

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 11

дисциплина: Операционные системы

Студент: Хосе Фернадо Леон Атупанья

Группа: НПМбд-02-20

МОСКВА 2021 г.

Цель работы

Изучить основы программирования в оболочке операционной системы UNIX / Linux. Узнайте, как писать небольшие пакетные файлы.

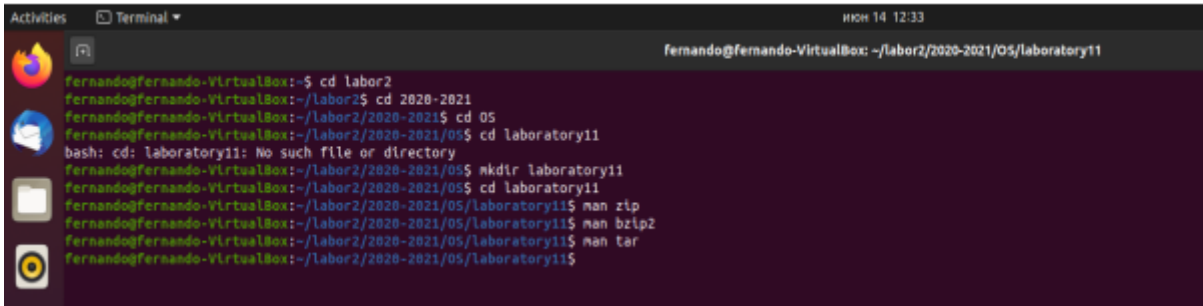
Задания

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет

количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

Выполнение лабораторной работы

1. Для начала я изучил команды архивации, используя команды «man zip», «man bzip2», «man tar» (рис. -@fig:001).



```
fernando@fernando-VirtualBox:~$ cd labor2
fernando@fernando-VirtualBox:~/labor2$ cd 2020-2021
fernando@fernando-VirtualBox:~/labor2/2020-2021$ cd 05
fernando@fernando-VirtualBox:~/labor2/2020-2021/05$ cd laboratory11
bash: cd: laboratory11: No such file or directory
fernando@fernando-VirtualBox:~/labor2/2020-2021/05$ mkdir laboratory11
fernando@fernando-VirtualBox:~/labor2/2020-2021/05$ cd laboratory11
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ man zip
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ man bzip2
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ man tar
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$
```

{ #fig:001 width=70% }

синтаксис команды zip для архивации файла (рис. -@fig:002):

zip [опции] [имя файла.zip] [файлы или папки, которые будем архивировать]

Синтаксис команды zip для разархивации/распаковки файла:

