ЛЕКЦИЯ 8. ОСНОВЫ КОМАНДНОГО ИНТЕРФЕЙСА В LINUX MINT

8.1. Командная строка

Основой командного интерфейса является командная строка, начинающаяся с приглашения для ввода. Далее оно будет обозначаться — $, если речь идёт о сеансе обычного пользователя, или символом решётки — #, для приглашения строки в сеансе администратора.

Командная директива образуется:

- именем команды, однозначно определяющим ее назначение;
- опциями, определяющими условия выполнения команды;
- аргументами — объектами, над которым осуществляются действия.

Очевидно, что имя команды является обязательным компонентом, тогда как опции и аргументы могут и отсутствовать (или подразумеваться в неявном виде по умолчанию).

Еще один непременный компонент командной директивы — это специальный невидимый символ конца строки: именно его ввод отправляет команду на исполнение. В обыденной жизни этот символ вводится нажатием и отпусканием клавиши Enter. Почему обычно и говорят: для исполнения команды нажмите клавишу Enter. Тот же эффект, как правило, достигается комбинацией клавиш Control+M. Символа конца командной строки, знаменующего исполнение команды, мы на экране не видим. Однако важно, что это — такой же символ, как и любой другой (хотя и имеющий специальное значение).

Команды, опции и аргументы обязательно разделяются между собой пробелами. Кроме того, опции обычно предваряются (без пробела) символом дефиса или двойного дефиса.

Для правильного применения команд, конечно же, нужно знать их имена и назначение. Однако есть метод автодополнения. Благодаря этому методу для любой команды достаточно ввести первые несколько ее символов — и нажать клавишу табуляции (Tab). И, если введённых буковок достаточно для однозначной идентификации, полное имя команды волшебным образом возникнет в строке. Если же наш ввод допускает альтернативы продолжения имени — все они высветятся на экране (сразу или после повторного нажатия на табулятор), и из них можно будет выбрать подходящую.

Указания только имени команды достаточно для выполнения лишь некоторых из них. Типичный пример — команда ls (от list), предназначенная для просмотра имен файлов (строго говоря, содержимого каталогов). Данная без аргументов, она выводит список имен файлов, составляющих текущий каталог, представленный в некоторой форме по умолчанию, например, в домашнем каталоге пользователя это будет выглядеть примерно так:

```
$ ls
Desktop/  Downloads/          Music/  Pictures/  Templates/
Documents/  lost+found/  mytmp/  Public/   Videos/
```

Исполнение же многих других команд невозможно без указания опций и (или) аргументов. Для них в ответ на ввод одного её имени часто следует не сообщение об ошибке (или не только оно), но и краткая справка по использованию команды. Например, в ответ на ввод команды для создания каталогов mkdir (от make directory) последует следующий вывод:

```
usage: mkdir [-pv] [-m mode] directory ...
```

Для одних опций достаточно факта присутствия в командой директиве, другие же

требуют указания их значений (даваемых после опции обычно через знак равенства). В приведённом примере команды mkdir к первым относятся опции -v (или —verbose), предписывающая выводит информацию о ходе выполнения команды, и -p, которая позволяет создать любую цепочку промежуточных каталогов между текущим и новообразуемым (в случае их отсутствия).

А вот опция -m, определяющая атрибуты доступа к создаваемому каталогу, обязательно требует указания значения — этих самых атрибутов, заданных в символьной форме.

Многие опции имеют две формы — краткую, односимвольную, и полную, или многосимвольную, Некоторые же опции могут быть даны только в многосимвольной форме. Общее правило здесь таково: если одного символа достаточно для однозначного определения опции, могут употребляться обе формы в качестве равноправных. Однако поскольку количество символов латинского алфавита ограниченно (а человеческая фантазия, конструирующая опции — безгранична), при большом количестве опций одной команды некоторые из них приходится делать исключительно многосимвольными.

Продемонстрирую это на примере опций все той же команды mkdir. Полный их список будет следующим:

-m, --mode=MODE установить код доступа
(как в chmod)
-p, --parents не выдавать ошибок,
если существует, создавать
родительские каталоги,
если необходимо
-v, --verbose печатать сообщение
о каждом созданном каталоге
--help показать помощь и выйти
--version вывести информацию
о версии и выйти

8.2. Командные конструкции

Командные конструкции — очень важный компонент интерфейса командной строки. Они позволяют объединять несколько команд воедино и выполнять различные команды последовательно или параллельно. Для этого служат специальные символы — операторы: фонового режима, объединения, перенаправления и конвейеризации.

