

Отчет по лабораторной работе №11

дисциплина: Операционные системы

Старков Никита Алексеевич

Содержание

| | | |
|----------|---------------------------------------|-----------|
| 1 | Цель работы | 4 |
| 2 | Выполнение лабораторной работы | 5 |
| 3 | Контрольные вопросы | 11 |
| 4 | Вывод | 14 |

Список иллюстраций

| | | |
|------|--------------------------------------|----|
| 2.1 | Создание файла | 5 |
| 2.2 | Скрипт №1 | 6 |
| 2.3 | Создание файлов | 6 |
| 2.4 | Содержимое a1.txt | 6 |
| 2.5 | Проверка работы программы | 7 |
| 2.6 | Создание файлов | 7 |
| 2.7 | Скрипт файла chslo.c | 7 |
| 2.8 | Скрипт файла chslo.sh | 8 |
| 2.9 | Проверка работы программы | 8 |
| 2.10 | Скрипт №3 | 9 |
| 2.11 | Проверка работы скрипта №3 | 9 |
| 2.12 | Проверка работы скрипта №3 | 9 |
| 2.13 | Скрипт №4 | 10 |
| 2.14 | Проверка работы скрипта №4 | 10 |

1 Цель работы

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

2 Выполнение лабораторной работы

1)Используя команды `getopts` `grep`, напишем 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;–
- n — выдавать номера строк.

а затем ищет в указанном файле нужные строки, определяемые ключом -р. Создаем файл `prog11.sh` и пишем соответствующий скрипт

```
nastarkov@dk8n67 ~ $ touch prog11.sh
nastarkov@dk8n67 ~ $ emacs &
```

Рис. 2.1: Создание файла

```
#!/bin/bash
iflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:op: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 "Шаблон не найден"
else
if (($iflag==0))
then echo "Файл не найден"
else
if (($oflag==0))
then if (($cflag==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 $nval
fi
fi
else if (($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
```

Рис. 2.2: Скрипт №1

Проверяем работу написанного скрипта, предварительно создав 2 файла a1.txt и a2.txt. В a1.txt записываем любой набор слов. Также даем доступ на исполнение файла

```
nastarkov@dk8n67 ~ $ touch a1.txt a2.txt
```

Рис. 2.3: Создание файлов

```
nastarkov@dk8n67 ~ $ cat a1.txt
water abc abcs
asd
prog1
water water
```

Рис. 2.4: Содержимое a1.txt

```
nastarkov@dk8n67 ~ $ chmod +x prog11.sh
[1]+  Завершён      emacs
nastarkov@dk8n67 ~ $ chmod +x prog11.sh
nastarkov@dk8n67 ~ $ ./prog11.sh -i a1.txt -o a2.txt -p water -n
./prog11.sh: строка 2: 0flag=0: команда не найдена
nastarkov@dk8n67 ~ $ emacs &
[1] 12279
nastarkov@dk8n67 ~ $ ./prog11.sh -i a1.txt -o a2.txt -p water -n
[1]+  Завершён      emacs
nastarkov@dk8n67 ~ $ ./prog11.sh -i a1.txt -o a2.txt -p water -n
nastarkov@dk8n67 ~ $ cat a2.txt
1:water abc abcs
4:water water
```

Рис. 2.5: Проверка работы программы

2) Напишем 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 о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.

Создаем два файла: `chslo.c` и `chslo.sh`

```
nastarkov@dk8n67 ~ $ touch chslo.c
nastarkov@dk8n67 ~ $ touch chslo.sh
nastarkov@dk8n67 ~ $ emacs &
```

Рис. 2.6: Создание файлов

Пишем соответствующие скрипты

```
File Edit Options Buffers Tools C Help
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    printf ("Введите число\n");
    int a;
    scanf("%d", &a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a==0) exit(2);
    return 0;
}
```

Рис. 2.7: Скрипт файла `chslo.c`

```
#!/bin/bash
gcc chslo.c -o chslo
./chslo
code=$?
case $code in
  0) echo "Число меньше 0";;
  1) echo "Число больше 0";;
  2) echo "Число равно нулю";;
esac
```

Рис. 2.8: Скрипт файла chslo.sh

Проверяем работу программы, предварительно открыв доступ на исполнение файла

```
nastarkov@dk8n67 ~ $ chmod +x chslo.sh
nastarkov@dk8n67 ~ $ ./chslo.sh
Введите число
0
Число равно нулю
nastarkov@dk8n67 ~ $ ./chslo.sh
Введите число
5
Число больше 0
nastarkov@dk8n67 ~ $ ./chslo.sh
Введите число
-1
Число меньше 0
```

Рис. 2.9: Проверка работы программы

3) Напишем командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до ∞ (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)

