**Отчёт по лабораторной работе №11**

### Дисциплина: Операционные системы

## Джеффри Родригес Сантос

**Содержание**

1. [Цель работы 5](#_TOC_250004)
2. [Задание 6](#_TOC_250003)
3. [Выполнение лабораторной работы 7](#_TOC_250002)
4. [Контрольные вопросы 17](#_TOC_250001)
5. [Выводы 23](#_TOC_250000)

**Список таблиц**

# Цель работы

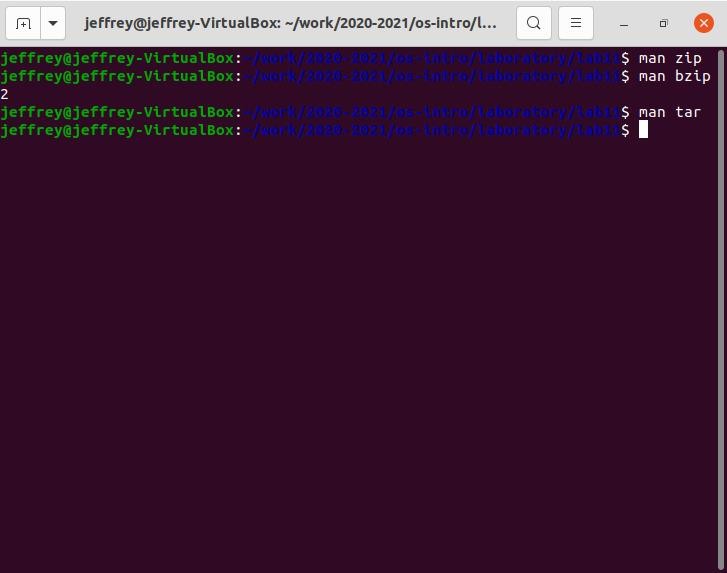
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

# Задание

1. Написать скрипт, который при запуске будет делать резервную копию са- мого 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 произволь- ное 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 команд- ной 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.

# Выполнениелабораторнойработы

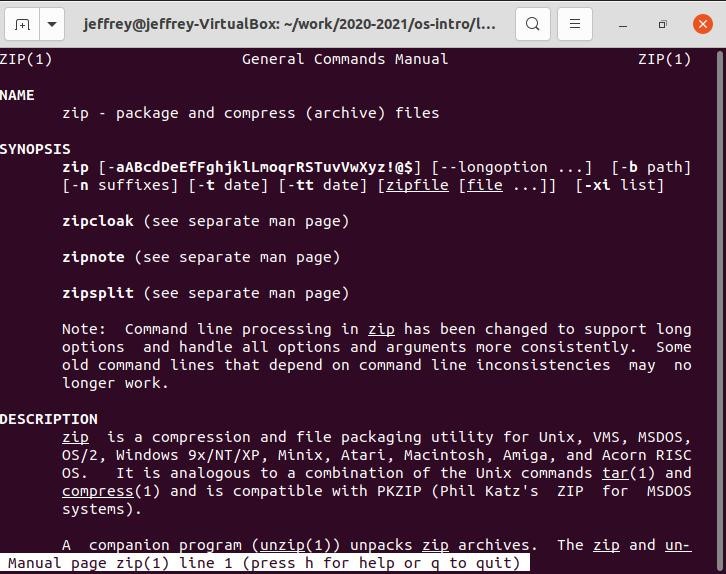
1. Для начала я изучил команды архивации, используя команды «man zip»,«man bzip2», «man tar» (рис. -fig. 3.1).



Синтаксис команды zip для архивации файла (рис. -fig. 3.2):

