

Отчет

Лабораторная работа №11

Павлова Варвара Юрьевна НПМбд-02-21

Содержание

Цель работы	5
Задание	6
Теоретическое введение	8
Выполнение лабораторной работы	9
Выводы	15
Список литературы	16

Список иллюстраций

0.1	создание файла	9
0.2	написание скрипта	10
0.3	написание скрипта	10
0.4	проверка первого файла	10
0.5	создание файла	11
0.6	написание 2.c	11
0.7	написание 2.sh	12
0.8	проверка второго файла	12
0.9	создание файла	12
0.10	написание скрипта	13
0.11	проверка третьего файла	13
0.12	создание файла	14
0.13	написание скрипта	14
0.14	проверка четвертого файла	14

Список таблиц

Цель работы

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

Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:
 - `-iinputfile` — прочитать данные из указанного файла;
 - `-ooutputfile` — вывести данные в указанный файл;
 - `-rшаблон` — указать шаблон для поиска;
 - `-C` — различать большие и малые буквы;
 - `-n` — выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом `-r`.
2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например `1.tmp`, `2.tmp`, `3.tmp`, `4.tmp` и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).
4. Написать командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы

запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`).

Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: - оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; - C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; - оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; - BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

Выполнение лабораторной работы

1. Создаю первый исполняемый файл 1.sh и открываю редактор emacs. (рис. [-@fig:001])

```
[vypavlova@fedora ~]$ touch 1.sh
[vypavlova@fedora ~]$ emacs &
[1] 3273
```

Рис. 0.1: создание файла

3. Используя команды getoptс grep, пишу командный файл, который анализирует командную строку с ключами:
 - -iinputfile — прочитать данные из указанного файла;
 - -ooutputfile — вывести данные в указанный файл;
 - -ршаблон — указать шаблон для поиска;
 - -С — различать большие и малые буквы;
 - -п — выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом -р. (рис. [-@fig:002]) (рис. [-@fig:003])

```
#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:o:p:C:n optletter
do case $optletter in
    i) iflag=1 ival=$OPTARG;;
    o) oflag=1 oval=$OPTARG;;
    p) pflag=1 pval=$OPTARG;;
    C) cflag=1;;
    n) nflag=1;;
    *) echo illegal option $optletter
    esac
done
if (($pflag==0))
then echo "not found"
else
    if (($iflag==0))
    then echo "file not found"
    else
        if (($oflag==0))
        then if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival
                else grep -n $pval $ival
                fi
            else if (($nflag==0))
                then grep -i $pval $ival
                else grep -i -n $pval $ival
                fi
            fi
        else if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival > $oval
                else grep -n $pval $ival > $oval
                fi
            fi
        fi
    fi
fi
```

Рис. 0.2: написание скрипта

```
else if (($cflag==0))
then if (($nflag==0))
    then grep $pval $ival > $oval
    else grep -n $pval $ival > $oval
    fi
else if (($nflag==0))
    then grep -i $pval $ival > $oval
    else grep -i -n $pval $ival > $oval
    fi
fi
fi
fi
```

Рис. 0.3: написание скрипта

4. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:004])

```
[vypavlova@fedora ~]$ chmod +x 1.sh
[vypavlova@fedora ~]$ ./1.sh -i 11.txt -C -n
not found
[vypavlova@fedora ~]$ ./1.sh -o 22.txt -p file -C -n
file not found
```

Рис. 0.4: проверка первого файла

5. Создаю второй исполняемый файл 2.sh и 2.c и открываю редактор emacs. (рис. [-@fig:005])

```
[vypavlova@fedora ~]$ touch 2.c 2.sh
[1]+  Done                  emacs
[vypavlova@fedora ~]$ emacs &
[1] 5490
```

Рис. 0.5: создание файла

6. Пишу на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции `exit(n)`, передавая информацию в о коде завершения в оболочку. (рис. [-@fig:006]) Командный файл вызывает эту программу и, проанализировав с помощью команды `$?`, выдает сообщение о том, какое число было введено. (рис. [-@fig:007])

