Командный интерпретатор bash — продолжение

2 марта 2017 г.

Чего не хватает до «полноценного» ЯП?

Чего не хватает до «полноценного» ЯП?

- Условия
- Циклы
- Функции
- Массивы

Условия if

- if-then

```
if [ $((2+2)) -eq 4 ]
then
   # условие выполняется
fi
```

Условия if

- if-then

```
if [ $((2+2)) -eq 4 ]
then
  # условие выполняется
fi
```

- if-then-else

```
if diff file1 file2; then
  echo "Equal"
else
  echo "Different"
fi
```

Условия if

- if-then-elif-else

```
if [ $CMD = "commit" ]; then
  do_commit
elif [ $CMD = "push" ]; then
  do_push
elif [ $CMD = "pull" ]; then
  do_pull
else
  echo "Unrecognized command: $CMD"
  exit 1
fi
```

[[...]]

- Зарезервированные «&&», «||», «(», «)», ...

```
if [ 1 -lt 2 -a \( "A" = "A" -o 2 -eq 3 \) ]
then
  echo "[ ] OK"
else
  echo "[ ] WTF"
fi
```

[[...]]

- Зарезервированные «&&», «||», «(», «)», ...

```
if [ 1 -lt 2 -a \( "A" = "A" -o 2 -eq 3 \) ]
then
  echo "[ ] OK"
else
  echo "[ ] WTF"
fi
```

- Введём ещё языковую конструкцию

```
if [[ 1 < 2 && ("A" == "A" || 2 == 3) ]]
then
  echo "[[ ]] OK"
else
  echo "[[ ]] WTF"
fi
```

Сравнения

Что из этого делает одно и то же?

1. `if [ $s == x* ];`
2. `if [ "$s" == "x*" ];`
3. `if [[ $s == x* ]];`
4. `if [[ "$s" == "x*" ]];`

Сравнения

Что из этого делает одно и то же?

1. `if [ $s == x* ];`
   - `x*` раскрывается в имена файлов
2. `if [ "$s" == "x*" ];`
   - Сравнение со строкой «x*»
3. `if [[ $s == x* ]];`
   - Сопоставление с образцом «x*»
4. `if [[ "$s" == "x*" ]];`
   - Сравнение со строкой «x*»

switch-технология

```
case "$CMD" in
  commit)
    do_commit
    ;;
  push)
    do_push
    ;;
  pull)
    do_pull
    ;;
  *)
    echo "Unrecognized command: $CMD"
esac
```

switch-технология

Очередной способ сопоставлять с образцом

```
case $CMD in
   [nN] | [Nn][Oo])
     echo "NOOOOOO"
     ;;
   [yY] | [Yy][Ee][Ss])
     echo "YEEES"
     ;;
   [0-9]*)
     echo "HAHA"
esac
```

Цикл while

- Кажется, хоть здесь нет никаких подстав

```
while [[ $(date +%H:%M:%S) != *0 ]]; do
  sleep 0.2
done
date +%H:%M:%S
```

Цикл while и until

- Кажется, хоть здесь нет никаких подстав

```
while [[ $(date +%H:%M:%S) != *0 ]]; do
  sleep 0.2
done
date +%H:%M:%S
```

- until — while наоборот

```
until [[ $(date +%H:%M:%S) == *0 ]]; do
  sleep 0.2
done
date +%H:%M:%S
```

Цикл for

```
for i in 1 2 3 4 5
do
  echo $i
done

for f in *; do
  if [ -d "$f" ]; then
    echo "$f is a directory"
  fi
done
```

Цикл for

```
cnt=0
for f in $(find . -name ".c")
do
  echo $f
  ((cnt++))
  if ((cnt >= 10)); then
    break
  fi
done
```

- continue тоже есть

Ещё цикл for

```
for ((i = 0; i < 10; i += 2)) do
  echo $i
done
```

Функции

```
function hello() {
  echo hello
}
```

Функции

```
function hello() {
  echo hello
}

hello2() {
  echo hello
  return 1
}
```

Функции

```
function hello() {
  echo hello
}

hello2() {
  echo hello
  return 1
}
```

Где аргументы функции?

Массивы

```
arr[0]=5
arr[1]=10
arr[2]=15
echo ${arr[1]} ${arr[2]}
```

Массивы

```
arr[0]=5
arr[1]=10
arr[2]=15
echo ${arr[1]} ${arr[2]}

arr2=(1 2 3 4 5)
echo ${arr2[2]} # -> 3
echo ${arr2[@]} # -> 1 2 3 4 5
echo ${arr2[*]} # -> 1 2 3 4 5
echo ${#arr2[*]} # -> 5 (длина)
```

Ассоциативные массивы

```
declare -A arr3
arr3["key1"]=value1
arr3["key2"]=value2
echo ${arr3[key1]} # value1
echo ${!arr3[@]} # key1 key2
echo ${!arr3[*]} # key1 key2
declare -A arr4
arr4=([key1]=value1 [key2]=value2)
echo ${!arr4[@]} # key1 key2
echo ${arr4[@]} # value1 value2
```

Ассоциативные массивы

```
declare -A arr3
arr3["key1"]=value1
arr3["key2"]=value2
echo ${arr3[key1]} # value1
echo ${!arr3[@]} # key1 key2
echo ${!arr3[*]} # key1 key2
declare -A arr4
arr4=([key1]=value1 [key2]=value2)
echo ${!arr4[@]} # key1 key2
echo ${arr4[@]} # value1 value2
```

Но в чём отличие `${!arr3[@]}` и `${!arr3[*]}`?