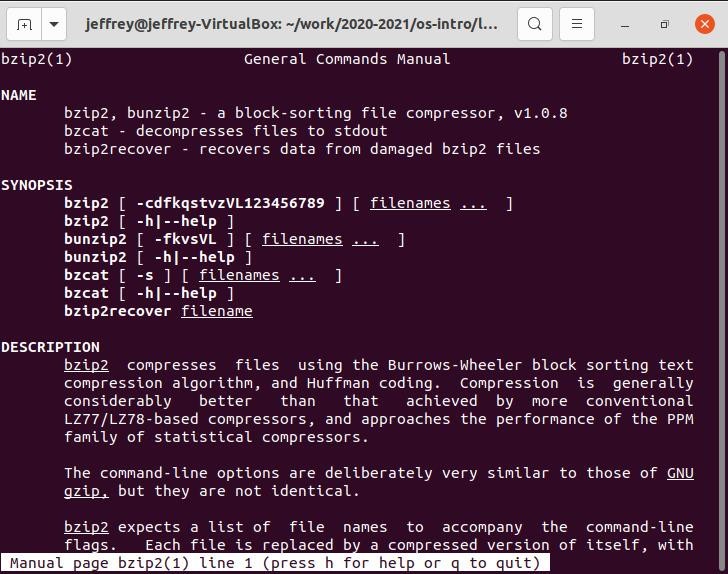
zip [опции] [имя файла.zip] [файлы или папки, которые будемархивировать] Синтаксис команды zip для разархивации/распаковки файла:

unzip [опции] [файл\_архива.zip] [файлы] -x [исключить] -d [папка]

Синтаксис команды zip

Синтаксис команды bzip2 для архивации файла (рис. -fig. 3.3): bzip2 [опции] [имена файлов]Синтаксис команды bzip2 для разархивации/распаковки файла: bunzip2 [опции] [архивы.bz2].

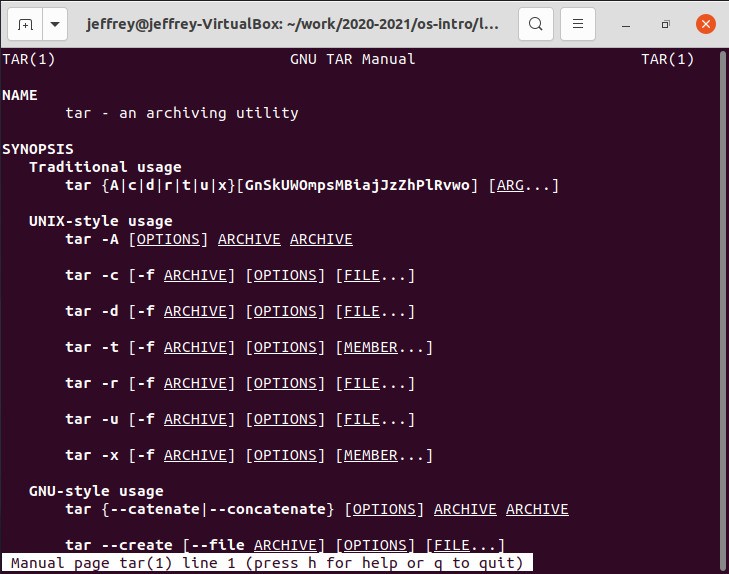
Рис. 3.2:



#### Рис. 3.3: Синтаксис команды bzip2

#### Синтаксис команды tar для архивации файла (рис. -fig. 3.4): tar [опции] [архив.tar] [файлы\_для\_архивации]

#### Синтаксис команды tar для разархивации/распаковки файла: tar [опции] [архив.tar].



#### Рис. 3.4: Синтаксис команды tar

#### Далее я создал файл, в котором буду писать первый скрипт, и открыл его в ре- дакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch backup.sh» и «emacs &») (рис. -fig. 3.5).

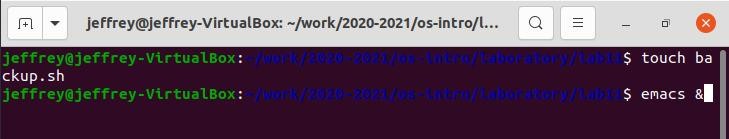


Рис. 3.5: создал файл для первого скрипта

После написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директориюbackupвмоёмдомашнемкаталоге. При этом файлдолженархиви- роваться одним из архиваторов на выбор zip, bzip2 или tar (рис. -fig. 3.6). При написании скрипта использовал архиватор bzip2.

