

# **Лабораторная работа No 10.**

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

Сарасбати Брасалес

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Выводы</b>	<b>13</b>
<b>6</b>	<b>Контрольные вопросы</b>	<b>14</b>
	<b>Список литературы</b>	<b>18</b>

# Список иллюстраций

4.1	script1 . . . . .	9
4.2	script1 . . . . .	9
4.3	script2 . . . . .	10
4.4	script2 . . . . .	10
4.5	script3 . . . . .	11
4.6	script3 . . . . .	11
4.7	script4 . . . . .	12
4.8	script4 . . . . .	12

## 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 Теоретическое введение

Командные процессоры (оболочки) 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; – C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation). 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

Использование арифметических вычислений. Операторы let и read Оболочка

bash поддерживает встроенные арифметические функции. 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 типа `radix#number`, где `radix` (основание системы счисления) — любое число не более 26. Для большинства команд используются следующие основания систем исчисления: 2 (двоичная), 8 (восьмеричная) и 16 (шестнадцатеричная). Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%). Команда `let` берет два операнда и присваивает их переменной. Положительным моментом команды `let` можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа `let sum=x+7`, и `let` будет искать переменную `x` и добавлять к ней 7. 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. Табл. 10.1 показывает полный набор `let`-операций. Подобно С оболочка `bash` может присваивать переменной любое значение, а произвольное выражение само имеет значение, которое может использоваться. При этом «ноль» воспринимается как «ложь», а любое другое значение выражения — как «истина»



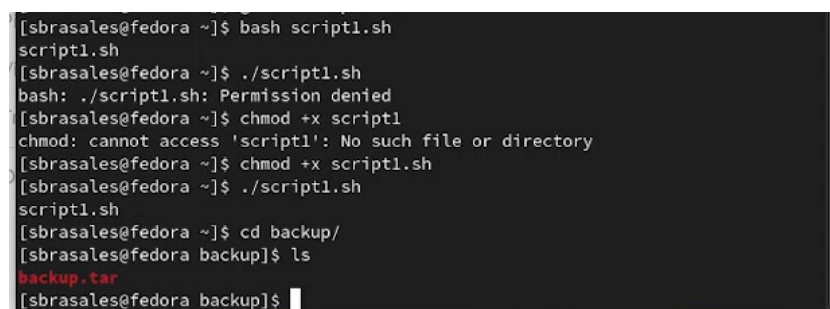
## 4 Выполнение лабораторной работы

Скрипт 1 (рис. 4.1) рис. 4.2).



```
1 #!/bin/bash
2 tar -cvf -backup/backup.tar script1
```

Рис. 4.1: script1



```
[sbrasales@fedora ~]$ bash script1.sh
script1.sh
[sbrasales@fedora ~]$ ./script1.sh
bash: ./script1.sh: Permission denied
[sbrasales@fedora ~]$ chmod +x script1
chmod: cannot access 'script1': No such file or directory
[sbrasales@fedora ~]$ chmod +x script1.sh
[sbrasales@fedora ~]$ ./script1.sh
script1.sh
[sbrasales@fedora ~]$ cd backup/
[sbrasales@fedora backup]$ ls
backup.tar
[sbrasales@fedora backup]$
```

Рис. 4.2: script1

Скрипт 2 (рис. 4.3) рис. 4.4).

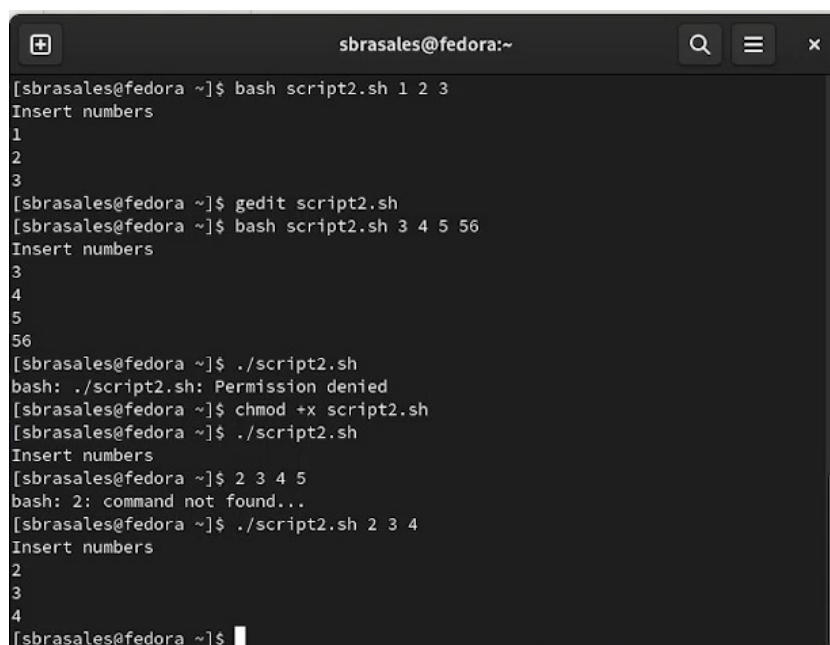


The screenshot shows a text editor window titled 'script2.sh' with a file icon, 'Open' button, and 'Save' button. The code is as follows:

```
1 #!/bin/bash
2 echo 'Insert numbers'
3 # read n
4 for A in $*
5 do echo $A
6 done
```

The status bar at the bottom indicates 'sh', 'Tab Width: 8', 'Ln 1, Col 5', and 'INS'.

