Кодирование переменных

Александр Ольферук,
наставник

Яндекс Практикум

Цели на консультацию

Первая часть – 40 минут

- Основы языка Bash

Перерыв – 10 минут

Вторая часть – 40 минут

- Основные команды Linux

Немного о языке

Bash проще всего воспринимать как смесь двух языков: Bash’а и C: будут конструкции как явно на Bash, так и заимствованные из C.

Немного о языке

Bash проще всего воспринимать как смесь двух языков: Bash’а и C: будут конструкции как явно на Bash, так и заимствованные из C.

Bash полезен, когда вы работаете на удаленном сервере, который БЕЗ МОНИТОРА И ВООБЩЕ.

Sha-bang

Скрипт на bash состоит из двух частей: **заголовка** и **тела**.

В **заголовке** мы даем компьюхтеру понять, чем этот скрипт открывать и, возможно, кодировку. А сочетание символов **#!** и называется sha-bang.
В теле скрипта – весь наш код.

```
#!/bin/sh          # так мы больше охватываем, но…
#!/bin/bash        # …лишаемся некоторых фишек bash'а
```

Основы языка

Создание переменных

```
a=123                    # a - int со значением 123
```

Создание переменных

```
a=123                    # a - int со значением 123

b=                       # b = null
```

Создание переменных

```
a=123                    # a - int со значением 123

b=                       # b = null

c="1 2"                  # c = массив из 1 и 2, кавычки
                         обязательны!
```

Создание переменных

```
a=123                   # a - int со значением 123
b=                      # b = null
c="1 2"                 # c = массив из 1 и 2, кавычки
                        обязательны!
d="Hello world"         # d – строка
```

Создание переменных

```
a=123                   # a - int со значением 123
b=                      # b = null
c="1 2"                 # c = массив из 1 и 2, кавычки
                        обязательны!
d="Hello world"         # d – строка
e=Hello\ world          # e – если пробел экранируется, то
                        кавычки не нужны
```

Создание переменных

```
a=123
b=
c="1 2"

d="Hello world"
e=Hello\ world
```

Обратите внимание, что пробел вокруг **=** нет ни в одном из примеров, — это приведет к ошибке!

Вывод переменных

```
echo e                  # e
```

Вывод переменных

```
echo e                  # e

echo $e                 # Hello World
```

Вывод переменных

```
echo e                  # e
echo $e                 # Hello World
echo “$e”               # Hello World
```

Вывод переменных

```
echo e                  # e
echo $e                 # Hello World
echo “$e”               # Hello World
e=“A    Z”
```

Вывод переменных

```
echo e                  # e
echo $e                 # Hello World
echo “$e”               # Hello World
e=“A    Z”
echo $e                 # A Z
echo “$e”               # A    Z
```

на самом деле, кавычки подавляют еще и переносы строк

Вывод переменных

```
echo e                  # e
echo $e                 # Hello World
echo “$e”               # Hello World
e=“A    Z”
echo $e                 # A Z
echo “$e”               # A    Z
echo '$e'               # $e
```

на самом деле, кавычки подавляют еще и переносы строк

let

```
f=2+2                    # 2+2
```

let

```
f=2+2                   # 2+2

let f=2+2               # 4
```

let

```
f=2+2                   # 2+2

let f=2+2               # 4

let f=f+2               # 6

let f=$f+2              # 8
```

readonly и declare

```
readonly a=13        # константа

declare -r a=13      # равноценная запись
```

let и declare

```
f=1                    # 1
```

let и declare

```
f=1                    # 1


f=$f+2                 # 1+2
let f=f+2              # 3
```

let и declare

```
f=1                     # 1


f=$f+2                  # f+2
let f=f+2               # 3


declare -i f            # явно объявили, что f - int
f=f+2                   # 3     , чем сняли необходимость в let
f=0.12312               # значение f не изменится, выведется warning
```

declare и readonly

```
readonly a=13          # константа

declare -r b=13        # тоже константа

b=12                   # значение b не изменится, выведется warning
```

Арифметика

```
a=1 b=2                    # можно, к слову, объявлять сразу несколько
```

Арифметика

```
a=1 b=2                 # можно, к слову, объявлять сразу несколько
let c=$a+$b             # 3
```

Арифметика

```
a=1 b=2                 # можно, к слову, объявлять сразу несколько
let c=$a+$b             # 3
let c=$a + $b           # 1
```

Арифметика

```
a=1 b=2                 # можно, к слову, объявлять сразу несколько
let c=$a+$b             # 3
let c=$a + $b           # 1
let c=$a-$b             # -1
```

