Отчёт по лабораторной работе №10

Дисциплина: Операционные системы

Елизавета Андреевна Алмазова

Содержание

Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX/Linux, научиться писать небольшие командные файлы.

Задание

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.

Теоретическое введение

Командные оболочки

Командная оболочка - 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.

Переменные и массивы в языке программирования bash

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. 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, 1 set -A states Delaware Michigan “New Jersey”. Далее можно сделать добавление в массив, например,states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

Арифметические вычисления

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.

Целые числа можно записывать как последовательность цифр или в любом базовом формате типа radix#number, где radix (основание системы счисления) — любое число не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (*), целочисленное деление (/) и целочисленный остаток от деления (%).

Команда let берет два операнда и присваивает их переменной. Положительным моментом команды let можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа let sum=x+7, и let будет искать переменную x и добавлять к ней 7. Команда let также расширяет другие выражения let, если они заключены в двойные круглые скобки. Таким способом вы можете создавать довольно сложные выражения. Команда let не ограничена простыми арифметическими выражениями.

Подобно C оболочка bash может присваивать переменной любое значение, а произвольное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина». Для сказанного является выполнение некоторого действия, одновременно декрементируя некоторое значение.

Наиболее распространённым является сокращение, избавляющееся от слова let в программах оболочек. Если объявить переменные целыми значениями, то любое присвоение автоматически будет трактоваться как арифметическое действие. Если использовать typeset -i для объявления и присвоения переменной, то при последующем её применении она станет целой. Также можно использовать ключевое слово integer (псевдоним для typeset -i) и объявлять таким образом переменные целыми. Выражения типа x=y+z будет восприниматься в это случае как арифметические.

Команда read позволяет читать значения переменных со стандартного ввода

Стандартные переменные

Переменные PS1 и PS2 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 >. Другие стандартные переменные:

- 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.

В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

Метасимволы и экранирование

Такие символы, как ’ < > * ? | ” &, являются метасимволами и имеют для командного процессора специальный смысл. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом.

Командные файлы и их параметры

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.

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 командного 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.

Getopts и флаги

Весьма необходимой при программировании является команда getopts, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Флаги — это опции командной строки, обычно помеченные знаком минус; Например, для команды ls флагом может являться -F. Иногда флаги имеют аргументы, связанные с ними. Программы интерпретируют

флаги, соответствующим образом изменяя своё поведение. 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.

Управление последовательностью действий

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.

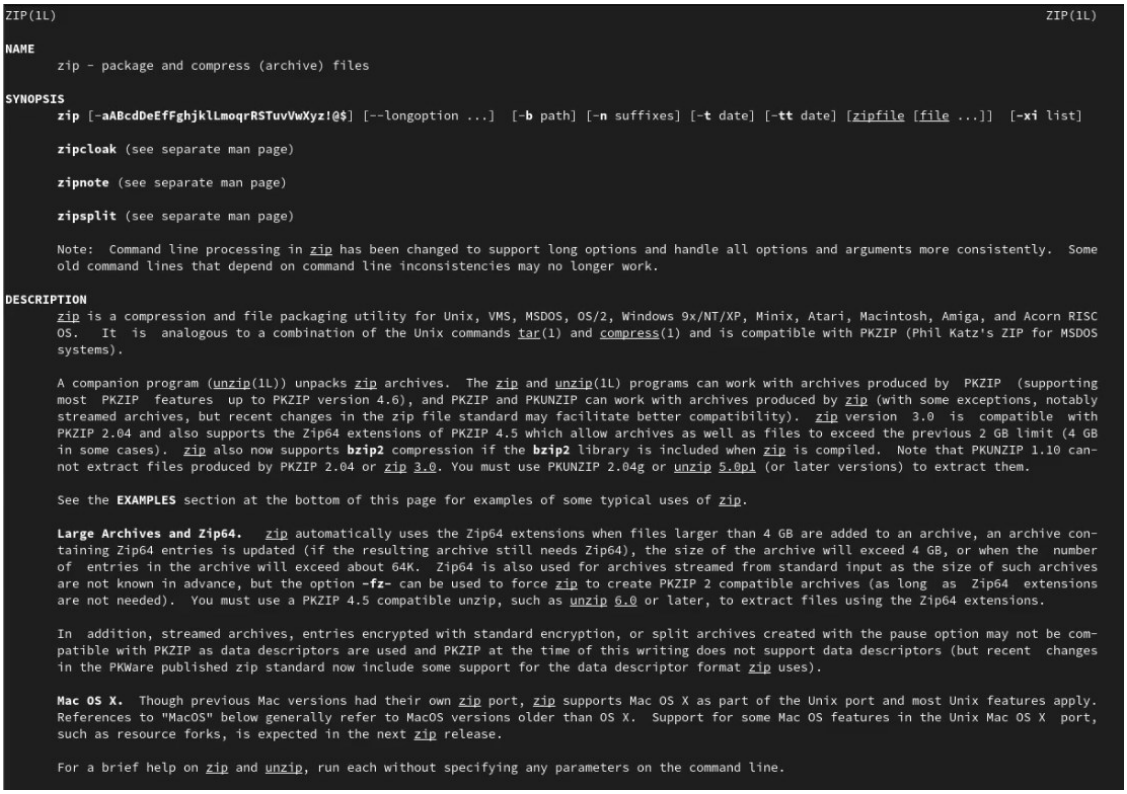
При каждом следующем выполнении оператора цикла for переменная имя принимает следующее значение из списка значений, задаваемых списком список -значений. Вообще говоря, список-значений является необязательным. При его отсутствии оператор цикла for выполняется для всех позиционных параметров или, иначе говоря, аргументов.

