

- пакет info – описание основных пакетов программ и языков программирования;
- система man – основное on-line руководство по ОС;

Руководство UNIX Reference Manual содержит 8 секций:

1. Commands – команды;
2. System calls – системные вызовы;
3. Subroutines – подпрограммы;
4. Special files – описания специальных файлов системы;
5. File format and convention – формат файлов и соглашения;
6. Games – игры;
7. Macro packages and language conventions – макропакеты и языковые соглашения (для обработки текстов);
8. Administrator commands and procedures – команды и процедуры администратора

<q> – выход из Руководства (quit);

· <пробел>, <f>, <число> – пролистывание вперед (forward);

· – пролистывание руководства назад (back);

· </pattern> – поиск (search) по шаблону вперед;

· </> – поиск по предыдущему шаблону следующего (next) совпадения;

· <?> – поиск по предыдущему шаблону предыдущего (previous) совпадения.

Описание команды состоит из следующих разделов:

- NAME – имя и функция;
- SYNOPSIS – синтаксис;
- DESCRIPTION – описание функции;
- OPTIONS – описание опций;
- SEE ALSO – смежные команды;
- FILE – используемые файлы;
- DIAGNOSTIC – ответы на ошибки;
- BUGS – замеченные некорректности;
- IMPLEMENTATION – кто сделал.

Команды

cat

Выводящая последовательно указанные файлы, таким образом, объединяя их в единый поток. команда полезна в двух случаях:

- Когда требуется вывести куда-то файл без изменений;
- Когда требуется объединить более одного файла (например части одного файла, разбитого командой `split`), либо файл(ы) с потоком стандартного ввода.
- Когда требуется просмотреть содержимое файла.

```
cat myfile > abc
```

cd

Смена директории

```
$ cd /home/tanya
```

cp

Предназначена для копирования файлов из одного в другие каталоги

Чтобы скопировать файл

```
cp [-f] [-h] [-i] [-p] [--] исходный_файл целевой_файл
```

Чтобы скопировать файл или файлы в другой каталог

```
cp [-f] [-h] [-i] [-p] [-r | -R] [--] исходный_файл ... целевая_директория
```

Чтобы скопировать каталог в другой каталог

```
cp [-f] [-h] [-i] [-p] [--] { -r | -R } исходная_директория ... целевая_директория
```

echo

Предназначена для отображения строки текста.

```
user@desktop:~$echo Hello!
```

```
Hello!
```

Также может служить для записи строки в файл, если используется ">", то файл будет перезаписан, если ">>", то строка "string" будет дописана в конец файла.

```
localhost:/new# echo "string" > filename
```

```
localhost:/new# cat filename
```

```
string
```

file

file — команда Unix, предназначенная для определения типа файла.

file [-zL] [-f file] file ...

Ключи

- -f file: Считывает из указанного файла список файлов для проверки.
- -L : Определяет тип файлов, указанных по ссылке.
- -z : Определяет тип файлов, находящихся в сжатых файлах.

mkdir

команда для создания новой директории. Пример использования:

mkdir имя_директории

more

утилита, которая отображает содержимое файла постранично

mv

используется для перемещения или переименования файлов или каталогов.

- Если в качестве аргументов заданы имена двух файлов, то имя первого файла будет изменено на имя второго.

mv file1 file_1 *переименовывает файл*

- Если последний аргумент является именем существующего каталога, то mv перемещает все заданные файлы в этот каталог.

mv file ./dir/ *перемещает 'file' в 'dir/file' относительно текущего каталога*

pwd

выводит полный путь от корневого [каталога](#) к текущему рабочему каталогу.

rm

используемая для удаления [файлов](#) из [файловой системы](#)

rmdir

команда удаляет директорию

find

Поиск файлов как простейший, так и удовлетворяющий сложным

условиям.

Синтаксис :

```
find [path...] [expression] [операция]
```

[path...] – путь для поиска, по умолчанию текущая директория;

[expression] – выражение условия поиска, по умолчанию –print;

[операция] – которую нужно произвести с найденными элементами.