Рис. 4.3: script2

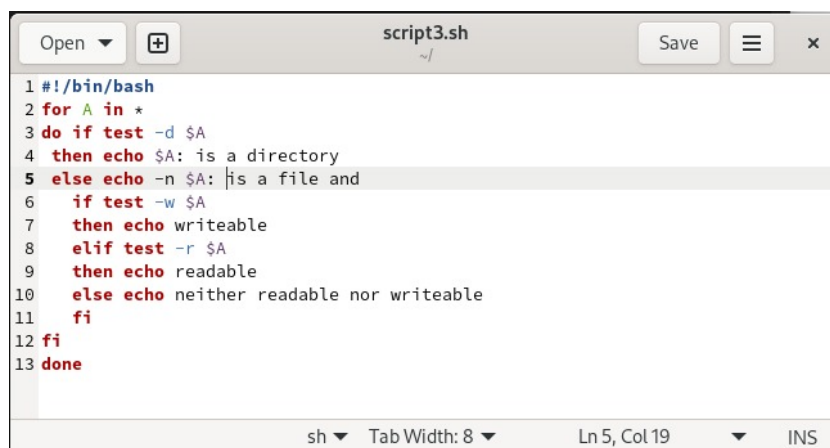


The screenshot shows a terminal window with the prompt 'sbrasales@fedora:~'. The following commands and their outputs are shown:

```
[sbrasales@fedora ~]$ bash script2.sh 1 2 3
Insert numbers
1
2
3
[sbrasales@fedora ~]$ gedit script2.sh
[sbrasales@fedora ~]$ bash script2.sh 3 4 5 56
Insert numbers
3
4
5
56
[sbrasales@fedora ~]$ ./script2.sh
bash: ./script2.sh: Permission denied
[sbrasales@fedora ~]$ chmod +x script2.sh
[sbrasales@fedora ~]$ ./script2.sh
Insert numbers
[sbrasales@fedora ~]$ 2 3 4 5
bash: 2: command not found...
[sbrasales@fedora ~]$ ./script2.sh 2 3 4
Insert numbers
2
3
4
[sbrasales@fedora ~]$
```

Рис. 4.4: script2

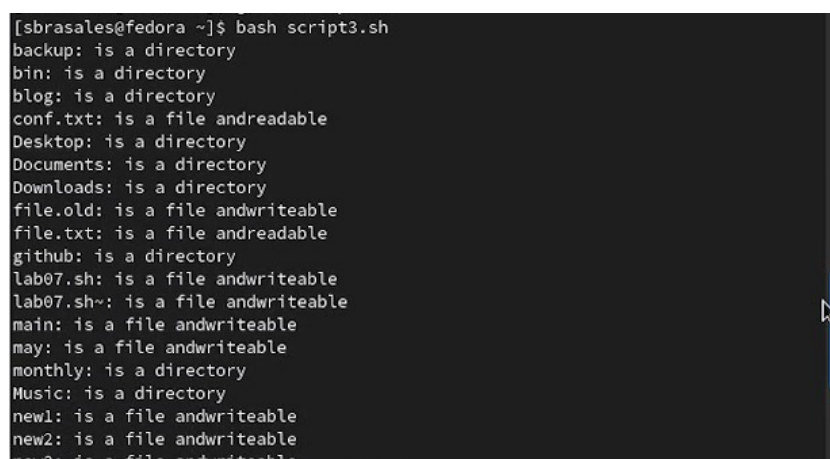
Скрипт 3 (рис. 4.5) (рис. 4.6).

A screenshot of a code editor window titled 'script3.sh'. The window has a menu bar with 'Open', 'Save', and a close button. The code is as follows:

```
1 #!/bin/bash
2 for A in *
3 do if test -d $A
4 then echo $A: is a directory
5 else echo -n $A: is a file and
6     if test -w $A
7 then echo writeable
8 elif test -r $A
9 then echo readable
10 else echo neither readable nor writeable
11 fi
12 fi
13 done
```

The status bar at the bottom shows 'sh', 'Tab Width: 8', 'Ln 5, Col 19', and 'INS'.

