокольный файл, позволяющий отследить ход выполнения программы. Трассировка счета уже отлаженной программы (процесс отладки — иное дело) должна реализоваваться именно так. Вывод текущей информации на экран — дело совершенно бессмысленное, поскольку, во-первых, при достаточно длительном счете конекция со счетным сервером практически наверняка рано или поздно оборвется, и Вы перестанете что-либо видеть. Во-вторых, выводя протокол счета в файл, Вы его сохраняете, и более чем вероятно, что эта информация Вам потом пригодится.

  Так вот, следить за пополнением протокольного файла "myprog.log" можно и примитивным способом — просто надо периодически набивать команду

  ```
  tail myprog.log
  ```

  и рассматривать последние строки выдачи. Но гораздо разумнее дать команду

  ```
  tail -f myprog.log
  ```

  Тогда система будет сама отслеживать появление новых строк в файле и выводить их на экран. По виду это практически то же самое, что и прямой вывод программы на экран, только информация еще и сохраняется в файле. Прервать выдачу можно, оборвав работу "`tail`", а оборвать работу команды (любой команды) можно нажатием <Ctrl-c>.

4.10 Архивация файлов.

В принципе под UNIX Вам могут встретиться и привычные Вам по работе под Windows архиваторы — например `zip/unzip`. Но лучше на это не полагаться. Потеря нервных клеток при освоении стандартного архиватора гораздо меньше, чем потеря нервных клеток при попытке открыть zip-файл на машине, где нет ни unzip, ни mc. К сожалению, такое случается. По историческим причинам стандартом архивации является комбинация программ `tar` и `gzip`. (Сейчас потихоньку распространяется еще и `bz2`).

Программа `tar` позволяет засунуть некоторое количество файлов и директорий в один tar-файл.[11] Команда

```
tar cvf myfile.tar *
```

позволяет заархивировать всю текущую директорию (включая содержимое поддиректорий) в tar-файл “myfile.tar”.

Соответственно, развернуть архив можно командой

```
tar xvf myfile.tar
```

Единственное неудобство `tar` заключается в том, что он не сжимает архив. Если размер архива играет роль, то его стоит сжать командой `gzip`. Эта команда согласна сжимать любые файлы:

```
gzip myfile
```

сожмет файл и переименует его в “myfile.gz”. В этой команде соблюдается неплохой баланс между качеством сжатия и скоростью работы — не блестящий, но неплохой. Соответственно, если напустить `gzip` на tar-файл:

```
gzip myfile.tar
```

то получится “myfile.tar.gz”. Иногда это имя урезается до “myfile.tgz”.

Обратный процесс запускается командой `gunzip`:[12]

```
gunzip myfile.gz
```

это распакует сжатый файл “myfile.gz” в исходное состояние и переименует его обратно в “myfile”.

Если под рукой есть mc, то распаковывать архивы совсем просто: следует попросту поставить полоску на файл “myfile.tar.gz” (или “myfile.tgz”) и нажать <Enter>. Архив откроется так, как будто он является директорией. Дальше можно копировать или все файлы архива, или отдельные файлы в любое место обычным образом (см. выше).

4.11 Поиск файлов

В mc все очень просто — жмем <F9> (подсказка "9 PullDn") и попадаем в меню. Далее выбираем: пункт меню “Command” ⇒ подпункт меню “Find File”. Появятся поля, куда надо вбить параметры поиска. Параметр “Start at” — это та точка дерева, начиная с которой будет производиться поиск. Иначе говоря, поиск будет идти только по вбитой сюда директории и по тем директориям, которые в нее вложены. По умолчанию там стоит “ . ”, т.е. текущая директория. Если Вы хотите искать везде, то или оставьте пустое место, или вбейте “корень” (“/”).

[11] Программа `tar` появилась для работы со стриммером — носителем информации на магнитной ленте (это своего рода кассетный магнитофон). В незапамятные времена, когда диск в 300 Mb считался большим, емкость стриммерной кассеты поражала воображение. Впрочем, скорость считывания была, разумеется, очень низкой. Как это ни смешно, стриммеры живут до сих пор: некоторая часть raw data с LHC хранится в стриммерном массиве. И понятно почему — поверхность пленки, свернутой в рулон, разумеется, превышает суммарную поверхность блинов в жестком диске, если объем диска и объем рулона совпадают. Так вот, на стриммер можно было положить tar-файл (а не дерево директорий), и наоборот, со стриммера можно было считать tar-файл, а потом развернуть его в дерево директорий.

[12] Забавно, что это не отдельная программа. Это тот же самый исполняемый файл `gzip`. При его запуске с флагом “`-d`”: “`gzip -d myfile.gz`” он разжимает файл. А “gunzip” — это так называемый “link”, т.е. имитация наличия файла — под другим именем в этой же директории, или под другим или тем же именем в другой директории. Система делает вид, что это обычный файл, но при попытке к нему обратиться (например, запустить) — “идет по линку”, т.е. использует исходный файл. При запуске программа `gzip` первым делом смотрит, под каким именем она вызвана: если `gzip`, то она начинает сжимать, если `gunzip`, то начинает распаковывать.

Смысл параметра “filename” очевиден, а вот возможностью “content” Вы должны пользоваться с осторожностью. При жестком задании имени файла все не так страшно, а вот если Вы не помните имя, не помните, где может лежать файл, и пытаетесь найти его по содержимому — постарайтесь правильно задать остальные параметры. Исследовать содержимое **всех** файлов на Вашем компьютере — дело долгое.

