

Отчет по лабораторной работе №10

Дисциплина: Операционные системы

Старикова Евгения Дмитриевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	12
5	Контрольные вопросы	13

Список иллюстраций

3.1	рис.14	11
-----	------------------	----

Список таблиц

1 Цель работы

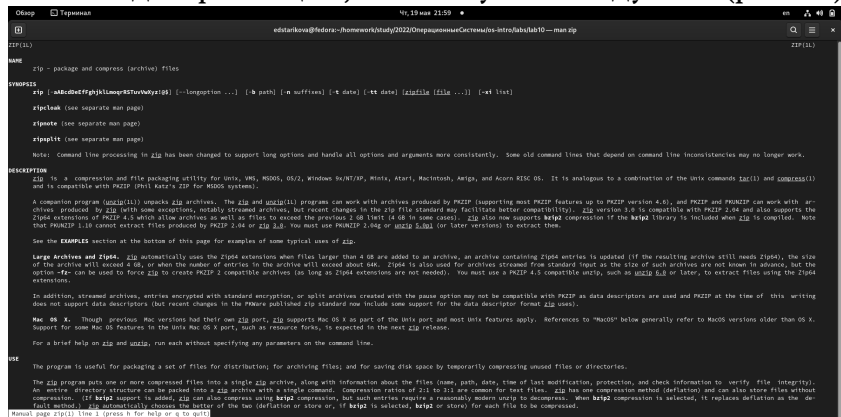
Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

2 Задание

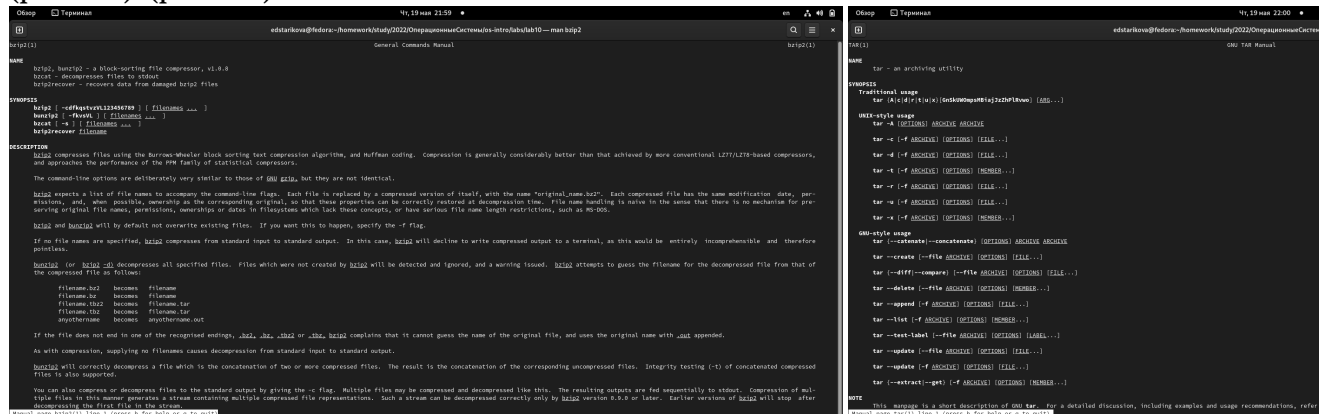
1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд 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.

3 Выполнение лабораторной работы

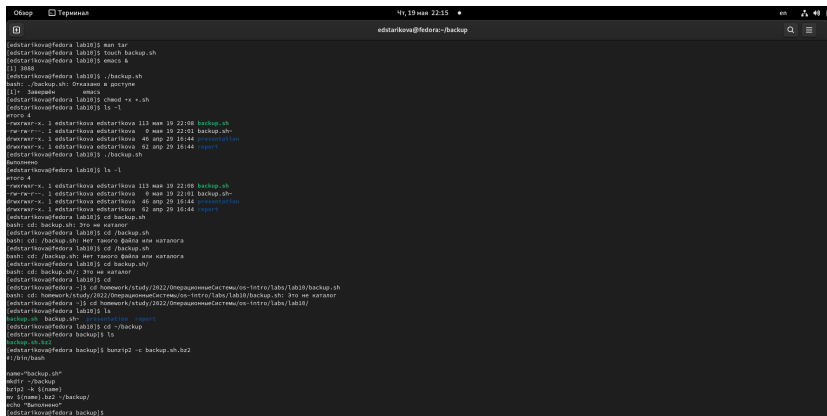
1. Для начала изучили команды архивации, используя команду man (рис. ??)



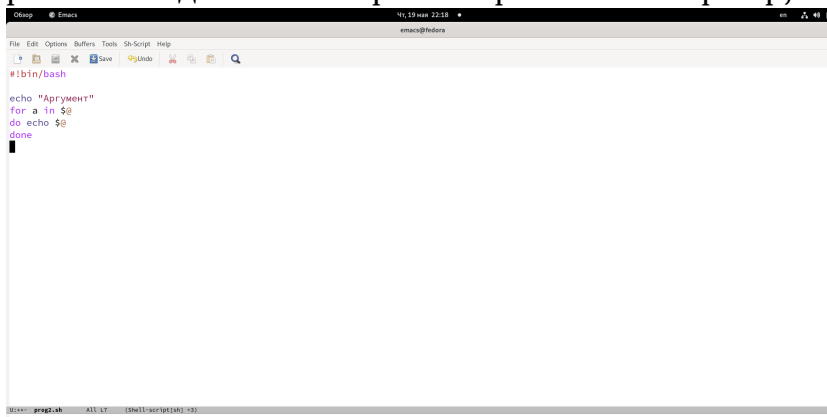
(рис. ??) (рис. ??)



