

Отчет по лабораторной работе № 13

Архитектура компьютера, раздел Операционные системы

Курсоро Майова Джеймс

Содержание

Цель.....	1
Задание.....	1
Теоретическое введение.....	2
Выполнение лабораторной работы.....	3
командный файл, который анализирует командную строку.....	3
программа, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.....	4
командный файл, создающий указанное число файлов.....	5
командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории.....	6
Выводы.....	7
Ответы на контрольные вопросы.....	7
Список литературы.....	9

Цель

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: `-inputfile` — прочитать данные из указанного файла; `-outputfile` — вывести данные в указанный файл; `-ршаблон` — указать шаблон для поиска; `-C` — различать большие и малые буквы; `-n` — выдавать номера строк. а

затем ищет в указанном файле нужные строки, определяемые ключом -p.

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

Bash (Bourne Again Shell) — это мощная командная оболочка Unix, которая используется для выполнения различных задач в терминале. Bash предоставляет интерактивный интерфейс, в котором пользователи могут вводить команды, а затем получать результаты. Она также поддерживает скрипты оболочки, которые представляют собой текстовые файлы, содержащие последовательность команд Bash для автоматизации задач. Bash широко используется в средах Unix и Linux, а также поддерживается Windows с помощью подсистемы Windows для Linux (WSL). Перечислим основные возможности этой оболочки. Обработка команд. Bash может обрабатывать как простые, так и сложные команды. Простые состоят из одного действия и, возможно, некоторых аргументов. Сложные команды могут содержать несколько простых, объединенных с помощью операторов конвейера (`|`), перенаправления ввода и вывода (`<`, `>`, `>>`), условных операторов (`if/else`, `case/esac`, `while/do`). О них мы расскажем позже. Расширенный ввод, редактирование строк. Оболочка предоставляет функции расширенного ввода, такие как автодополнение, которое предлагает возможные варианты завершения команд и имен файлов по мере их ввода. Она также поддерживает историю, позволяя пользователям просматривать ранее введенные команды, а затем повторно их использовать. Возможность создания и запуска скриптов. Скрипты оболочки — это текстовые файлы, содержащие

последовательность команд. Их можно создавать с помощью текстового редактора, а затем запускать в терминале, что дает возможность пользователям автоматизировать зада

Выполнение лабораторной работы

командный файл, который анализирует командную строку

Создаю файл file1 и в нем написала код, который анализирует командную строку с ключами -i (прочитать данные из указанного файла), -o (вывести данные в указанный файл), -p (указать шаблон для поиска), -C (различать большие и малые буквы), -n (выдавать номера строк) используя команды getoptс grep:



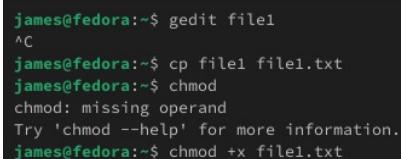
```
1 while getoptс "i:o:p:C:n" opt
2 do
3 case $opt in
4 i)inputfile="$OPTARG";;
5 o)outputfile="$OPTARG";;
6 p)template="$OPTARG";;
7 c)register="$OPTARG";;
8 n)number="";;
9 esac
10 done
11
12 grep -n "$template" "$inputfile.txt" > "$outputfile.txt"
```

Код для анализа командной строки

```
while getoptс "i:o:p:C:n" opt
do
case $opt in
i)inputfile="$OPTARG";;
o)outputfile="$OPTARG";;
p)template="$OPTARG";;
c)register="$OPTARG";;
n)number="";;
esac
done
```

```
grep -n "$template" "$inputfile.txt" > "$outputfile.txt"
```

Далее я установила права на исполнение и запустила файл:



```
james@fedora:~$ gedit file1
^C
james@fedora:~$ cp file1 file1.txt
james@fedora:~$ chmod
chmod: missing operand
Try 'chmod --help' for more information.
james@fedora:~$ chmod +x file1.txt
```

право на исполнение

```
james@fedora:~$ ./file1.txt -i file1 -o output -p n etconf -C -n
james@fedora:~$ cat output.txt
1:while getopt "i:o:p:C:n" opt
3:case $opt in
4:i)inputfile="$OPTARG";;
8:n)number="";;
10:done
12:grep -n "$template" "$inputfile.txt" > "$outputfile.txt"
james@fedora:~$
```

Запуск file1

программа, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю.

