

Отчёт по лабораторной работе № 11

Операционные системы

Ильина Любовь Александровна

Содержание

| | | |
|----------|---------------------------------------|-----------|
| 1 | Цель работы | 5 |
| 2 | Выполнение лабораторной работы | 6 |
| 3 | Выводы | 16 |

Список таблиц

Список иллюстраций

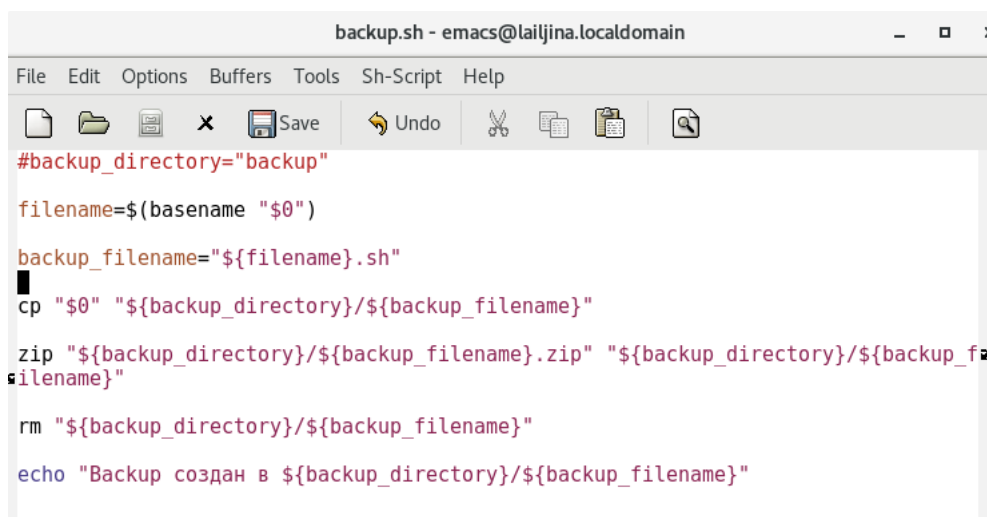
| | | |
|-----|--|----|
| 2.1 | Скрипт командного файла, создающего свой backup | 6 |
| 2.2 | Выполнение командного файла, создающего свой backup | 6 |
| 2.3 | Скрипт командного файла, обрабатывающего любое произвольное число аргументов | 7 |
| 2.4 | Выполнение командного файла, обрабатывающего любое произвольное число аргументов | 7 |
| 2.5 | Скрипт командного файла, аналога команды ls | 8 |
| 2.6 | Выполнение командного файла, аналога команды ls | 9 |
| 2.7 | Скрипт командного файла, который вычисляет количество файлов указанного формата в указанной директории | 10 |
| 2.8 | Выполнение командного файла, который вычисляет количество файлов указанного формата в указанной директории | 10 |
| 2.9 | Арифметические операции bash | 12 |

1. Цель работы

Изучение основ программирования в оболочке ОС UNIX/Linux. Написание небольших командных файлов.

2. Выполнение лабораторной работы

1. Напишем 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. (рис. 2.1 - 2.2)



```
backup.sh - emacs@lailjina.localdomain
File Edit Options Buffers Tools Sh-Script Help
#backup_directory="backup"
filename=$(basename "$0")
backup_filename="${filename}.sh"
cp "$0" "${backup_directory}/${backup_filename}"
zip "${backup_directory}/${backup_filename}.zip" "${backup_directory}/${backup_filename}"
rm "${backup_directory}/${backup_filename}"
echo "Backup создан в ${backup_directory}/${backup_filename}"
```

Рис. 2.1: Скрипт командного файла, создающего свой backup

```
[lailjina@lailjina report]$ bash backup.sh
adding: backup/backup.sh.sh (deflated 63%)
Backup создан в backup/backup.sh.sh
```

Рис. 2.2: Выполнение командного файла, создающего свой backup

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. (рис. 2.3 - 2.4)

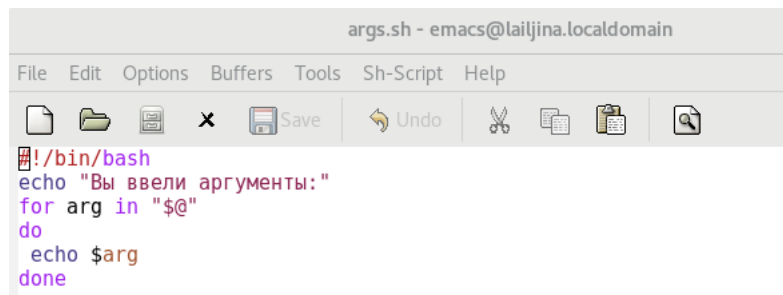


Рис. 2.3: Скрипт командного файла, обрабатывающего любое произвольное число аргументов

```
[lailjina@lailjina report]$ bash args.sh 10 9 8 7 6 5 4 3 2 1 hgjk @
Вы ввели аргументы:
10
9
8
7
6
5
4
3
2
1
hgjk
@
```