Пример:

Найти файл, начинающийся на букву и заканчивающийся на цифру:

```
$ find / -name [a-zA-Z]*[0-9]
```

Найти все файлы в системе, которые модифицировались за последнюю неделю:

```
$ find / -mtime -7
```

Найти все файлы в системе, которые модифицировались за последнюю неделю, и показать “длинный” листинг для этих файлов:

```
$ find / -mtime -7 -exec ls -l {} \;
```

Удалить все tmp-файлы из домашней директории текущего пользователя:

```
$ find ~ -name *.tmp -exec rm -f {} \;
```

ln

```
ln [-s] <source-file> <destination-file>
```

Shell

Shell обычно использует текстовые командные интерпретаторы sh, bash, csh, ksh, tcsh, zsh и графические среды XFree86, CDE, Photon.

Посмотреть список доступных shell можно в файле /etc/shells.

```
$ more /etc/shells
```

переназначение shell-a

```
$ chsh -s
```

В полученном списке вы можете обнаружить следующие shell:

- /bin/sh – Bourne shell (в системе есть почти наверняка);
- /bin/bash – Bourne Again shell (развитый клон Bourne shell);
- /bin/csh – C-shell с синтаксисом подобным языку C;
- /bin/tcsh – Turbo C-shell – развитие C-shell;
- /bin/ksh – Korn-shell с возможностями csh, но с синтаксисом sh, разработан Д. Корном из AT&T;
- /dev/null – Псевдо-shell, т.е. пустой, ничего не делающий.

Экспансия (подстановка) в shell

Экспансия – это процесс анализа командной строки или строки shell-script с целью нахождения в ней специальных обозначений и подстановки на их место соответствующих значений. Для просмотра результата экспансии набранной строки нажмите <Esc>+<Ctrl>+<E>. Для возврата прежней строки - <Ctrl>+<->.

Например, командная строка:

```
$ ls -l `echo ${HOME}/..`
```

интерпретируется как

```
$ ls -l /home/vasja/..
```

Порядок выполнения экспансии:

1. Проверка синтаксиса команды, если ошибка, то shell сообщает об этом и останавливает выполнение команды.

2. Подстановка результата работы команд, которые взяты в знаки ударения ` `.
3. Подстановка значений параметров и переменных начинающихся с \$.
4. Арифметическая подстановка \$(()) или \${}.
5. Разбиение строки на слова посредством интерпретации <Space>, <Tab>, <Enter>, которые не в кавычках или не экранированы.
6. Генерация имён файлов. Каждое слово, содержащее знаки ? * [по возможности, раскрывается на упорядоченный по алфавиту список имён файлов.
7. Удаление кавычек, всех кроме экранированных слешем \". Генерация имен файлов:
 - ? – любой непустой символ в заданной позиции шаблона, например, *.??? – все файлы, оканчивающиеся на “точку и три символа”;
 - * – любое количество разрешенных символов, в том числе и пустое. Например, * – все файлы, кроме начинающихся с точки;
 - [list] – любой один символ из заданных в списке, например: [-abk3] или [A-Za-z0-9_];

Понятие о стандартных потоках

В UNIX имеется разница между физической реализацией файла и его логической организацией.

Физически файл представляет собой стохастический набор блоков размещенных на диске.

Логически все файлы в UNIX организованы одинаково и представляют собой последовательность байтов. Эта логическая организация файлов распространяется и на данные в операциях ввода, вывода. Для каждого запущенного процесса UNIX создает в ОЗУ три файла – stdin, stdout, stderr и связывает их с этим процессом.

Данные, вводимые с терминала (клавиатуры), направляются в поток данных, организованный как непрерывная совокупность байтов и называющийся, стандартный ввод (stdin). Поток stdin связан с терминалом (клавиатурой).

Данные, выводимые из команды или программы, направляются в поток организованный как непрерывная совокупность байтов и называющийся стандартный вывод (stdout). Поток stdout связан с терминалом (дисплеем). Существует также стандартный поток ошибок (stderr) или поток диагностических сообщений, который по умолчанию выводится на терминал, как и stdout

Например, команда

```
$ > new-file
```

создаст или перезапишет файл new-file, а команда

```
$ ls > spisok
```

переадресует вывод команды ls в файл spisok, если файл spisok не существует, он будет создан, если существует, то его содержимое заменится на новое, а старое содержимое будет безвозвратно потеряно. Чтобы предотвратить случайное изменение существующего файла, можно включить для shell специальную переменную среды noclobber. Это можно сделать в командной строке или в сценарии инициализации командой

```
$ set -o noclobber
```

выключить режим noclobber можно командой

```
$ set +o noclobber
```

Синтаксис “>!” позволяет отменить режим noclobber и провести принудительную перезапись существующего файла, но он работает только в C-shell и Korn-shell

Оператор переадресации “>>” работает аналогично оператору “>”, за исключением того, что если целевой файл уже существует, то он не перезаписывается, а происходит дописывание в конец.

