

# **Лабораторная работа 10**

**Программирование в командном процессоре. Командные файлы**

**Абдулфазов Мансур Али оглы**

# **Содержание**

Цель работы . . . . .	2
Выполнение лабораторной работы . . . . .	2
Вывод . . . . .	5
Контрольные вопросы . . . . .	5

## **Цель работы**

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

## **Выполнение лабораторной работы**

1. Скрипт 1 (рис. 1)

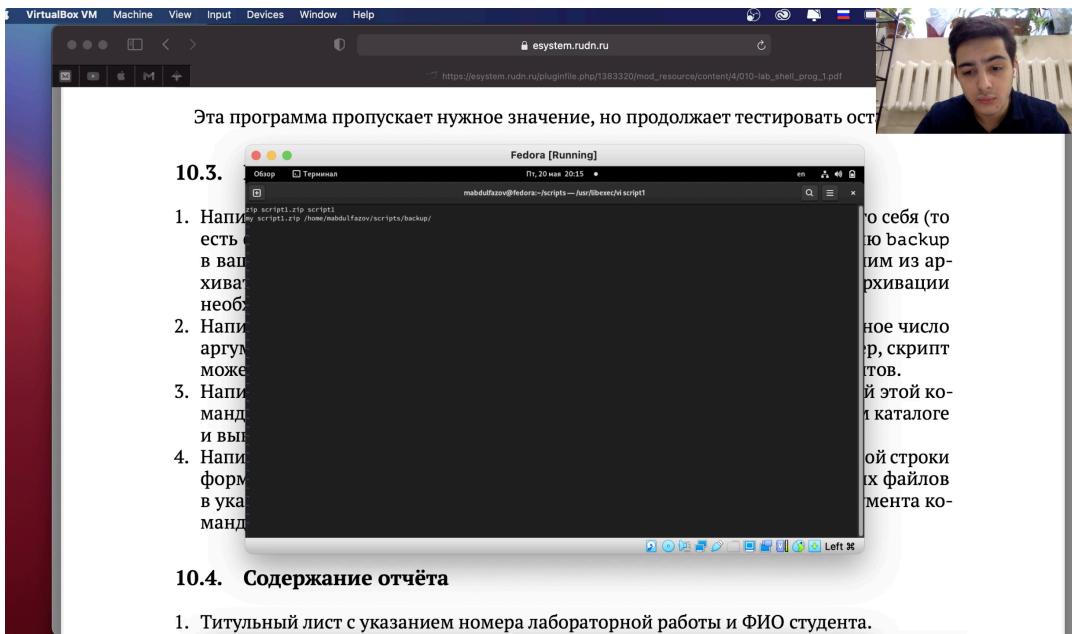


Рис. 1: Скрипт 1

## 2. Скрипт 2 (рис. 2)

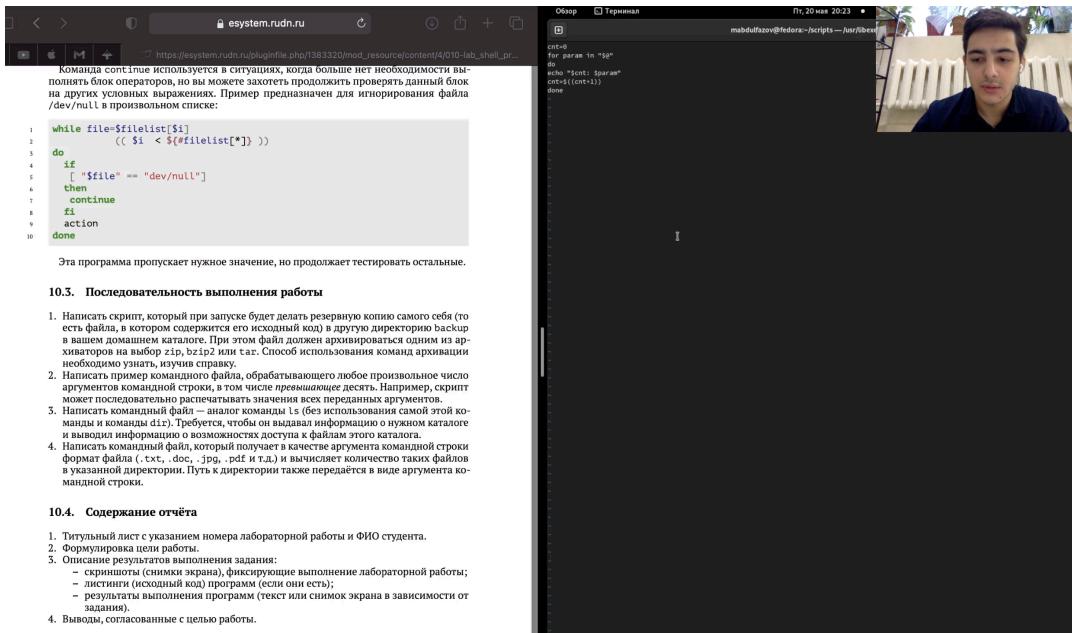


Рис. 2: Скрипт 2

## 3. Скрипт 3 (рис. 3)

The screenshot shows a video conference interface with a student on the right and a terminal window on the left. The terminal window displays a script code:

```

1 #!/bin/bash
2
3 if test -d $1
4   then echo "$1 : is dir"
5   if test -w $1
6     then echo "writable"
7     elif test -r $1
8       then echo "readable"
9     fi
10 elseecho "it is not a dir"
11 fi

```

Below the terminal, a message reads: "Эта программа пропускает нужное значение, но продолжает тестировать остальные."