Рис. 2.4: Выполнение командного файла, обрабатывающего любое произвольное число 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 (рис. 2.5 - 2.6)

The image shows a screenshot of an Emacs editor window. The title bar at the top reads "ls2 - emacs@lailjina.localdomain". Below the title bar is a menu bar with "File", "Edit", "Options", "Buffers", "Tools", "Sh-Script", and "Help". Underneath the menu bar is a toolbar with icons for file operations: a document icon, a folder icon, a save icon, a close icon, a save icon labeled "Save", an undo icon labeled "Undo", a scissors icon, a copy icon, a paste icon, and a search icon. The main text area contains a shell script. The script starts with a shebang line "#!/bin/bash". It defines a function that takes an argument "\$1". If the argument is empty, it sets "directory" to "."; otherwise, it sets "directory" to "\$1". The function then prints "Информация о каталоге \$directory:" followed by a dashed line separator. It prints "Файлы:" and then iterates over all files in the directory. For each file, it checks if it is a regular file (-f) and if so, it prints the filename. After listing files, it prints "Директории:" and iterates over all subdirectories. For each subdirectory, it checks if it is a directory (-d) and if so, it prints the directory name. Finally, it prints "Права доступа:" followed by a dashed line separator, and then iterates over all files and subdirectories, printing their permissions using the "stat -c '%A'" command. The script is color-coded: keywords like "if", "then", "else", "fi", "do", "done", "for", "in", "echo", and "file" are in purple; variables like "\$1", "\$directory", "\$file", "\$dir", and "\$stat" are in pink; and the "stat" command is in magenta. The script ends with a "done" keyword.

```
#!/bin/bash

if [ -z "$1" ]
then
    directory="."
else
    directory="$1"
fi

echo "Информация о каталоге $directory:"
echo "-----"
echo "Файлы:"
for file in "$directory"/*
do
    if [ -f "$file" ]
    then
        echo "$file"
    fi
done

echo "Директории:"
for dir in "$directory"/*
do
    if [ -d "$dir" ]
    then
        echo "$dir"
    fi
done

echo "Права доступа:"
echo "-----"
for file in "$directory"/*
do
    if [ -f "$file" ]
    then
        echo "$file $(stat -c '%A' "$file")"
    fi
done

for dir in "$directory"/*
do
    if [ -d "$dir" ]
    then
        echo "$dir $(stat -c '%A' "$dir")"
    fi
done
```

Рис. 2.5: Скрипт командного файла, аналога команды ls


```
lailjina@lailjina:~/work/study/2022-2023/Операцио
File Edit View Search Terminal Help
lailjina@lailjina ~]$ chmod u+x ls2
[lailjina@lailjina ~]$ ./play
bash: ./play: Is a directory
[lailjina@lailjina ~]$ bash ls2
Информация о каталоге .:
-----
Файлы:
./abc1
./c-feathers
./get-pip.py
./#hello.sh#
./install-tl-unx.tar.gz
./#lab07.sh#
./lab07.sh
./logfile.txt
./ls2
./my_os
Директории:
./australia
./backup
./Desktop
./Documents
./Downloads
./install-tl-20230505
./Music
./pandoc
./Pictures
./play
./Public
./ski.plases
./Templates
./usr
./Videos
./work
./Work
Права доступа:
-----
./abc1 -rw-rw-r--
./c-feathers -rw-rw-r--
./get-pip.py -rw-rw-r--
./#hello.sh# -rw-rw-r--
./install-tl-unx.tar.gz -rw-rw-r--
./#lab07.sh# -rw-rw-r--
./lab07.sh -rw-rw-r--
./logfile.txt -rw-rw-r--
./ls2 -rwxrw-r--
./my_os -r-xr--r--
./australia drwxr--r--
./backup drwxr-xr-x
```

Рис. 2.6: Выполнение командного файла, аналога команды ls

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 (рис. 2.7 - 2.8)

Рис. 2.7: Скрипт командного файла, который вычисляет количество файлов указанного формата в указанной директории

```
[lailjina@lailjina ~]$ bash formats.sh ~ txt
Количество файлов с расширением .txt в директории /home/lailjina: 2563
[lailjina@lailjina ~]$ bash formats.sh Downloads pdf
Количество файлов с расширением .pdf в директории Downloads: 1
```

Рис. 2.8: Выполнение командного файла, который вычисляет количество файлов указанного формата в указанной директории

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 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 оболочкой Борна, использующая С подобный 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).

