Unix-скрипты

ВАЖНО

Любой bash-скрипт должен начинаться со строки:
#!/bin/bash
(Последовательность #! называется Sha-Bang – указание интерпретатора)

Сделать исполняемым:
chmod 555 scriptname
Или
chmod +x scriptname
а затем просто запустить:
./scriptname

ПЕРЕМЕННЫЕ

Могут быть строками, целыми числами и массивами.

Можно объявлять константы.

- export foo=93 – экспорт во все дочерние оболочки.
- foo=93 – установка для текущей оболочки

вокруг знака = пробелов быть не должно!

- printenv – вывести глобальные переменные.

Пример: HOME, MAIL, PATH, LC_ALL, PWD, UID.

- set – вывести все переменные и функции.

- a=1
- b=$a
- echo $b # 1
- hello="A B C"
- echo hello # hello
- echo $hello # A B C
- echo "$hello" # A B C
- echo '$hello' # $hello
- var=23
- unset var

МАССИВЫ

- arr[0]=0
- arr[5]=45
- arr2=(zero one two three)
- arr3=([17]=семнадцать [21]=двадцатьОдин)
- echo ${arr[5]} # нужны скобки!

ОСОБЫЕ ПЕРЕМЕННЫЕ

- $@ – параметры скрипта (все!)
- $0 – имя скрипта
- $1, $2, $3, . . . – параметры скрипта по одному
- $# – количество переданных аргументов
- $*, $@ – специальная переменная, содержащая все
- аргументы

ПРИСВАИВАНИЕ

- a=1
- let a=16+5 # let – арифметические действия

- for a in 7 8 9 11
  do
  . . .
  done
- read a
- a=`echo Hello!`
- echo $a # Hello!
- a=`ls -l`
- a=$(ls) # то же самое

EXPR

Вычисляет заданное выражение

Аргументы должны отделяться пробелами!!

Примеры:

- expr 3 + 5 #8
- expr 5 * 3 #15 экранируем *
- y=$(expr $y + 1) #инкремент

ЦИКЛ

```
for arg in list
do
        command(s)
done
```

- for arg in $(seq n) # 1 2 3 ... n
- for arg in {1..n} # 1 2 3 ... n
- for arg in "$@“
- for i in "${arrayName[@]}"

TEST

- Команда test (или "[ ]")проверяет выполнение некоторого условия. С ее помощью формируются операторы выбора и цикла.

- Минус команды test:

[privet]
echo $? # 0
[ ]
echo $? # 1

- Test возвращает 0 (истина), если в скобках стоит непустое слово. **Причина: 0 – код возврата, если все хорошо, т.е. истина**

IF

```
if condition
then
        command1
else
        command2
fi
```

if test condition точно тоже, что и if [condition]
if [[condition]] - возможны &&, | |
if ((арифметическое выражение))

УСЛОВНЫЕ ВЫРАЖЕНИЯ

- [[$foo > 7]] – сравнивает строки
- [$foo > 7] – перенаправление вывода! :)

Правильно

- (($foo > 7)) - сравнивает числа
- [$foo -gt 7]
- [[$foo -gt 7]]

УСЛОВНЫЕ ВЫРАЖЕНИЯ

- [bar == "$foo"] - неверно

Правильно

- [bar = "$foo"]
- [[bar == "$foo"]]

УСЛОВНЫЕ ВЫРАЖЕНИЯ

- if EXPR; then команды; fi
- if EXPR; then команды; else другое; fi
- [! EXPR] – отрицание
- [(EXPR)] – скобки
- [EXPR1 – a EXPR2] – И
- [[EXPR1 && EXPR2]] – И
- [EXPR1 – o EXPR2] – ИЛИ
- [[EXPR1 | | EXPR2]] – ИЛИ
- ((арифим. выражение))
- man test

УСЛОВНЫЕ ВЫРАЖЕНИЯ

- [−e FILE] - файл существует
- [−d FILE] - это директория
- [−f FILE] - это обычный файл
- [−s FILE] - размер ненулевой
- [−r FILE] - доступен для чтения
- [−w FILE] - доступен для записи
- [−x FILE] - исполняемый

```
if ! grep -q $USER /etc/passwd; then
        echo "not a local account"
fi

grep -q $USER /etc/passwd if [ $? – ne 0 ]; then
        echo "not a local account"
fi

if [ "$(whoami)"! = root ]; then echo "Oh"; fi
```

CASE

```
echo "1 Запуск программы nano"
echo "2 Запуск программы vi"
echo "3 Выход"
read doing
case $doing in
1) /usr/bin/nano ;;
2) /usr/bin/vi ;;
3) exit 0 ;;
*) echo "Введено неправильное действие"
esac
```

ФУНКЦИИ

- function function_name { command... }
- function_name () { command... }

НЕ СТОИТ СМЕШИВАТЬ ОПРЕДЕЛЕНИЯ!

Функции могут принимать входные аргументы и возвращать код завершения.

- return – завершает исполнение функции.
- Может возвращать "код завершения"(int), который записывается в переменную $?.
- Если "код завершения« не указан – возвращается код последней команды в функции

ФУНКЦИИ

Нет опережающего объявления функции, но...

```
foo1 (){
echo Вызов функции "foo2"из "foo1"
foo2
}
foo2 (){
echo "Функция foo2"
}
foo1
```

```
if [ $USER = student ]
then
student_greet () {
echo "Привет, student!"
}
fi
student_greet

# Работает только у пользователя student, другие получат сообщение об ошибке.
```

BC

bc – утилита, выполняющая вычисления с произвольной точностью.

- echo "scale=4;(321-123)/123" | bc -l

scale=4 – количество знаков после запятой

- echo "obase=16;ibase=10;123" | bc

преобразование из десятичного в шестнадцатеричный вид

- var3=$(bc -l << EOF

scale = 9; s (1.7)

EOF)

СТРОКИ

- Длина строки ${#string}
- ·Длина подстроки в строке

expr "$string": '$substring'

stringZ=abcABC123ABCabc

echo `expr "$stringZ": 'abc[A-Z]*.2'`# 8

- expr index $string $substring - номер позиции совпадения в $string с символом в $substring.
- Извлечение подстроки ${string:position} либо

${string:position:length}

СТРОКИ

- expr "$string": '\($substring\)'

Находит и извлекает первое совпадение $substring в $string, где $substring – это регулярное выражение.

- Удаление части строки

${string#substring} - удаление самой короткой

stringZ=abcABC123ABCabc

echo ${stringZ#a*C} # 123ABCabc

- Замена подстроки -

${string/substring/replacement} – первое

${string//substring/replacement} – все substring

СТРОКИ

- expr "$string": '\($substring\)'

Находит и извлекает первое совпадение $substring в $string, где $substring – это регулярное выражение.

- Удаление части строки

${string#substring} - удаление самой короткой

stringZ=abcABC123ABCabc

echo ${stringZ#a*C} # 123ABCabc

- Замена подстроки -

${string/substring/replacement} – первое

${string//substring/replacement} – все substring

СТРОКИ

- Смотреть тут:

http://gordin.us/sergo/abs-guide/x4165.html