**10.3. Последовательность выполнения работы**

- Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию `бакспир` в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор `zip`, `bz2` или `tar`. Способ использования команды архивации необходимо учесть, изучив соответствующие документации.
- Написать пример скрипта, обрабатывающего любое произвольное число аргументов командной строки, в том числе преамбулье десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
- Написать командный файл – аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выдавал примерно в виде листинга, похожего на то, что получается при выполнении команды `ls`.
- 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 передается в виде аргумента командной строки.

**10.4. Содержание отчёта**

- Титульный лист с указанием номера лабораторной работы и ФИО студента.
- Формулировка цели работы.
- Описание результатов выполнения задания:
  - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
  - листинги (исходный код) программ (если они есть);
  - результаты выполнения программ (текст или снимок экрана в зависимости от задания).
- Выводы, согласованные с целью работы.

Рис. 3: Скрипт 1

## 4. Скрипт 4 (рис. 4)

The screenshot shows a video conference interface with a student on the right and a terminal window on the left. The terminal window displays a script code:

```

1 #!/bin/bash
2
3 while file=$filelist[$i]
4   do
5     if
6       [ "$file" == "dev/null" ]
7       then
8         continue
9       fi
10      action
11 done

```

Below the terminal, a message reads: "Эта программа пропускает нужное значение, но продолжает тестировать остальные."

**10.3. Последовательность выполнения работы**

- Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию `бакспир` в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор `zip`, `bz2` или `tar`. Способ использования команды архивации необходимо учесть, изучив соответствующие документации.
- Написать пример скрипта, обрабатывающего любое произвольное число аргументов командной строки, в том числе преамбулье десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
- Написать командный файл – аналог команды `ls` (без использования самой этой команды и команды `dir`). Требуется, чтобы он выдавал информацию о нужном каталоге и выдавал примерно в виде листинга, похожего на то, что получается при выполнении команды `ls`.
- 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 передается в виде аргумента командной строки.

**10.4. Содержание отчёта**

- Титульный лист с указанием номера лабораторной работы и ФИО студента.
- Формулировка цели работы.
- Описание результатов выполнения задания:
  - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
  - листинги (исходный код) программ (если они есть);
  - результаты выполнения программ (текст или снимок экрана в зависимости от задания).
- Выводы, согласованные с целью работы.

Рис. 4: Скрипт 4

## **Вывод**

Научился писать небольшие командные файлы.

## **Контрольные вопросы**

1. Командные процессоры или оболочки - это программы, позволяющие пользователю взаимодействовать с компьютером. Их можно рассматривать как настоящие интерпретируемые языки, которые воспринимают команды пользователя и обрабатывают их. Поэтому командные процессоры также называют интерпретаторами команд. На языках оболочек можно писать программы и выполнять их подобно любым другим программам. UNIX обладает большим количеством оболочек. Наиболее популярными являются следующие четыре оболочки: – оболочка Борна (Bourne) - первоначальная командная оболочка UNIX: базовый, но полный набор функций; –С-оболочка - добавка университета Беркли к коллекции оболочек: она надстраивается над оболочкой Борна, используя Сподобный синтаксис команд, и сохраняет историю выполненных команд; –оболочка Корна - 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).
2. POSIX (Portable Operating System Interface for Computer Environments)-интерфейс переносимой операционной системы для компьютерных сред. Представляет собой набор стандартов, подготовленных институтом инженеров по электронике и радиотехнике (IEEE), который определяет различные аспекты построения операционной системы. POSIX включает такие темы, как программный интерфейс, безопасность, работа с

сетями и графический интерфейс. POSIX-совместимые оболочки являются будущим поколением оболочек UNIX и других ОС. Windows NT рекламируется как система, удовлетворяющая POSIX-стандартам. POSIX-совместимые оболочки разработаны на базе оболочки Корна; фонд бесплатного программного обеспечения (Free Software Foundation) работает над тем, чтобы и оболочку BASH сделать POSIX-совместимой.