10

2. Что такое POSIX? POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ
3. Как определяются переменные и массивы в языке программирования bash? Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Присвоение с помощью `=`. Значение, присвоенное некоторой переменной, может быть впоследствии использовано по символу `$`. Например, для переменной `mark`: `mv afile ${mark}`. 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"`
4. Каково назначение операторов `let` и `read`? Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный. 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. 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 клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введённую информацию и игнорировать её.
5. Какие арифметические операции можно применять в языке программирования bash? (рис. 2.9)

Таблица 8.1

Арифметические операторы оболочки bash

| Оператор | Синтаксис | Результат |
|----------|--------------|---|
| ! | !exp | Если exp равно 0, то возвращает 1; иначе 0 |
| != | exp1 !=exp2 | Если exp1 не равно exp2, то возвращает 1; иначе 0 |
| % | exp1%exp2 | Возвращает остаток от деления exp1 на exp2 |
| %= | var=%exp | Присваивает остаток от деления var на exp переменной var |
| & | exp1&exp2 | Возвращает побитовое AND выражений exp1 и exp2 |
| && | exp1&&exp2 | Если и exp1 и exp2 не равны нулю, то возвращает 1; иначе 0 |
| &= | var &= exp | Присваивает переменной var побитовое AND var и exp |
| * | exp1 * exp2 | Умножает exp1 на exp2 |
| *= | var *= exp | Умножает exp на значение переменной var и присваивает результат переменной var |
| + | exp1 + exp2 | Складывает exp1 и exp2 |
| += | var += exp | Складывает exp со значением переменной var и результат присваивает переменной var |
| - | -exp | Операция отрицания exp (унарный минус) |
| - | exp1 - exp2 | Вычитает exp2 из exp1 |
| -= | var -= exp | Вычитает exp из значения переменной var и присваивает результат переменной var |
| / | exp / exp2 | Делит exp1 на exp2 |
| /= | var /= exp | Делит значение переменной var на exp и присваивает результат переменной var |
| < | exp1 < exp2 | Если exp1 меньше, чем exp2, то возвращает 1, иначе возвращает 0 |
| << | exp1 << exp2 | Сдвигает exp1 влево на exp2 бит |
| <=< | var <=< exp | Побитовый сдвиг влево значения переменной var на exp |
| <= | exp1 <= exp2 | Если exp1 меньше или равно exp2, то возвращает 1; иначе возвращает 0 |
| = | var = exp | Присваивает значение exp переменной var |
| == | exp1==exp2 | Если exp1 равно exp2, то возвращает 1; иначе возвращает 0 |
| > | exp1 > exp2 | 1, если exp1 больше, чем exp2; иначе 0 |
| >= | exp1 >= exp2 | 1, если exp1 больше или равно exp2; иначе 0 |
| >> | exp >> exp2 | Сдвигает exp1 вправо на exp2 бит |
| >>= | var >>=exp | Побитовый сдвиг вправо значения переменной var на exp |
| ^ | exp1 ^ exp2 | Исключающее OR выражений exp1 и exp2 |
| ^= | var ^= exp | Присваивает переменной var побитовое XOR var и exp |
| | exp1 exp2 | Побитовое OR выражений exp1 и exp2 |
| = | var = exp | Присваивает переменной var результат операции XOR var и exp |
| | exp1 exp2 | 1, если или exp1 или exp2 являются ненулевыми значениями; иначе 0 |
| ~ | ~exp | Побитовое дополнение до exp |

Рис. 2.9: Арифметические операции bash

6. Что означает операция (())? условия оболочки let для удобства вкладывают в (())
7. Какие стандартные имена переменных Вам известны? PATH - стандартная (по умолчанию) последовательность поиска файла: текущий каталог, каталог /bin, каталог /usr/bin. Переменной можно присвоить иное значение. – 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. Что такое метасимволы? Такие символы, как `' < * ? | " &`, являются метасимволами и имеют для командного процессора специальный смысл. Снятие специального смысла с метасимвола называется экранированием метасимвола.
9. Как экранировать метасимволы? 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. Как создавать и запускать командные файлы? Создание файла с помощью редактора `vim`: `vim .sh`. Также можно создать командой `touch` и редактировать в терминале: `cat > .sh`. Сохранить изменения по Исполнение файла: `chmod u+x .sh` Вы можете запустить скрипт любым из указанных способов: `sh .sh` `bash .sh` `./sh`
11. Как определяются функции в языке программирования `bash`? Группу команд можно объединить в функцию. 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? `for A`

in * do if test -d \$A then echo \$A: is a directory else echo -n \$A: is a file

13. Каково назначение команд set, typeset и unset? Команда set — это встроенная команда оболочки, которая позволяет отображать или устанавливать переменные оболочки и среды. Команда typeset имеет четыре опции для работы с функциями: – -f — перечисляет определённые на текущий момент функции; – -ft — при последующем вызове функции иницирует её трассировку; – -fx — экспортирует все перечисленные функции в любые дочерние программы оболочек; – -fu — обозначает указанные функции как автоматически загружаемые. Автоматически загружаемые функции хранятся в командных файлах, а при их вызове оболочка просматривает переменную FPATH, отыскивая файл с одноимёнными именами функций, загружает его и вызывает эти функции. Удалить функцию можно с помощью команды unset с флагом -f.
14. Как передаются параметры в командные файлы? Символ \$ используется для ссылки на параметры, точнее, для получения их значений в командном файле.
15. Назовите специальные переменные языка bash и их назначение. \$# - осуществляет подстановку числа параметров, указанных в командной строке при вызове данного командного файла на выполнение. – \$* — 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.

3. Выводы

Изучила основы программирования в оболочке ОС UNIX/Linux. Научилась писать небольшие командные файлы.