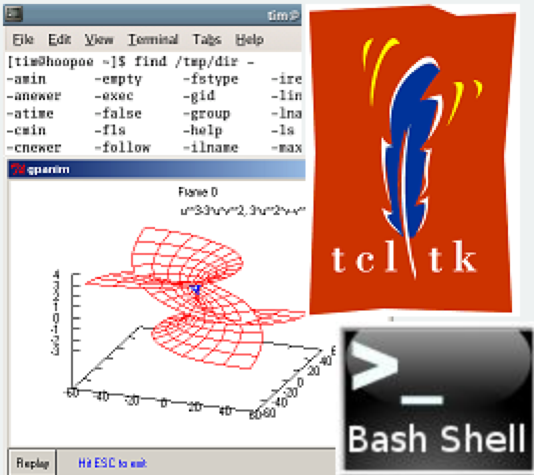
МИЭТ

Национальный исследовательский университет «МИЭТ»

Кафедра ПКИМС

Интерпретируемые языки программирования



Лабораторная работа №2

Работа с командной строкой в Linux

Язык командного интерпретатора BASH

Начало работы – установка прав и написание скрипта:

```
[topgun@4132-s scripts]$ touch script.sh
[topgun@4132-s scripts]$ chmod +x ./script.sh
[topgun@4132-s scripts]$ sublime_text3 script.sh &
```

```
#!/bin/bash

echo "Hello, world"
```

```
[topgun@4132-s scripts]$ ./script.sh
```

Форматированный вывод на консоль: команда echo (1)

Общий синтаксис для echo:

```
echo [опции] [строка]
```

Опции:

`-e` – обрабатывать управляющие символы в составе строки
`-n` – не выводить перевод строки в конце строки

Управляющие символы:

`\n` – перевод строки
`\t` – горизонтальная табуляция
`\v` – вертикальная табуляция
`\b` – аналог backspace
`\a` – вывод звукового сигнала
`\r` – удаление текста слева до этого места

Форматированный вывод на консоль: команда echo (3)

Управляющие последовательности для манипуляции с цветом:

Цвет текста

```
\033[30m – чёрный
\033[31m – красный
\033[32m – зелёный
\033[33m – жёлтый
\033[34m – синий
\033[35m – фиолетовый
\033[36m – голубой
\033[37m – серый
```

Цвет фона

```
\033[40m – чёрный
\033[41m – красный
\033[42m – зелёный
\033[43m – жёлтый
\033[44m – синий
\033[45m – фиолетовый
\033[46m – голубой
\033[47m – серый
```

Сброс до значений по умолчанию

```
\033[0m
```

Вывод на консоль: команда printf

Общий синтаксис для printf:

```
printf [формат] [аргументы]
```

Управляющие символы:

- `\n` – перевод строки
- `\t` – горизонтальная табуляция
- `\v` – вертикальная табуляция
- `\b` – аналог backspace
- `\a` – вывод звукового сигнала
- `\r` – удаление текста слева до этого места

Варианты исполнения:

```
printf "%s : %d" "Аудитория" 4131
printf "%10s%10s" "String1" "String2"
printf "%-10s%-10s" "String1" "String2"
```

Работа с переменными (1)

Синтаксис инициализации:

```
y=11                # правильный синтаксис
B=""                # правильный синтаксис
x = 3               # неправильный синтаксис
s=String string     # неправильный синтаксис
s="String string"   # правильный синтаксис
```

Синтаксис обращения:

```
echo Var value is $var
```

или

```
echo Var value is ${var}
```

Использование в вычислениях:

```
r=$(( $x + $y ))
```

или

```
r=$(( ${x} + ${y} ))
```

Работа с переменными (2)

В переменную можно записать результат выполнения команды:

```
a=`ls`
echo $a
```

или

```
a=$(ls)
echo $a
```

Чтение с консоли: команда read

Общий синтаксис для read:

```
read [опции] <переменная>
```

Значимые опции:

```
-s
-p строка
-n число
-d разделитель
```

- `-s` – «тихий» режим, символы не выводятся
- `-p строка` – вывести приглашение
- `-n число` – считать только заданное число символов
- `-d разделитель` – считать не до конца строки, а до разделителя