С командной строки тоже довольно просто:

```
find /usr -name ‘‘hosts’’ -print
```

Здесь первый параметр (`/usr`) — та точка дерева, начиная с которой будет производиться поиск (еще раз напомним, что поиск идет по дереву “вглубь”, т.е. в “/usr/lib” и “/usr/include” мы попадем, а в “/etc” — нет); второй параметр — критерий поиска (`-name ‘‘hosts’’`), здесь в имени файла допускаются метасимволы, т.е. можно писать “ `-name ‘‘host?’’` ”; третий параметр — что делать с найденным файлом — (`-print`) означает, что надо печатать полное имя файла.

5 Редакторы

Надо отчетливо понимать, что удобство редактора — понятие относительное. Редактор, к которому Вы привыкли, всегда самый удобный.

Поэтому речь пойдет только о том, какие редакторы Вам могут встретится на счетном сервере, и как внести минимальную правку в Ваш с-файл с их помощью.

5.1 Редактор mcedit

Во-первых, есть встроенный редактор `mc` — “`mcedit`”. Его можно пускать как непосредственно из `mc` (ставитесь на файл и жмете <F4>, подсказка в нижней строке — “4 Edit”), так и находясь в командной строке: команда

```
mcedit myfile.c
```

запускает редактирование файла myfile.c. При этом неважно, существует ли уже этот файл — так что это прекрасный способ создать новый с-файл.

В этом редакторе есть все примитивно-необходимые функции: копирование и перемещение блоков, поиск, поиск с заменой, и т.п. Подсказка по клавишам выведена в самой нижней строке. В принципе следовало бы почитать встроенный Help (он пускается нажатием <F1>, подсказка “1 Help”), но, как нам кажется, большинство функций является self-explanatory. (Кстати, выход из Help — нажатие <F10>, или двукратное нажатие <Esc>).

Тем не менее, опыт показал, что возникают вопросы. Поэтому перейдем к самой нелепой части этой шпаргалки.

- Чтобы отмаркировать блок, надо поставить курсор на его начало, нажать <F3> (подсказка “3 Mark”), переместить курсор на конец блока и еще раз нажать <F3>. Блок будет так или иначе подсвечен.
- Чтобы передвинуть или скопировать уже отмаркированный блок, надо поставить курсор на то место, куда Вы его хотите поместить, и нажать <F5> (подсказка “5 Copy”) для копирования, либо <F6> (подсказка “6 Move”) для перемещения.
- Чтобы стереть строку, на которой стоит курсор, нажмите <Ctrl-y>.
- Чтобы найти ключевое слово, достаточно нажать <F7> (подсказка “7 Search”), при этом появится поле, в которое надо вколотить ключевое слово и нажать <Enter>. Для повторного поиска надо нажать <Shift-F7>.

- Чтобы заменить ключевое слово на что-то другое, надо нажать <F4> (подсказка “4 Replac”), при этом появятся 2 поля, в которые надо вколотить ключевое слово и то, на что оно заменяется. Настройки по умолчанию тут довольно странные[13] — считается, что заменять надо ровно 1 слово ровно 1 раз — в первом месте, которое встретится. Чтобы заменить слово по всему документу, надо выставить флажок “Replace All”, и только потом жать <Enter>. И все равно, в каждом очередном месте она будет методично Вас переспрашивать — “а в этом месте тоже заменить?”. Предусмотрены стандартные ответы: заменить (repl), пропустить (skip), заменить все (all), прервать (cancel).

- Чтобы сохранить текущий результат редактирования, достаточно нажать <F2> (подсказка “2 Save”).

- Чтобы выйти из редактирования, достаточно нажать <F10> (подсказка “10 Quit”), либо 2 раза нажать <Esc>.

5.2 Редактор joe

Это редактор с командами редактирования, которые использовались фирмой Borland в интегрированных средах Turbo Pascal, Tubo Basic, Turbo C, Turbo Prolog и т.д. еще до появления Windows. Тут надо сказать, что оконные редакторы, с одной стороны, внесли унификацию (что хорошо), а с другой — чудовищно обеднили возможности редактирования (что плохо). Ни одному оконному редактору недоступно все разнообразие возможностей текстового редактора MultiEdit, если только Вы хорошо изучили этот самый MultiEdit.

Чтобы запустить `joe`, наберите:

```
joe myfile.c
```

- Чтобы посмотреть встроенный Help, нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-h> (<Ctrl> можно не отпускать). (Кстати, выход из Help — повторное нажатие <Ctrl-k>, <Ctrl-h>.)

- Чтобы отмаркировать блок, поставьте курсор на его начало, и нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-b>. Потом поставьте курсор на конец блока и нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-k>.

- Чтобы скопировать блок, поставьте курсор на то место, куда Вы хотите скопировать блок и нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-c>.

- Чтобы переместить блок, поставьте курсор на то место, куда Вы хотите его переместить и нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-m>.

- Чтобы стереть строку, на которой стоит курсор, нажмите <Ctrl-y>.

- Чтобы найти ключевое слово или заменить ключевое слово на что-то другое, нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-f>. В нижней строке появится поле, куда надо сначала вколотить исходное ключевое слово и нажать <Enter>. Далее Вас спросят, что делать с ключевым словом — просто искать (ignore) или заменять (replace).