Оператор выбора case реализует возможность ветвления на произвольное число ветвей.

Выполнение условного оператора if сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово if. Затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово then. Фраза elif проверяется в том случае, когда предыдущая проверка была ложной. Строка, содержащая служебное слово else, является необязательной. Если она присутствует, то последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово else, будет выполнена только при условии, что последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово if или elif, возвращает ненулевой код завершения (ложь).

Выполнение лабораторной работы

1. С помощью команды man изучала справку по командам архивации: man zip, man bzip2, man tar (рис.1,2,3).



ZIP(1L) ZIP(1L)

NAME

zip - package and compress (archive) files

SYNOPSIS

zip [-**aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$**] [--longoption ...] [-**b** path] [-**n** suffixes] [-**t** date] [-**tt** date] [zipfile [file ...]] [-**xi** 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(1L)) unpacks zip archives. The zip and unzip(1L) 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 5.0p1 (or later versions) to extract them.

See the **EXAMPLES** section at the bottom of this page for examples of some typical uses of zip.

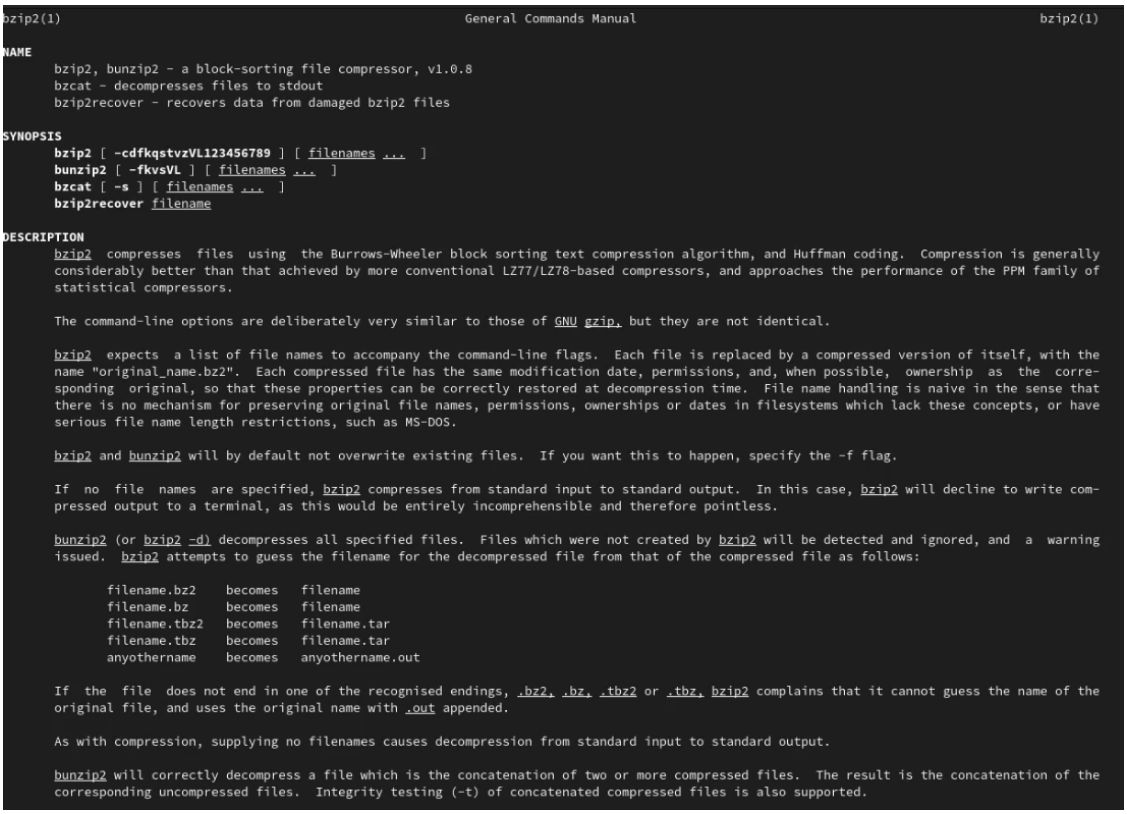
Large Archives and Zip64. zip automatically uses the Zip64 extensions when files larger than 4 GB are added to an archive, an archive containing Zip64 entries is updated (if the resulting archive still needs Zip64), the size of the archive will exceed 4 GB, or when the number of entries in the archive 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 **-fz-** 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.