unzip [опции] [файл_архива.zip] [файлы] -x [исключить] -d [папка]



```
ZIP(1)
NAME
zip - package and compress (archive) files
SYNOPSIS
zip [-aABcdDeFFghjklLnOqrRSTuvVwXyzI!$] [--longoption ...] [-b path] [-m suffixes] [-t date] [-tt date] [zipfile [file ...]] [-x! list]
zipcloak (see separate man page)
zipnote (see separate man page)
zipsplit (see separate man page)
Note: Command line processing in zip has been changed to support long options and handle all options and arguments more consistently. Some old command lines that depend on command line inconsistencies may no longer work.
DESCRIPTION
zip is a compression and file packaging utility for Unix, VMS, MSDOS, OS/2, Windows 9x/NT/XP, Minix, Atari, Macintosh, Amiga, and Acorn RISC OS. It is analogous to a combination of the Unix commands tar(1) and compress(1) and is compatible with PKZIP (Phil Katz's ZIP for MSDOS systems).
A companion program (unzip(1)) unpacks zip archives. The zip and unzip(1) programs can work with archives produced by PKZIP (supporting most PKZIP features up to PKZIP version 4.6), and PKZIP and PKUNZIP can work with archives produced by zip (with some exceptions, notably streamed archives, but recent changes in the zip file standard may facilitate better compatibility). zip version 3.0 is compatible with PKZIP 2.04 and also supports the Zip64 extensions of PKZIP 4.5 which allow archives as well as files to exceed the previous 2 GB limit (4 GB in some cases). zip also now supports bzip2 compression if the bzip2 library is included when zip is compiled. Note that PKUNZIP 1.10 cannot extract files produced by PKZIP 2.04 or zip 3.0. You must use PKUNZIP 2.04g or unzip 5.0g (or later versions) to extract them.
See the EXAMPLES section at the bottom of this page for examples of some typical uses of zip.
Large Archives and Zip64. zip automatically uses the Zip64 extensions when files larger than 4 GB are added to an archive, an archive containing Zip64 entries is updated (if the resulting archive still needs Zip64), the size of the archive will exceed 4 GB, or when the number of entries in the archive will exceed about 64k. Zip64 is also used for archives streamed from standard input as the size of such archives are not known in advance, but the option -Pz- can be used to force zip to create PKZIP 2 compatible archives (as long as Zip64 extensions are not needed). You must use a PKZIP 4.5 compatible unzip, such as unzip 6.0 or later, to extract files using the Zip64 extensions.
In addition, streamed archives, entries encrypted with standard encryption, or split archives created with the pause option may not be compatible with PKZIP as data descriptors are used and PKZIP at the time of this writing does not support data descriptors (but recent changes in the PKWare published zip standard now include some support for the data descriptor format zip uses).
Mac OS X. Though previous Mac versions had their own zip port, zip supports Mac OS X as part of the Unix port and most Unix features apply. References to "MacOS" below generally refer to MacOS versions older than OS X. Support for some Mac OS features in the Unix Mac OS X port, such as resource forks, is expected in the next zip release.
For a brief help on zip and unzip, run each without specifying any parameters on the command line.
USE
The program is useful for packaging a set of files for distribution; for archiving files; and for saving disk space by temporarily compressing unused files or directories.
The zip program puts one or more compressed files into a single zip archive, along with information about the files (name, path, date, time of last modification, protection, and check information to verify file integrity). An entire directory structure can be packed into a zip archive with a single command. Compression ratios of 2:1 to 3:1 are common for text files. zip has one compression method (deflation) and can also store files without compression. (If bzip2 support is added, zip can also compress using bzip2 compression, but such entries require a reasonably modern unzip to decompress. When bzip2 compression is selected, it replaces deflation as the default method.) zip automatically chooses the better of the two (deflation or store or, if bzip2 is selected, bzip2 or store) for each file to be compressed.
Command format. The basic command format is
zip options archive [npath inpath ...]
where archive is a new or existing zip archive and inpath is a directory or file path optionally including wildcards. When given the name of an existing zip archive, zip will replace identically named entries in the zip archive (matching the relative names as stored in the archive) or add entries for new names. For example, if foo.zip exists and contains foo/file1 and foo/file2, and the directory foo contains the files foo/file1 and foo/file3, then:
zip -r foo.zip foo
Manual page zip(1) line 1 (press h for help or q to quit)
```

{ #fig:002 width=70% }

Синтаксис команды bzip2 для архивации файла (рис. -@fig:003):

bzip2 [опции] [имена файлов]

Синтаксис команды bzip2 для разархивации/распаковки файла:

bunzip2 [опции] [архивы.bz2]