- В случае, если Вы выбрали замену (replace), в нижней строке появится поле, в которое надо вколотить замену. После этого для каждого найденного слова будет задан вопрос:

[13] По крайней мере, у той версии, с которой Вы будете работать. Есть версии `mc` с более разумным поведением.

заменить (yes), пропустить (no), заменить все (rest). Кроме этих, стандартных вариантов, есть еще нестандартный вариант — вернуться к предыдущему найденному слову (backup).

В случае, если Вы выбрали поиск (ignore), слово ищется один раз. Чтобы повторить поиск, нажмите <Ctrl-l>.

- Чтобы сохранить результаты редактирования без выхода из редактора, нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-d>.
- Чтобы выйти и сохранить результаты редактирования, нажмите подряд 2 клавиши: <Ctrl-k>, <Ctrl-x>.
- Чтобы выйти без сохранения результатов редактирования, нажмите <Ctrl-c>. Разумеется, Вас переспросят, соображаете ли Вы, что делаете, и только потом выйдут.

5.3 Редактор vi (или vim).

Безусловно, к этому редактору можно привыкнуть. Мало того, совершенно понятно, отчего он такой странный: он был создан в те незапамятные времена, когда ключевых клавиш на клавиатурах стандартных терминалов практически не было (нередко даже стрелок не было). Поэтому в нем есть несколько режимов. В "нормальном" режиме обычные буквенные клавиши играют роль ключевых (например, клавиши h,j,k,l играют роль стрелок), в режиме "вставки" Вы просто набиваете текст, в "визуальном" режиме Вы маркируете текст, и т.д.

- В "нормальный" режим попадают нажатием <Esc>.
- Простое редактирование текста реализовано так: ходить по тексту Вы можете с помощью стрелок (или клавиш h,j,k,l — как Вам больше нравится) стирать буквы Вы можете клавишей <Del>. А вот чтобы что-то допечатать, следует перейти в режим "вставки". Для этого надо нажать клавишу <i> (при этом внизу высветится слово "Insert"). Вернуться в "нормальный" режим можно нажатием клавиши <Esc>.
- Чтобы посмотреть Help, следует, находясь в нормальном режиме (если есть сомнения насчет режима — жмите <Esc> без колебаний: от лишнего <Esc> беды не будет), набрать команду

  ```
  :help
  ```

  (она высветится в самой нижней строке) и нажать <Enter>. Выход из Help — набрать команду

  ```
  :q
  ```

  (она высветится в самой нижней строке) и нажать <Enter>. Фактически, это выход из редактирования файла, в котором содержится Help. Заметьте, что пока Вы читаете Help, Вы, в сущности, редактируете сразу два файла (Help и свой собственный).
- Чтобы отмаркировать блок, надо поставить курсор на начало блока, перейти в "визуальный" режим нажатием клавиши <v>, и переместиться на конец блока.

  Далее блок можно удалить, нажав клавишу <d>. Кроме того, его можно скопировать, если нажать клавишу <y>, переместиться в нужное место (туда, куда Вы хотите его вставить), и нажать клавишу <p>.

- Чтобы удалить строку, в которой находится курсор, надо 2 раза подряд нажать клавишу <d>.
- Чтобы найти ключевое слово, в нормальном режиме наберите команду

  ```
  /myword
  ```

  (она высветится в нижней строке) и нажмите <Enter>. Слово запоминается, так что еще раз найти его можно, просто нажав </> и <Enter>.
- Чтобы заменить ключевое слово на что-то другое, в нормальном режиме наберите команду

  ```
  :%s/oldword/newword/gc
  ```

  (она высветится в нижней строке) и нажмите <Enter>. После этого для каждого найденного слова будет задан вопрос: заменить (y), пропустить (n), заменить все (a), прервать (q).
- В нормальном режиме отменить последнюю команду редактирования можно, нажав клавишу <u>.
- Чтобы сохранить результаты редактирования без выхода из редактора, в нормальном режиме наберите команду

  :w

  и нажмите <Enter>.
- Чтобы выйти и сохранить результаты редактирования, в нормальном режиме наберите команду

  :wq

  и нажмите <Enter>.
- Чтобы выйти без сохранения результатов редактирования, в нормальном режиме наберите команду

  :q!

  и нажмите <Enter>. При этом никто у Вас ничего не переспросит.
- Если Вы ухитрились случайно наоткрывать несколько файлов и в каждом что-то поменять, то предыдущая команда не сработает. Чтобы гарантированно выйти из vi без сохранения результатов редактирования, нажмите на всякий случай <Esc>, чтобы заведомо оказаться в нормальном режиме, наберите команду

  :qa!

  и нажмите <Enter>. Это абсолютное оружие, vi будет вынужден Вас выпустить.

5.4 Другия редакторы

Еще есть редактор "`nano`". Обычный текстовый редактор, при его написании старались, чтобы он поедал мало памяти и быстро работал.

Еще есть редактор "`emacs`". В определенном смысле это антипод `nano` — при прочих равных он занимает довольно много памяти и довольно медленно работает. Это, пожалуй, самый развесистый из перечисленных редакторов — по замыслу его можно превратить в интегрированную среду для любого языка — от "C" до "MAXIM'ы".[14] Следует подчеркнуть, что, по нашему мнению, удобство таких сред сильно преувеличено. Возможность скомпилировать программу и пустить ее на счет (или запустить отладчик) прямо из редактора дает экономию времени, которая, к сожалению, иллюзорна. При отладке программ большая часть времени тратится вовсе не на выход из редактора и отдельный запуск компилятора. Что касается встроенного Help'а, боевой раскраски текста, проверки и подсказки по синтаксису команд и стандартных функций, и т.п., то это обоюдоострое оружие. Когда эти удобства есть, они действительно помогают, но устойчивая привычка к ним делает человека довольно беспомощным, если они вдруг исчезают. А исчезают они часто. Например, при работе с удаленного терминала на вычислительном сервере.