Рисунок 1 - man zip.



bzip2(1) General Commands Manual bzip2(1)

NAME

bzip2, bunzip2 - a block-sorting file compressor, v1.0.8
bzcat - decompresses files to stdout
bzip2recover - recovers data from damaged bzip2 files

SYNOPSIS

bzip2 [**-cdfkqstvzVL123456789**] [filenames ...]
bunzip2 [**-fkvsVL**] [filenames ...]
bzcat [**-s**] [filenames ...]
bzip2recover filename

DESCRIPTION

bzip2 compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Compression is generally considerably better than that achieved by more conventional LZ77/LZ78-based compressors, and approaches the performance of the PPM family of statistical compressors.

The command-line options are deliberately very similar to those of GNU gzip, but they are not identical.

bzip2 expects a list of file names to accompany the command-line flags. Each file is replaced by a compressed version of itself, with the name "original_name.bz2". Each compressed file has the same modification date, permissions, and, when possible, ownership as the corresponding original, so that these properties can be correctly restored at decompression time. File name handling is naive in the sense that there is no mechanism for preserving original file names, permissions, ownerships or dates in filesystems which lack these concepts, or have serious file name length restrictions, such as MS-DOS.

bzip2 and bunzip2 will by default not overwrite existing files. If you want this to happen, specify the -f flag.

If no file names are specified, bzip2 compresses from standard input to standard output. In this case, bzip2 will decline to write compressed output to a terminal, as this would be entirely incomprehensible and therefore pointless.

bunzip2 (or bzip2 -d) decompresses all specified files. Files which were not created by bzip2 will be detected and ignored, and a warning issued. bzip2 attempts to guess the filename for the decompressed file from that of the compressed file as follows:

filename.bz2	becomes	filename
filename.bz	becomes	filename
filename.tbz2	becomes	filename.tar
filename.tbz	becomes	filename.tar
anyothername	becomes	anyothername.out

If the file does not end in one of the recognised endings, .bz2, .bz, .tbz2 or .tbz, bzip2 complains that it cannot guess the name of the original file, and uses the original name with .out appended.

As with compression, supplying no filenames causes decompression from standard input to standard output.

bunzip2 will correctly decompress a file which is the concatenation of two or more compressed files. The result is the concatenation of the corresponding uncompressed files. Integrity testing (-t) of concatenated compressed files is also supported.

Рисунок 2 - man bzip2.