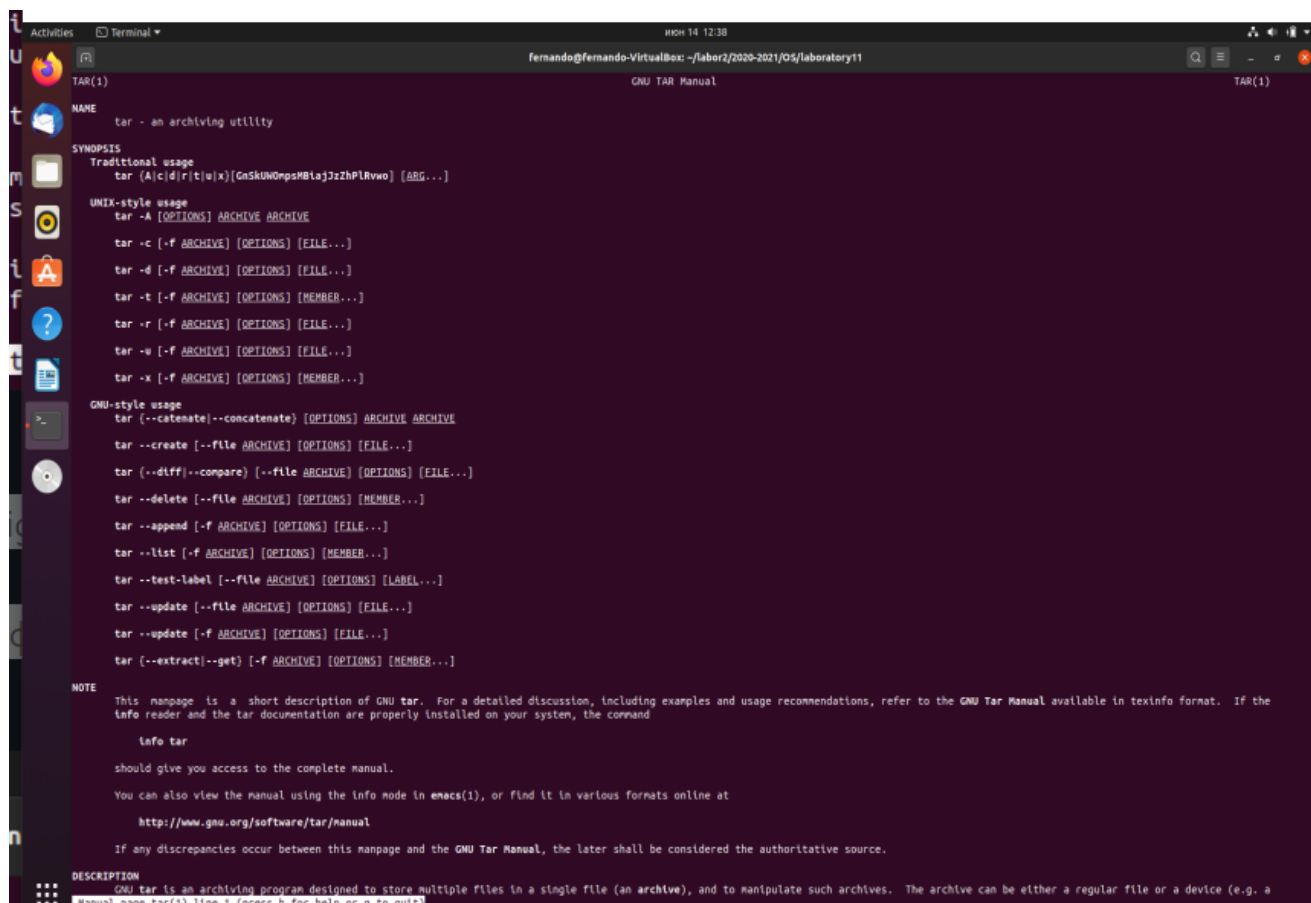
{ #fig:003 width=70% }

Синтаксис команды tar для архивации файла (рис. -@fig:004):

tar [опции] [архив.tar] [файлы_для_архивации]

Синтаксис команды tar для разархивации/распаковки файла:

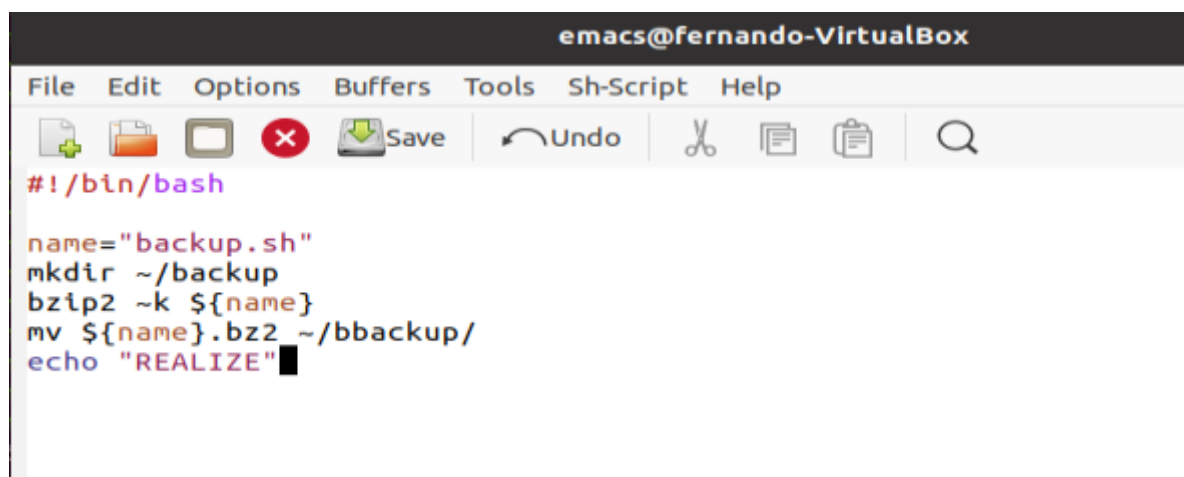
tar [опции] [архив.tar]