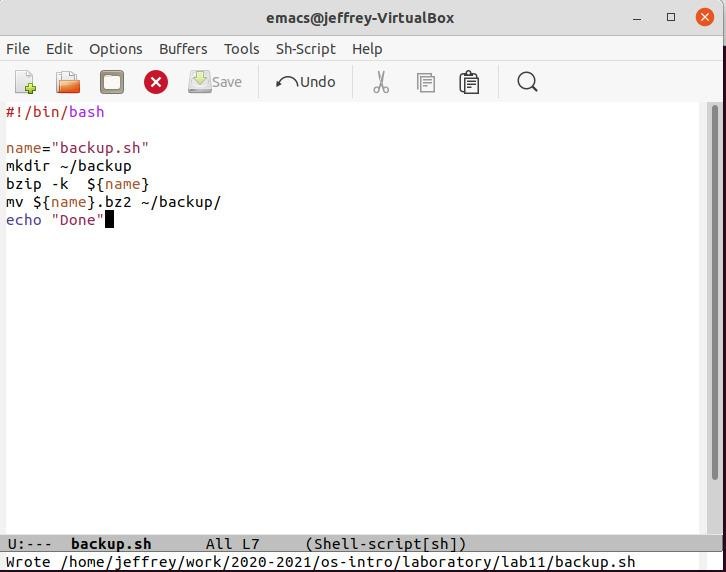


Рис. 3.6: Написал первый скрипт

Проверил работу скрипта (команда «./backup.sh»), предварительно добавив для него право на выполнение (команда «chmod +x .sh»). Проверил, появился ли каталог backup/, перейдя в него (команда «cd backup/»), посмотрел его содер- жимое (команда «ls») и просмотрел содержимое архива (команда «bunzip2 -c backup.sh.bz2») (рис. -fig. 3.7) (рис. -fig. 3.8). Скрипт работает корректно.

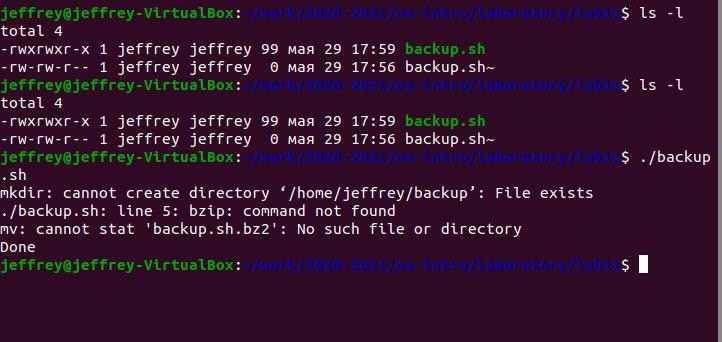


Рис. 3.7: Проверил первый скрипт

* 1. 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 &») (рис. -fig. 3.9).

Рис. 3.9: Создал файл для второго скрипта

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. 3.10).

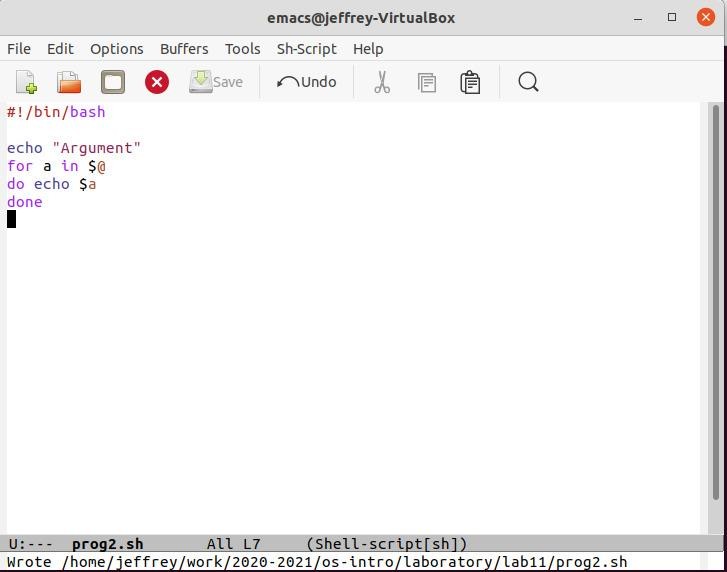


Рис. 3.10: Написал второй скрипт

Проверил работу написанного скрипта (команды «./prog2.sh 1 2 3 4 5»

и«./prog2.sh 1 2 3 4 5 6 7 8 9 10 11 12»), предварительно добавив для него право навыполнение (команда «chmod +x .sh»). Вводил аргументы, количество которых меньше 10 и больше 10 (рис. -fig. 3.11). Скрипт работает корректно.