Еще есть очень много других текстовых и оконных (под XWindows) редакторов.

6 Компилятор gcc

Исторически компилятор "`cc`" был неотъемлемой частью UNIX, и в каждом диалекте был свой компилятор, и этим компилятором вполне можно было пользоваться при компиляции своих счетных программ.

В настоящее время все (почти все) пользуются GNU компилятором gcc ("гнутый" компилятор). Дело в том, что существует проект GNU, в рамках которого разрабатывается не просто бесплатное, а open-source программное обеспечение. Вы можете скачать исходные c-файлы, (если Вы очень любопытны, Вы можете попытаться в них разобраться, а если Вы страдаете манией величия, Вы можете даже попытаться что-то там поменять), а потом скомпилировать программу. Собственно, относительно недавно это и был основной способ инсталляции программ на UNIX'овский компьютер. Он в определенном смысле оптимален — ведь у Вас есть возможность скомпилировать программу с точным учетом свойств Вашего конкретного компьютера (железа) и операционной системы. Типовая процедура выглядит так: вы скачиваете исходники (source), обыкновенно в виде prog-4.56.tar.gz, разархивируете их в какую-либо директорию (см. выше), заходите в эту директорию, пускаете скрипт "configure":

```
./configure
```

который смотрит на Ваше железо и на уже установленное программное обеспечение, чтобы сконфигурировать будущую компиляцию. Когда он отработает, Вы даете команду "make":

```
make
```

После этого можно пойти попить чаю, обыкновенно времени как раз хватает. Если возникли ошибки, то следует читать файлы "README" или "INSTALL". Если ошибок не было, то обычно программой уже можно пользоваться. Иногда бывает необходима установка. К сожалению, для установки по умолчанию, которая пускается командой

[14] В эпоху, когда на персоналку еще невозможно было втиснуть UNIX (винчестер — 40 Mb, оперативная память — 500 Kb, 286-й процессор на частоте 20 MHz), а Windows (к счастью) еще не существовало, основной операционной системой был DOS. Так вот, под DOS разными фирмами писалось бесчисленное количество как компиляторов, так и интегрированных сред для разных языков: Turbo Basic, Turbo Pascal, Turbo C, Turbo Prolog (Borland); Quick Basic, Quick C (Microsoft); Watcom C, и многия другия. Нынешний Visual C является наследником Quick C, и заодно великолепной иллюстрацией направления работы фирмы Microsoft: забота об оформлении продукта превалирует над заботой об эффективности. Достаточно сказать, что код, сгенерированный компилятором Watcom от 1998 года, работает раза в полтора быстрее, чем код Visual C от 2010 года. А ведь компилятор Watcom от 1998 года, разумеется, абсолютно ничего не знает про современные процессоры. Кстати, код, сгенерированный gcc, работает в полтора-два раза быстрее, чем код Watcom.

```
make install
```

скорее всего понадобятся права администратора. Как провести установку "только для себя", т.е. не в `/usr` или `/usr/local`, а в свою домашнюю директорию, обычно изложено в файлах "README" или "INSTALL".

К сожалению, в последнее время такой способ инсталляции превращается в бег с препятствиями — как правило любая достаточно сложная программа использует не только стандартные библиотеки, но и полустандартные, а то и совсем экзотические. Кроме того, процесс компиляции существенно зависит от версии компилятора — нередко оказывается, что, скажем, (это пример из довольно древних времен) под gcc 2.3 и под gcc 2.5 программа не собирается, а под gcc 2.4 — все чудесно проходит. Поэтому в настоящее время разработчики Linux как правило заранее компилируют более или менее стандартный набор программ, частично включают его в дистрибутив системы (именно поэтому размер дистрибутива варьируется от одного CD [700Mb] до шести DVD [около 25 Gb]), частично выкладывают на сайте, и включают в систему Package Manager, который позволяет более-менее автоматически установить нужную программу (он ставит не только заказанную программу, но и необходимые ей библиотеки и другие программы)[15].

Вернемся, однако, к `gcc`. Сейчас это де факто стандартный компилятор в мире UNIX. Он поддерживает почти все известные процессоры (и учитывает их особенности при оптимизации), имеет громадное количество настроек, большие возможности в оптимизации кода, допускает cross-компиляцию (т.е. можно, скажем, работая на машине с Intel'овским процессором, скомпилировать код для Sun'овской рабочей станции. Впрочем, как показывает опыт, этой возможностью лучше не пользоваться), и т.п.

Описание возможностей `gcc`, можно посмотреть, дав команду

```
info gcc
```

Это описание содержит довольно много информации (даже если выкинуть разделы, относящиеся к специфике различных процессоров, с большинством из которых Вы скорее всего не встретитесь). В принципе, с ним желательно было бы ознакомиться в полном объеме, но мы ограничимся только той информацией, без которой обойтись нельзя.

