Bash

> Командная оболочка, написанная на C; скриптовый, интерпретируемый язык

Запуск файла

Sha-Bang строка:
`#!/bin/bash` путь к используемому интерпретатору.
`chmod +x name_file.sh` - Дать файлу скрипта права на запуск
`chmod +(добавить права) -(забрать права)`

- `r` - read - права на чтение
- `w` - write - права на запись
- `x` - execute - права на запуск\
  `./name_file.sh` | `bash name_file.sh` - запуск bash-скрипта

Оформление кода

```
#комментарий
- #TODO - сделать что-то
- #NOTE - заметка
```

```
Отступы для красоты
```

Если есть дублирование, то надо вынести в отдельную константу

Переменные

Переименованный адрес к памяти
`a=10` - объявление
`$a` - обращение к переменной

Математические операции

- в круглых скобках
- результат с помощью $
  `a=$(($a+1))`

Вывод

`echo текст`
`echo "текст с переменной $var"`
`echo TEXT > file` - перенаправленный поток вывода

Ввод

`read var`
`read -p "TEXT" var`

Выполнение команд

`result=$(cat file.txt)` - выполнение команды и запись результата в переменную
`result=`cat file.txt`` - второй вариант, косые ковычки!

Ветвление

```
if [[ условие ]]
then
    # код, если True
else
    # код, если False
fi
```

Логические операции

Вместо <,>,== используются:

- `-gt` (greater than - больше, чем)
- `-ge` (greater equal - больше или равно)
- `-lt` (lower than - меньше, чем)
- `-le` (lower equal - меньше или равно)
- `-eq` (equal - равно)
- `-ne` (not equal - не равно)
- `-a` (and)
- `-o` (or)

Циклы

Цикл *for*

> Первый вариант

```
for student in Ivan Petr Vasya
do
    echo "$student"
done
```

> Второй вариант

```
for student in $(cat students) #В файле обязательно пустую строчку в конце
do
    echo "$student"
done
```

> Третий вариант (как в C)

```
for ((i=0; i<5; i++))
do
    echo $i
done
```

Цикл *while*

> Первый вариант

```
i=0
while [[ $i -lt 5 ]]
do
    echo $i
    i=$(($i+1))
done
```

> Второй вариант

```
while read student #В файле обязательно пустую строчку в конце
do
   echo $student
done < students
```

Циклы *switch case*

> Осуществляет сопоставление текущего значения переменной заданным вариантам, выполняя соответсвующий им код

```
case var in
    value_1) #код ;;
    value_2) #код ;;
    value_3) #код ;;
    *) #код, если предыдущие варианты не подошли ;;
esac
```

Приём аргументов (параметров)

- `$0` - имя скрипта
- `$1` - первый аргумент
- `$2` - второй аргумент - и так далее, вплоть до переменной `$9`
  Параметры командной строки разделяются пробелами
  Если нужно обратиться к параметру, индекс которого состоит больше, чем из одной цифры, надо ииспользовать фигурные скобки: `${10}`, `${11}` и т.д.

Особые конструкции работы с параметрами

- `$#` - Количество параметров
- `$*` - Все параметры (вместе)
- `$@` - Все параметры (по отдельности)
- `${!#}` - Последний параметр

Ключевое слово *shift*

Сдвигает значение позиционных параметров влево.

> Например, значение переменной `$3` становится значение переменной `$2`, значение `$2` переходит `$1`, а то, что было до этого в `$1` теряется.
> Пример, на нахождение параметра

```
while [[ -n "$1" ]]
do
    case "$1" in
        -a) echo "Найден параметр -a" ;;
        -b) echo "Найден параметр -b" ;;
        -c) echo "Найден параметр -c" ;;
        *) echo "$1 не параметр" ;;
    esac
    shift
done
```

Потоки вывода и ввода

- 0, STDIN - стандартный поток ввода
- 1, STDOUT - стандартный поток вывода
- 2, STDERR - стандартный поток ошибок

Поток можно перенаправить исполюзуя его номер, к примеру, `2>ls_errors`
Временной перенапрвление:

> - Присоединяя к содержимому файла командой `>>`
> `echo "This is error" >> list_of_errors`
> - Перезаписывая файл комадой `>`
> `echo "This is error" > list_of_errors`

Постоянное перенаправление командой `exec`

> К примеру

```
exec 1>outfile
echo "Test output"
echo "ERROR" >&2
```

`-f` - Проверить существование файла
`exit` - Досрочно выходит из скрипта
Вместо `echo -n > file_name` лучше в конце цикла `...done > file_name`
Аргументы надо сравнивать как строки
`break` - завершить цикл

Сигнал Linux

Сигналы закрытия и остановки

- 1 SIGHUP - Закрытие терминала
- 2 SIGINT - Сигнал остановки процесса пользователя с терминала (CTRL + C)
- 3 SIGQUIT - Сигнал остановки процесса пользователя с терминала (CTRL +)
- 9 SIGKILL - Безусловное завершение процесса
- 15 SIGTERM - Сигнал запроса завершения процесса
- 17 SIGSTOP - Принудительное приостановка выполнения процесса, но не завершение его работы
- 18 SIGTSTP - Приостановка процесса с терминала (CTRL + Z), но не завершение работы

Перехват сигналов

`trap` - позволяет скрипту реагировать на сигналы

```
trap "echo ' Trapped Ctrl+C'" SIGINT

for ((i=0; i<=10; i++))
do
    echo "Итерация №$i"
    sleep 1
done
```

`kill -[signal or code] PID(Proccess ID)`

```
kill -9 pid # Убьёт процесс в любом случае
kill -SIGKILL pid # Убьёт процесс в любом случае
kill -SIGTERM pid
```

`trap -- [SIGNAL]` - убирает ловушку

Запуск в фоновом режме

Используется `&`

`bash main.sh &`

nohup

`nohup bash main.sh &` - отвязывает процесс от терминала. Процесс потеряет ссылки на STDOUT и STDERR

По умолчанию в файл *nohup.out*

Функции

```
function fun_name {
    # тело цикла
}
```

Аргументы передаются через пробел, в функции аргументы `$1`, `$2`, `$3`

`return` - возвращает код завершения функции

Чтобы вернуть значенние, нужно через `echo` внутри функции

Можно занести результат в переменную, оформив как команду `result=$(fun_name $arg)`

Глобальные и локальные переменные

Если имя переменной совпадают, то это глобальные перменные
Для локальной переменной используется `local`

Импорт функции из другого скрипта

`. ./script.sh` - импорт кода (всего, не только функции) в текущий скрипт
Нельзя импортировать только часть кода

Регулярное выражение

`([0-9]+[A-Za-z]*)_?){1,3}` - Регулярное выражение

> `+` - 1 и более символов (в примере хотя бы одно число)
> `*` - 0 и более символов (в примере латинские буквы)
> `{1,3}` - повторение от 1 до 3 раз
> **Регулярное выражение** - это строка, задающая шаблон поиска подстрок в тексте

Шаблона для одного символа

- `.` - Один любой символ, кроме новой строки `\n`
- `\d` - Любая цифра
- `\D` - Любой символ, кроме цифры
- `\s` - Любой пробельный символ(пробел, табуляция, конец строки и т.п.)
- `\S` - Любой непробельный символ
- `\w` - Любая буква, а также цифры и нижнее подчёркивание
- `\W` - Любая не буква, не цифра и не подчёркивание
- `[..]` - Один из символов в скобках, а также любой символ диапозона a-b
- `[^..]` - Любой символ, кроме перечисленных
- `\b` - Начало или конец слова (слева пусто или не-буква, справа буква или наоборот
- `\B` - Не граница слова: либо и слева и справа буквы, либо слева и справа не-буквы

Квантификаторы (количество повторений)

- `{n}` - Ровно `n` раз
- `{m, n}` - От `m` до `n` включительно
- `{m,}` - Не менне `m` повторений
- `{,n}` - Не более `n` повторений
- `?` - Ноль или одно (аналогично `{0,1}`)
- `*` - Ноль и более (аналогично `{0,}`)
- `+` - Один и более (аналогично `{1,}`)

`echo $text | awk '/re/{print $1}'` - проверка текста через регулярку