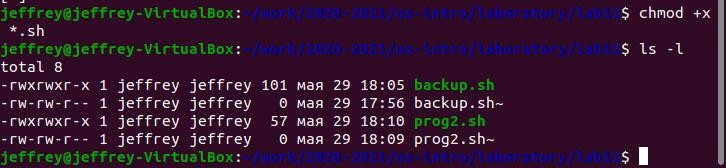


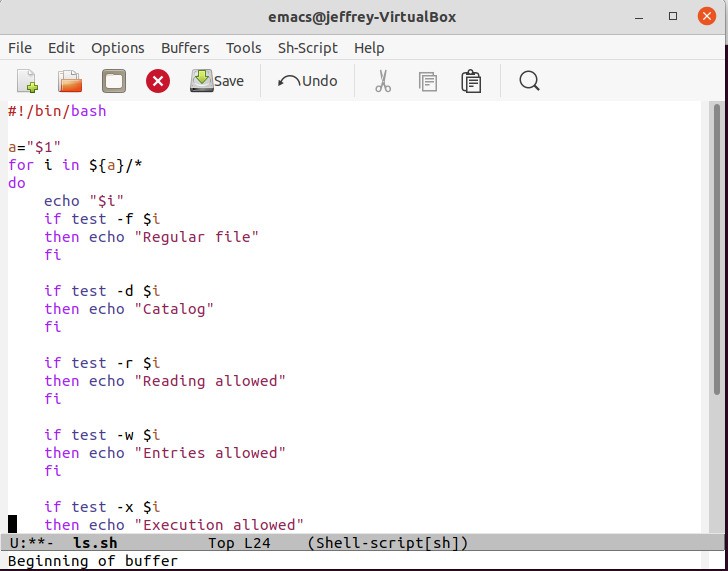


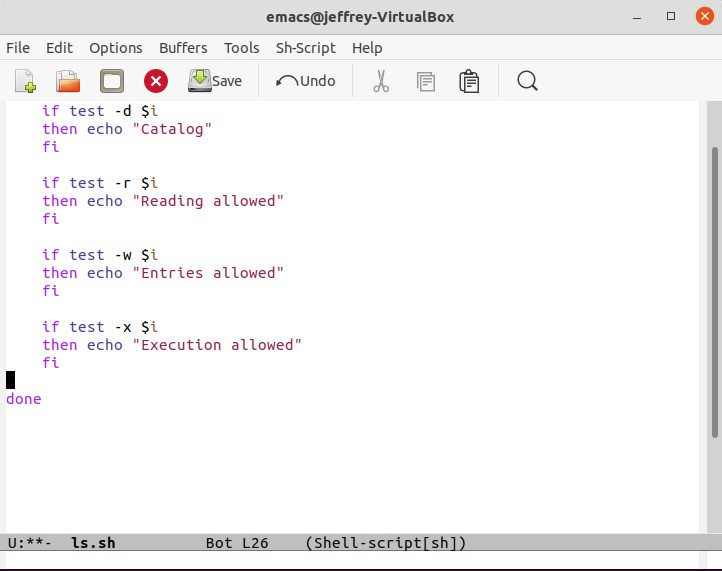
Рис. 3.11: проверил второй скрипт

* 1. Создал файл, в котором буду писать третий скрипт, и открыла его вредак- торе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команды «touch ls.sh» и«emacs &») (рис. -fig. 3.12).

Рис. 3.12: Создал файл для третьего скрипта

Написал командный файл − аналог команды ls (без использования самой этой команды и команды dir). Он должен выдавать информациюо нужном каталоге и выводить информацию о возможностях доступа к файлам этого каталога (рис.-fig. 3.13).





Далее проверил работу скрипта (команда «./progls.sh ~»), предварительно до- бавив для него право на выполнение (команда «chmod +x .sh») (рис. -fig. 3.14). Скрипт работает корректно.