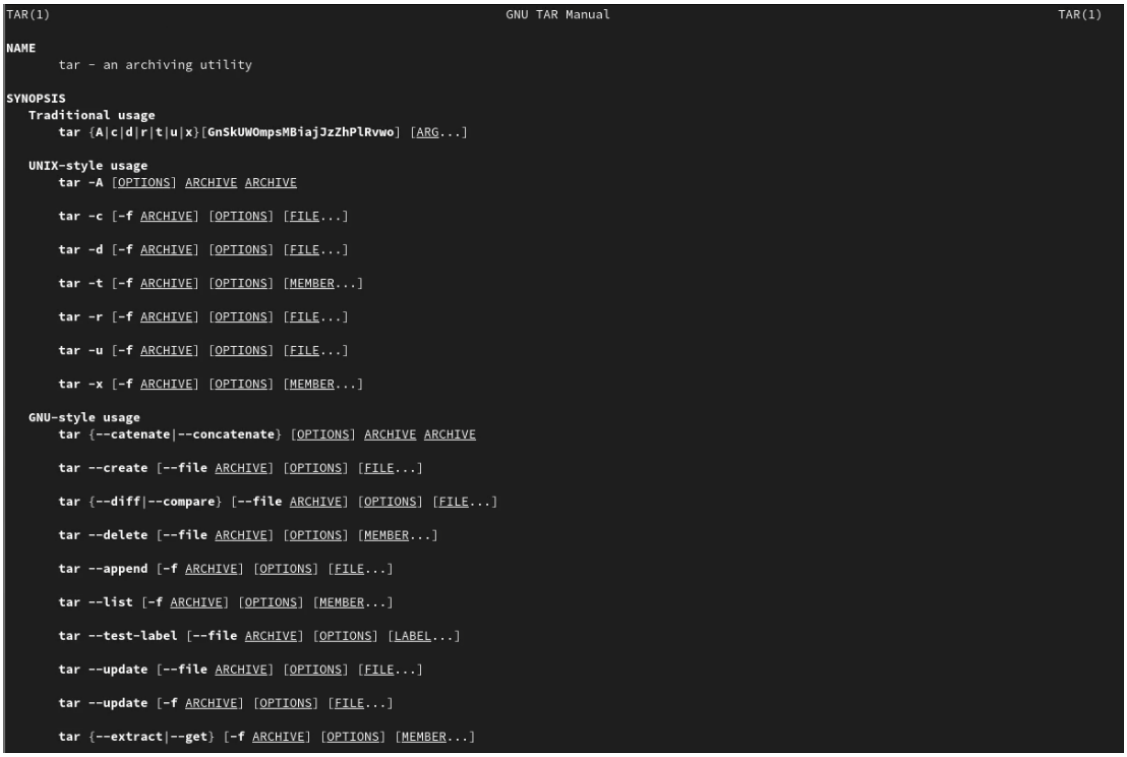


Рисунок 3 - man tar.

2. Написала скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моем домашнем каталоге, при этом файл архивируется bzip2 (mcedit backup.sh). С помощью команды chmod +x *.sh сделала файл исполняемым и проверила работу скрипта с помощью ./backup.sh (рис.4,5).

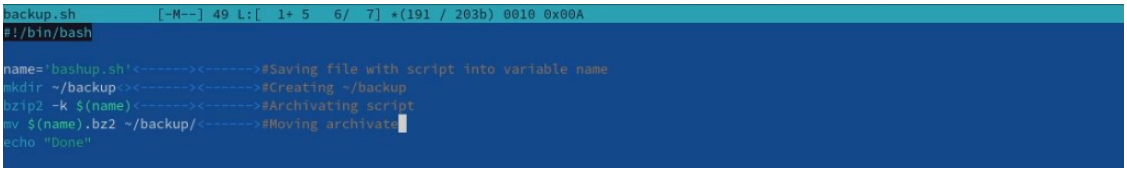


Рисунок 4 - Редактирование файла backup.sh.

Рисунок 5 - Результат работы скрипта.

3. Написала пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять: скрипт может последовательно распечатывать значения всех переданных аргументов (mcedit print.sh). С помощью команды chmod +x *.sh сделала файл исполняемым и проверила работу скрипта с помощью ./print.sh (рис.6).

Рисунок 6 - Результат работы скрипта.