Вы должны отчетливо понимать, что создание исполняемой программы состоит из двух фаз — компиляции и линкования. Компиляция суть создание двоичного кода для каждого отдельного с-файла (при этом из файла prog1.c создается объектный файл prog1.o). Линкование в статическом режиме суть сборка из нескольких объектных файлов и стандартных библиотечных файлов (из библиотечных файлов берутся стандартные функции) исполняемого файла. При этом разница между библиотечными и объектными файлами очень невелика — просто объектный файл всегда вставляется целиком, даже если не все его функции реально используются в исполняемом файле, а из библиотеки выбирается только то, что используется. При динамическом линковании стандартные функции в исполняемый файл не вставляются, они подгружаются в момент запуска исполняемого файла. Если Вы собираетесь компилироваться на одной машине, а пускаться на другой, обязательно линкуйтесь в статическом режиме (флаг "`-static`"). В противном случае может оказаться, что на другой машине динамические библиотеки лежат не там (и поэтому исполняемый файл их не найдет), либо на другой машине лежит другая версия динамических библиотек (и поэтому исполняемый файл не сможет их использовать), или еще что-нибудь в этом духе.

В настоящее время причин, по которым Вашу счетную программу следовало бы распихать по нескольким с-файлам, можно сказать, не существует. Раньше процесс компиляции шел довольно долго, так что имело смысл отладить и откомпилировать раз и навсегда один кусочек

[15] В принципе, эти программы можно устанавливать и вручную, если скачать соответствующие rpm-файлы и разобраться в их "зависямостях" ("dependencies").

программы, потом другой, и т.д., а потом собрать из кусочков целое. На современных машинах компиляция идет достаточно быстро, так что компиляция всего Вашего кода занимает не очень много времени.[16] Некоторые не любят редактировать один большой файл. В этом случае можно разбить его на кусочки, а потом вставить эти кусочки в головной файл директивой "#include". С точки зрения компилятора это будет означать, что существует ровно один с-файл с кодом.

Поэтому далее мы будем считать, что с-файл с программой один.

Команда "`gcc`" позволяет запустить сразу оба процесса (компиляцию и линкование) и передать им соответствующие настроечные параметры ("флаги"). Впрочем, можно ограничиться только компиляцией или только линкованием (например, флаг "`-c`" заставит ограничиться только компиляцией).

"Минимальный" запуск gcc может выглядеть так:

```
gcc myprog.c
```

Результат может получиться довольно разнообразный. Если в "myprog.c" содержатся синтаксические ошибки, то будет выдан их перечень, с номерами строк и короткой (и довольно невразумительной) диагностикой. Следует понимать, что нередко первые ошибки, встретившиеся в файле, приводят к тому, что компилятор весь последующий текст воспринимает наперекосяк ("наведенные ошибки"). Например, лишняя закрытая фигурная скобка наведет его на мысль, что тело функции кончилось, и он будет ругаться на строки, которые были бы совершенно законны внутри тела функции. Поэтому ошибки следует исправлять строго сверху вниз.

Если ошибок нет, и если в "myprog.c" никаких особенных функций не используется (для нас с Вами особенные функции — это главным образом математические функции, скажем, "sin" или "sqrt"), то все пройдет успешно, и будет создан исполняемый файл с диким именем "a.out". Его можно будет запустить из командной строки:

```
a.out
```

(это если текущая директория "`.`" содержится в переменной (environment variable) PATH — "путь содержит текущую директорию"), или

```
./a.out
```

(в противном случае). Что произойдет дальше — целиком на Вашей совести. Что Вы написали в "myprog.c", то машина и сделает.

Если же в "myprog.c" используется хоть одна математическая функция (скажем, "sin" или "sqrt"), то компиляция пройдет нормально, а линкер начнет ругаться. Он не найдет эти функции, пока Вы не укажете явно библиотеку, в которой их следует искать. Так что надо добавить флаг "`-l`" (буква <*l*> нижнего регистра). Если Вы добавите в команду "`-lm`", это будет означать команду линкеру поискать недостающие функции в математической библиотеке:[17]

```
gcc myprog.c -lm
```

В этом случае опять-таки будет создан исполняемый файл с именем "`a.out`".

Разумеется, использовать имя "`a.out`" для всех исполняемых файлов, которые Вы собираетесь создавать, было бы неразумно. Поэтому следует использовать флаг "`-o`" (буква <o> нижнего регистра):

```
gcc -o ispoln_file_myprog myprog.c -lm
```

Вообще-то лучше не фантазировать, а называть исполняемые файлы попросту:

```
gcc -o myprog myprog.c -lm
```

[16] Единственное исключение — cuda-компилятор nvcc. Если в Вашей программе используется большое количество математических функций, он может компилировать и 10 минут.

[17] Имя файла с математической библиотекой — "`libm.a`", т.е., как нетрудно догадаться, математические функции как правило прилинковываются статически.

И, напоследок, коротенькая информация об оптимизации кода.

- Во-первых, "универсальная" оптимизация. Ее уровень можно задать флагом "`-O`" (буква <O> верхнего регистра). Как правило, можно не думая писать "`-O3`" (по умолчанию установлен не 3-й, а 2-й уровень). Но при этом надо отчетливо понимать, что оптимизация — это не только "unwind loops" и "inline functions". Такие виды оптимизации довольно естественны и не должны приводить к ошибкам в работе. Оптимизация — это еще и отказ от некоторых проверок. Вы должны всегда помнить, что разное поведение программы при разных уровнях оптимизации — это отчетливый признак той или иной неаккуратности в написании. Это как раз те неаккуратности, которые приводят к непредсказуемому поведению программы (при каких-то параметрах — считает, а при других — не считает). Наиболее распространенная неаккуратность такого рода — это попытка завести много автоматических переменных внутри функции. Вы должны знать, что если внутри любой функции заводится переменная, не снабженная модификатором "static", то она живет не в области данных, а в стеке. Так что, заведя, скажем, double массив длиной в 1000 элементов, Вы заведомо стек переполните. Проверка переполнения стека выкинута ради большей скорости работы. Правильно работать программа с переполненным стеком, разумеется, не может, но что-то она делать будет, иногда даже что-то похожее на то, что Вы написали. Но время от времени (или часто, или всегда), будет происходить что-то неожиданное.