Арифметика

```
a=1 b=2                 # можно, к слову, объявлять сразу несколько
let c=$a+$b             # 3
let c=$a + $b           # 1
let c=$a-$b             # -1
let c=$a*$b             # 2
```

Арифметика

```
a=1 b=2                 # можно, к слову, объявлять сразу несколько
let c=$a+$b             # 3
let c=$a + $b           # 1
let c=$a-$b             # -1
let c=$a*$b             # 2
let c=$a/$b             # 0, так как по умолчанию деление целочисленное
```

Считывание с клавиатуры

```
read a                                  # считать переменную a
```

Считывание с клавиатуры

```
read a                                  # считать переменную a

read -p “Enter your age: “ a            # считать переменную a, но
                                        при этом выведется
                                        приглашение ко вводу
```

Считывание с клавиатуры

```
read a                              # считать переменную a

read -p “Enter your age: “ a        # считать переменную a, но
                                    при этом выведется
                                    приглашение ко вводу

read -s password                    # ввод будет скрыт
```

Сочетание флажков

Эти записи равноценны:

```
read -p -s “Enter your password: “ pwd

read -ps “Enter your password: “ pwd
```

Parameter expansion

```
read -p “Enter your age: “ age      # вводим возраст
read -p “Enter your name: “ name    # и имя
echo “${name}’s age is: $age“       # Выводим так:
                                    Alex’s age…
```

Здесь **${…}** – тот самый parameter expansion. Запомните его, мы к нему еще вернемся.

Вызов “старших братьев”

```
c=$((b**b + a))          # в $((…)) блок на C
d=$(ls)                  # в $(…) блок на Bash
```

Конструкция $(...)

Команда `c=$(ls)`, как и `c=`ls`` выполнят одно и то же: поместят в переменную с список файлов в текущей директории.

Это очень важный концепт! Затем-то мы и открываем Bash, чтобы упростить себе жизнь и автоматизировать командами выполнение других, более сложных команд 😊

Условный оператор

```
if [ … ] then          # … - условие
  …                    # операторы
else                   # или elif [ … ] then
  …                    # операторы
fi                     # fi = if наоборот
```

Условия

```
-eq             # равно
-ne             # не равно
-gt             # строго больше
-lt             # строго меньше
-ge             # больше или равно
-le             # меньше или равно
```

Условный оператор на C

```
if ((…)); then          # … - условие
  …                     # операторы
else                    # или elif (( … )); then
  …                     # операторы
fi
```

Условия на C

```
==          # равно
!=          # не равно
>           # строго больше
<           # строго меньше
>=          # больше или равно
<=          # меньше или равно
```

Предикаты

```
&&                  # … И …
||                  # … ИЛИ …
!                   # НЕ …
```

Невеселые задачки №1 *(Bash)*

Ваших знаний на текущий момент должно быть достаточно, чтобы сделать:

- программу, которая по введенному возрасту определяет и выводит, можно ли курить, пить и голосовать
- программу-калькулятор сложных процентов: `x*(1+a)^n`, где `x` – первый взнос, `a` – процентная ставка, `n` – количество лет

Полезные условия

```
-d my_folder       # существует ли папка my_folder
-e my_file.txt     # существует ли файл my_file.txt
-r my_file.txt     # можно ли из файла читать
-w my_file.txt     # можно ли в файл писать
-x my_file.txt     # можно ли файл запустить (как программу)
```

case

```
case $age in                    # выбираем по значению переменной из вариантов:

esac                            # esac = case наоборот
```

case

```
case $age in                    # выбираем по значению переменной из вариантов:
[0-4])                          # от 0 до 4


5)
6-9|1[0-8])                     # паттерн матчинг


*)                              # для всех остальных вариантов


esac                            # esac = case наоборот
```

case

```
case $age in                                # выбираем по значению переменной из вариантов:
[0-4])                                      # от 0 до 4
  echo “too young for school”               # пишем
  ;;                                        # закрываем этот вариант
5)
6-9|1[0-8])                                 # паттерн матчинг


*)                                          # для всех остальных вариантов


esac                                        # esac = case наоборот
```

case

```
case $age in                                  # выбираем по значению переменной из вариантов:
[0-4])                                        # от 0 до 4
  echo “too young for school”                 # пишем
  ;;                                          # закрываем этот вариант
5) echo “go to Kindergarten” ;;               # не обязательно, но желательно разносить по разным строкам
6-9|1[0-8])                                   # паттерн матчинг


*)                                            # для всех остальных вариантов


esac                                          # esac = case наоборот
```

case

```
case $age in                                  # выбираем по значению переменной из вариантов:
[0-4])                                        # от 0 до 4
  echo “too young for school”                 # пишем
  ;;                                          # закрываем этот вариант
5) echo “go to Kindergarten” ;;               # не обязательно, но желательно разносить по разным строкам
6-9|1[0-8])                                   # паттерн матчинг
  let grade=$age-5
  echo “go to the $grade grade”
  ;;
*)                                            # для всех остальных вариантов
  echo "you are too old for school"
  ;;
esac                                          # esac = case наоборот
```

