Обработка текста

Случайные данные

Есть несколько простых для использования источников случайных данных:

- Переменная окружения `$RANDOM`, которая при каждом чтении из неё возвращает случайное число от 0 до 32767.
- Файл `/dev/random`, чтение из которого порождает случайные байты.
- Файл `/dev/urandom`

Подробнее можно почитать в `random(7)`.

Экранирование

`"`, `'`, `;`, `\`, `#` воспринимаются shell-ом по-особому.

Но можно их экранировать

```
$ echo " ' ; \ # комментарий

$ echo \" \' \; \\ \# комментарий
" ' ; \ # комментарий

$ echo '"' "'" ';' '\' '#' комментарий
" ' ; \ # комментарий

$ echo "\" ' ; \ # комментарий"
" ' ; \ # комментарий

$ echo '" '\'' ; \ # комментарий'
" ' ; \ # комментарий
```

Экранирование

```
$ echo \ \ \ \ \\ # экранируем пробелы
    \

# можно экранировать даже перенос строки
ls / \
/var # экранировали перенос строки

# эквивалентно
ls / /var
```

Экранирование

Иногда то, что выглядит как экранирование, ведет себя ровно наоборот: если у символа не было особого смысла, то его можно создать.

```
echo -e 'n' # вывести n
echo -e '\n' # вывести перенос строки
printf '0' # вывести ноль
printf '\0' # вывести нулевой байт
```

sed

sed — Stream EDitor — выполняет для каждой строчки входных данных набор выражений.

```
# для каждой строки выполнит только выражение_1
sed выражение_1

# для каждой строки выполнится сначала выражение_1,
# затем выражение_2, затем выражение_3
sed -e выражение_1 -e выражение_2 -e выражение_3
```

sed. Выражения

Выражения в **sed** имеют следующий вид:

```
[адрес]команда[опции]
```

Если адрес отсутствует, команда выполнится всегда.

Иначе только на строках, которые соответствуют адресу.

После некоторых команд могут быть добавлены дополнительные опции.

sed. Адреса

Некоторые из возможных адресов:

```
1 # первая строка
$ # последняя строка
/apple/ # строки, содержащие `apple`
/8.7/ # строки, в которых есть подстрока `8x7`, где `x` —
произвольный символ
АДРЕС! # строки, не подходящие под АДРЕС
АДРЕС_1,АДРЕС_2 # строки между той, которая подходит под
`АДРЕС_1`, и той,
                # которая подходит под `АДРЕС_2`
АДРЕС,+5 # пять строк после той, которая подходит под АДРЕС
```

sed. Что происходит на самом деле

В **sed** есть два буфера: **pattern space** и **hold space**. Изначально оба пустые.

- считываем строку из входных данных и кладет ее в **pattern space**,
- исполняем выражения,
- по достижению конца скрипта печатаем содержимое **patern space**,
- начинаем заново

Pattern space между двумя итерациями очищается. Содержимое **hold space** не меняется.

sed. Команды

Некоторые из возможных команд:

```
d # удалить содержимое ps и начать цикл заново
p # напечатать содержимое ps
a ТЕКСТ # дописать ТЕКСТ в конец ps
y/АБВ/абв # в ps заменить "А" на "а", "Б" на "б", "В" на "в"
{ cmd ; cmd ... } # группировка команд

s/регулярное выражение/на что заменить/ # замена вхождений в ps
s;регулярное выражение;на что заменить; # альтернативный
синтаксис, чтобы не надо было экранировать /
s&регулярное выражение&на что заменить& # можно использовать
почти любой символ
```

sed.

```
cat file
# 1
# 2
# 3
# 4

cat file | sed -e 'y/0123456789/1234567890/' -e '1, 2 {s:\([0-
9]\):\1\1\1:g; p;}' -e '$d'
# ?
```

sed. Дополнительные команды

```
D # удалить строку из ps, т.е. до первого переноса строки и
начать цикл заново

G # добавить строку из hs в ps, добавив перенос строки перед
H # аналогично добавить строку из ps в hs

N # считать строку и добавить в ps
```

sed.

```
sed -n -e '1! G; h; $p;'
# ?

sed '/./{H;$!d} ; x ; s/^Here we go/$Here we end/'
# ?
```

sed. Флаги

`-n` — не печатать содержимое ps после каждой итерации

`-i` (`--in-place`) — применить преобразования к файлам

grep.

grep — ищет вхождения регулярного выражения и выводит строки с такими вхождениями (**g**/**re**/**p** — **g**lobally search for **re**gular expression and **print** matching lines).

```
grep 'regexp'


grep -e 'regexp_1' -e 'regexp_2'
```

grep. Флаги

```
-o # only-matching, т.е. печатать только те части строк,
которые совпали

-l # печатать названия файла, в котором строка была найдена

-i # ignore-case, нечувствительный к регистру поиск

-v # invert, т.е. только те строки, которые не совпали

-A n
-B n
-C n # печатать n строк после (after-context), перед (before-
context) и вокруг (context) совпадений
```

grep.

```
find /home/gigi -name '*.c' ! -type d  | xargs grep 'hello'

find /home/gigi -name '*.c' ! -type d  -exec grep 'hello' {} +


grep *.txt
# или
grep '*.txt'
```

cut

В Unix часто используются табличные данные в текстовом виде.

cut хорошо работает, если между полями заранее известно количество разделителей.

Например, если гарантируется, что разделитель — один пробел:

```
echo '1 2 3 4' | cut -d ' ' -f 3
# 3
```

cut

В случае, когда разделителей несколько, **cut** становится бесполезен:

```
echo  '   1  2    3 4' | cut -d ' ' -f 3
# ?
```

awk

awk — утилита для выбора определенных записей из файла и выполнения операция над ними.

Разбивает текст в поданных ему строках на поля по заданному разделителю (по умолчанию разделитель — любое количество пробельных символов).

```
    1   hi  7  2 3
```

Первое поле содержит `1`, второе — `hi`, третье — `7` и так далее.

awk

```
# выведем третье поле
$ echo '    1   hi  7  2 3' | awk '{ print $3 }'
7

# выведем 3 и 2 поле
# обязательна ли тут запятая?
$ echo '    1   hi  7  2 3' | awk '{ print $3, $2 }'
7 hi
```

awk. Команды

В **awk** можно задать несколько блоков команд, и у каждого блока можно указать условие, при котором блок исполнится:

```
$ cat /etc/passwd | awk 'BEGIN { FS=":"; OFS=";" }; /x/ {print
$1, $2}; /y/ { print $0 }'
```

awk. BEGIN END

Команды в блоке `BEGIN` будут выполняться не для каждой строки, а один раз при запуске.

В `END` — аналогично, но после считывания всего ввода.

В `BEGIN` имеет смысл установить переменные:

- `OFS` — какой строкой разделять поля при выводе,
- `FS` — по какой строке разбивать поля при чтении,
- `ORS` и `RS` — разделяются записи при выводе и чтении соответственно; по умолчанию запись — это одна строка.

awk

В **awk** есть:

- циклы,
- условные переходы,
- синусы и косинусы,
- объявление функций,
- сохранение в переменные строк, массивов, словарей и пр.

Подумайте хорошо, а нужно ли оно вам?

Дополнительный материал

Лучший способ понять **awk**, **sed**, **grep**

```
info sed
info grep
info gawk
```

Также самостоятельно изучите: `split`, `join`, `paste`, `tr`.