Рис. 4.5: script3

A screenshot of a terminal window showing the output of the script3.sh script. The prompt is '[sbrasales@fedora ~]\$ bash script3.sh'. The output lists various files and directories with their permissions:

```
[sbrasales@fedora ~]$ bash script3.sh
backup: is a directory
bin: is a directory
blog: is a directory
conf.txt: is a file andreadable
Desktop: is a directory
Documents: is a directory
Downloads: is a directory
file.old: is a file andwriteable
file.txt: is a file andreadable
github: is a directory
lab07.sh: is a file andwriteable
lab07.sh~: is a file andwriteable
main: is a file andwriteable
may: is a file andwriteable
monthly: is a directory
Music: is a directory
new1: is a file andwriteable
new2: is a file andwriteable
new3: is a file andwriteable
```

Рис. 4.6: script3

Скрипт 4 (рис. 4.7) (рис. 4.8).

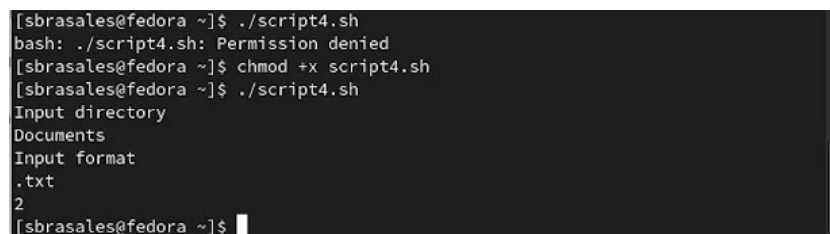


The screenshot shows a text editor window titled "script4.sh" with a file icon and "Open" and "Save" buttons. The script content is as follows:

```
1 #!/bin/bash
2 echo 'Input directory'
3 read directory
4 echo 'Input format'
5 read format
6 find ${directory} -maxdepth 1 -name "*${format}" -type f | wc -l
```

The status bar at the bottom indicates "sh", "Tab Width: 8", "Ln 6, Col 65", and "INS".

Рис. 4.7: script4



The screenshot shows a terminal window with the following output:

```
[sbrasales@fedora ~]$ ./script4.sh
bash: ./script4.sh: Permission denied
[sbrasales@fedora ~]$ chmod +x script4.sh
[sbrasales@fedora ~]$ ./script4.sh
Input directory
Documents
Input format
.txt
2
[sbrasales@fedora ~]$
```

Рис. 4.8: script4

## 5 Выводы

Мы изучили основы программирования в оболочке ОС UNIX/Linux и научились писать небольшие командные файлы.

## 6 Контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Че

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; – C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation) 2. Что такое POSIX? 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. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже. 3. Как определяются переменные и масси-

вы в языке программирования `bash`? 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 `$`. Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: `${имя переменной}`. Например, использование команд `b=/tmp/andy2 ls -l myfile > blssudoapt—getinstalltexlive—luatexls/tmp/andy—ls,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 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. Каково назначение операторов `let` и `read`? `let` - команда, после которой аргументы представляют собой выражение, подлежащее вычислению. `read` - команда, позволяющая читать переменные, вводимые с компьютера. 5. Какие арифметические операции можно применять в языке программирования `bash`? Простейшими математическими выражениями являются сложение (+), вычитание (-), умножение (\*), целочисленное деление (/) и целочисленный остаток от деления (%). Однако их намного больше 6. Что означает операция (( ))? Условия оболочки `bush`. 7. Какие стандартные имена переменных Вам известны? `PATH`; `PS1`; `PS2`; `HOME`; `IFS`; `MAIL`;

TERM; LOGNAME 8. Что такое метасимволы? Такие символы, как ' < > ? | " &, являются метасимволами и имеют для командного процессора специальный смысл. 9. Как экранировать метасимволы? 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 \$, ', , " 10. Как создавать и запускать командные файлы? Нужно просто создать файл через touch, а затем сделать его исполняемым через chmod +x/название 11. Как определяются функции в языке программирования bash? Указать ключевое слово function, затем написать её название и открыть фигурную скобку. 12. Каким образом можно выяснить, является файл каталогом или обычным файлом? Командой ls -lrt. Есть d, то каталог. 13. Каково назначение команд set, typeset и unset? set - установка переменных оболочек и сред unset - удаление переменной из командной оболочки typeset - наложение ограничений на переменные 14. Как передаются параметры в командные файлы? 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 командного 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. Назовите специальные переменные языка bash и их назначение. При использовании в командном файле комбинации символов \$# вместо неё будет осуществлена подстановка числа параметров, указанных в командной



строке при вызове данного командного файла на выполнение. Вот ещё несколько специальных переменных, используемых в командных файлах: – `$*` — отображается вся командная строка или параметры оболочки; – `$?` — 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; – `${name[n]}` — обращение к n-му элементу массива; – `${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`);

## **Список литературы**