Тернарный оператор

Python:

```
age = 15

can_vote = 1 if age ⩾ 18 else 0
```

Тернарный оператор

Python:

```
age = 15

can_vote = 1 if age ⩾ 18 else 0
```

Bash:

```
age=15

((age ⩾ 18 ? (can_vote=1) : (can_vote=0)))

can_vote=$((age ⩾ 18 ? 1 : 0))
```

Цикл while

```
while […]; do                    # условие как в if со всеми ne, и gt
  …
done                             # фуу, лишнее ключевое слово!
```

Цикл while

```
while […]; do                    # условие как в if со всеми ne, и gt
  …
done                             # фуу, лишнее ключевое слово!
```

Точно так же, как и в Python, здесь есть ключевые слова **continue** и **break**, работающие точно так же.

Цикл for

```
for a in 7 8 9 11          # как в python :)
do                         # фуу, лишнее ключевое слово!
  echo -n "$a "            # -n не дает переносить на новую строку
done                       # фуу, лишнее ключевое слово!
```

Цикл for

```
for a in {A..Z}                 # перечисление букв от A до Z
do
  echo -n "$a "
done
```

Цикл for

```
for ((i=0; i<10; i++));      # как в C* :)
do
  echo $i
done
```

* — на самом деле, не совсем как в C: в C нужно еще объявить тип переменной-счетчика:

```
((int i=0; i<10; i++))
```

Пример применения цикла

```
read -p "Enter a filename: " fname
line_num=0
while read line; do
  let line_num+=1
  echo "$line_num $line"
done < $fname
```

Консоль:

```
Enter a filename: hello.txt
1 Первая строка файла
2 Вторая
3 Вот и все!
```

Строки

```
a=“hello world”          # объявили
echo ${#a}               # выведет 5 — длину строки
                         #
```

Главные фичи Bash’a

Перенаправление

```
ls > list_of_files.txt
```

```
# символ > означает перенаправление вывода в файл. Перетирает файл, если файл присутствует
```

```
ls >> list_of_files.txt
```

```
# символ >> означает перенаправление вывода в файл. Дописывает в файл, если файл присутствует
```

Перенаправление

```
ls > /dev/null          # заглушает вывод
```

Можно воспринимать `/dev/null` как черную дыру. Если записать, и сразу прочитать, то ничего не выведется.

Собираем команды в цепочки

```
cat long_file | less
```

```
# символ | означает
перенаправление вывода одной
команды в другую
```

Собираем команды в цепочки

```
cat long_file | less
```

```
# символ | означает
перенаправление вывода одной
команды в другую
```

```
cat a.txt
```

```
#       b
        c
        a
        a
```

```
cat a.txt | sort | uniq
```

```
#       a
        b
        c
```

Алиасы

```
alias gc=“git checkout”          # сделали новую команду gc
alias ga=“git add -A”            # сделали новую команду ga

unalias gc                       # надоела? забыли gc
unalias ga                       # надоела? забыли ga
```

Как стать крутым хакером?

1) Написать свой скрипт и сделать его исполняемым:

   ```
   chmod +x my_cool_script.sh
   ```

2) Положить его в домашнюю директорию **~**
3) Открыть **~/.bashrc**
4) Дописать в него:

   ```
   alias cool="~/my_cool_script.sh"
   ```

5) Вызвать команду **source ~/.bashrc** (или просто перезагрузить терминал)
6) Пользоваться командой **`cool`**!

