

# **Лабораторная работа №10**

**НКАбд-03-22**

Шубнякова Дарья

# 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>8</b>
<b>5</b>	<b>Выводы</b>	<b>17</b>
	<b>Список литературы</b>	<b>24</b>

## Список иллюстраций

4.1	Программа . . . . .	9
4.2	Исполнение в терминале . . . . .	10
4.3	Программа . . . . .	11
4.4	Исполнение в терминале . . . . .	12
4.5	Программа . . . . .	13
4.6	Исполнение в терминале . . . . .	14
4.7	Программа . . . . .	15
4.8	Исполнение в терминале . . . . .	16

## Список таблиц

# 1 Цель работы

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

## 2 Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

### 3 Теоретическое введение

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.

## 4 Выполнение лабораторной работы

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.



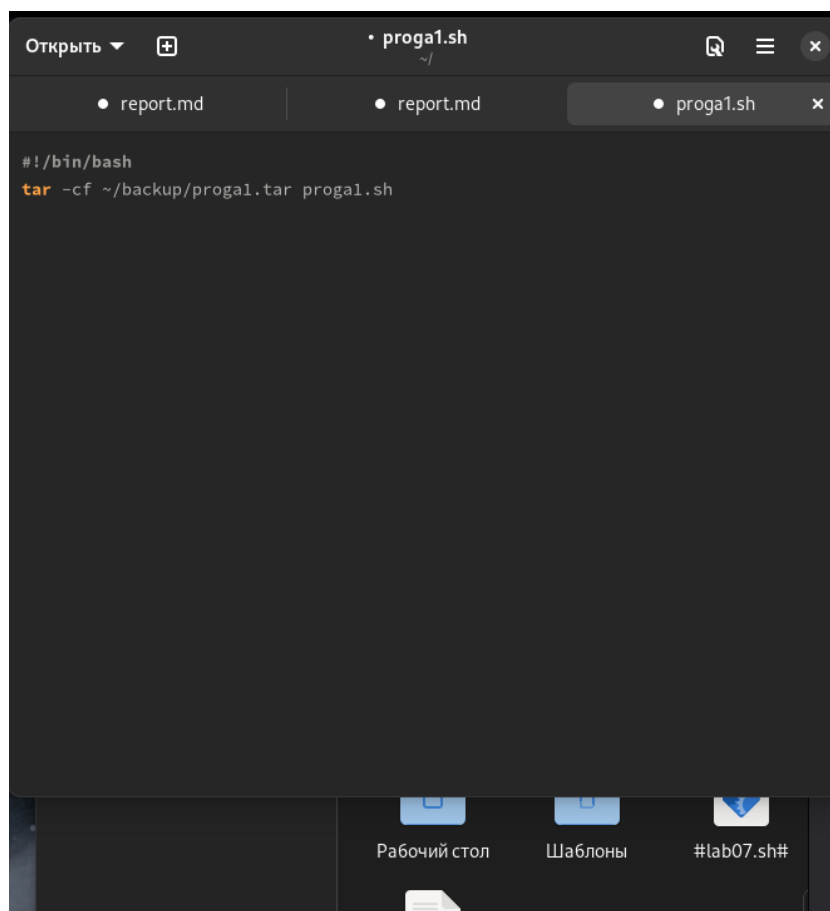


Рис. 4.1: Программа

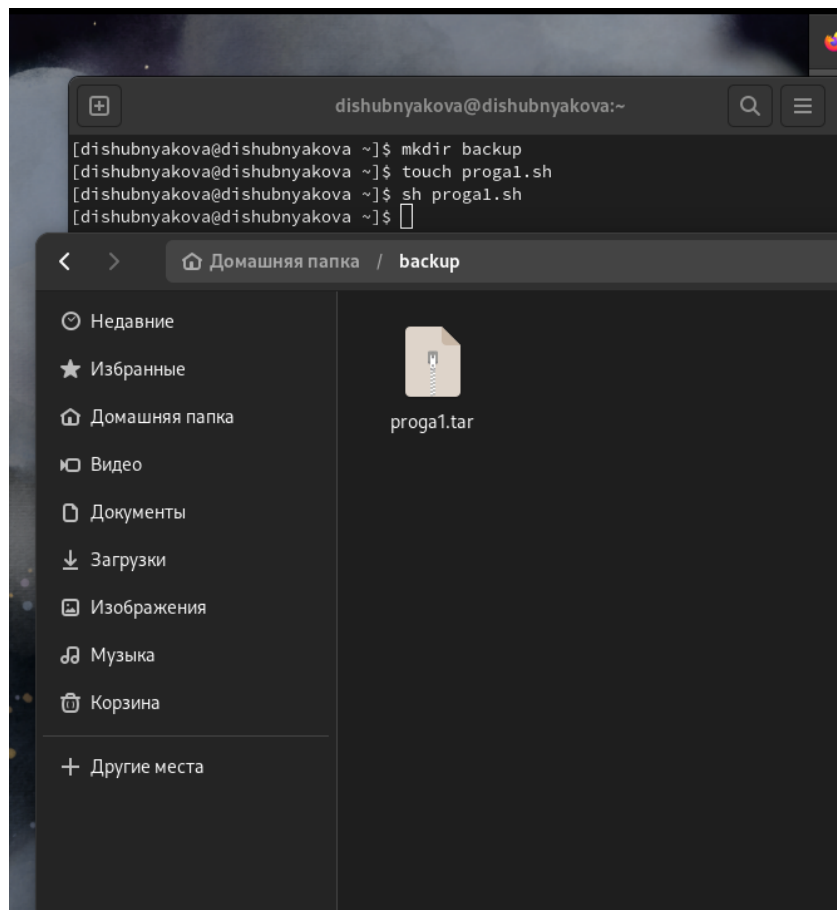


Рис. 4.2: Исполнение в терминале

Пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.

