Семинар #7: Скрипты Bash

Запуск скриптов Bash

`chmod +x script.sh`	дать права скрипту на выполнение
`./script.sh`	запуск скрипта

Манипуляции со строками

`$var`	значение переменной (строки) `var`
`${var}`	значение переменной (строки) `var`
`${#var}`	длина строки `var`
`${var:N}`	подстрока, начиная с символа с индексом `N` и до конца строки
`${var:N:K}`	подстрока, начиная с символа с индексом `N` длинной `K` символов
`${var:-default}`	если `var` не установлена или пуста, то вернёт значение `"default"`; иначе вернёт `$var`
`${var:=default}`	если `var` не установлена или пуста, то присвоит `var` значение `"default"`; вернёт `$var`
`${var:+default}`	если `var` установлена и не пуста, то вернёт значение `"default"`; иначе вернёт пустую строку
`${var:?error}`	если `var` не установлена или пуста, то будет ошибка с сообщением `"error"`; иначе `$var`
`${var#pattern}`	строка с удалённым префиксом, соответствующим wildcard-паттерну `pattern` берётся наименьшее вхождение
`${var##pattern}`	то же самое, но берётся наибольшее вхождение
`${var%pattern}`	строка с удалённым суффиксом, соответствующим wildcard-паттерну `pattern` берётся наименьшее вхождение
`${var%%pattern}`	то же самое, но берётся наибольшее вхождение
`${var/pattern/str}`	заменить первое вхождение паттерна `pattern` на строку `str`
`${var//pattern/str}`	заменить все вхождения паттерна `pattern` на строку `str`
`${var^}`	сделать первый символ заглавным
`${var^^}`	сделать все символы заглавными
`${var,}`	сделать первый символ строчным
`${var,,}`	сделать все символы строчными

wildcard-паттерны – это те же паттерны, которые используются в командах linux (`*`, `?`, `[...]`).

Примеры использования манипуляции со строками

Если есть строки:

```
$ a="appleXbananaXcherryXdurian"
$ b=""
```

то

```
echo ${a}                 # напечатает appleXbananaXcherryXdurian
echo ${#a}                # напечатает 26
echo ${a:5:3}             # напечатает Xba

echo ${a:-default}        # напечатает appleXbananaXcherryXdurian
echo ${b:-default}        # напечатает default

echo ${a#*X}              # напечатает bananaXcherryXdurian
echo ${a##*X}             # напечатает durian
                          # паттерн *X это любая подстрока, заканчивающаяся на X

echo ${a%X*}              # напечатает appleXbananaXcherry
echo ${a%%X*}             # напечатает apple
                          # паттерн X* это любая подстрока, начинающаяся на X

echo ${a/cherry/coconut}  # напечатает appleXbananaXcoconutXdurian
echo ${a//X/AAA}          # напечатает appleAAAbananaAAAcherryAAAdurian
echo ${a//??X/ABC}        # напечатает appABCbanaABCcherABCdurian

echo ${a^^}               # напечатает APPLEXBANANAXCHERRYXDURIAN
echo ${a,,}               # напечатает applexbananaxcherryxdurian
```

Программа awk

Скриптовый язык awk служит для обработки текстовых файлов. Он обрабатывает текстовый файл построчно. Общий вид программы на awk имеет вид:

```
'BEGIN { код } условие { код } ... условие { код } END { код }'
```

где

- Программа делится на блоки: блок BEGIN, средние блоки, и блок END.
- Код в блоке BEGIN выполняется перед обработкой строк файла.
- Условие вычисляется перед обработкой каждой строки.
- Код в средних блоках выполняется при обработке каждой строки, если соответствующее условие верно.
- Код в блоке END выполняется после обработки строк файла.
- Любой из блоков программы можно опустить.
- Условие тоже можно опустить, в этом случае оно считается истинным.

