Лабораторная работа №11 по предмету Операционные системы

### Группа НПМбв-01-19

## Бондаренко Артем Федорович

Содержание

[Цель работы](#_bookmark0) 5

[Задание](#_bookmark1) 6

[Выполнение лабораторной работы](#_bookmark2) 7

[Выводы](#_bookmark12) 13

[Ответы на контрольные вопросы](#_bookmark13) 14

Список иллюстраций

1. [Изучение способа использования команд архивации](#_bookmark3) 7
2. [Скрипт, который при запуске будет делать резервную копию самого себя](#_bookmark4) 8
3. [Результат выполнения скрипта](#_bookmark5) 9
4. [Командный файл, обрабатывающий любое произвольное число аргу-](#_bookmark6) [ментов командной строки](#_bookmark6) 10
5. [Результат выполнения командного файла](#_bookmark7) 10
6. [Командный файл — аналог команды ls (без использования самой этой](#_bookmark8) [команды и команды dir)](#_bookmark8) 11
7. [Результат выполнения командного файла. Информация о каталогах и](#_bookmark9) [файлах, а также их правах доступа](#_bookmark9) 11
8. [Командный файл, который получает в качестве аргументов формат](#_bookmark10) [файла и директорию, и показывают количество файлов указанного](#_bookmark10) [формата](#_bookmark10) 12
9. [Процесс исполнения командного файла в командной строке](#_bookmark11) 12

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 само- го 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. Напри- мер, 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

Написал скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в моём домашнем каталоге. (Ссылка: Рис.2) При этом файл архивируется архиватором bzip2. Способ использования команд архивации узнал, изучив справку по ним. (Ссылка: Рис.1). В результате исполнения скрипта получил резервную копию самого скрипта и архивированную копию в папке backup в моем домашнем каталоге.



Рис. 1: Изучение способа использования команд архивации

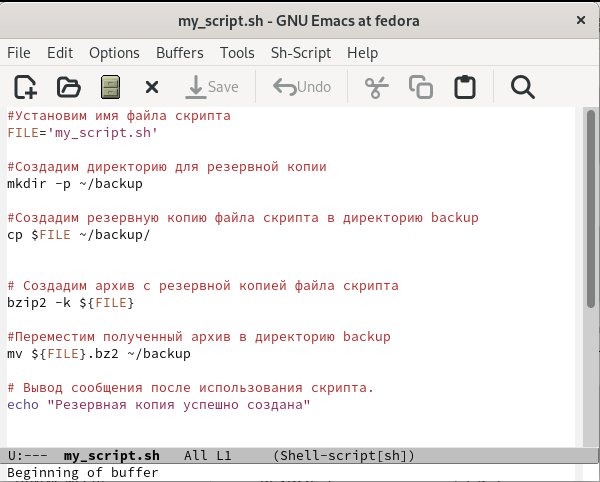


Рис. 2: Скрипт, который при запуске будет делать резервную копию самого себя

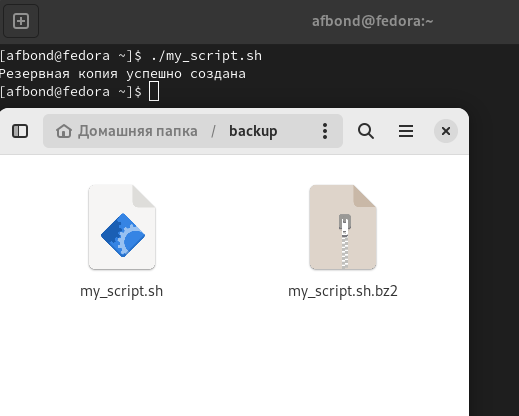


Рис. 3: Результат выполнения скрипта

1. Написал пример командного файла, обрабатывающего любое произвольное чис- ло аргументов командной строки, в том числе превышающее десять.(Ссылка: Рис.4) Например, скрипт может последовательно распечатывать значения всех переданных аргументов.(Ссылка: Рис.5)



Рис. 4: Командный файл, обрабатывающий любое произвольное число аргументов командной строки

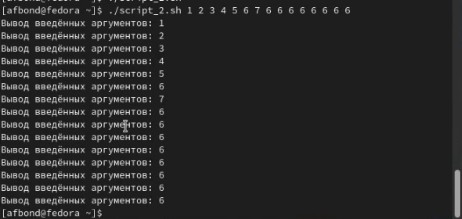


Рис. 5: Результат выполнения командного файла

1. Написал командный файл — аналог команды ls (без использования самой этой команды и команды dir).(Ссылка: Рис.6) Он выдает информацию о каталогах в указанной директории и выводит информацию о возможностях доступа к файлам и каталогам в этой директории. (Ссылка: Рис.7)

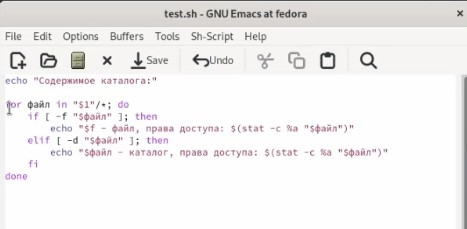


Рис. 6: Командный файл — аналог команды ls (без использования самой этой ко- манды и команды dir)