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a;
    printf("Input the number: \n");
    scanf ("%d", &a);
    if (a<0)
        exit(0);
    if (a>0)
        exit(1);
    if (a==0)
        exit(2);
    return 0;
}
```

Рис. 0.6: написание 2.c

```
#!/bin/bash
gcc 2.c -o g
./g
code=$?
case $code in
    0) echo "<0";;
    1) echo ">0";;
    2) echo "=0";;
esac
```

Рис. 0.7: написание 2.sh

7. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:008])

```
[vypavlova@fedora ~]$ chmod +x 2.sh
[vypavlova@fedora ~]$ ./2.sh
Input the number:
0
=0
[vypavlova@fedora ~]$ ./2.sh
Input the number:
7
>0
[vypavlova@fedora ~]$ ./2.sh
Input the number:
-8
<0
```

Рис. 0.8: проверка второго файла

8. Создаю третий исполняемый файл 3.sh и открываю редактор emacs. (рис. [-@fig:009])

```
[vypavlova@fedora ~]$ touch 3.sh
[vypavlova@fedora ~]$ emacs &
[1] 6980
```

Рис. 0.9: создание файла

9. Пишу командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (например 1.tmp, 2.tmp, 3.tmp, 4.tmp и т.д.). Число

файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же командный файл должен уметь удалять все созданные им файлы (если они существуют). (рис. [-@fig:010])

```
#!/bin/bash
opt=$1;
format=$2;
number=$3;
function g ()
{
    for (( i=1; i<=$number; i++)) do
        file=$(echo $format | tr '#' "$i")
        if (( $opt == "-r" ))
        then
            rm -f $file
        elif (( $opt == "-c" ))
        then
            touch $file
        fi
    done
}
g
```

Рис. 0.10: написание скрипта

10. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:011])

```
[vypavlova@fedora ~]$ chmod +x 3.sh
[vypavlova@fedora ~]$ ./3.sh -c et#.txt 3
[vypavlova@fedora ~]$ ls
11.txt  2.c  3.sh  bin  g  Public  Videos
1.sh~  2.c~  3.sh~ Desktop  Music  reports  work
1.sh~  2.sh  3.txt Documents  my_os  second.sh~
1.txt  2.sh~ backup Downloads  opsystemlab  Templates
22.txt 2.txt backup.sh~ fourth.sh~ Pictures  third.sh~
[vypavlova@fedora ~]$ ./3.sh -r et#.txt 3
[vypavlova@fedora ~]$ ls
11.txt  2.c  3.sh  bin  fourth.sh~ opsystemlab  second.sh~  work
1.sh~  2.c~  3.sh~ Desktop  g  Pictures  Templates
1.sh~  2.sh  backup Documents  Music  Public  third.sh~
22.txt  2.sh~ backup.sh~ Downloads  my_os  reports  Videos
```

Рис. 0.11: проверка третьего файла

11. Создаю четвертый исполняемый файл 4.sh и открываю редактор emacs. (рис. [-@fig:012])

```
[vypavlova@fedora ~]$ touch 4.sh
[1]+  Done                  emacs
[vypavlova@fedora ~]$ emacs &
[1] 9355
```

Рис. 0.12: создание файла

12. Пишу командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицирую его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использую команду find). (рис. [-@fig:013])

```
#!/bin/bash
files=$(find ./-maxdepth 1 -mtime -7)
listing=""
for file in "$files"; do
    file=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

Рис. 0.13: написание скрипта

13. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:014])

```
[vypavlova@fedora ~]$ chmod +x 4.sh
[vypavlova@fedora ~]$ ./4.sh backup
find: './-maxdepth': No such file or directory
4.sh
tar: 1.txt: Cannot stat: No such file or directory
```

Рис. 0.14: проверка четвертого файла

Выводы

Выполняя данную лабораторную работу я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.

Список литературы