Написала 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:

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main()
5 {
6     int n;
7     printf("Enter a number: ");
8     scanf("%d", &n);
9     if(n>0)
10    {
11        exit(1);
12    }
13
14    else if (n==0) {
15        exit(0);}
16
17    else
18    {
19        exit(2);
20    }
```

Программа на языке Си

```
#include <stdlib.h>
#include <stdio.h>
```

```
int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    if(n>0)
    {
        exit(1);
    }
```

```

    else if (n==0) {
        exit(0);}
    else
    {
        exit(2);
    }
}

```

Далее создала командный файл который вызывает эту программу и, проанализировав с помощью команды \$?, выдает сообщение о том, какое число было введено:

```

1 gcc -o cprog file2.c
2 ./cprog
3
4 case $? in
5 0) echo "равно нулю";;
6 1) echo "больше нуля";;
7 2) echo "меньше нуля";;
8
9 esac

```

Командный файл программы на Си

```

gcc -o cprog file2.c
./cprog

```

```

case $? in
0) echo "равно нулю";;
1) echo "больше нуля";;
2) echo "меньше нуля";;

```

```

esac

```

Создала исполняемый файл и запустила:

```

james@fedora:~$ gedit file2.c
^C
james@fedora:~$ gedit command_file.sh
^C
james@fedora:~$ chmod +x command_file.sh
james@fedora:~$ ./command_file.sh
Enter a number: 7
больше нуля

```

Результаты программы

командный файл, создающий указанное число файлов

Я написала командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до *N*. Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же

командный файл должен уметь удалять все созданные им файлы (если они существуют):



```
1 for ((i=1; i<=$*; i++))
2 do
3 if test -f "$i".tmp
4 then rm "$i".tmp
5 else touch "$i".tmp"
6 fi
7 done
```

Командный файл для создания файлов

```
for((i=1; i≤$*; i++))
do
if test -f "$i".tmp
then rm "$i".tmp
else touch "$i.tmp"
fi
done
```

Создала исполняемый файл и запустила:

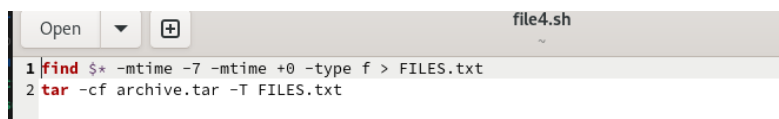


```
james@fedora:~$ chmod +x file3.sh
james@fedora:~$ ./file3.sh 3
james@fedora:~$ ./file3.sh 3
james@fedora:~$ gedit file3.sh
```

Создание файлов с помощью командного файла

командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории.

создала командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).



```
1 find $* -mtime -7 -mtime +0 -type f > FILES.txt
2 tar -cf archive.tar -T FILES.txt
```

Создание архива

```
find $* -mtime -7 -mtime +0 -type f > FILES.txt
tar -cf archive.tar -T FILES.txt
```

```

james@fedora:~$ ./file4.sh /home/james/work
james@fedora:~$ ls ~/work
blog James-4321.github.io os study_2024-2025_os-intro
james@fedora:~$ ls
1.tmp      CHANGELOG.md      Downloads  FILES.txt      mtheme        project3.sh      rs
2.tmp      command_file.sh   equipment  file.txt       Music         project.sh       sh
3.tmp      config            feathers   fun            output.txt    Public           st
abc1      conf.txt          file1     git-extended   Pictures      README.en.md    tw
archive.tar  COURSE           file1.txt LICENSE        play          README.git-flow.md  Td
australia  cprog            file2.c   Makefile       play.bak      README.md        te
backup     Desktop          file3.sh  may            project1.sh   report.docx     ts
bin        Documents        file4.sh  monthly        project2.sh   report.md        V
james@fedora:~$

```

Результаты кода

Выводы

При выполнении проделанной работы я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Ответы на контрольные вопросы

1. Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable`. Флаги – это опции командной строки, обычно помеченные знаком минус; Например, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -ifile_in.txt -ofile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае: `while getopts o:i:Ltr optletter do case $optletter in o) oflag=1; oval=OPTARG;; i) iflag=1; ival=$OPTARG;; L) Lflag=1;; t) tflag=1;; r) rflag=1;; *) echo Illegal option $optletter esac done` Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равно `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый аргумент. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для

синтаксического анализа аргументов функций, но и для анализа введенных 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`. Например, `echo *` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls .c` – выведет все файлы с последними двумя символами, совпадающими с `.c`. `echo prog.?` – выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.` `[a-z]` – соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.
3. Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов 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, не равный нулю (т.е. ложь).

6. Строка `if test -f mans/i.sn` проверяет, существует ли файл `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.

Список литературы

Архитектура коипьютеров