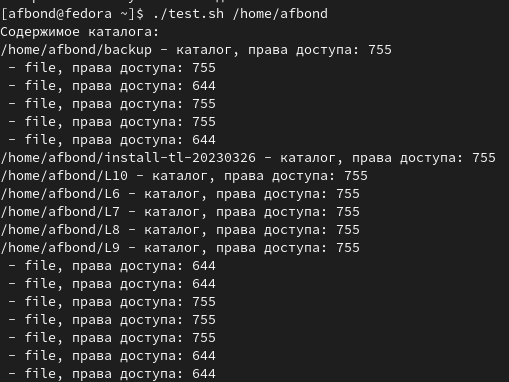


Рис. 7: Результат выполнения командного файла. Информация о каталогах и файлах, а также их правах доступа

1. 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.(Ссылка: Рис.8) Путь к директории также передаётся в виде аргумента командной строки.(Ссылка: Рис.9)

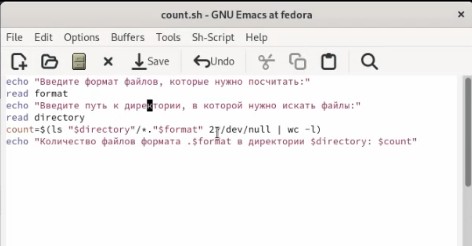


Рис. 8: Командный файл, который получает в качестве аргументов формат файла и директорию, и показывают количество файлов указанного формата

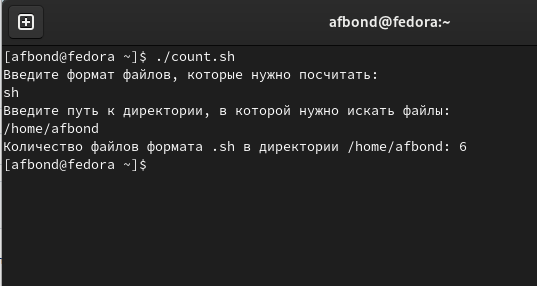


Рис. 9: Процесс исполнения командного файла в командной строке

# Выводы

Таким образом, мы выучили основы программирования в оболочке ОС UNIX/Linux. Научились писать небольшие командные файлы, например, которые обрабатывают произвольное число аргументов, командные файлы которые используются как аналог другим командам, а также научились писать скрипты.

# Ответы на контрольные вопросы

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

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто использу- ются 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 оболочкой Борна, использующая Сподоб- ный синтаксис команд с возможностью сохранения истории выполнения команд;
* оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления 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).

POSIX (Portable Operating System Interface for Computer Environments) — на- бор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

1. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) — на- бор 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/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна.

1. Как определяются переменные и массивы в языке программирования bash?

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи:

${имя переменной}

Например, использование команд b=/tmp/andyls -l myfile > *blssudoapt − getinstalltexlive−luatex*ïðèâåäĳòêïåðåíàçíà@åíèþñòàíäàðòíîãîâûâîäàêîìàíäû*ls*ñòåðìèíàëàíàôàéë*/tmp/andy ls,* àèñïîëüçîâàíèåêîìàíäû*ls − l >*bls приведёт к подстановке в командную строку

значения переменной bls. Если переменной bls не было предварительно присвоено никакого значения, то её значением будет символ пробела.

Оболочка bash позволяет работать с массивами. Для создания массива использу- ется 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. Индекса- ция массивов начинается с нулевого элемента.

1. Каково назначение операторов let и read?

Оператор let в языке программирования bash используется для выполнения ариф- метических операций. Он позволяет задавать арифметические выражения с исполь- зованием различных операторов, таких как +, -, \*, / и %. Например, let x=5+3 присваивает переменной x значение 8.

Оператор read используется для чтения данных из стандартного ввода и со- хранения их в переменных. Он позволяет пользователю вводить данные во время выполнения скрипта. Например, read name прочитает введенные пользователем данные и сохранит их в переменной name. Оператор read также может использо- ваться для чтения нескольких значений, разделенных пробелами, и сохранения их в массиве. Например, read -a array прочитает введенные пользователем данные и сохранит их в массиве array.

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

В языке программирования bash можно использовать следующие арифметические операции:

Сложение (+) Вычитание (-) Умножение (\*) Деление (/) Модуль (остаток от деления) (%) Возведение в степень (\*\*) Операции могут быть выполнены как в контексте арифметического выражения, так и внутри командного файла (скрипта) с использованием оператора let или двойных круглых скобок (( )).

1. Что означает операция (( ))?

Операция (( )) в языке программирования bash используется для выполнения арифметических вычислений. Эта операция обрамляет арифметическое выражение, которое может содержать числа, переменные и операторы, и вычисляет его значение. Например, выражение (( 2 + 3 )) вернет значение 5, а выражение (( x = 10 \* 2 ))

установит значение переменной x равным 20.

Также, внутри (( )) можно использовать условные операторы и операторы сравне- ния, которые позволяют проверять и изменять значения переменных в зависимости от выполнения определенных условий.

1. Какие стандартные имена переменных Вам известны?