Текст программы для awk можно передавать через аргументы или через файл. Текст для обработки передаётся через файл, указанный в аргументах или через стандартный вход. Итоговый результат программа печатает в стандартный выход. Для примера, пусть у нас будет файл animals.txt, содержащий строки:

```
Alpaca loves apples.
Bison can run up to 35 miles per hour.
Camel is a very unique mammal.
Dog barks very loudly.
```

Такой запуск awk с блоком BEGIN, одним средним блоком без условия и блоком END:

```
$ awk 'BEGIN {print "Hello"} {print "Processing line"} END {print "Goodbye"}' animals.txt
```

напечатает:

```
Hello
Processing line
Processing line
Processing line
Processing line
Goodbye
```

Любой из блоков программы, можно опускать, например такая команда без блоков BEGIN и END:

```
$ awk '{print "Processing line"}' animals.txt
```

напечатает:

```
Processing line
Processing line
Processing line
Processing line
```

Выбор слов (полей) в строке

Для обозначения всей строки или одного слова строки используются специальные выражения. Под словом понимается любая последовательность непробельных символов, ограниченная пробельными символами и/или началом/концом строки. Слова в контексте awk также называются полями (*fields*). Разделитель для полей в строке можно настроить с помощью переменной FS.

$0	вся строка
$1	первое слово строки
$5	пятое слово строки
$NF	последнее слово строки
$(NF-1)	предпоследнее слово строки

Например, такая команда:

```
$ awk '{print $2,$(NF-1)}' animals.txt
```

Напечатает второе и предпоследнее слово в каждой строке:

```
loves loves
can per
is unique
barks very
```

Функции `print` языка `awk` можно передать аргументы через запятую. В этом случае она будет печатать их через разделитель, заданный в переменной `OFS` (по умолчанию `OFS` равен пробелу). Если же передавать аргументы без запятой, то они будут склеиваться:

```
$ awk '{print $2 $(NF-1)}' animals.txt
```

Напечатает второе и предпоследнее слово в каждой строке без разделителей:

```
lovesloves
canper
isunique
barksvery
```

Основные переменные

`NR`	номер текущей обрабатываемой строки.
`NF`	количество полей (слов) в текущей обрабатываемой строке.
`FS`	разделитель, используемый при считывании полей; определяет, что является полем.
`OFS`	разделитель, используемый при печати.

Например, следующая команда:

```
$ awk '{print NR,NF}' animals.txt
```

Напечатает значение переменных `NR` и `NF` для каждой строки файла `animals.txt`:

```
1 3
2 9
3 6
4 4
```

Часто разделителем полем в файле нужно считать не пробельные, а другие символы. Например в известном файле `/etc/passwd` разделителем выступает символ `":"`.

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync

...
```

Для того, чтобы разделить этот файл на поля, нужно задать переменную `FS` значением `":"`.

```
$ awk 'BEGIN {FS=":"} {print $1,$3}' /etc/passwd
```

Данная команда напечатает только имя пользователя и его `UID`:

```
$ cat /etc/passwd
root 0
daemon 1
bin 2
sys 3
sync 4
...
```

Условия

Перед каждым средним блоком кода можно указать некоторое условие или некоторый паттерн, по которому будет определяться будет ли обрабатываться текущая строка. Например, следующая команда:

```
$ awk 'NR == 2 || NR == 4 {print $0}' animals.txt
```

Напечатает только строку номер 2 и строку номер 4 из файла `animals.txt`:

```
Bison can run up to 35 miles per hour.
Dog barks very loudly.
```

Можно также использовать паттерны регулярных выражений используя синтаксис `str ~ /regex/`
Например, следующая команда напечатает только строки, которые содержат подстроку `very`:

```
$ awk '$0 ~ /very/ {print $0}' animals.txt
Camel is a very unique mammal.
Dog barks very loudly.
```

А следующая команда напечатает только строки, содержащие подстроку из двух цифр:

```
$ awk '$0 ~ /[[:digit:]]{2}/ {print $0}' animals.txt
Bison can run up to 35 miles per hour.
```

`awk` использует регулярные выражения, совместимые POSIX, поэтому выражения `\s`, `\w` и т. д. тут не поддерживаются. Вместо этого нужно использовать `[[:space:]]`, `[[:alpha:]]` и т. д.

Примеры команд

- Напечатать все поля в строка, начиная со второго поля:

  ```
  $ awk '{for (i = 2; i < NF; i++) {printf $i " "} printf "\n"}'
  ```

- Напечатать все слова в файле:

  ```
  $ awk 'BEGIN {myvar=0} {myvar += NF} END {print myvar}' file.txt
  ```