4. Написала командный файл — аналог команды ls (без использования самой этой команды и команды dir), чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога (mcedit ls.sh). С помощью команды chmod +x *.sh сделала файл исполняемым и проверила работу скрипта с помощью ./ls.sh (рис.7,8).

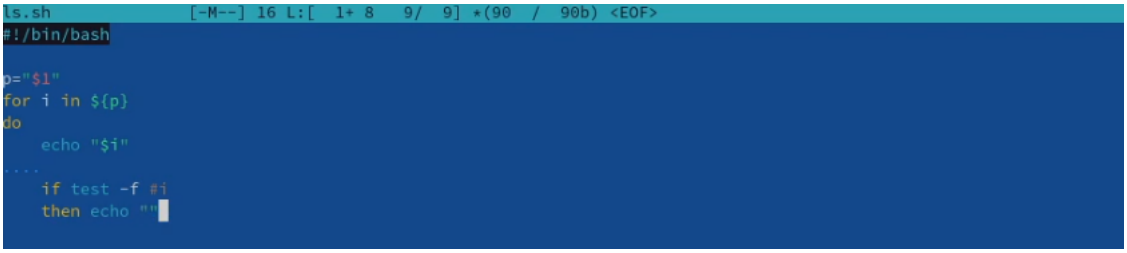


Рисунок 7 - Редактирование файла ls.sh.

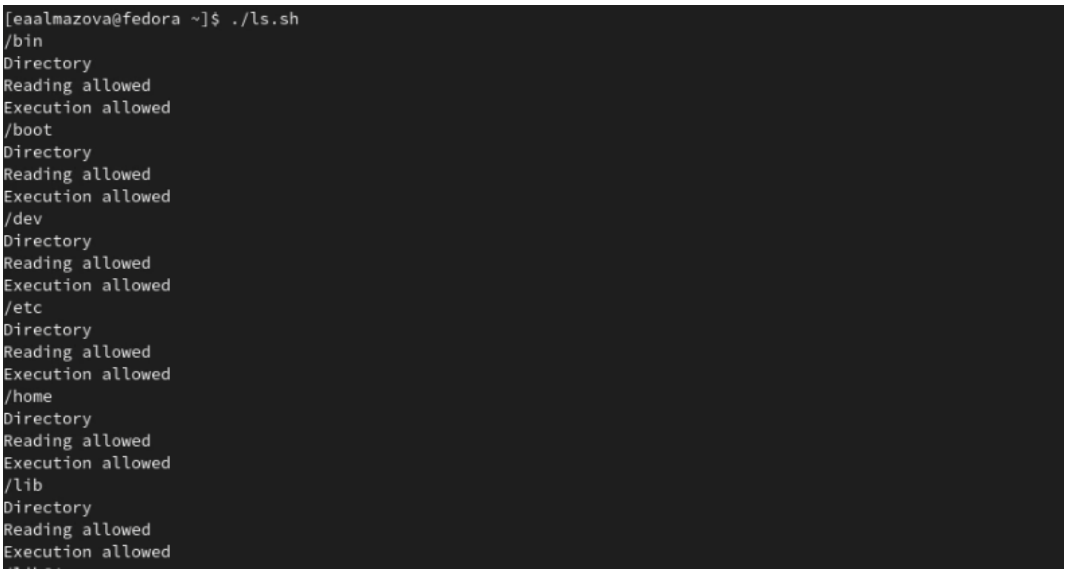


Рисунок 8 - Результат работы скрипта.

5. Написала 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 (mcedit format.sh). Путь к директории также

передаётся в виде аргумента командной строки. С помощью команды chmod +x *.sh сделала файл исполняемым и проверила работу скрипта с помощью ./format.sh (рис.9).