- Во-вторых, оптимизация floating-point математики. Стандарт ANSI предполагает, что все знаки, выдаваемые математическими функциями во всей области определения функции и во всем диапазоне изменения double всегда "правильные" (т.е. удовлетворяют соглашениям, установленным стандартом ANSI). Иногда это сильно замедляет работу. Если допустить, что ответы будут слегка[18] отличаться от стандартных, то счет может существенно ускориться. Флаг "`-f`" управляет работой floating-point математики, и если Вы напишете "`-ffast-math`", то Вы разрешите компилятору отойти от стандарта ANSI, и программа начнет работать заметно быстрее.

- В-третьих, учет архитектуры конкретного процессора. При том, что компилятор располагает достаточно полной информацией о машине, на которой он установлен, довольно часто он генерирует просто "generic i386" код.[19] Так что если Вы прямо скажете "`-march=i686`", то ускорение может оказаться вполне заметным.[20] Более тонкие различия между процессорами тоже можно учесть (см. подробное описание gcc), но это, скорее всего, не окупится — несколько процентов ускорения не стоят затраченных усилий.

Итак, если Вы компилируете отлаженную счетную программу, то Вы, скорее всего, должны использовать такую команду:

```
gcc -o myprog -O3 -ffast-math -march=i686 myprog.c -lm
```

порядок флагов здесь более-менее произволен, только флаг линкера "`-lm`" лучше ставить в конце.

7 Запуск программ

В сущности, все, что до сих пор было изложено — это запуск программ. Команда "`cat`" — это двоичный исполняемый файл, который лежит в директории "`/usr/bin`", мало того, Вы

[18] Разумеется, речь не идет о неверном порядке ответа, или об ошибке где-нибудь в первых 10 знаках ответа.

[19] Вероятно, чтобы была возможность скопировать исполняемый файл на любую другую машину.

[20] Бывает, что компилятор по умолчанию генерит "i686" код. В этом случае ускорения не будет, но и вреда от этого лишнего флага тоже скорее всего не будет.

должны уметь такую нехитрую программу писать самостоятельно. То же самое (почти то же самое) можно сказать про все остальное — “`ls`”, “`mc`”, “`vi`”, “`gcc`”, “`a.out`”, и т.д.

7.1 Запуск программ

Простейший запуск заключается в том, что Вы набираете полное имя исполняемого файла (это если директория, в которой он лежит, не содержится в “пути” — environment variable “PATH”), либо просто имя исполняемого файла (это если директория, где он лежит, в пути есть), и нажимаете <Enter>. Иначе говоря, если в Вашей домашней директории лежит Ваша счетная программа “myprog”, то пустить ее можно командой:

```
/home/studfve37/myprog
```

Либо, если сделать “`/home/studfve37`” текущей директорией, командой:

```
./myprog
```

А если текущая директория содержится в пути, то попросту командой

```
myprog
```

Посмотреть, что содержится в пути, можно с помощью команды

```
echo $PATH
```

Вы увидите перечень директорий, разделенных двоеточиями, что-нибудь вроде:

```
/usr/bin:/usr/local/bin:.:/home/username/bin:/usr/bin/X11:/usr/X11R6/bin:/bin
```

Вообще говоря, текущей директории “ `.` ” в этом перечне может и не быть (это делается из соображений сетевой безопасности), но иногда она есть.

При простейшем запуске программа работает, пишет что-то на терминал (лучше, конечно, писать в файл), теоретически готова к вводу с клавиатуры (хотя, конечно, счетная программа не должна этого делать!), т.е. программа занимет (“захватывает”) терминал. Других команд (программ) с этого терминала запустить нельзя, пока программа не завершится. Более того, если убить терминал, то умрет (прекратит работу) и сама программа. Довольно ясно, что для счетной программы, которая считает, скажем, неделю, это совершенно неприемлемо. Дело в том, что с очень высокой вероятностью за неделю коннекция со счетным сервером по тем или иным причинам оборвется, и, следовательно, умрет терминал, а вместе с ним умрет и Ваша программа.

Есть еще возможность пустить программу в фоновом режиме. Для этого достаточно в командной строке после имени программы добавить конъюнкцию <`&`> (то, что лежит рядом с цифрой “7”):

```
./myprog &
```

Это приведет к тому, что терминал до некоторой степени освободится. Именно, появится командная строка, с которой можно будет пускать другие программы. К сожалению, пользы от этого не очень много. Например, неразумно было бы пустить программу на счет, а потом заняться редактированием какого-нибудь файла. Целесообразнее пустить два терминала. Дело в том, что когда Ваша программа, пущенная в фоновом режиме, что-то печатает на экран (а не в протокольный файл), то эта выдача чудесным образом появляется там, где находится курсор. Разумеется, редактируемый файл от этого не испортится, но на экрана будет твориться нечто не слишком вразумительное. Совершенно так же, не очень разумно пускать с одного терминала на счет сразу несколько программ.[21] Довольно ясно, что их выдача начнет интерферировать.