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 $mark` переместит файл `afile` из текущего каталога в каталог с абсолютным полным именем `/usr/andy/bin`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того, чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи:  `${имя переменной}` например, использование команд `b=/tmp/andy-ls -l myfile > bls` `ls /tmp/andy -l >bls` приведет к подстановке в командную строку значения переменной `bls`. Если переменной `bls` не было предварительно присвоено никакого значения, то ее значением является символ пробел. Оболочка bash позволяет создание массивов. Для создания массива используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. 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. Команда `let` является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение - это единичный терм (term), обычно целочисленный. Целые числа можно записывать как последовательность цифр или в любом базовом формате. Этот формат – `radix#number`, где `radix` (основание системы счисления) – любое число не более 26. Для большинства команд основания систем счисления это – 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток (%). Команда `let` берет два операнда и присваивает их переменной.
5. Какие арифметические операции можно применять в языке программирования `bash`? Оператор Синтаксис Результат ! !exp Если `exp` равно 0, возвращает 1; иначе 0 != exp1 != exp2 Если `exp1` не равно `exp2`, 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, возвращает 1; иначе 0 &= var &= exp Присваивает `var` побитовое AND переменных `var` и выражения `exp * exp1 * exp2` Умножает `exp1` на `exp2 = var = exp` Умножает `exp` на значение `var` и присваивает результат переменной `var + exp1 + exp2` Складывает `exp1` и `exp2 += var += exp` Складывает `exp` со значением `var` и результат присваивает `var - -exp` Операция отрицания `exp` (называется унарный минус) - `expl - exp2` Вычитает `exp2` из `exp1` -- var -- exp Вычитает `exp` из значения `var` и присваивает результат `var / exp / exp2` Делит `exp1` на `exp2 /= var /= exp` Делит `var` на `exp` и присваивает результат `var < expl < exp2` Если `exp1` меньше, чем `exp2`, возвращает 1, иначе возвращает 0 « `exp1«`

`exp2` Сдвигает `exp1` влево на `exp2` бит  
`<= var <= exp` Побитовый сдвиг влево  
значения `var` на `exp`  
`<= exp1 <= exp2` Если `exp1` меньше, или равно `exp2`,  
возвращает 1; иначе возвращает 0 = `var = exp` Присваивает значение  
`exp` переменной `va == exp1==exp2` Если `exp1` равно `exp2`. 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 `var` на `exp ^ exp1 ^ exp2` Исключающее OR выражений `exp1` и  
`exp2 ^= var ^= exp` Присваивает `var` побитовое исключающее OR `var` и `exp | exp1 | exp2` Побитовое OR выражений `exp1` и `exp2 |= var |= exp` Присваивает  
`var «исключающее OR»` переменной `var` и выражения `exp || exp1 || exp2` 1  
если или `exp1` или `exp2` являются ненулевыми значениями; иначе 0 ~  
`~exp` Побитовое дополнение до `exp`

6. Условия оболочки bash, в двойные скобки —(( )).
7. Имя переменной (идентификатор) — это строка символов, которая отличает эту переменную от других объектов программы (идентифицирует переменную в программе). При задании имен переменным нужно соблюдать следующие правила: § первым символом имени должна быть буква. Остальные символы — буквы и цифры (прописные и строчные буквы различаются). Можно использовать символ «\_»; § в имени нельзя использовать символ «.»; § число символов в имени не должно превышать 255; § имя переменной не должно совпадать с зарезервированными (служебными) словами языка. Var1, PATH, trash, mon, day, PS1, PS2 Другие стандартные переменные: –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. Это символы 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. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

8. Такие символы, как ' < > \* ? | " & являются метасимволами и имеют для командного процессора специальный смысл.
9. 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, ее нужно заключить в одинарные кавычки. Стока, заключенная в двойные кавычки, экранирует все метасимволы, кроме \$, ' , , “. Например, –echo \* выведет на экран символ, –echo ab'cd выдаст строку ab|\*cd.
10. 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 ее интерпретацию.

11. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определенные на текущий момент функции; `--ft` — при последующем вызове функции инициирует ее 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 одноименными именами функций, загружает его и вызывает эти функции.
12. `ls -lrt` Если есть `d`, то является файл каталогом
13. Используется команда `set` с флагом `-A`. За флагом следует имя переменной, а затем список значений, разделенных пробелом. Например, `set -A states Delaware Michigan "New Jersey"` Далее можно сделать добавление в массив, например, `states[49]=Alaska`. Индексация массивов начинается с нулевого элемента. В командном процессоре Си имеется еще несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды `set`. Наиболее распространенным является сокращение, избавляющееся от слова `let` в программах оболочек. Если объявить переменные целыми значениями, любое присвоение автоматически трактуется как арифметическое. Используйте `typeset -i` для объявления и присвоения переменной, и при последующем использовании она становится целой. Или можете использовать ключевое слово `integer` (псевдоним для `typeset -l`) и объявлять переменные целыми. Таким образом, выражения типа `x=y+z` воспринимаются как арифметические. Группу команд можно объединить в функцию. Для этого существует ключевое слово `function`, после которого следует имя функции и список

команд, заключенных в фигурные скобки. Удалить функцию можно с помощью команды `unset` с флагом `-f`. Команда `typeset` имеет четыре опции для работы с функциями: `-f` — перечисляет определенные на текущий момент функции; `-ft` — при последующем вызове функции инициирует ее 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 одноименными именами функций, загружает его и вызывает эти функции. В переменные `mon` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобрать всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.