Например, команда

```
$ uname -a >> HackerVasja
```

```
$ cat /etc/passwd >> HackerVasja
```

Например, команда cat без аргументов читает свой ввод с клавиатуры:

```
$ cat
```

```
This is a new line
```

```
This is a new line
```

```
for the cat command
```

```
for the cat command
```

```
^D
```

```
$
```

Здесь ^D управляющая комбинация, означающая конец файла (обычно <Ctrl>+<D>). Несмотря на то, что всё дублируется, результат нулевой, т.к. файл с текстом не создан. Объединим cat и “>”

```
$ cat > myfile
```

```
This is a new line
```

```
for the cat command
```

```
^D
```

```
$ more myfile
```

```
This is a new line
```

```
for the cat command
```

```
$
```

Так уже лучше, файл myfile был создан и содержит текст. Попробуем переадресовать stdin команды cat с клавиатуры на файл:

```
$ cat < myfile
```

*This is a new line
for the cat command*

\$

Операторы “<” и “>” можно объединять. В следующем примере cat не имеет аргументов, но работает: cat читает текст из файла myfile и выводит его в создаваемый файл myletter:

\$ cat < myfile > myletter

Проверьте, будет ли работать команда с другим порядком операторов переадресации:

\$ cat > myfile < myletter

Встроенные документы

Иногда возникает необходимость поместить поток ввода вместе с командой. Для этого используется конструкция “ввод здесь”, состоящую из оператора “<<” и “символа-флага”. Например:

\$ mail root << w

Hello, root!

w

\$

Синтаксис конструкции “ввод здесь”:

\$ command_name arguments <<[-]symbol_flag Текст встроенного документа

...

symbol_flag где symbol_flag – это символ, ограничивающий документ; если после << задан “-”, то из вводимого документа удаляются все лидирующие табуляторы, что удобно для сценариев shell. Пример чуть сложнее, с перенаправлением в канун Нового Года текста письма для Васи из файла happy.new.year:

\$ at Dec 31 << !

cat happy.new.year | mail -s 'Happy NY!'

vasja@pupkin.lv

!\$

Проверьте, можно ли:

- использовать сам “символ-флаг” в тексте встроенного документа;
- отменить специальное значение символа-флага обратным слешем <>;
- использовать не “символ-флаг”, а “слово-флаг”.

Программные каналы

Иногда возникает ситуация, когда необходимо передать данные из одной команды в другую, т.е. послать стандартный вывод одной команды на стандартный ввод другой. Такая конструкция называется программным каналом или конвейером. Синтаксис использования оператора программного канала “|”:

```
$ command1 | command2
```

Допустим, вы хотите посчитать количество пользователей зарегистрировавшихся на данный момент в системе. Для этого нужны три команды `who`, `wc`, `rm`. Это можно сделать так:

```
$ who > temp.tmp
```

```
$ wc -l temp.tmp
```

```
$ rm temp.tmp
```

Наш пример с использованием конвейера выглядит так:

```
$ who | wc -l
```

Используя конвейер можно создавать практически неограниченное количество специализированных мощных команд, используя ограниченные функции различных простых команд. Не интерактивные команды, такие как `ls`, `pwd`, не имеющие `stdin` должны стоять в конвейере первыми. Совместно с каналами часто используются программы-фильтры, такие как `grep`, `egrep`, `sort`, `head`, `tail`, `wc`, `spell`, `sed`, `diff`. В конвейеры можно объединять не только две, но и большее количество команд. Например, нижеприведенный конвейер команд построчно выведет отсортированный в алфавитном порядке файл с информацией о пользователях системы, пронумеровав каждую строку вывода:

```
$ sort /etc/passwd | cat -n | more
```

Дублирование стандартного вывода

Программа `tee` читает данные со стандартного ввода и записывает их, во-первых, на стандартный вывод, а во-вторых, на один или более указанных в команде файлов; если файла нет, то он создается, если есть, то переписывается. Для дозаписи в файл используется ключ `-a` (от слова `append` – добавка): `tee -a file`.