[21] А пускать на счет несколько программ одновременно имеет смысл даже на обычной современной персоналке — если у вас четырехядерный процессор, то Вы можете пустить четыре счетные программы, и они почти не будут друг другу мешать. Разумеется, если эти программы не ведут себя слишком нагло при использовании ресурсов системы.

К сожалению, запуск в фоновом режиме не решает проблему конечной жизни терминала. Вообще-то в документации утверждается, что программа, пущенная в фоновом (background) режиме, умирает вместе с терминалом, так же, как и программы foreground-режима. Опыт показывает, что это не вполне так. Бывают системы, где умирает, а бывают системы, где не умирает. Разумеется, полагаться на это не следует. Поэтому при реальном запуске долгоиграющей программы ее надо отцепить от терминала. Это делается с помощью команды "`nohup`":

```
nohup ./myprog &
```

После этого все, что программа печатает, будет перенаправлено в файл "`nohup.out`", лежащий в текущей директории.[22] Кроме того, программа перестанет зависеть от терминала. Вы можете сказать "`exit`", даже погасить ту машину, с которой Вы подсоединялись к серверу, а Ваша программа все равно продолжит работать.

7.2 Управление приоритетами

Еще одна вещь, которую необходимо знать — это управление приоритетами. В отличие от Windows, система приоритетов UNIX работает очень качественно. Если Вы выставите низкий приоритет для, скажем, шести счетных программ, то Вы даже не заметите их присутствия, сидя за терминалом. В то же время они охотно съедят все оставшееся от Вас свободное процессорное время.

Политика приоритетов на разных счетных серверах бывает очень разная. Обыкновенно доминирует система джентельменской анархии. Правила хорошего тона заключаются в том, что пуская программу надолго, Вы даете ей наинизший приоритет. Соответственно, если Вы знаете, что Ваша программа считает около получаса, то Вы вправе не снижать ей приоритет — фоновые программы с наинизшим приоритетом могут на полчаса и остановиться: если бы Вы делили с ними процессорное время, Вам пришлось бы зря сидеть лишний час. Иногда администраторы вводят принудительное джентельменство — пишут скрипт, который делает это автоматически. Он просыпается, скажем, каждые 5 минут, и проверяет, не слишком ли долго считает программа с несниженным приоритетом. Если долго — снижает приоритет на пяток ступеней, если совсем долго — еще на пяток, и т.д.

Понизить приоритет очень просто — для этого существует команда "`nice`":

```
nice -19 ./myprog &
```

В действительности, надо, разумеется, еще и отцепиться от терминала:

```
nohup nice -19 ./myprog &
```

Параметр "19" означает наинизший приоритет. В общем-то, можно, конечно, экспериментировать с промежуточными значениями в диапазоне $0 \ldots 19$ (условно говоря, 10 — наполовину пониженный, 5 — на четверть пониженный, и т.д.), но опыт показывает, что в реальной работе чаще всего встречаются крайности: или совсем без "`nice`", или "`nice -19`".

7.3 Просмотр процессов, живущих в системе

Довольно ясно, что кроме всего перечисленного, надо еще уметь выяснять, как поживает Ваша программа (спустя, скажем, пять дней счета) и уметь убивать ее, если она делает что-то не то.

Для того, чтобы выяснить, какие процессы живут на машине, существует команда "`ps`". В простейшем варианте можно сказать просто

[22] Заметьте, что проблема интерференции выдачи нескольких счетных программ сохраняется. Если Вы запустите с помощью "`nohup`" несколько программ из одной директории, то все они начнут писать в один и тот же файл "`nohup.out`".

```
ps
```

Выдача будет очень коротенькая, более того, фоновые процессы показаны не будут. Поэтому следует добавить флаги. К сожалению, как Вы, наверное уже заметили, синтаксис флагов не вполне унифицирован. Для некоторых команд в некоторых системах дефис добавляется ("`ls -l`"), для некоторых — нет ("`tar xvf`"). В данном случае, скорее всего, не добавляется. Это синтаксис, принятый в BSD (один из диалектов UNIX), но он (как правило) допускается и в других системах. Впрочем, иногда система допускает оба варианта синтаксиса.

Флаги, которые Вам могут пригодиться, это:

- `x` — показывать процессы, не привязанные к какому-либо терминалу.
- `a` — (all) показывать процессы всех пользователей
- `l` — давать гораздо более подробную информацию по каждому процессу.

При этом допускается любая комбинация флагов, например

```
ps ax
```

или

```
ps alx
```

К сожалению, эти флаги не очень унифицированы. Например, Вы можете встретиться с системой, где вместо флага "`a`" ("all") надо использовать "`e`" ("every"). В любом случае, локальные особенности следует уточнять с помощью "`man ps`" или "`info ps`".
В общем-то, выдача команды "`ps`" должна быть Вам более-менее понятна. Вас, главным образом, должен интересовать "`PID`" процесса ("индекс", номер процесса), потому что все управление процессом идет через этот индекс.[23] Например, если Вы хотите понизить приоритет своей программы (допустим, ее `PID` — 16384), Вы можете набрать команду

```
renice 15 16384
```

(Обратите внимание, что в команде "`renice`" дефис перед параметром не ставится, в отличие от команды "`nice`"). Тут надо заметить, что сами Вы можете только понижать приоритет своих процессов. Повысить приоритет может только администратор, и это правильно.