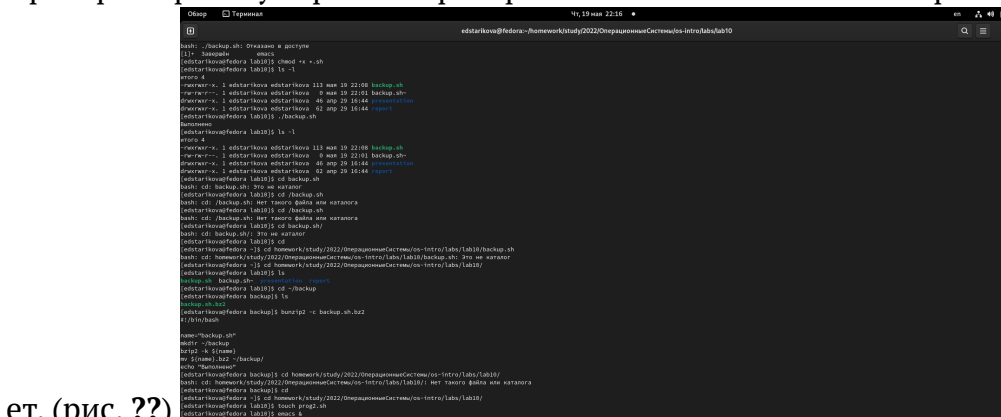
2. Далее создаем файл, в котором буду писать первый скрипт и открыла его с помощью редактора emacs, используя клавиши (C-x C-f) (рис. ??)



3. Пишем скрипт, который при запуске будет делать резервную копию самого себя в другую дерикторию. При этом файл должен архивироваться одни м из архиваторов на выбор zip, bzip2, tar. (рис. ??)



4. Проверила работу скрипта. Проверила появился ли каталог. Скрипт работа-



ет. (рис. ??)

5. Создала файл, в котором буду писать второй скрипт и открыла его тоже в ре-



Обзор

Emiss

File Edit Options Buffers Tools Sh-Script Help

#!/bin/bash

a="51"
for i in \$(j)/*
do

 echo "\$i"
 if test -f \$i
 then echo "Обычный файл"
 fi

 if test -d \$i
 then echo "Каталог"
 fi

 if test -r \$i
 then echo ""Чтение разрешено"
 fi

 if test -w \$i
 then echo "Запись разрешена"
 fi

 if test -x \$i
 then echo "Выполнение разрешено"
 fi

done

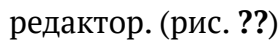
100% 1 KB All 128 Shell scripts (1 MB) 24

Welcome to Emiss, an open component of the GNU nano operating system.

[illegible]

Проверила работу.Скрипт работает (рис. ??)

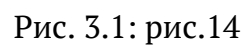
9

[illegible]

■ Проверила работу.



7. Для четвертого скрипта создала файл Написала командный файл, который получает в качестве аргумента командной строки формат файла и вычисля-



4 Выводы

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

5 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,

C-оболочка (или csh) – надстройка на оболочкой Борна, использующая Сподобный синтаксис,

оболочка Корна (или ksh) – напоминает оболочку C, но операторы управления программой,

BASH – сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей со

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 states Delaware Michigan "New Jersey"» 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 яв
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, введённые с к

В языке программирования bash можно применять такие арифметические операции

), умножение(*), целочисленное деление (/) и целочисленный остаток от деления

В (()) можно записывать условия оболочки bash, а также внутри двойных скобок

Стандартные переменные:

PATН: значением данной переменной является список каталогов, в которых командный

PS1 и PS2: эти переменные предназначены для отображения промптера командного про

то интерактивная программа, запущенная командным процессором, требует ввода, то и

HOME: имя домашнего каталога пользователя. Если команда cd вводится без аргументов

IFS: последовательность символов, являющихся разделителями в командной строке, на

MAIL: командный процессор каждый раз перед выводом на экран промптера проверяет с

TERM: тип используемого терминала.

LOGNAME: содержит регистрационное имя пользователя, которое устанавливается автом

Такие символы, как ' < > * ? | " &, являются метасимволами и имеют для командного процессора специальный смысл. 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`. 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. 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`. 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). 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. 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 файл;

\$! – номер процесса, в рамках которого выполняется последняя вызванная на выполнение 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 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 сообщение;

\${name+value} – это выражение работает противоположно \${name-value}. Если переменная определена, то выводит сообщение;

\${name#pattern} – представляет значение переменной name с удалённым самым коротким подстроком, соответствующим pattern;

`${#name[*]}` и `${#name[@]}` – эти выражения возвращают количество элементов в массиве