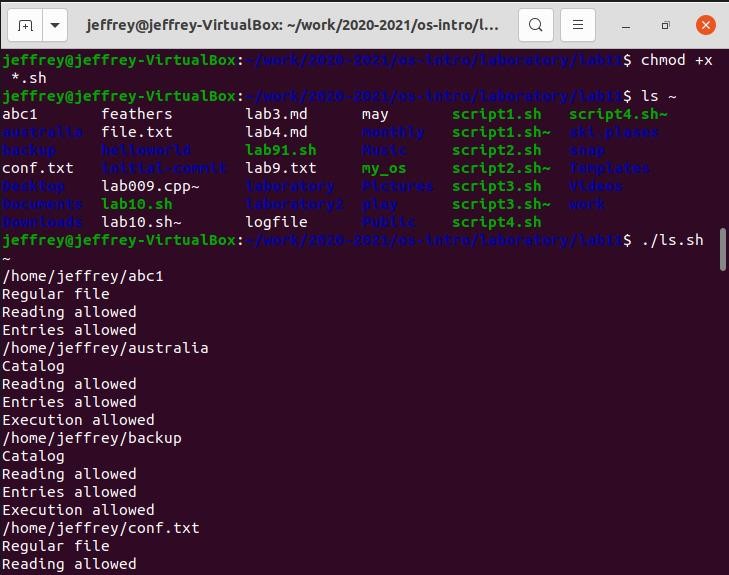


Рис. 3.14: Проверил третий скрипт

* 1. Для четвертого скрипта также создал файл (команда «touch format.sh») и от- крыл его в редакторе emacs, используя клавиши «Ctrl-x» и «Ctrl-f» (команда«emacs &») (рис. -fig. 3.15).



Рис. 3.15: Создал файл для четвёртого скрипта

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. 3.16).

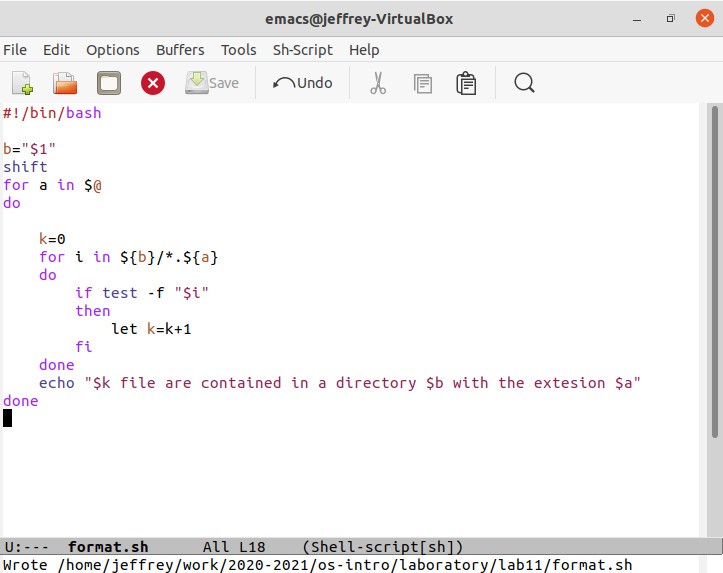


Рис. 3.16: Написал четвёртый скрипт

Проверилработунаписанногоскрипта(команда«./format.sh~pdfshtxtdoc»), 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. 3.17). Скрипт работает корректно.

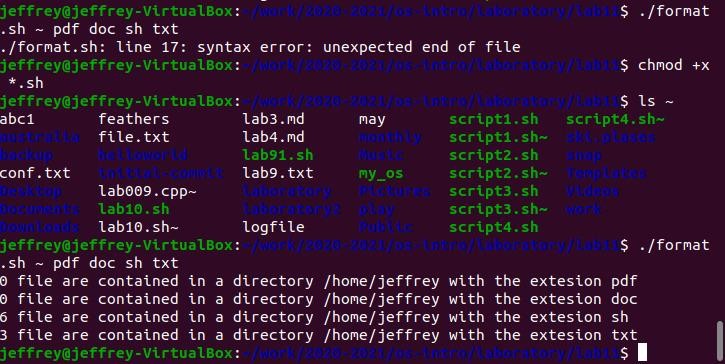


Рис. 3.17: Проверил четвёртый скрипт

* 1. 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;
  + С-оболочка (или csh) − надстройка на оболочку Борна, использующая Спо- добный синтаксис команд с возможностью сохранения истории выполне- ния команд;
  + оболочка Корна (или ksh) − напоминает оболочку С, но операторы управле- ния программой совместимы с операторами оболочки Борна;
  + BASH − сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

1. POSIX (Portable Operating System Interface for Computer Environments) − набор стандартов описания интерфейсов взаимодействия операционной системы и прикладныхпрограмм.