$HOME - путь к домашней директории пользователя. $PWD - текущий рабочий каталог. $USER - имя пользователя, от которого запущен скрипт. $SHELL - путь к

исполняемому файлу оболочки. $PATH - список путей, разделенных двоеточием, в которых ищутся исполняемые файлы. $@ - список всех аргументов, переданных скрипту. $# - количество аргументов, переданных скрипту.

1. Что такое метасимволы?

Метасимволы в языке программирования bash (и в других Unix-подобных си- стемах) - это специальные символы, которые используются для выполнения опре- деленных действий при работе с файлами и директориями. Некоторые примеры метасимволов:

(звездочка) - соответствует любой последовательности символов (например, все файлы с расширением .txt можно выбрать с помощью маски \*.txt). ? (вопроси- тельный знак) - соответствует любому одному символу. [] (квадратные скобки) - соответствуют любому символу в указанном диапазоне (например, [a-z] соответству- ет любой букве в нижнем регистре). Эти метасимволы используются в командах, таких как ls, cp, rm и других, чтобы выбрать файлы и директории, с которыми нужно работать. Кроме того, они могут быть использованы внутри скриптов bash для работы с файлами и директориями.

1. Как экранировать метасимволы?

Для экранирования метасимволов в языке программирования bash используется символ обратной косой черты . Этот символ позволяет отключить специальное значение следующего за ним символа. Например, если необходимо вывести символ $ как обычный символ, а не как начало переменной, то нужно написать $. Аналогично, для вывода символа обратной косой черты нужно написать \.

1. Как создавать и запускать командные файлы?

Для создания командного файла в языке программирования bash необходимо создать текстовый файл с расширением .sh, содержащий команды и инструкции, которые вы хотите выполнить. Например, вы можете создать командный файл с именем myscript.sh, содержащий следующий код:

echo "Hello, world!"

Затем сохраните этот файл в желаемой директории и сделайте его исполняемым с помощью команды chmod:

chmod +x myscript.sh

Теперь вы можете запустить командный файл, указав его полный путь:

./myscript.sh

Для запуска командного файла вы также можете добавить его путь в переменную окружения $PATH, чтобы он был доступен в любой директории.

1. Как определяются функции в языке программирования bash?

В языке программирования bash функции определяются с помощью ключевого слова function, после которого идет имя функции, затем открываются фигурные скобки и пишется код функции, а затем закрываются фигурные скобки.

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

В языке программирования bash есть несколько способов проверить, является ли файл каталогом или обычным файлом:

Использование команды test:

Для проверки, является ли файл каталогом: test -d /path/to/file

Для проверки, является ли файл обычным файлом: test -f /path/to/file

\*Использование квадратных скобок и команды test:

Для проверки, является ли файл каталогом: [ -d /path/to/file ]

Для проверки, является ли файл обычным файлом: [ -f /path/to/file ]

Использование оператора файлового теста -d (для каталогов) и -f (для обычных файлов):

Для проверки, является ли файл каталогом: if [[ -d /path/to/file ]]; then . . .

Для проверки, является ли файл обычным файлом: if [[ -f /path/to/file ]]; then . . .

1. Каково назначение команд set, typeset и unset?

Команды set, typeset и unset относятся к управлению переменными в языке программирования bash.

Команда set используется для установки или изменения различных опций для те- кущего сеанса оболочки. Например, можно использовать команду set для включения режима отображения ошибок скрипта (set -e), который приведет к автоматическому выходу из скрипта, если произойдет ошибка выполнения любой команды. Команда set может также использоваться для установки и изменения значений системных переменных.

Команда typeset используется для объявления переменных и установки их атри- бутов. Например, можно использовать команду typeset для объявления переменной, которая должна быть только для чтения, или для установки переменной как мас- сива. Команда typeset может также использоваться для установки переменных в различных контекстах, например, в функциях или скриптах.

Команда unset используется для удаления переменных и функций. Например, можно использовать команду unset для удаления переменной, которая больше не нужна в скрипте, или для удаления функции, которая больше не используется в программе.

1. Как передаются параметры в командные файлы?

Параметры передаются в командные файлы через аргументы командной строки. Когда командный файл запускается, любые аргументы, переданные в командную строку после его имени, становятся доступными внутри скрипта через специальные переменные.

1. Назовите специальные переменные языка bash и их назначение

В языке bash есть множество специальных переменных, которые предназначены для выполнения различных задач. Некоторые из них:

$0: имя скрипта командного файла, который был запущен;

$1, $2, $3, . . . : аргументы, переданные в командный файл;

$\*: список всех аргументов, переданных в командный файл;

$@: список всех аргументов, переданных в командный файл, с сохранением пробе- лов и кавычек;

$#: количество аргументов, переданных в командный файл;

$?: код возврата последней выполненной команды;

$!: PID последнего фонового процесса, запущенного в командном файле;

$\* — отображается вся командная строка или параметры оболочки;

$? — код завершения последней выполненной команды;

$$ — уникальный идентификатор процесса, в рамках которого выполняется ко- мандный процессор;

$! — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; $- — значение флагов командного про- цессора;

${#} — 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.