{ #fig:004 width=70% }

Далее я создал файл, в котором буду писать первый скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &»)

После этого я написал скрипт, который при запуске будет создавать резервные копии себя (т. е. файл, содержащий его исходный код) в другой каталог резервного копирования в моем домашнем каталоге. В этом случае файл должен быть заархивирован одним из картотек на выбор zip, bzip2 или tar (рис. - @рис.005). При написании скрипта я использовал архиватор bzip2.



{ #fig:005 width=70% }

Посмотрел его содержимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис. -@fig:006). Скрипт работает корректно

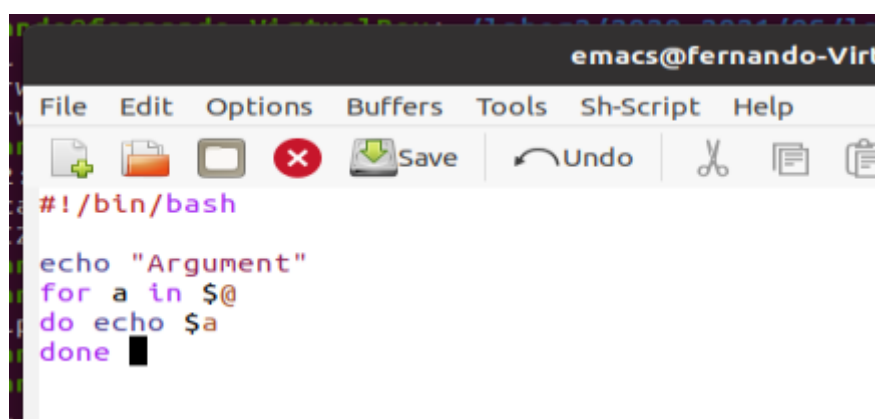
```
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ chmod +x *.sh~
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ bunzip2 -c backup.sh.bz2
#!/bin/bash

name="backup.sh"
mkdir ~/backup
bzip2 -k ${name}
mv ${name}.bz2 ~/bbackup/
echo "REALIZE"S
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$
```

{ #fig:006 width=70% }

2. Создал файл, в котором буду писать второй скрипт, и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch prog2.sh» и «emacs &»).

Написал пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов (рис. -@fig:007).



```
#!/bin/bash

echo "Argument"
for a in $@
do echo $a
done
```