8.2.1. Совместное выполнение команд

Простейшая командная конструкция — это выполнение команды в фоновом режиме, что вызывается вводом символа амперсанда после списка опций и (или аргументов):

```
$ command [options] [arguments] &
```

Команды для параллельного исполнения можно задать и в той же строке:

```
$ command1 & command2 & ... & commandN
```

В результате все команды, перечисленные в строке, кроме той, что указана последней, будут выполняться в фоновом режиме.

Существуют и конструкции для последовательного выполнения команд. Так, если

ряд команд разделен в строке символом точки с запятой (;)

```
$ command1 ; command2 ; ... ; commandN
```

то сначала будет выполнена команда command1, затем — command1 и так далее.

Возможна ситуация, когда результаты предыдущей команды из такой конструкции используются в команде последующей. В этом случае ошибка исполнения любой составляющей команды, кроме последней, делает невозможным продолжение работы всей конструкции. Для предотвращения таких ситуаций в конструкции из взаимосвязанных команд существует другой оператор, обозначаемый удвоенным символом амперсанда — &&. Он указывает, что последующая команда конструкции должна исполняться только в том случае, если предыдущая завершилась успешно:

```
$ ./configure && make && make install
```

Впрочем, предусмотрена и командная конструкция, в которой последующей команде предписано исполняться в том и только в том случае, если предыдущая команда завершилась неудачно. Она имеет вид

```
$ command1 || command2
```

и может служить, в частности, для вывода сообщений об ошибках.

8.2.2. Перенаправление

Следующая командная конструкция — это так называемое перенаправление ввода/вывода. Чтобы понять, что это такое, нужно помнить две вещи:

1. любая команда получает данные для своей работы (например, список опций и аргументов) со стандартного устройства ввода (клавиатуры), а результаты своей работы представляет на стандартном устройстве вывода (экран монитора);
2. в POSIX-системах любое устройство — не более чем имя специального файла, именуемого файлом устройства.

Таким образом, ничто не запрещает нам подменить специальный файл устройства ввода или устройства вывода любым иным файлом (например, обычным текстовым). Откуда и будут в этом случае браться входные данные или куда будет записываться вывод команды.

Перенаправление вывода команды обозначается следующим образом:

```
$ command > filename
```

или

```
$ command >> filename
```

В первом случае (одиночный символ >) вывод команды command образует содержимое нового файла с именем filename, не появляясь на экране. Или, если файл с этим именем существовал ранее, то его содержимое подменяется выходным потоком команды (точно также, как при копировании одного файла в другой, уже существующий). Такое перенаправление называется замещающим (или перенаправлением в режиме замещения).

Во втором же случае (двойной символ >>) происходит добавление вывода команды command в конец существующего файла filename (при отсутствии же его в большинстве случаев просто образуется новый файл). И потому это называется присоединяющим перенаправлением, или перенаправлением в режиме присоединения.

Перенаправление ввода выглядит так:

$ command < filename

Простейший случай перенаправления вывода — сохранение результата исполнения команды в обычном текстовом файле. Например, конструкция

$ ls dir1 > list

создаст файл, содержанием которого будет список файлов каталога dir1. А в результате выполнения конструкции

$ ls dir2 >> list

к этому списку добавится и содержимое каталога dir2.

При перенаправлении ввода команда получает данные для своей работы из входящего в командную конструкцию файла. Например, конструкция

$ sort < list

выведет на экран строки файла list, отсортированных в порядке возрастания значения ASCII-кода первого символа, а конструкция

$ sort -r < list

осуществит сортировку строк того же файла в порядке, обратном алфавитному.

В одной конструкции могут сочетаться перенаправления ввода и вывода, как в режиме замещения, так и в режиме присоединения. Так, конструкция

$ sort -r < list > list_r

не только выполнит сортировку строк файла list (это — назначение команды sort) в обратном алфавитному порядке (что предписывается опцией -r, происходящей в данном случае от *reverce*), но и запишет ее результаты в новый файл list_r, а конструкция

$ sort -r < list >> list

добавит по-новому отсортированный список в конец существующего файла list.

8.2.3. Конвейеры

Возможности построения командных конструкций не ограничиваются перенаправлением ввода/вывода: результаты работы одной команды могут быть переданы для обработки другой команде. Это достигается благодаря механизму программных каналов (*pipe*) или конвейеров — последний термин лучше отражает существо дела.