Выполнение вычислений в bash (1)

```
x=4

a=$[ $x + 1 + 5 ]

b=$(( $x + 1 + 5 ))

let c=$x+1+5

d=$( expr $x + 1 + 5 )
```

```
x=4

a=$[ $x * 2 + 1 + 5 ]

b=$(( $x * 2 + 1 + 5 ))

let c=$x*2+1+5

d=$( expr $x*2 + 1 + 5 )

d=$( expr $x * 2 + 1 + 5 )

d=$( expr $x \* 2 + 1 + 5 )

d=$( expr $x\*2 + 1 + 5 )
```

( ), (( )), [ ], [[ ]] (1)

$(...) – выполнить то, что внутри скобок, результат вернуть
выполнить – средствами операционной системы (команда, программа)

$((...)) – вычислить математическое выражение в скобках

[...] – внутри скобок – условное выражение, помним, что внутри таких скобок никакой пробел лишним не будет!
Является заменой команде test.

[[...]] – альтернативная форма условного выражения, позволяет писать логические выражения внутри себя, а также использовать регулярные выражения.

Условный оператор if-then-else-fi

Общий синтаксис для if:

```
if [ <условие> ]
then

else

fi
```

Общий синтаксис для if с then в одной строке:

```
if [ <условие> ]; then

else

fi
```

Условный оператор if-then-else-fi для работы с файлами

Варианты использования условия в операторе if для работы с файлами и директориями

```
[ -d FILE ] Правда если FILE существует и это директория.
[ -e FILE ] Правда если FILE существует.
[ -f FILE ] Правда если FILE существует и это регулярный файл.
[ -h FILE ] Правда если FILE существует и это символическая ссылка.
[ -r FILE ] Правда если FILE существует и он доступен на чтение.
[ -s FILE ] Правда если FILE существует и больше нуля.
[ -w FILE ] Правда если FILE существует и он доступен на запись.
[ -x FILE ] Правда если FILE существует и он доступен на исполнение.
```

```
if [ -d $fname ]; then
  echo "$fname is a directory"
else
  echo "$fname is not a directory"
fi
```

Оператор множественного выбора case

Синтаксис:

```
case <переменная> in
  <шаблон1> )
     <набор команд 1>
     ;;
  <шаблон2> )
     <набор команд 1>
     ;;
esac
```

Синтаксис с использованием дефолтной секции:

```
case <переменная> in
  <шаблон1> )
     <набор команд 1>
     ;;
  <шаблон2> )
     <набор команд 1>
     ;;
  *)
  <набор команд 1>
     ;;
esac
```

Оператор множественного выбора case

```
#!/bin/bash

echo;
echo "Press a key: "
read -n 1 k

case "${k}" in
  [a-z] ) echo " lowercase letter";;
  [A-Z] ) echo " uppercase letter";;
  [0-9] ) echo " digit";;
  " "   ) echo " space";;
  "." | "," ) echo " separator";;
  * ) echo "Something different";;
esac
```

Циклы в bash: цикл for (1)

```
#!/bin/bash

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

```
#!/bin/bash

for i in {1..5}
do
  echo "Value is $i"
done
```

```
#!/bin/bash

for i in {0..20..2}
do
  echo "Value is $i"
done
```

```
#!/bin/bash

for i in $(seq 1 2 20)
do
  echo "Value is $i"
done
```

Циклы в bash: цикл for (2)

```
#!/bin/bash

for (( c=1; c<=5; c++ ))
do
  echo "Value is $c"
done
```

```
#!/bin/bash

for file in /home/topgun/*
do
  if [ "${file}" == ".bashrc" ]
  then
    echo ".bashrc found!"
    break
  fi
done
```

Циклы в bash: цикл while