{ #fig:007 width=70% }

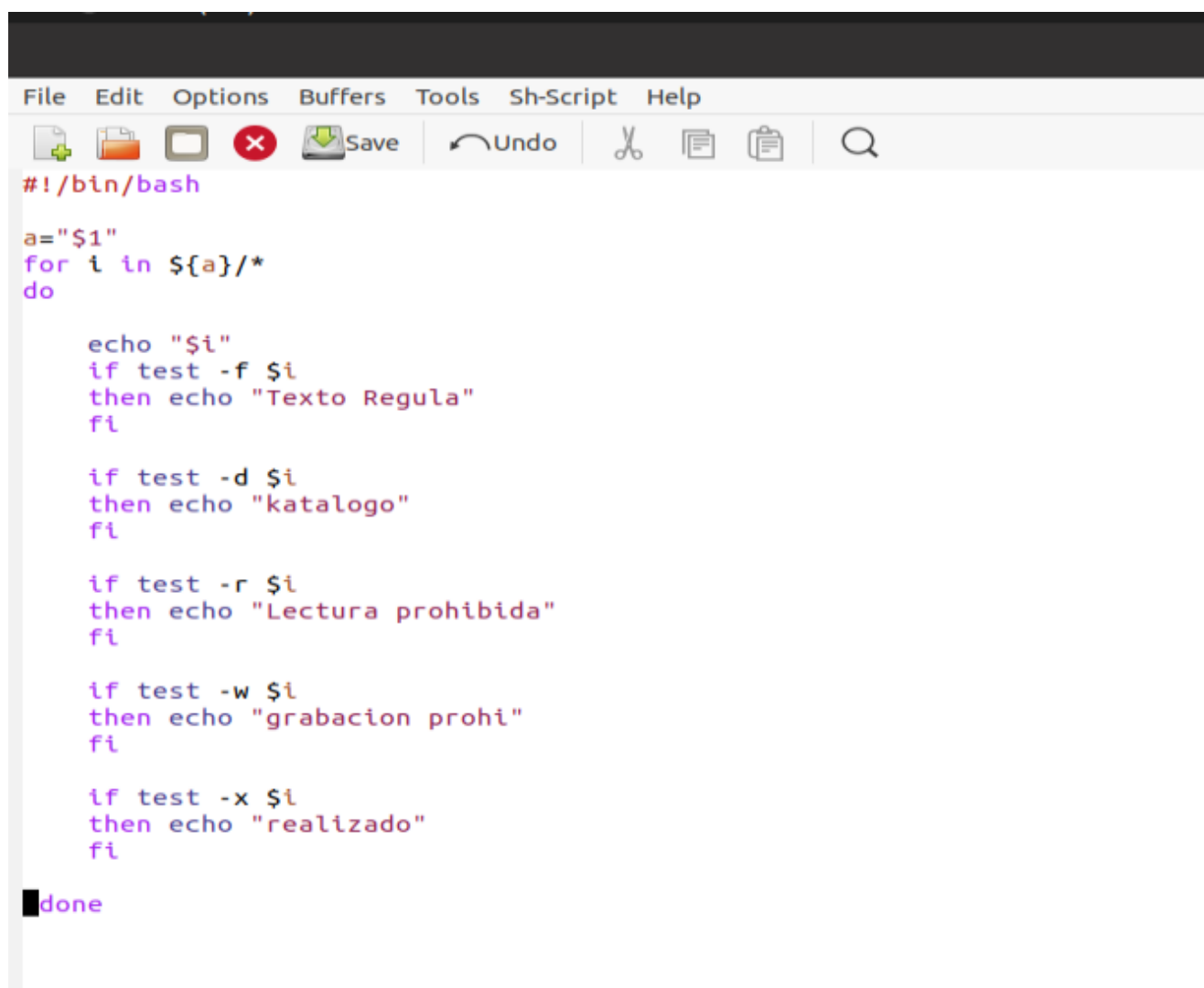
Проверил работу написанного скрипта (команды «./prog2.sh 1 2 3 4 5» и «./prog2.sh 1 2 3 »), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»). Скрипт работает корректно. (рис. -@fig:008).

```
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ chmod +x *.sh
[1]-  Done                  emacs
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ ls -l
total 8
-rwxrwxr-x 1 fernando fernando  0 июн 14 12:41 backup.sh~
-rwxrwxr-x 1 fernando fernando 125 июн 14 12:47 backup.sh.bz2
-rwxrwxr-x 1 fernando fernando  58 июн 14 13:02 prog2.sh
-rw-rw-r-- 1 fernando fernando  0 июн 14 13:00 prog2.sh~
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ ./prog2.sh 1 2 3 4 5
Argument
1
2
3
4
5
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$ ./prog2.sh 1 2 3 4
Argument
1
2
3
4
fernando@fernando-VirtualBox:~/labor2/2020-2021/05/laboratory11$
```

{ #fig:008 width=70% }

3. Создал файл, в котором буду писать третий скрипт, и открыла его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch ls.sh» и «emacs &»).

Написал командный файл – аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информацию о нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис. -@fig:009).

A screenshot of a terminal window with a menu bar (File, Edit, Options, Buffers, Tools, Sh-Script, Help) and a toolbar with icons for file operations and editing. The terminal displays a shell script being executed. The script starts with a shebang line, followed by a variable assignment and a for loop. Inside the loop, there are several if statements checking file types and permissions, each with a corresponding echo command. The script ends with a 'done' statement.

```
#!/bin/bash

a="$1"
for i in ${a}/*
do

    echo "$i"
    if test -f $i
    then echo "Texto Regula"
    fi

    if test -d $i
    then echo "katalogo"
    fi

    if test -r $i
    then echo "Lectura prohibida"
    fi

    if test -w $i
    then echo "grabacion prohi"
    fi

    if test -x $i
    then echo "realizado"
    fi

done
```

