

Лабораторная работа №10

**Программирование в командном процессоре ОС UNIX. Командные
файлы**

Галиева Аделина Руслановна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Контрольная работа	10
4	Выводы	13
	Список литературы	14

Список иллюстраций

2.1	Прописываем команды	6
2.2	Пишем скрипт	6
2.3	Прописываем команды	7
2.4	Пишем пример	7
2.5	Прописываем команды	8
2.6	Пишем командный файл	8
2.7	Прописываем команды	9
2.8	Пишем командный файл	9

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Пишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в нашем домашнем каталоге. (рис. 2.1) (рис. 2.2)

```
argalievadk4n69 ~ $ touch lab10_1.sh
argalievadk4n69 ~ $ chmod +x lab10_1.sh
argalievadk4n69 ~ $ ./lab10_1.sh
argalievadk4n69 ~ $ ls ~/backup/
backup.sh.gz
```

Рис. 2.1: Прописываем команды

```
1 #!/bin/bash
2 mkdir ~/backup
3 cp lab10_1.sh ~/backup/backup.sh
4 gzip ~/backup/backup.sh
```

Рис. 2.2: Пишем скрипт

2. Пишем пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. (рис. 2.3) (рис. 2.4)

```

argaliev@dk4n69 ~ $ touch lab10_2.sh
argaliev@dk4n69 ~ $ chmod +x lab10_2.sh
argaliev@dk4n69 ~ $ ./lab10_2.sh A P P L E 1 3
A
P
P
L
E
1
3

```

Рис. 2.3: Прописываем команды

```

1 #!/bin/bash
2 for i
3     do echo $1
4         shift
5 done

```

Рис. 2.4: Пишем пример

3. Пишем командный файл — аналог команды `ls` (без использования самой этой команды и команды `dir`). (рис. 2.5) (рис. 2.6)

```

argaliev@dk4n69 ~ $ touch lab10_3.sh
argaliev@dk4n69 ~ $ chmod +x lab10_3.sh
argaliev@dk4n69 ~ $ ./lab10_3.sh
READ
.
./public
./public_html
./profile
./bashrc
./bash_profile
./Xauthority
./xsession-errors
./local
./Рабочий стол
./Загрузки
./Шаблоны
./Общедоступные

```

Рис. 2.5: Прописываем команды

```

1 #!/bin/bash
2 echo "READ"
3 find $1 -maxdepth 1 -perm /u=r
4 echo "WRITE"
5 find $1 -maxdepth 1 -perm /u=w
6 echo "EXECUTE"
7 find $1 -maxdepth 1 -perm /u=x

```

Рис. 2.6: Пишем командный файл

4. Пишем 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) (рис. 2.8)


```

argalieva@dk4n69 ~ $ touch lab10_4.sh
argalieva@dk4n69 ~ $ chmod +x lab10_4.sh
argalieva@dk4n69 ~ $ ./lab10_4.sh
write format
sh
write directory
backup
0
1.py      feathers  lab09.sh  main.map  public_html  work      Общедоступные
abc1      file.txt   lab09.sh~ monthly   __pycache__  Видео     'Рабочий стол'
australia GNUstep   lab10_1.sh my_os      reports      Документы  Шаблоны
backup    hello.asm  lab10_2.sh passwd     ski_places  Загрузки
bin       lab05      lab10_3.sh play       text.txt    Изображения
conf.txt  lab07.sh   lab10_4.sh public     tmp         Музыка
argalieva@dk4n69 ~ $ 

```

Рис. 2.7: Прописываем команды

```

1 #!/bin/bash
2 direct=''
3 form=''
4 echo 'write format'
5 read form
6 echo 'write directory'
7 read direct
8 find "$direct" -name "$form" -type f | wc -l
9 ls

```

Рис. 2.8: Пишем командный файл

3 Контрольная работа

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?
 - a) sh — стандартная командная оболочка UNIX/Linux, содержащая базовый, полный набор функций
 - b) csh — использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд
 - c) ksh — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна
 - d) bash — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна
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? Переменные вызываются \$var, где var=чему-то, указанному пользователем, неважно что бы то не было, название файла, каталога или еще чего. Для массивов используется команда set -A
4. Каково назначение операторов let и read? let — вычисляет далее заданное математическое значение read — позволяет читать значения переменных со стандартного ввода

5. Какие арифметические операции можно применять в языке программирования `bash`? Прибавление, умножение, вычисление, деление), сравнение значений, экспонирование и др.
6. Что означает операция `(())`? Это обозначение используется для облегчения программирования для условий `bash`
7. Какие стандартные имена переменных Вам известны? Нам известны `HOME`, `PATH`, `BASH`, `ENV`, `PWD`, `UID`, `OLDPWD`, `PPID`, `GROUPS`, `OSTYPE`, `PS1` - `PS4`, `LANG`, `HOSTFILE`, `MAIL`, `TERM`, `LOGNAME`, `USERNAME`, `IFS` и др.
8. Что такое метасимволы? Метасимволы это специальные знаки, которые могут использоваться для сокращения пути, поиска объекта по расширению, перед переменными, например «\$» или «*» .
9. Как экранировать метасимволы? Добавить перед метасимволом метасимвол «\»
10. Как создавать и запускать командные файлы? При помощи команды `chmod`. Надо дать права на запуск `chmod +x название файла`, затем запустить `bash ./название файла` Например у нас файл `lab` Пишем: `chmod +x lab ./lab`
11. Как определяются функции в языке программирования `bash`? Объединяя несколько команд с помощью `function`
12. Каким образом можно выяснить, является файл каталогом или обычным файлом? Можно задать команду на проверку директория ли это `test -d директория`
13. Каково назначение команд `set`, `typeset` и `unset`? `Set` — используется для создания массивов `Unset` — используется для изъятия переменной `Typeset` — используется для присваивания каких-либо функций
14. Как передаются параметры в командные файлы? Добавлением аргументов после команды запуска `bash` скрипта

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 команда;
- `$!` – номер процесса, в рамках которого выполняется последняя вызванная на выполнение команда;
- `$-` – значение флагов командного процессора;
- `${#*}` – возвращает целое число – количество слов, которые были результатом выполнения `$*`;
- `${#name}` – возвращает целое значение длины строки в переменной `name`;
- `${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 об ошибке;
- `${name+value}` – это выражение работает противоположно `${name-value}`. Если переменная определена, то подставляется `value`;
- `${name#pattern}` – представляет значение переменной `name` с удалённым самым коротким началом, соответствующим `pattern`;
- `${#name[*]}` и `${#name[@]}` – эти выражения возвращают количество элементов в массиве `name`.

4 Выводы

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

Список литературы