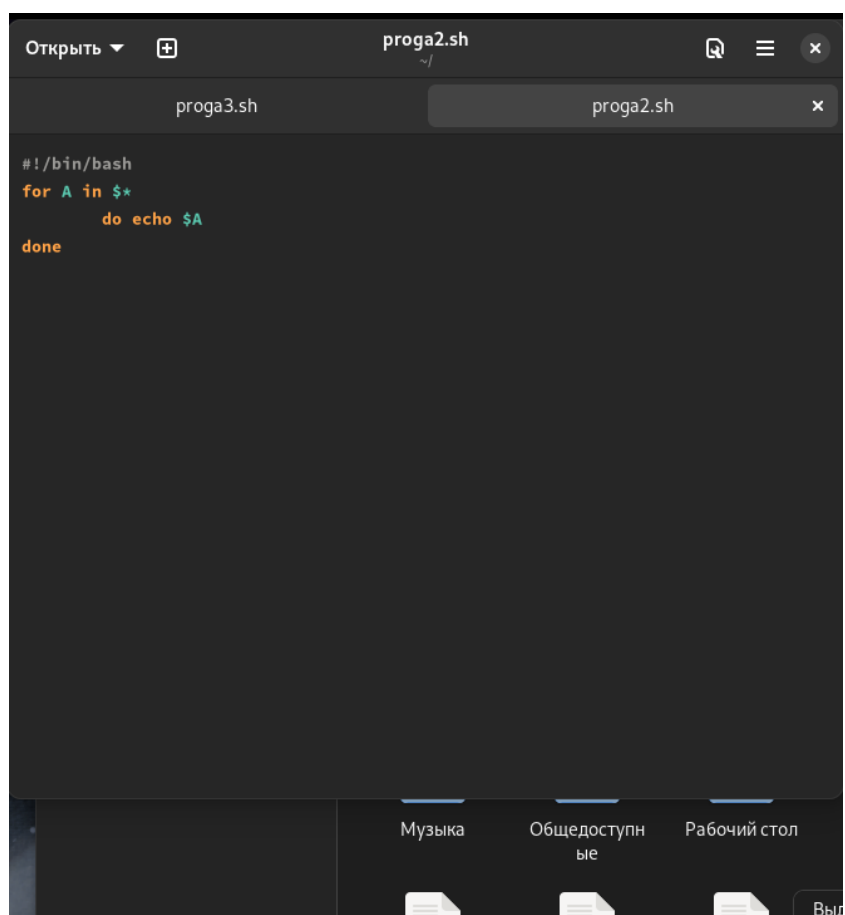


Рис. 4.3: Программа

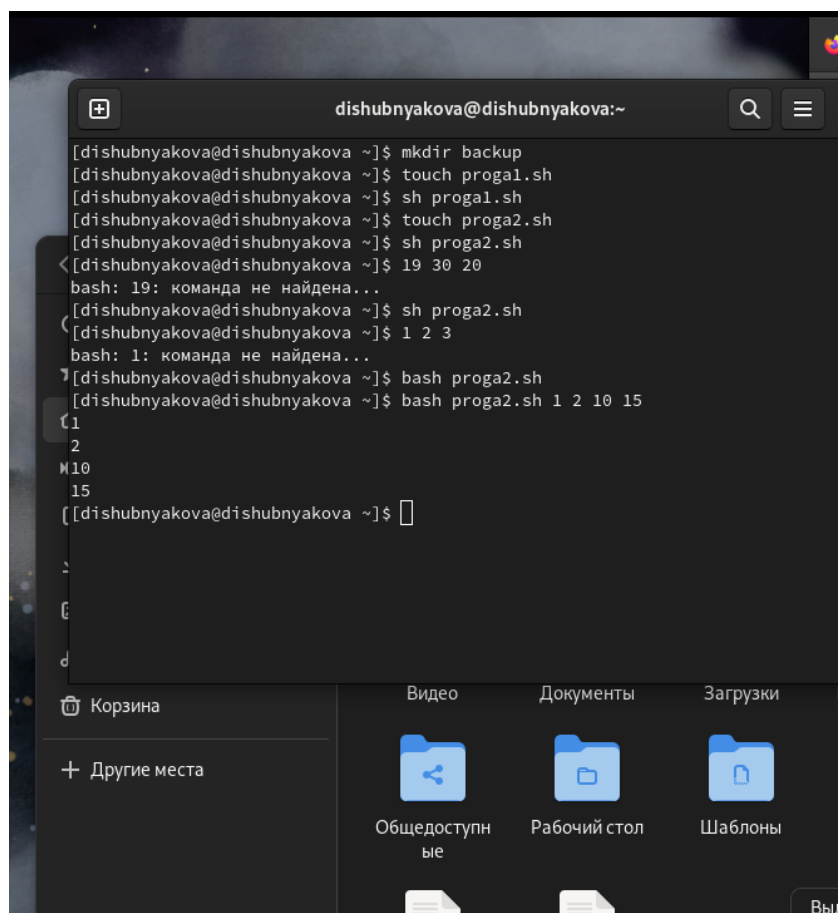


Рис. 4.4: Исполнение в терминале

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

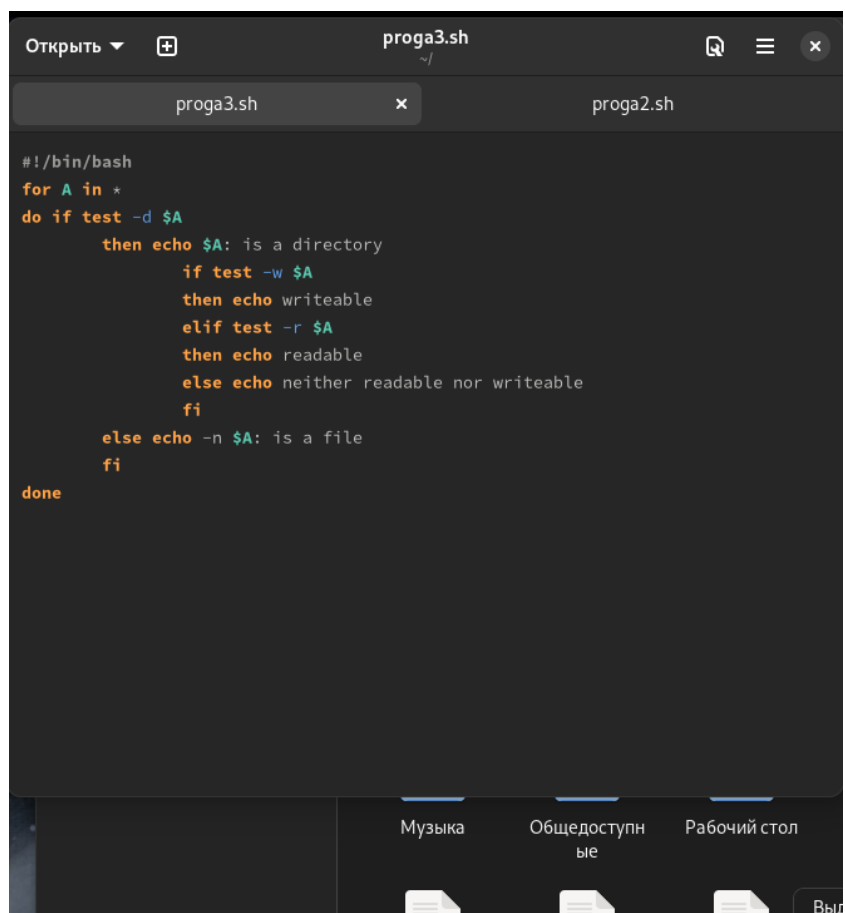
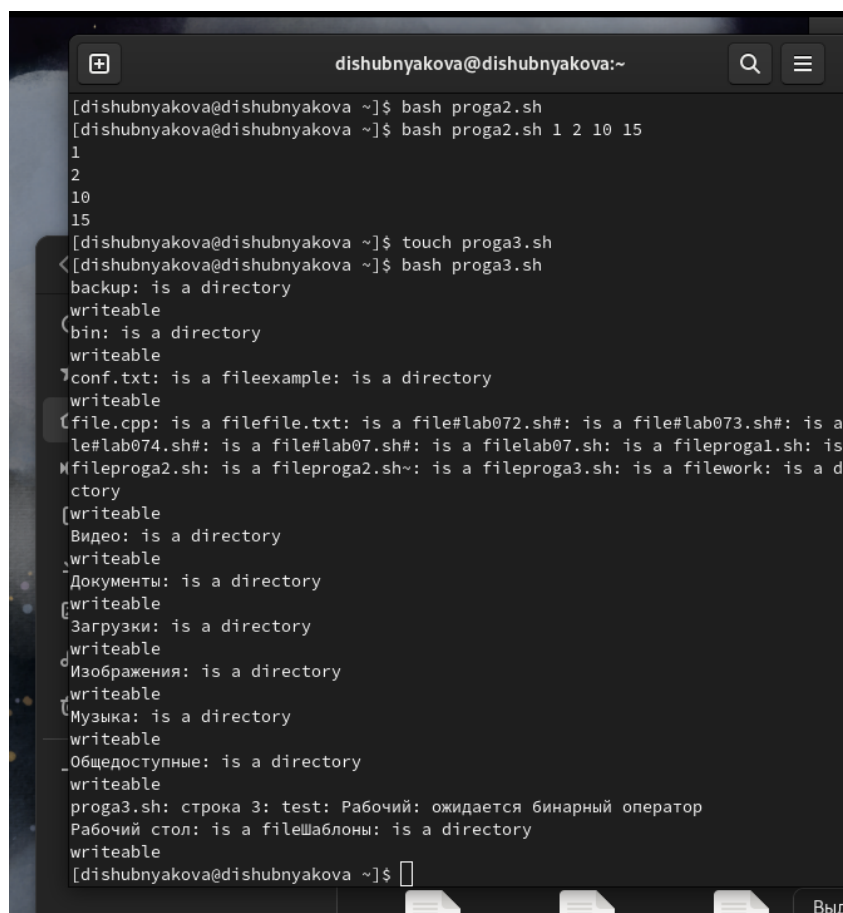


Рис. 4.5: Программа

A terminal window titled 'dishubnyakova@dishubnyakova:~' with search and menu icons in the top right. The terminal shows the following sequence of commands and outputs:

```
[dishubnyakova@dishubnyakova ~]$ bash proga2.sh
[dishubnyakova@dishubnyakova ~]$ bash proga2.sh 1 2 10 15
1
2
10
15
[dishubnyakova@dishubnyakova ~]$ touch proga3.sh
[dishubnyakova@dishubnyakova ~]$ bash proga3.sh
backup: is a directory
writeable
bin: is a directory
writeable
conf.txt: is a fileexample: is a directory
writeable
file.cpp: is a filefile.txt: is a filelab072.sh#: is a filelab073.sh#: is a
le#lab074.sh#: is a file#lab07.sh#: is a filelab07.sh: is a fileproga1.sh: is
Mfileproga2.sh: is a fileproga2.sh~: is a fileproga3.sh: is a filework: is a d
ctory
[writeable
Видео: is a directory
writeable
Документы: is a directory
writeable
Загрузки: is a directory
writeable
Изображения: is a directory
writeable
Музыка: is a directory
writeable
_Общедоступные: is a directory
writeable
proga3.sh: строка 3: test: Рабочий: ожидается бинарный оператор
Рабочий стол: is a fileШаблоны: is a directory
writeable
[dishubnyakova@dishubnyakova ~]$
```

Рис. 4.6: Исполнение в терминале

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.