{ #fig:009 width=70% }

Далее проверил работу скрипта (команда «./progl.sh ~»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh») (рис. -@fig:010). Скрипт работает корректно.

```

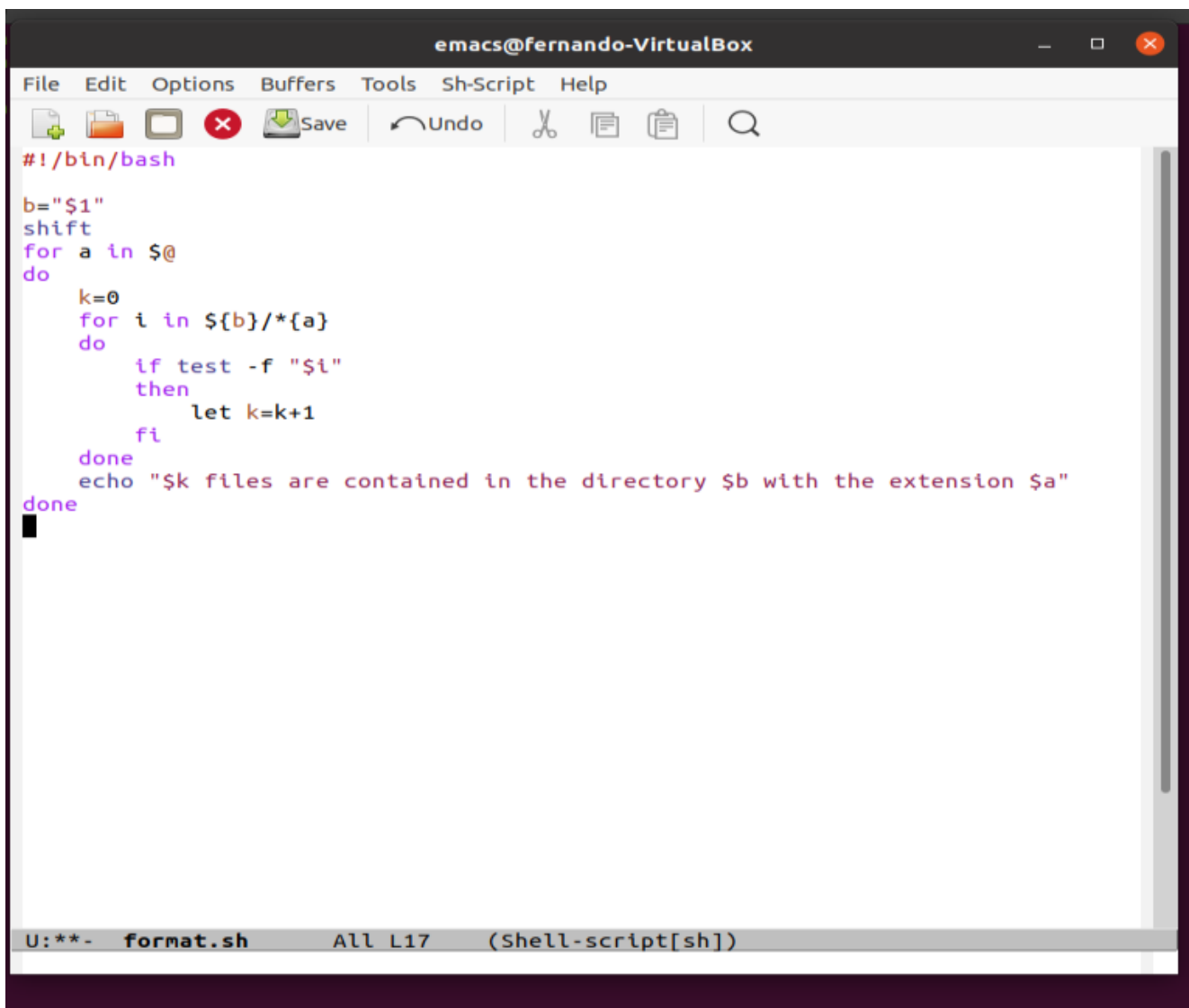
[2]- Done
fernando@fernando-VirtualBox: ~/Labor2/2020-2021/05/Laboratory11$ chmod +x *.sh
fernando@fernando-VirtualBox: ~/Labor2/2020-2021/05/Laboratory11$ ls -
01 australia conf.txt docs Downloads Images Labor2 monthly.00 Pictures program Templates Videos
abcd backup Desktop Documents file.txt lab2 monthly Music '~.play' Public text.txt work
fernando@fernando-VirtualBox: ~/Labor2/2020-2021/05/Laboratory11$ ./ls.sh ~
/home/fernando/01
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/abcd
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/australia
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/backup
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/conf.txt
texto Regula
lectura prohibida
grabacion prohi
/home/fernando/Desktop
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/docs
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/Documents
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/Downloads
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/file.txt
texto Regula
lectura prohibida
grabacion prohi
/home/fernando/images
katalogo
lectura prohibida
grabacion prohi
realizado
/home/fernando/lab2
katalogo
lectura prohibida
grabacion prohi