При конвейеризации команд стандартный вывод первой команды передается не в файл, а на стандартный ввод следующей команды. Простой пример такой операции — просмотр списка файлов:

```
$ ls -l | less
```

Перенаправление вывода команды ls, то есть списка файлов, который при использовании полного формата записи (опция -l) может занимать многие экраны, на ввод команды less позволяет просматривать результат с ее помощью постранично или построчно в обоих направлениях.

Конвейеризация команд может быть сколь угодно длинной. Возможно также объединение конвейеризации команд и перенаправления в одной конструкции. Кроме того, команды в конструкции могут быть сгруппированы с тем, чтобы они выполнялись как единое целое. Для этого группа команд разделяется символами ; и пробелами, как при последовательном выполнении команд, и заключается в фигурные скобки. Так, если нам требуется перенаправить вывод нескольких команд в один и тот же файл, вместо неуклюжей последовательности типа

```
$ command1 > file ; command2 >> file ; ... ; commandN >> file
```

можно прибегнут к более изящной конструкции:

```
$ { command1 ; command2 ; ... ; commandN } > file
```

8.2.4. Сценарии оболочки

В самом простом случае сценарий — это просто одна или несколько команд или (и) командных конструкций с необходимыми опциями и аргументами, сохраненные в виде обычного именованного текстового файла. И предназначены они в первую очередь для автоматизации часто исполняемых рутинных операций, в частности, ввода длинных последовательностей в командной строке.

Создание пользовательского сценария — просто, для этого всего и нужно:

- создать командную конструкцию;
- поместить ее в простой текстовый файл;
- по потребности и желанию снабдить комментариями;
- тем или иным способом запустить файл на исполнение.

С принципами создания команд и командных конструкций мы в первом приближении разобрались раньше. А вот способов помещения их в файл существует множество. Можно просто ввести (или вызвать из буфера истории) нужную команду и оформить ее как аргумент команды echo, вывод которой перенаправить в файл:

```
$ echo "cp -rf workdir backupdir" > mybackup
```

Таким образом мы получили простейший скрипт для копирования файлов из рабочего каталога в каталог для резервного хранения данных.

Аналогичную процедуру можно выполнить с помощью команды cat — она способна не только к объединению файлов и выводу их содержимого, но и к вводу в файл каких-либо данных. Делается это так. Вызываем cat с перенаправлением ее вывода в файл:

```
$ cat > myarchive
```

и нажимаем Enter. После этого команда остается в ожидании ввода данных для помещения их в новообразованный файл. Причем можно выполнить ввод в несколько строк, например:

```
cd $HOME/archivedir tar cf archive.tar
    ../workdir gzip archive.tar
```

Завершив ввод тела скрипта, все той же клавишей Enter открываем новую строку и набираем комбинацию Control+D, выдающую символ окончания файла.

В результате получаем сценарий для архивирования в специально предназначенном для этого каталоге archivedir наших рабочих данных (командой tar), а заодно и их компрессии (командой gzip) — в Unix, в отличие от DOS/Windows, архивирование и компрессия обычно рассматриваются как разные процедуры.

Наконец, сценарий можно создать в любом текстовом редакторе.

Комментариями в шелл-сценариях считаются любые строки, начинающиеся с символа решетки (#) — они не учитываются интерпретатором и не принимаются к исполнению. Хотя комментарий может быть начат и внутри строки — важно только, что между символом # и концом её больше ничего не было бы. Но одна строка, начинающаяся символом решётки, в сценарии практически обязательна. И должна она быть первой и выглядеть следующим образом:

```
#!/path/shell_name
```

В данном случае восклицательный знак подчеркивает, что предваряющий его символ решетки (#) — не комментарий, а указание (т.н. *sha-bang*) на точный абсолютный путь к исполняемому файлу оболочки, для которой наш сценарий предназначен.

8.3. Утилиты командной строки

8.3.1. Команда man

Команда man предназначена для вызова экранной документации в одноименном формате. А такая man-документация почти обязательно сопровождает любую уважающую себя программу.

Для файлов man-документации предназначен специальный каталог. Обычно это /usr/share/man, разделяемый на подкаталоги, соответствующие восьми нумерованным группам. Назначение этих групп следующее:

1. man1 — команды и прикладные программы пользователя;
2. man2 — системные вызовы;
3. man3 — библиотечные функции;
4. man4 — драйверы устройств;
5. man5 — форматы файлов;
6. man6 — игры;
7. man7 — различные документы, не попадающие в другие группы (в том числе относящиеся к национальной поддержке);
8. man8 — команды администрирования системы.