Далее, допустим, что вы хотите прекратить исполнение программы. Тогда следует набрать команду

```
kill 16384
```

Как правило, это действительно убивает (завершает) Ваш процесс. Бывает, однако, что этой команды не хватает. Тогда следует усилить команду:

```
kill -9 16384
```

Эта команда убьет Ваш процесс с вероятностью, очень близкой к единице. Тут невредно напомнить, что если Вы пустили Вашу программу простейшим образом (в foreground-режиме), то оборвать ее можно попросту — нажатием <Ctrl-c>.

Еще одна очень полезная программа, которую необходимо знать — это команда

```
top
```

Эта команда — что-то вроде сводного монитора загрузки ресурсов компьютера. Она выдает, насколько загружен процессор, память, жеский диск, и печатает упорядоченный список программ, которые наиболее активно поедают время процессора. Опять-таки, как нам кажется, выдача должна быть Вам вполне понятна. Тут есть и доля памяти (`%MEM`), и доля CPU (`%CPU`), занимаемые процессом (при этом не надо удивляться, что на четырехядерной машине каждая

[23] Еще Вас может заинтересовать "`UID`" — это индекс пользователя, жестко привязанный к его имени; статус процесса (run, sleep, zombie, и т.п.); номер терминала; память, занятая процессом; NI — насколько снижен приоритет; время исполнения процесса; команда, которой был запущен процесс.

из четырех счетных программ будет занимать 100% CPU); время работы процесса; NI (насколько снижен приоритет командами "`nice`" или "`renice`"); команда, которой был запущен процесс; память, которую занимает процесс.[24]

7.4 Очень примитивные скрипты

Ну и, наконец, немного о писании скриптов. Вернее, о писании тех примитивных скриптов, которые Вам могут пригодиться.

Допустим, у Вас есть счетная программа, которая должна пускаться много раз с разными наборами начальных данных. В общем-то можно было бы добавить в эту программу цикл, который эти данные перебирает. Можно написать программу на "C", которая будет пускать исходную программу с разными начальными данными. Но иногда бывает удобнее написать скрипт. Это текстовый файл (например, "myscript") со, скажем, таким содержанием:

```
nohup nice -19 myprog 1.1 infile outfile1
gzip outfile1
nohup nice -19 myprog 1.2 infile outfile2
gzip outfile2
nohup nice -19 myprog 1.3 infile outfile3
gzip outfile3
```

Дальше Вы делаете его исполняемым:

```
chmod +x myscript
```

И пускаете его на исполнение:

```
nohup myscript &
```

После этого все команды, содержащиеся в файле, начнут исполняться — одна за другой, причем пока не отработает предыдущая, следующая не запустится.

Следует заметить, что на машине с несколькими ядрами (например, четырьмя) соблазнительно было бы написать:

```
nohup nice -19 myprog 1.8 infile outfile1 &
nohup nice -19 myprog 2.7 infile outfile2 &
nohup nice -19 myprog 3.6 infile outfile3 &
nohup nice -19 myprog 4.5 infile outfile4
nohup nice -19 myprog 5.4 infile outfile5 &
nohup nice -19 myprog 6.3 infile outfile6 &
nohup nice -19 myprog 7.2 infile outfile7 &
nohup nice -19 myprog 8.1 infile outfile8
```

К сожалению, это не очень правильная идея. Она была бы правильной, если бы первые четыре запуска программы (строки 1-4) работали строго одно и то же время. Так почти никогда не бывает. Поэтому может случиться так, что четвертый запуск еще не завершился, а первые три уже отработали — в этом случае будут простаивать три ядра из четырех. Может случиться и наоборот: четвертый запуск уже отработал, а первые три еще не завершились — в этом случае в добавок к ним заработают следующие четыре запуска (строки 5-8), итого 7 процессов на четыре ядра.

Как уже было сказано, на практикуме Вам не удастся поэкспериментировать с "`chmod`". Но это не значит, что Вы не сможете писать и пускать скрипты. Дело в том, что "myscript" необязательно делать исполняемым. Его можно запустить и другим[25] способом:

[24] Тут надо быть внимательным — память бывает всякая: VIRT, RES, SHR, и т.п. Например, RES=CODE+DATA, VIRT=RES+SWAP, SHR — это подгруженные динамические библиотеки, которые процесс может "share" (использовать совместно) с другими процессами, и т.п. Неважно, что все это значит, Вам лучше ориентироваться по доле памяти `%MEM`.

[25] В действительности, тем же самым.

```
sh myscript
```

Или, скажем,

```
bash myscript
```

8 Заключение

Как уже было сказано, эта шпаргалка представляет собой некоторое подмножество сведений о UNIX. При отборе сведений мы руководствовались, во-первых, необходимостью для практической работы, во-вторых, понятностью в рамках этого подмножества, в третьих, замкнутостью подмножества (Вы должны помнить определение замкнутости).

Однако восприятие человека, знакомого с предметом, существенно отличается от восприятия человека, знакомящегося с ним в первый раз. (Именно с этим связана невозможность преподавательской деятельности. “В электродинамике никакого материала нет, все это очевидные вещи, что они учат там целый год?!”).

Поэтому мы обращаемся к читателям с настоятельным требованием: во-первых, указать на непонятные места в этой шпаргалке. Во-вторых, указать на места, где происходит ненужное разжевывание очевидных вещей.