```

{ #fig:0010 width=70% }

4. Для четвертого скрипта также создал файл (команда «touch format.sh») и открыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда «emacs &»).

Написал командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки (рис. -@fig:011).

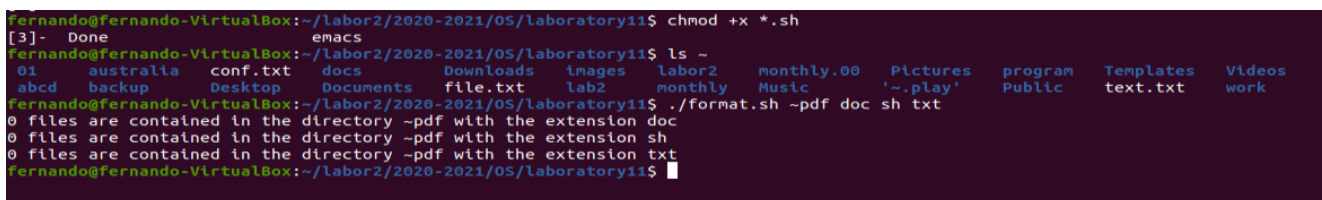


```
#!/bin/bash

b="$1"
shift
for a in $@
do
    k=0
    for i in ${b}/*{a}
    do
        if test -f "$i"
        then
            let k=k+1
        fi
    done
    echo "$k files are contained in the directory $b with the extension $a"
done
```

{ #fig:0011 width=70% }

Проверил работу написанного скрипта (команда «./format.sh ~ pdf sh txt doc»), предварительно добавив для него право на выполнение (команда «chmod +x *.sh»), а также создав дополнительные файлы с разными расширениями (команда «touch file.pdf file1.doc file2.doc») (рис. -@fig:012). Скрипт работает корректно.



```
fernando@fernando-VirtualBox:~/Labor2/2020-2021/05/Laboratory11$ chmod +x *.sh
[3]- Done emacs
fernando@fernando-VirtualBox:~/Labor2/2020-2021/05/Laboratory11$ ls ~
01_australia  conf.txt  docs  Downloads  images  labor2  monthly.00  Pictures  program  Templates  Videos
abcd  backup  Desktop  Documents  file.txt  lab2  monthly  Music  Public  text.txt  work
fernando@fernando-VirtualBox:~/Labor2/2020-2021/05/Laboratory11$ ./format.sh ~pdf doc sh txt
0 files are contained in the directory ~pdf with the extension doc
0 files are contained in the directory ~pdf with the extension sh
0 files are contained in the directory ~pdf with the extension txt
fernando@fernando-VirtualBox:~/Labor2/2020-2021/05/Laboratory11$
```

{ #fig:0012 width=70% }

Контрольные вопросы

1. 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 Сподобный 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).

2. 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.

3. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда «mark=/usr/andy/bin» присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов.

Значение, присвоенное некоторой переменной, может быть впоследствии использовано. Для этого в соответствующем месте командной строки должно быть употреблено имя этой переменной, которому предшествует метасимвол \$. Например, команда «mv afile \${mark}» переместит файл afile из текущего каталога в каталог с абсолютным полным именем /usr/andy/bin. Оболочка bash позволяет работать с массивами. Для создания массива используется команда set с флагом -A. За флагом следует имя переменной, а затем список значений, разделённых пробелами. Например, «set-A штаты Делавэр Мичиган " Нью-Джерси"»

Далее можно сделать добавление в массив, например, states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

4. Оболочка bash поддерживает встроенные арифметические функции.

Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение – это единичный терм (term), обычно целочисленный. Команда let берет два операнда и присваивает их переменной.

Команда read позволяет читать значения переменных со стандартного ввода:

«echo "Please enter Month and Day of Birth ?"»

«read mon day trash»

В переменные mon и day будут считаны соответствующие значения, введенные с клавиатуры, а переменная trash нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать её.

5. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение(*), целочисленное деление (/) и целочисленный остаток от деления (%).
6. В (()) можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать результат.
7. Стандартные переменные:
 - PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /. Если имя команды содержит хотя бы один символ /, то последовательность поиска, предписываемая значением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущего каталога.
 - PS1 и PS2: эти переменные предназначены для отображения промптера командного процессора. PS1 – это промптер командного процессора, по умолчанию его значение равно символу \$ или #. Если какая-то интерактивная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа >.
 - HOME: имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной.
 - IFS: последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line).
 - MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта).
 - TERM: тип используемого терминала.
 - LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему.
8. Такие символы, как ' < > * ? | \ " &, являются метасимволами и имеют для командного процессора специальный смысл.

9. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом.

Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , ". Например,

– `echo *` выведет на экран символ ,

– `echo ab'|'cd` выведет на экран строку `ab|*cd`.

10. Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: «`bash командный_файл [аргументы]`»

Чтобы не вводить каждый раз последовательности символов `bash`, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды «`chmod +x имя_файла`»

Теперь можно вызывать свой командный файл на выполнение, просто вводя его имя с терминала так, как будто он является выполняемой программой.

Командный процессор распознает, что в Вашем файле на самом деле хранится не выполняемая программа, а программа, написанная на языке программирования оболочки, и осуществит её интерпретацию.

11. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`.

12. Чтобы выяснить, является ли файл каталогом или обычным файлом, необходимо воспользоваться командами «`test -f [путь до файла]`» (для проверки, является ли обычным файлом) и «`test -d [путь до файла]`» (для проверки, является ли каталогом).

13. Команду «`set`» можно использовать для вывода списка переменных окружения. В системах Ubuntu и Debian команда «`set`» также выведет список функций командной оболочки после списка переменных командной оболочки. Поэтому для ознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «`set | more`».

Команда «`typeset`» предназначена для наложения ограничений на переменные.

Команду «`unset`» следует использовать для удаления переменной из окружения командной оболочки.

14. При вызове командного файла на выполнение параметры ему могут быть переданы точно таким же образом, как и выполняемой программе. С точки зрения командного файла эти параметры являются позиционными. Символ \$ является метасимволом процессора. Он используется, в частности, для

ссылки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использовании где-либо в командном файле комбинации символов \$i, где $0 < i < 10$, вместо неё будет осуществлена подстановка значения параметра с порядковым номером i, т. е. аргумента командного файла с порядковым номером i. Использование комбинации символов \$0 приводит к подстановке вместо неё имени данного командного файла.

15. Специальные переменные:

- \$* – отображается вся командная строка или параметры оболочки;
- \$? – код завершения последней выполненной команды;
- \$\$ – уникальный идентификатор процесса, в рамках которого выполняется командный процессор;
- \$! – номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда;
- \$- – значение флагов командного процессора;
- \${#} – *возвращает целое число – количество слов, которые были результатом \$;*
- \${#name} – возвращает целое значение длины строки в переменной name;
- \${name[n]} – обращение к n-му элементу массива;
- \${name[*]} – перечисляет все элементы массива, разделённые пробелом;
- \${name[@]} – то же самое, но позволяет учитывать символы пробелы в самих переменных;
- \${name:-value} – если значение переменной name не определено, то оно будет заменено на указанное value;
- \${name:value} – проверяется факт существования переменной;
- \${name=value} – если name не определено, то ему присваивается значение value;
- \${name?value} – останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке;
- \${name+value} – это выражение работает противоположно \${name-value}. Если переменная определена, то подставляется value;
- \${name#pattern} – представляет значение переменной name с удалённым самым коротким левым образцом (pattern);
- \${#name[*]} и \${#name[@]} – эти выражения возвращают количество элементов в массиве name.

Выводы

Выполняя эту лабораторную работу, я изучил основы программирования в оболочке операционной системы UNIX/Linux и научился писать небольшие пакетные файлы.