

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Лабораторная работа №12

Азимов М.

29 апреля 2023

Российский университет дружбы народов, Москва, Россия

Информация

- Азимов Миразим
- студент 1 курса, группа НММбд-01-22
- Российский университет дружбы народов



Вводная часть

- Командный процессор ОС UNIX
- Командные файлы

- Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

- Ознакомиться с теоретическим материалом.
- Выполнить упражнения.
- Ответить на контрольные вопросы.

Выполнение лабораторной работы №12

Первая программа

Открыть ▾  lab12_1.sh
~/

```
#!/bin/bash
t1=$1
t2=$2
s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))
while ((t < t1)) do
    echo "Ожидайте"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done
s1=$(date +%s")
s2=$(date +%s")
((t=$s2-$s1))
while (( t < t2)) do
    echo "Выполнение"
    sleep 1
    s2=$(date +%s")
    ((t=$s2-$s1))
done
```

```
[mazimov@fedora ~]$ chmod +x lab12_1.sh
```

```
[mazimov@fedora ~]$ ./lab12_1.sh 4 5
Ожидайте
Ожидайте
Ожидайте
Ожидайте
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
```

Первая программа доработка

```
        sleep 1
        s2=$(date +%s")
        ((t=$s2-$s1))
done
t1=$1
t2=$2
command=$3
while true
do
    if [ "$command" == "Выход" ]
    then echo "Выход"
        exit 0
    fi
    if [ "$command" == "Ожидание" ]
    then pass
    fi
    if [ "$command" == "Выполнение" ]
    then pass
    fi
    echo "Следующее действие"
    read command
done
```

```
Выполнение
[mazimov@fedora ~]$ ./lab12_1.sh 4 5 3
Ожидайте
Ожидайте
Ожидайте
Ожидайте
Выполнение
Выполнение
Выполнение
Выполнение
Выполнение
Следующее действие
Следующее действие
Следующее действие
```

Вторая программа

```
[mazimov@fedora ~]$ cd /usr/share/man/man1
[mazimov@fedora man1]$ ls
.:1.gz
'[:1.gz'
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
```

```
[mazimov@fedora ~]$ touch lab12_2.sh
```

Вторая программа


```
Открыть ▾ + lab12_2.sh
~

#!/bin/bash
c=$1
if [ -f /usr/share/man/man1/${c}.1.gz ]
then
    gunzip -c /usr/share/man/man1/${c}.1.gz | less
else
    echo "Справки по данной команде нет"
fi
```

```
[mazimov@fedora ~]$ ./lab12_2.sh ls
[3]+  Остановлен ./lab12_2.sh ls
[mazimov@fedora ~]$ ./lab12_2.sh cd
[4]+  Остановлен ./lab12_2.sh cd
```

```
mazimov@fedora: ~ — /bin/bash ./lab12_2.sh ls
\# DO NOT MODIFY THIS FILE! It was generated by help2man 1.48.5.
.TH LS "1" "March 2022" "GNU coreutils 9.0" "User Commands"
.SH NAME
ls \- list directory contents
.SH SYNOPSIS
.B ls
[\fI,\fO,\fR]... [\fI,\fO,\fR]...
.SH DESCRIPTION
\# Add any additional description here
.PP
List information about the files (the current directory by default).
Sort entries alphabetically if none of \fB\-\fR nor \fB\-\sort\fR is sp
ecified.
.PP
Mandatory arguments to long options are mandatory for short options too.
.TP
\fB\-\a\fR, \fB\-\all\fR
do not ignore entries starting with .
.TP
\fB\-\A\fR, \fB\-\almost-all\fR
do not list implied, and ..
.TP
\fB\-\author\fR
```

Третья программа

```
Открыть ▾  lab12_3.sh  
~/  
#!/bin/bash  
k=$1  
for (( i=0; i<$k; i++ )) do  
  (( char=$((RANDOM%26+1)) ))  
  case $char in  
    1) echo -n a;;  
    2) echo -n b;;  
    3) echo -n c;;  
    4) echo -n d;;  
    5) echo -n e;;  
    6) echo -n f;;  
    7) echo -n g;;  
    8) echo -n h;;  
    9) echo -n i;;  
    10) echo -n j;;  
    11) echo -n k;;  
    12) echo -n l;;  
    13) echo -n m;;  
    14) echo -n n;;  
    15) echo -n o;;  
    16) echo -n p;;  
    17) echo -n q;;  
    18) echo -n r;;  
    19) echo -n s;;  
    20) echo -n t;;  
    21) echo -n u;;  
    22) echo -n v;;  
    23) echo -n w;;  
    24) echo -n x;;  
    25) echo -n y;;  
    26) echo -n z;;  
  esac  
done
```

```
[mazimov@fedora ~]$ chmod +x lab12_3.sh
```

```
[mazimov@fedora ~]$ ./lab12_3.sh 12  
uscprvqddnha  
[mazimov@fedora ~]$ ./lab12_3.sh 20  
msjppjagvsxicyjapss0
```

1. `while [$1 != "exit"]` В данной строчке допущены следующие ошибки: • не хватает пробелов после первой скобки [и перед второй скобкой] • выражение `$1` необходимо взять в `" "`, потому что эта переменная может содержать пробелы. Таким образом, правильный вариант должен выглядеть так: `while ["$1" != "exit"]`
2. Чтобы объединить несколько строк в одну, можно воспользоваться несколькими способами:

Первый: `VAR1="Hello", "VAR2=" World" VAR3="☒ ☒☒1VAR2" echo "VAR3" :
Hello,World : VAR1 = "Hello",VAR1+ = "World"echo"VAR1"` Результат: Hello, World

4. Результатом данного выражения `$((10/3))` будет 3, потому что это целочисленное деление без остатка.

5. Отличия командной оболочки `zsh` от `bash`:

В `zsh` более быстрое автодополнение для `cd` помощью `Tab` В `zsh` существует калькулятор `zcalc`, способный выполнять вычисления внутри терминала В `zsh` поддерживаются числа с плавающей запятой В `zsh` поддерживаются структуры данных «хэш» В `zsh` поддерживается раскрытие полного пути на основе неполных данных В `zsh` поддерживается замена части пути В `zsh` есть возможность отображать разделенный экран, такой же как разделенный экран `vim`

6. `for((a=1; a<= LIMIT; a++))` синтаксис данной конструкции верен, потому что, используя двойные круглые скобки, можно не писать `$` перед переменными

Результаты

В ходе выполнения лабораторной работы были изучены основы программирования в оболочке ОС UNIX. Научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.