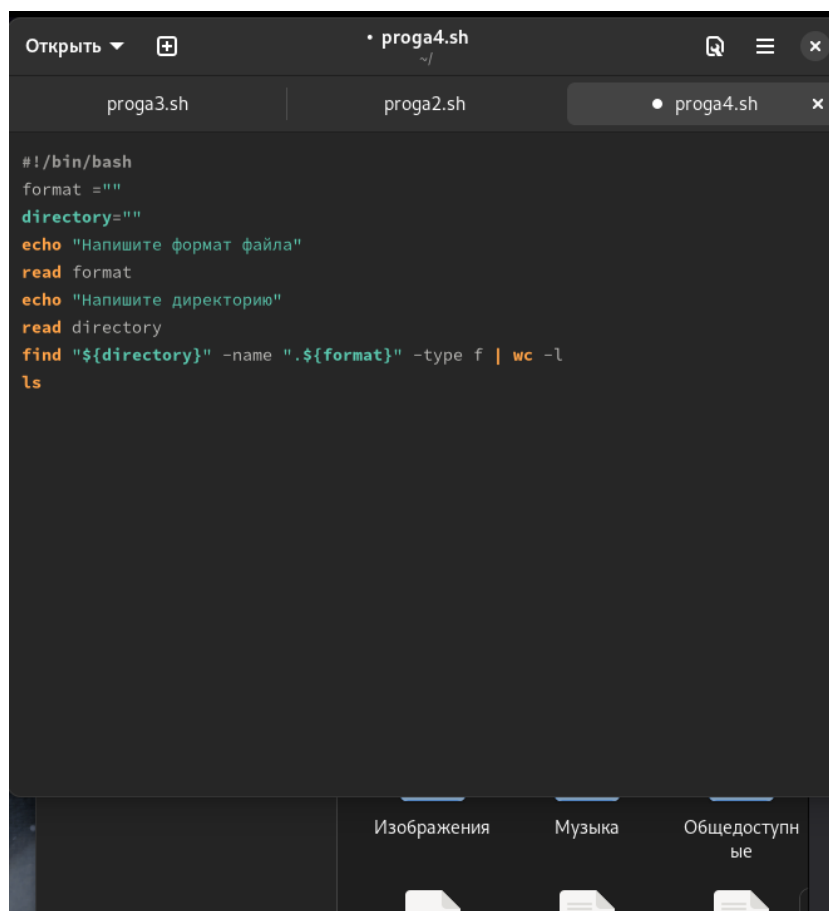


Рис. 4.7: Программа

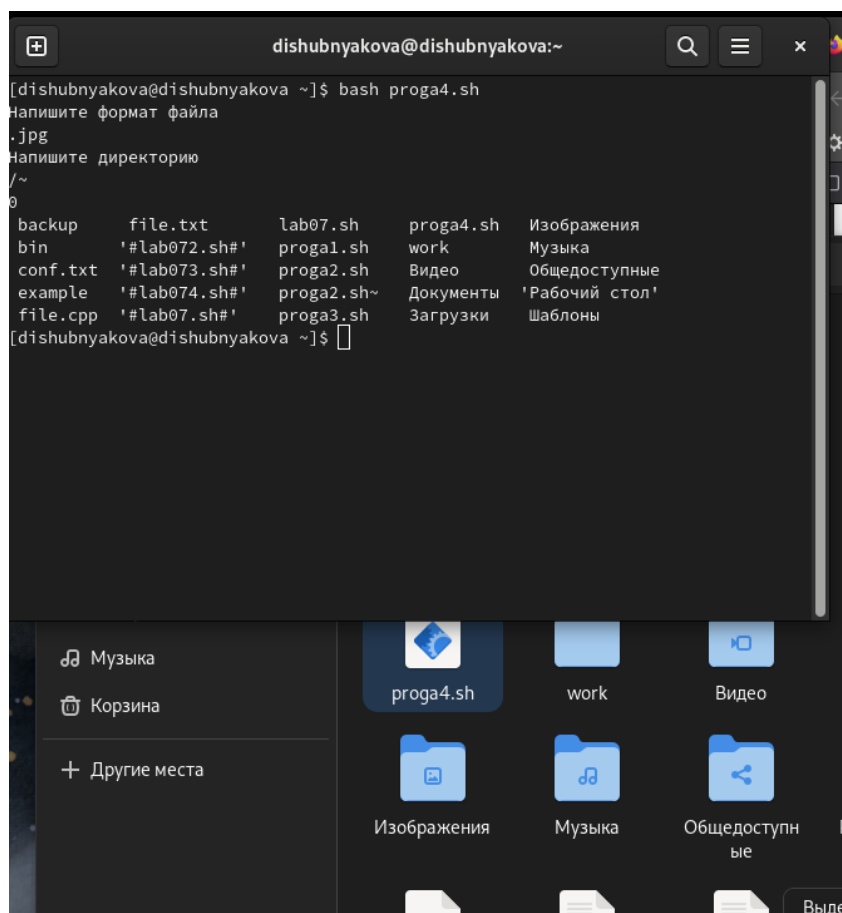


Рис. 4.8: Исполнение в терминале



## 5 Выводы

Научились писать небольшие командные файлы после изучения основ программирования в оболочке ОС UNIX/Linux. 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 С-подобный 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. 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 следующие 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. `!` *!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 *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`. 6. Условия оболочки `bash`. 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. Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа, который, в свою очередь, является метасимволом.

9. 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. 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 иницирует 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 `Cи` 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 иницирует 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 `top` и `day` будут считаны соответствующие значения, введенные с клавиатуры, а переменная `trash` нужна для того, чтобы отобразить всю избыточно введенную информацию и игнорировать ее. Изъять переменную из программы можно с помощью команды `unset`.

14. 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 нее 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 нее имени данного командного файла. Примере: пусть к командному файлу `where` имеется доступ по выполнению и этот командный файл содержит следующий конвейер: `who | grep $1` Если Вы введете с терминала команду: `where andy`, то в случае, если пользователь, зарегистрированный в ОС UNIX под именем `andy`, в данный момент работает в ОС UNIX, на терминал будет выведена строка, содержащая номер терминала, используемого указанным пользователем. Если же в данный момент этот пользователь не работает в ОС UNIX, то на терминал не будет выведено ничего. Команда `grep` производит контекстный поиск в тексте, поступающем со стандартного ввода, для нахождения в этом тексте строк, содержащих последовательности символов, переданные ей в качестве аргументов, и выводит результаты своей работы на стандартный вывод. В этом примере команда `grep` использует-ся как фильтр, обеспечивающий ввод со стандартного ввода и вывод всех строк, содержащих последовательность символов `andy`, на стандартный вывод. В ходе интерпретации этого файла командным процессором вместо комбинации симво-