Перерыв

Жду тебя через 5 минут

Основные команды

Навигация по директориям

```
cd …                    # перейти в папку …
cd                      # перейти в домашнюю директорию
cd ~                    # перейти в домашнюю директорию
pushd …                 # запомнить текущий путь и перейти в
                        папку …
popd                    # “вспомнить” путь и вернуться
```

Отображаем содержимое

```
ls …                    # отображает список файлов в папке …
ls -a …                 # в том числе и скрытые
ls -l …                 # в том числе и показать права
pwd                     # напечатать абсолютный путь к текущей
                        папке
```

Узнаем больше о мире вокруг

```
file …                      # узнать, что это за файл
locate …                    # найти файл с именем …
find … -name “<name>”       # найти файл с именем <name> в
                            папке …
sudo updatedb               # обновиться, чтобы поиск работал
which …                     # узнать путь к исполняемой команде
```

Отображаем содержимое

```
history                 # отобразить список последних введенных
                        команд
whatis …                # рассказать про команду …
man …                   # вывести подробную справку
                        по команде …
```

Оперируем папками и файлами

```
mkdir …                 # создать директорию
touch …                 # обновить время последнего изменения,
                        если файл существует, или создать его,
                        если нет
cp … …                  # копировать что-то куда-то
mv … …                  # переместить что-то куда-то
rm -rf …                # удалить по-хардкору
```

Выводим на экран содержимое

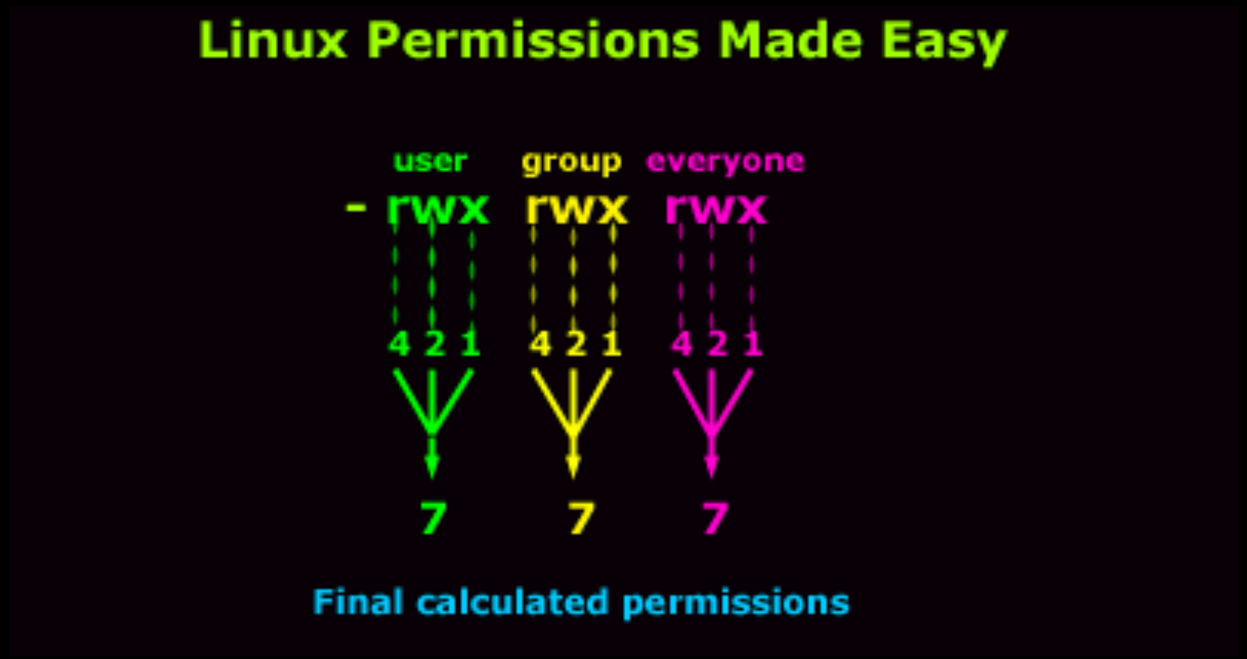
```
cat …                   # вывести содержимое файла
more …                  # вывести содержимое файла (с кое-
                        какими элементами управления)
less …                  # вывести содержимое (с продвинутыми
                        элементами управления)
```

Становимся супер-пользователями

```
sudo …                  # выполнить команду … от имени
                        суперпользователя
sudo -s                 # начать сессию суперпользователя
exit                    # выйти из режима суперпользователя
```

Разбираемся в правах

```
ls -l                    # вывести список файлов с правами
chmod 644 …              # добавить права, зная их коды файлу
chmod +x …               # добавить права на исполнение
                         конкретному файлу
```



Домашняя работа

По Shell:

- решать задачи здесь
- и здесь

Основы Bash, основные команды Linux

Александр Ольферук,
наставник

Яндекс Практикум