```
#!/bin/bash

n=1

while [ $n -le 5 ]
do
  echo "Value is $n"
  n=$(( n+1 ))
done
```

```
#!/bin/bash

n=1

while (( $n <= 5 ))
do
  echo "Value is $n"
  n=$(( n+1 ))
done
```

Аргументы командной строки (1)

```
[topgun@4132-s scripts]$ ./script.sh Hello world
```

Обратиться к каждому из аргументов:

$1, $2 и так далее

```
#!/bin/bash

echo $1
echo $2
```

Узнать число аргументов:

$#

Обратиться к списку аргументов:

$@

Проход по аргументам командной строки

Как пробежаться по всем аргументам, переданным программе?

Вариант 1:

```
for var in "$@"
do
  echo "$var"
done
```

Вариант 2:

```
while (( "$#" )); do
  echo $1
  shift
done
```

Использование подпрограмм (1)

```
#!/bin/bash

function print_msg {
  echo Function call!
}

print_msg
```

```
#!/bin/bash

function print_msg () {
  echo Function call!
}

print_msg
```

```
#!/bin/bash

function print_msg {
  echo $1
}

print_msg "Function call with args!"
```

Использование подпрограмм (2)

```
#!/bin/bash

function calc_sum {
  return $(($1 + $2))
}

print_msg 3 4
x=$?
```

```
#!/bin/bash

function calc_sum {
  return $(($1 + $2))
}

x=$(print_msg 3 4)
```

Обработка результатов работы программ (1)

```c
#include <stdio.h>

int main() {
  printf("PKIMS RULEZZZ!\n");
  return 0;
}
```

```
$proga > file.log
```

```c
#include <stdio.h>

int main() {
  printf("PKIMS RULEZZZ!\n");
  fprintf(stderr, "Some error!\n");
  return 0;
}
```

```
$proga > file.log

$proga > file.log 2> err.log

$proga > file.log 2>&1

$proga > file.log 2>/dev/null
```

Обработка результатов работы программ (2)

```c
#include <stdio.h>

int main() {
  printf("PKIMS RULEZZZ!\n");
  return 0;
}
```

```c
#include <stdio.h>

int main() {
  printf("PKIMS RULEZZZ!\n");
  return 42;
}
```

```bash
#!/bin/bash

./proga
echo Return code: $?
```

Обработка файлов: cat

```
$cat file.txt

$cat file1.txt file2.txt

$cat > file.txt

$cat -n file.txt
$cat -b file.txt
$cat -e file.txt
$cat -s file.txt
$cat -t file.txt
$cat -v file.txt
```

Фильтрация текстовых данных: grep

```
$cat circuit.log | grep "Simulation finished"
```

```
$cat file.txt | grep "abc"
$cat file.txt | grep -v "abc"
$cat file.txt | grep -w "abc"
$cat file.txt | grep -i "abc"
$cat file.txt | grep -c "abc"
```

```
$cat file.txt | grep "abc|bcd"
```

```
$cat file.txt | grep "^[0-9]"

$cat file.txt | grep "[0-9]$"
```

Обработка данных командой awk

```
$awk ' { print } ' /etc/passwd

$awk ' { print $0 } ' /etc/passwd

$awk ' { print $1 } ' /etc/passwd

$awk -F  ":" ' { print $1 } ' /etc/passwd
```

Задание

Написать скрипт, работающий следующим образом.

1. Принимает 1 аргумент командной строки – имя каталога, в котором мы работаем.
   Если аргумента нет – ругаемся и выходим.
   Если это не каталог – ругаемся и выходим.
2. Переходит в этот каталог и пробегается по всем его подкаталогам.
3. Заходит в каждый подкаталог и смотрит, есть ли там файлы .cir.
4. Если находит файлы .cir, то для них он вызывает симулятор symspice.
5. Считает число успешных и неуспешных запусков симулятора, выводит эти данные в конце своей работы.
6. Результаты моделирования успешных запусков складывает в папку Ok, результаты неуспешных – в папку Failed. Все папки – внутри папки, задаваемой первым аргументом. Если они не существуют – создать.