Кроме того, имеется несколько подкаталогов с локализованными man-страницами, в том числе и русскоязычными, имеющими ту же структуру, хотя и обычно не полную. Так, подкаталог с русскоязычными страницами, /usr/share/man/ru, включает в себя только группы man1, man5, man7 и man8.

Для вызова интересующей документации требуется дать команду man с аргументами

— номером группы и именем man-страницы, например:

```
$ man 1 ls
```

Причём номер группы необходим только в том случае, если одноимённые документы имеются в разных группах. Для пользовательских команд он обычно не нужен, так как все равно просмотр групповых каталогов идёт сначала в man1, затем — в man8, и только потом — во всех остальных (в порядке возрастания номеров). Так что для получения информации, например, по команде ls достаточно ввести следующее:

```
$ man ls
```

после чего можно будет лицезреть примерно такую картину:

```
LS(1)          FSF        LS(1)
NAME
    ls — list directory contents
SYNOPSIS
    ls [OPTION]... [FILE]...
DESCRIPTION
    List  information  about
    the FILEs (the current directory by default).
    Sort entries alphabetically if none
    of -cftuSUX nor --sort.
```

То есть в начале man-страницы даются имя команды, которую она описывает (ls), ее групповая принадлежность (1 — пользовательские команды) и авторство (в данном случае — FSF, Free Software Foundations), или название системы. После чего обычно дается обобщенный формат вызова (SYNOPSIS) и краткое описание.

Следующая, основная, часть man-страницы — описание опций команды, и условия их применения. Далее иногда (но, к сожалению, не всегда) следуют примеры использования команды (Examples) в разных типичных ситуациях. В заключении, как правило, даются сведения о найденных ошибках (Bug Report) и приведен список man-страниц, тематически связанных с данной (See also), с указанием номера группы, к которой они принадлежат, иногда — историческая справка, а также указываются данные об авторе.

Обращение к man-страницам позволяет получить практически исчерпывающую информацию по любым командам, но только в том случае, если пользователь знает название той команды, которая требуется в данном случае. А если он только в общих чертах представляет, что это команда должна делать?

Что ж, тогда можно прибегнуть к поиску man-страниц по ключевым словам, отражающим требуемые функции. Чему призвана служить опция -k команды man. Например, директива

```
$ man -k print
```

выведет на экран список всех man-страниц, описывающих команды, имеющие отношение к печати (причем не только на принтере, но и к выводу на экран — по английски подчас это тоже будет обозначаться как print).

8.3.2. Команда sudo

Команда sudo — вторая по важности команда в Mint. Это — основной способ получения прав администратора обычным пользователем. А по умолчанию — так просто единственный, ибо при инсталляции этого дистрибутива пароль root'а не задаётся и, соответственно, доступа к аккаунту «чистого» суперпользователя нет (хотя при желании его можно получить). Команда эта дополняется утилитами visudo и sudoedit. Это узкоспециализированные средства для редактирования общесистемных конфигурационных файлов (в том числе и конфига самой sudo) обычным пользователем. Главные особенности sudo таковы:

1. во-первых, sudo по умолчанию требует указания пароля того пользователя, который получает права другого, а не пароля того, чьи права приобретаются; правда, это может быть изменено;
2. Во-вторых, действие sudo распространяется по умолчанию только на одну команду — ту, которая указывается в качестве ее аргумента; хотя и такое поведение можно изменить с помощью соответствующих опций, о чём будет сказано позднее;
3. в-третьих, sudo обеспечивает более гибкое разграничение доступа пользователей к административным правам — не только разрешая или запрещая получение оных, но и позволяя выполнять только определённый круг действий.

Этим достигается две цели: а) возможность выполнения пользователем административных действий без сообщения ему суперпользовательского пароля (и даже, как в Mint, при его отсуствтии), и б) снижение риска повредить систему вследствие забывчивости. Да, есть еще и третья, дополнительная возможность, предоставляемая sudo — протоколирование действий, выполненных в режиме администратора.

В элементарном виде применение команды sudo — элементарно же просто: требуется лишь указать в качестве ее аргумента имя команды, требующей исполнения, со всеми необходимыми последней опциями и аргументами. После этого запрашивается пароль запустившего её пользователя — и команда исполняется. Например, команда

```
$ sudo fdisk -l /dev/sda
```

данная от лица обычного пользователя, выведет информацию об указанном дисковом устройстве точно так же, как и данная root'ом.

Если от лица алминистратора нужно выполнить подряд несколько команд, делать это следует быстро — введенный первый раз пароль по умолчанию «действует» в течении 5 минут. То есть в течении этого времени в ответ на команду sudo пароль повторно запрашиваться не будет.