лов \$1 осуществляется подстановка значения первого и единственного параметра *andy*. Если предположить, что пользователь, зарегистрированный в ОС UNIX под именем *andy*, в данный момент работает в ОС UNIX, то на терминале Вы увидите примерно следующее: *\$ where andy andy ttyG Jan 14 09:12 \$* Определим функцию, которая изменяет каталог и печатает список файлов: *\$ function clist { > cd \$1 > ls > }*. Теперь при вызове команды *clist* каталог будет изменен каталог и выведено его содержимое.

15. – \$ — отображается вся командная строка или параметры оболочки; – \$? — код завершения последней выполненной команды; – \$\$ — уникальный идентификатор процесса, в рамках которого выполняется команд- ный процессор; – \$! — номер процесса, в рамках которого выполняется последняя вызванная на выпол- нение в командном режиме команда; – \$- — значение фла- гов 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 пере- менных; – \${name:-value} — если значение переменной name не определе- но, то оно будет заме- нено 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 пе- ременная определена, то подставляется value; – \${name#pattern} — представ- ляет значение переменной name с удалённым самым коротким левым образцом (pattern); – \${#name[\*]} и \${#name[@]} — эти выражения возвращают количество элементов в массиве name.

## Список литературы

1. Dash P. Getting started with oracle vm virtualbox. Packt Publishing Ltd, 2013. 86 p.
2. Colvin H. Virtualbox: An ultimate guide book on virtualization with virtualbox. CreateSpace Independent Publishing Platform, 2015. 70 p.
3. van Vugt S. Red hat rhcsa/rhce 7 cert guide : Red hat enterprise linux 7 (ex200 and ex300). Pearson IT Certification, 2016. 1008 p.
4. Робачевский А., Немнюгин С., Стесик О. Операционная система unix. 2-е изд. Санкт-Петербург: БХВ-Петербург, 2010. 656 p.
5. Немет Э. et al. Unix и Linux: руководство системного администратора. 4-е изд. Вильямс, 2014. 1312 p.
6. Колисниченко Д.Н. Самоучитель системного администратора Linux. СПб.: БХВ-Петербург, 2011. 544 p.
7. Robbins A. Bash pocket reference. O'Reilly Media, 2016. 156 p.