Например:

```
$ cat my-letter | tee new-letter
```

– письмо скопировано в `new-letter` и на экран;

```
$ cat my-letter | tee new-letter | wc -w > size
```

– письмо скопировано в `new-letter` и отправлено в `wc`, который подсчитал количество слов в нем и сохранил это число в файле `size`.

Как и при использовании операторов перенаправления “>” и “>>”, команда `tee` сначала создает файл, а потом наполняет его содержимым.

Переадресация стандартного потока диагностических сообщений Стандартным потоком ввода, вывода и ошибок (поток диагностики) присваиваются числовые дескрипторы, соответственно 0, 1, 2. Используя эти числовые значения можно переадресовывать стандартные потоки. Как правило, диагностические сообщения отображаются на экран вместе со stdout. Однако UNIX различает потоки stdout и stderr, поэтому их можно переадресовать отдельно. Например, команда find, наткнувшись на каталог с правами, запрещающими работу для данного пользователя, выводит много диагностических сообщений на экран.

```
$ find / -name *
```

Попробуем подавить эти диагностические сообщения:

```
$ find / -name * 2>/dev/null
```

Чтобы переадресовать и stdout и stderr можно использовать два оператора “>” или “>>” и два файла, различных или одинаковых, например:

```
$ command 1> file_out 2> file-err
```

```
$ command 1> file_out 2> file-out
```

```
$ command > file_out 2>> file-err
```

Слияние потоков

Командный интерпретатор предоставляет возможность слияния потоков stdin, stdout, stderr, ссылаясь на них по номеру с предшествующим “&”. По умолчанию входным потоком операции >& является stderr, а выходным потоком – stdout. Например, вам нужно вывести сообщения об ошибках в тот же файл, куда отправлен stdout. Все нижеприведенные команды эквивалентны, они переадресуют stdout в файл myfile и туда же переадресуют stderr.

```
$ command 1> file_out_err 2>&1
```

```
$ command > file_out_err 2>&1
```

```
$ command >& file_out_err__
```

Регулярные выражения (REs)

Для того чтобы найти некоторый текст, иногда приходится формировать сложные запросы по образцу. Многие из утилит, обладающих редактирующими возможностями, используют при поиске по образцу стандартный набор специальных символов. Образец, содержащий такие символы, называется регулярным выражением (RE – Regular Expression). Регулярные выражения позволяют вести поиск вида: найти все слова из четырех букв, начинающиеся на d; или все строки, содержащие вещественные числа; или все строки, начинающиеся с правильного IP-адреса и тому подобное.

Утилиты и программы поддерживающие REs

Существует большое количество различных программ в той или иной степени поддерживающих механизм регулярных выражений. Более того, некоторые программы просто немислимы без REs, например язык perl или фильтр grep.

Вот только некоторые из таких программ:

- редакторы: emacs, vi, vim, ed, sed, ex, Nisus Writer;
- фильтры: grep, egrep, more, less;
- командные процессоры: Korn Shell;
- специальные инструменты: expr, less, flex, Expect;
- языки сценариев: Perl, PHP, Java Script, TCL, Python, awk;
- среды программирования: Delphi, MS Visual C++, Balise.

К сожалению, синтаксис и семантика RE различается в различных программах, поэтому перед использованием конкретной программы рекомендуется прочесть соответствующее руководство (man regex, man sed, man perl).

```
$ grep Vasja econstudent radiostudent compstudent
```

```
compstudent: Vasja Pupkin 7
```

или:

```
$ grep Vasja *student
```

```
compstudent: Vasja Pupkin 7
```

Примеры использования регулярных выражений в grep

Просто напечатать все строки файла:

```
$ grep ".*" compstudent
```

Отбросить все пустые строки в файле file и результат записать в new-file:

```
$ grep -v "^$" file > new-file
```

Отбросить все пустые строки и строки, состоящие из пробелов:

```
$ grep -v "^ *$" file > new-file
```

Получить список только каталогов (строки, начинающиеся на "d"):

```
$ ls -l /etc | grep '^d'
```