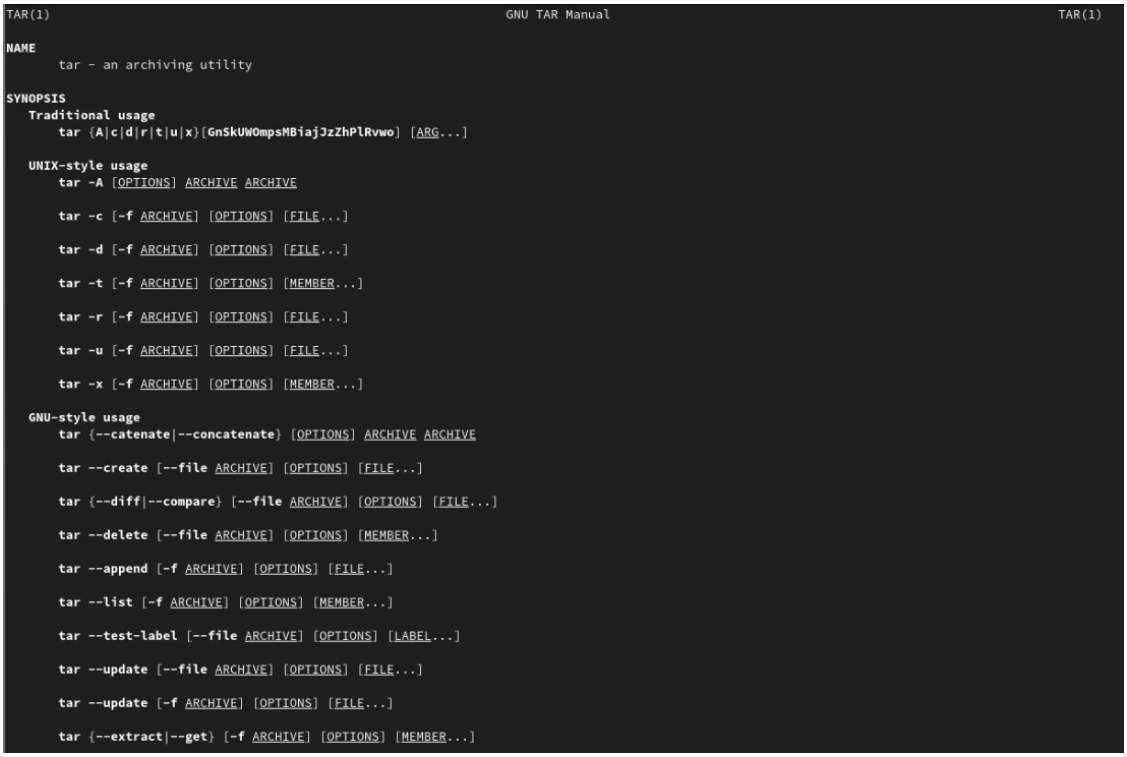


Рисунок 9 - Подсчет файлов домашнего каталога с расширениями pdf, txt, sh.

Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux, научилась писать небольшие командные файлы.

Ответы на контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командная оболочка - 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).

2. Что такое POSIX?

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. Как определяются переменные и массивы в языке программирования bash?

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. 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, 1 set -A states Delaware Michigan “New Jersey”. Далее можно сделать добавление в массив, например,states[49]=Alaska. Индексация массивов начинается с нулевого элемента.

4. Каково назначение операторов let и read?

Команда let берет два операнда и присваивает их переменной. Команда read позволяет читать значения переменных со стандартного ввода.

5. Какие арифметические операции можно применять в языке программирования bash?

Сложение, вычитание, умножение, деление, сдвиг влево и право на некоторое количество бит, сравнение и другие.

6. Что означает операция (( ))?

Для облегчения программирования можно записывать условия оболочки bash в двойные скобки.

7. Какие стандартные имена переменных Вам известны?

PS1, PS2, HOME, IFS, MAIL, TERM, LOGNAME.

8. Что такое метасимволы?

Такие символы, как ’ < > * ? | ” &, являются метасимволами и имеют для командного процессора специальный смысл.

9. Как экранировать метасимволы?

Экранирование может быть осуществлено с помощью предшествующего метасимволу символа , который, в свою очередь, является метасимволом.

10. Как создавать и запускать командные файлы?

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. Как определяются функции в языке программирования bash?

Функция – это объединение группы команд. Для этого существует ключевое слово – function, после которого следует имя функции и список команд, заключённых в фигурные скобки.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом?

С помощью команд test –d и test –f соответственно.

13. Каково назначение команд set, typeset и unset?

Set используется для вывода списка переменных окружения, typeset для наложения ограничений на переменные, unset для удаления переменной из окружения командной оболочки.

14. Как передаются параметры в командные файлы?

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 командного 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. Назовите специальные переменные языка bash и их назначение

- $* — 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.