Создаем файл files.sh и пишем соответствующий скрипт


```

File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
opt=$1
format=$2;
number=$3;
function Files()
{
    for (( i=1; i<=$number; i++ )) do
        file=$(echo $format | tr '#' "$i")
        if [ $opt == "-r" ]
        then
            rm -f $file
        elif [ $opt == "-c" ]
        then
            touch $file
        fi
    done
}
Files

```

Рис. 2.10: Скрипт №3

Далее проверяем работу написанного скрипта, добавив право на исполнение.

```

nastarkov@dk8n67 ~ $ chmod +x files.sh
nastarkov@dk8n67 ~ $ ls
a1.txt      a2.txt      australia  backup      backup.sh  backup.sh~  bin        blog        chslo      chslo.c     chslo.c~  chslo.sh  chslo.sh~  conf.txt  course-directory-student-template  equipment
f1.txt      f2.txt      f3.txt     f4.txt      file2.doc  file.doc    file.pdf   files.sh    files.sh~  file.txt    format.sh  format.sh~  GNUstep  lab07.sh  lab07.sh~  may        my_os       'NOVIY KATALOG'  play        prog11.sh  prog11.sh~  prog1.sh  prog1.sh~  prog2.sh  prog2.sh~  progls.sh  progls.sh~  progls.sh

```

Рис. 2.11: Проверка работы скрипта №3

```

nastarkov@dk8n67 ~ $ ./files.sh -r abc.txt 3
nastarkov@dk8n67 ~ $ ls
a1.txt      backup.sh~  chslo.c~  equipment  f1.txt     file2.doc  file.txt  lab07.sh~  prog11.sh  prog2.sh~  text.txt  загрузки  Шаблоны
a2.txt      bin        chslo.sh  f1.txt     f2.txt     file.doc   format.sh  my        prog11.sh~  prog1.sh  tmp       Изображения
australia  blog       chslo.sh~  f2.txt     f4.txt     files.pdf  format.sh~  my_os    prog1.sh  progls.sh~  public    work      Музыка
backup      chslo      conf.txt  f3.txt     files.sh   GNUstep   'NOVIY KATALOG'  play      prog1.sh~  progls.sh  public_html  Видео     Общедоступные
backup.sh  chslo.c   course-directory-student-template  f4.txt~    feathers   files.sh~  lab07.sh  prog2.sh  progls.sh  progls.sh~  'Рабочий стол'

```

Рис. 2.12: Проверка работы скрипта №3

4) Напишем 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).

Создаем файл prog4.sh и пишем в нем соответствующий скрипт

```

#!/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. 2.13: Скрипт №4

Далее проверяем работу написанного скрипта, предварительно добавив право на исполнение файла и создав отдельный каталог Catalog1 с несколькими файлами.

```

nastarkov@dkn67 ~ $ cd Catalog1
nastarkov@dkn67 ~/Catalog1 $ ./prog4.sh
bash: ./prog4.sh: команда не найдена
nastarkov@dkn67 ~/Catalog1 $ ./prog4.sh
a1.txt
a2.txt
ch10 -
ch10.c
ch10.sh
nastarkov@dkn67 ~/Catalog1 $ tar -tf Catalog1.tar
a1.txt
a2.txt
ch10 -
ch10.c
ch10.sh

```

Рис. 2.14: Проверка работы скрипта №4

3 Контрольные вопросы

1). Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg...]` Флаги – это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`. Строка опций `option-string` – это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введённые данные с помощью оператора `case`. Функция `getopts` включает две специальные переменные среды `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.

2). При перечислении имён файлов текущего каталога можно использовать следующие символы:

`*` – соответствует произвольной, в том числе и пустой строке;

`?` – соответствует любому одинарному символу;

`[c1-c2]` – соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`;

`z]*` – соответствует произвольному имени файла в текущем каталоге, начинающемуся с `z`.

3). Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов проверки некоторого условия. Для решения подобных задач язык программирования `bash` предоставляет возможность использовать такие управляющие конструкции, как `for`, `case`, `if` и `while`. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда `test`, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.

4). Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. Команда `break` завершает выполнение цикла, а команда `continue` завершает данную итерацию блока операторов. Команда `break` полезна для завершения цикла `while` в ситуациях, когда условие перестаёт быть правильным. Команда `continue` используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

5). Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования `bash`: это команда `true`, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда `false`, которая всегда возвращает код завершения, неравный нулю (т.е. ложь). Примеры бесконечных циклов: `while true do echo hello andy done` и `until false do echo hello mike done`.

6). Строка `if test -f mans/i.s`, `mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение

(ложь).

7). Выполнение оператора цикла `while` сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `do`, после чего осуществляется безусловный переход на начало оператора цикла `while`. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово `while`, возвратит ненулевой код завершения (ложь). При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла `while` и оператор цикла `until` идентичны.

4 Вывод

Вывод: в ходе выполнения лабораторной работы я изучил основы программирования в оболочке ОС UNIX. Научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.