Ассоциативные массивы, «*» vs «@»

```
declare -A arr5
arr5=([a b c]=1 [def]=2)
```

1. ```
   for k in ${!arr5[@]};
   do echo -n "[$k] "; done
   ```

2. ```
   for k in "${!arr5[@]}";
   do echo -n "[$k] "; done
   ```

3. ```
   for k in ${!arr5[*]};
   do echo -n "[$k] "; done
   ```

4. ```
   for k in "${!arr5[*]}";
   do echo -n "[$k] "; done
   ```

Ассоциативные массивы, «*» vs «@»

```
declare -A arr5
arr5=([a b c]=1 [def]=2)
```

1. ```
   for k in ${!arr5[@]};
   do echo -n "[$k] "; done
   ```
   - *# [a] [b] [c] [def]*
2. ```
   for k in "${!arr5[@]}";
   do echo -n "[$k] "; done
   ```
   - *# [a b c] [def]*
3. ```
   for k in ${!arr5[*]};
   do echo -n "[$k] "; done
   ```
   - *# [a] [b] [c] [def]*
4. ```
   for k in "${!arr5[*]}";
   do echo -n "[$k] "; done
   ```
   - *# [a b c def]*

Операции со строками

```
s=abacaba.c.tar.gz
echo ${#s} # 16
echo ${s:3} # caba.c.tar.gz
echo ${s:3:4} # caba
echo ${s:(-5)} # ar.gz
echo ${s#*.} # c.tar.gz
echo ${s##*.} # gz
echo ${s%.*} # abacaba.c.tar
echo ${s%%.*} # abacaba
echo ${s/aba/zyz} # zyzcaba.c.tar.gz
echo ${s//aba/zyz} # zyzczyz.c.tar.gz
echo ${s/#aba/zyz} # zyzcaba.c.tar.gz
echo ${s/#caba/zyz} # abacaba.c.tar.gz
echo ${s/%.gz/} # abacaba.c.tar
echo ${s/%.tar/} # abacaba.c.tar.gz
```

Ура!

Ура, bash закончился!

Регулярные выражения

- Способ работы с текстом
  - Обобщение glob-ов
  - Найти шаблон в тексте
  - Найти и заменить

Регулярные выражения

- Способ работы с текстом
  - Обобщение glob-ов
  - Найти шаблон в тексте
  - Найти и заменить
- Полезные
- Часто используются

Регулярные выражения

- Способ работы с текстом
  - Обобщение glob-ов
  - Найти шаблон в тексте
  - Найти и заменить
- Полезные
- Часто используются
- Много разных несовместимых реализаций
  - PCRE (Perl-Compatible Regular Expressions)

Регулярные выражения

- Слишком мощные

```
$ python -c \
"print(\"\n\".join(\"1\"*i for i in range(2, 15)))" \
| grep -P "^.?$|^(..+?)\1+$"
1111
111111
11111111
111111111
1111111111
11111111111
111111111111
```

Регулярные выражения

- Слова:

```
echo abacaba | egrep -o aba # aba
```

Регулярные выражения

- Слова:

```
echo abacaba | egrep -o aba # aba
```

- Или: ... `"a(b|c)ab"` # *acab*

Регулярные выражения

- Слова:

```
echo abacaba | egrep -o aba # aba
```

- Или: ... `"a(b|c)ab"` `# acab`
- .: ... `"a..b"` `# acab`

Регулярные выражения

- Слова:

```
echo abacaba | egrep -o aba # aba
```

- Или: ... `"a(b|c)ab"` *# acab*
- .: ... `"a..b"` *# acab*
- Группы: ... `"c[a-c][bc][^b]"` *# caba*

Регулярные выражения

- Слова:

  ```
  echo abacaba | egrep -o aba # aba
  ```

- Или: `... "a(b|c)ab" # acab`
- .: `... "a..b" # acab`
- Группы: `... "c[a-c][bc][^b]" # caba`
- Повторы:
  - `*` — ноль или более раз (`[a-z]*`)
  - `+` — один или более раз
  - `?` — ноль или один раз
  - $\{n\}$ — n раз
  - $\{n,\}$ — n или более раз
  - $\{,n\}$ — n или менее раз
  - $\{n,m\}$ — от n до m раз

Регулярные выражения

- ^ — начало строки
- $ — конец строки

Регулярные выражения

- ^ — начало строки
- $ — конец строки
- backreference
  - `$ echo abacaba | egrep -o "([a-b]{3}).*\1"`
    `abacaba`
  - `$ echo abacabb | egrep -o "([a-b]{3}).*\1"`
- ... и ещё куча всего

grep

- Этимология
  - g/re/p

grep

- Этимология
  - g/re/p
- Ищет регулярное выражение в тексте
  - fgrep, grep, egrep, grep -P

sed

- «Текстовый редактор»

sed

- «Текстовый редактор»
- Язык программирования
  - Умельцы пишут тетрисы и арканоиды

sed

- «Текстовый редактор»
- Язык программирования
  - Умельцы пишут тетрисы и арканоиды
- s///
  - s/что/на что/флаги
  - $ strace ls |& grep open
    | sed -re "s/.*\"(.*)\".*/\1/"
    /etc/ld.so.cache
    /usr/lib/libcap.so.2
    /usr/lib/libc.so.6
    /usr/lib/locale/locale-archive
    .

Ещё полезные утилиты

- head, tail
- sort
- uniq
- find
- xargs
- pushd/popd