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.

1. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присво- ить 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 states Delaware Michigan “New Jersey”»

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

1. 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 вводда:

«echo “Please enter Month and Day of Birth?” »

«read mon day trash»

В переменные mon и day будут считаны соответствующие значения, вве- дённые с клавиатуры, а переменная trash нужна для того, чтобы отобрать всю избыточно введённую информацию и игнорировать её.

1. В языке программирования bash можно применять такие арифметические операции как сложение (+), вычитание (-), умножение (), целочисленное деление (/) и целочисленный остаток от деления (%).
2. В (()) можно записывать условия оболочки bash, а также внутри двойных скобок можно вычислять арифметические выражения и возвращать резуль- тат.
3. Стандартные переменные:
   * PATH: значением данной переменной является список каталогов, в которых командный процессор осуществляет поиск программы или команды, ука- занной 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 зна- чением переменной PATH, нарушается. В этом случае в зависимости от того, является имя команды абсолютным или относительным, поиск начинается соответственно от корневого или текущегокаталога.
   * 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 интерактив- ная программа, запущенная командным процессором, требует ввода, то используется промптер PS2. Он по умолчанию имеет значение символа>.
   * HOME: имядомашнего каталога пользователя. Есликоманда 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 промпте- ра проверяет содержимое файла, имя которого указано вэтой 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: содержитрегистрационное имя пользователя, котороеустанав- ливается автоматически при входе в систему.

1. Такие символы, как ’ <>? |” &, являются метасимволами и имеют для командного процессора специальный смысл.
2. 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 одинар- ные 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.

1. 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, необхо- димо 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 еёинтерпретацию.

1. Группу команд можно объединить в функцию. Для этого существует ключе-

вое слово function, послекоторого следуетимяфункциии списоккоманд, заключённых в фигурные скобки. Удалить функцию можно с помощью команды unset c флагом-f.

1. Чтобы выяснить, является ли файл каталогом или обычным файлом, необ- ходимо воспользоваться командами «test -f [путь до файла]» (для проверки, является ли обычным файлом) и «test -d [путь до файла]» (для проверки, является ли каталогом).
2. 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. Поэтомудляознакомления со всеми элементами списка переменных окружения при работе с данными системами рекомендуется использовать команду «set | more».

Команда «typeset» предназначена для наложения ограничений на переменные.

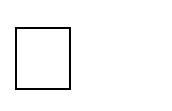
Команду «unset» следует использовать для удаления переменной из окружения командной оболочки.

1. 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 метасиволом процессора. Он используется, вчастности, дляссыл- ки на параметры, точнее, для получения их значений в командном файле. В командный файл можно передать до девяти параметров. При использова- нии где-либо в командном файле комбинации символов $i, где 0 <i <10, вместо неё будет осуществлена подстановка значения параметра с поряд- ковым номером i, т. е. аргумента командного файла с порядковым номером

i. Использование комбинации символов $0 приводит к подстановкевместо неё имени данного командного файла.

1. 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 командногопроцессора;
  + ${#*} − возвращает целое число − количество слов, которые были результатом*

*$*;

* + ${#name} − возвращает целоезначениедлиныстрокивпеременной name;
  + ${name[n]} − обращение к n-му элементу массива;
  + ${name[]} − перечисляет все элементы массива, разделённые пробелом;
  + ${name [@]} −тожесамое,нопозволяетучитыватьсимволыпробелы всамих 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 опре- делено, 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.

**5.Выводы**

В ходе выполнения данной лабораторной работы я изучил основы программи- рования в оболочке ОС UNIX/Linux и научился писать небольшие командные файлы.