

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики
Кафедра технологий программирования

Лабораторная работа №4
По курсу “Системное программирование”

Синтаксис языка Bash и разработка сценариев

Методические указания по выполнению лабораторной работы

Подготовила: Давидовская М.И.,
Ст. преподаватель кафедры ТП

Минск, 2023 г.

СОДЕРЖАНИЕ

Методические материалы	2
Командная оболочка bash	2
Что хранится в окружении?	2
Что такое сценарий командной оболочки?	3
Как написать сценарий командной оболочки?	4
Формат файла сценария	4
Разрешения на выполнение	5
Местоположение файла сценария	5
Задания лабораторной работы	7
Критерии оценивания	7
Содержание отчета	8
Требования к отчёту	8
1 Примеры для изучения	9
1.1 Исполнительная среда оболочки	9
1.2 Исполнение команд	10
1.3 Сценарии Bash	11
1.4 Программирование Bash	11
2 Задания для самостоятельной работы	15
2.1 Исполнительная среда оболочки	15
2.2 Исполнение команд	16
2.3 Программирование сценариев Bash: общие задания	16
2.4 Программирование сценариев Bash: индивидуальные задания	16
Варианты	17
2.5 Программирование сценариев Bash: творческое задание	21

Методические материалы

Командная оболочка bash

Оболочка — это интерпретатор команд. Это больше, чем просто изолирующий слой между ядром операционной системы и пользователем, это довольно мощный язык программирования.

Что хранится в окружении?

Командная оболочка хранит в окружении данные двух основных типов, хотя bash практически не делает различий между типами. Эти данные хранятся в переменных окружения и в переменных командной оболочки. **Переменные командной оболочки** — это фрагменты данных, инициализируемые командой bash, а **переменные окружения** — практически все остальное. Помимо переменных командная оболочка хранит также программируемые данные, а именно псевдонимы и функции командной оболочки.

Можем использовать окружение для настройки некоторых параметров командной оболочки. Для этого используются следующие команды:

- `printenv` — выводит часть или все окружение;
- `set` — устанавливает параметры командной оболочки;
- `export` — экспортирует окружение для программ, которые будут выполняться;
- `alias` — создает псевдоним команды.

В таблице представлены примеры переменных окружения:

Переменная	Содержит
DISPLAY	Имя вашего дисплея, если вы работаете в графическом окружении. Обычно это <code>:0</code> , что означает первый дисплей, сгенерированный X сервером.
EDITOR	Имя программы, используемой в качестве текстового редактора.
SHELL	Имя командной оболочки.
HOME	Путь к домашнему каталогу.
LANG	Определяет набор символов и порядок сортировки для вашего языка.
PATH	Список каталогов, разделенных двоеточием, в которых производится поиск выполняемых программ по их именам.
PS1	Строка приглашения к вводу No 1. Определяет содержимое строки приглашения к вводу в командной оболочке.
PWD	Текущий каталог.
TERM	Тип терминала. Unix-подобные системы поддерживают множество протоколов для работы с терминалами; эта переменная определяет протокол, который будет использоваться при обмене данными с эмулятором терминала.
USER	Имя пользователя.

Что такое сценарий командной оболочки?

Что такое сценарий командной оболочки? Выражаясь простым языком, **сценарий командной оболочки** — это файл, содержащий последовательность команд. Командная оболочка читает этот файл и выполняет команды, как если бы они вводились вручную в командной строке. **Командная оболочка** — это одновременно и мощный **ин-**

терфейс командной строки системы, и **интерпретатор языка сценариев**. Как вы увидите далее, многое из того, что можно сделать в командной строке, также можно сделать в сценариях, а многое из того, что можно сделать в сценариях, можно сделать в командной строке. Практически весь набор команд, утилит и инструментов Linux доступен для вызова с помощью сценария оболочки. Если этого не достаточно, то внутренние команды оболочки, такие как конструкции условий и циклов, придают сценариям дополнительную мощь и гибкость. Сценарии оболочки особенно хорошо подходят для решения задач управления системой и других рутинных, повторяющихся, задач.

Как написать сценарий командной оболочки?

Чтобы успешно создать и запустить сценарий командной оболочки, нам нужно:

1. **Написать сценарий.** *Сценарии командной оболочки* — это обычные текстовые файлы. Поэтому для их создания нам понадобится текстовый редактор. Лучше использовать текстовый редактор, обладающий функцией подсветки синтаксиса, позволяющей видеть элементы сценариев с цветной маркировкой. Подсветка синтаксиса помогает замечать некоторые типичные ошибки. Для создания сценариев хорошо подходят vim, gedit, kate и многие другие редакторы.
2. **Сделать сценарий выполняемым.** Система не позволяет интерпретировать любой старый текстовый файл как программу, и небезосновательно! Поэтому, чтобы выполнить сценарий, файлу сценария нужно дать разрешения на выполнение.
3. **Поместить сценарий в каталог, где командная оболочка сможет найти его.** Командная оболочка автоматически выполняет поиск выполняемых файлов в нескольких каталогах, если путь к файлу не указан явно. Для максимального удобства необходимо помещать ваши сценарии в такие каталоги.

Формат файла сценария

Следуя традициям программирования, напомним программу «hello world», чтобы продемонстрировать чрезвычайно простой сценарий. Итак, запустите текстовый редактор и введите следующий сценарий:

```
#!/bin/bash
# Это наш первый сценарий
echo 'Hello world!'
```

Последняя строка в сценарии хорошо знакома — это простая команда echo со строковым аргументом. Вторая строка — комментарий. Комментарии можем использовать и в командной строке:

```
$ echo 'Hello world!' # Это тоже комментарий
Hello world!
```

```
maryia@ThinkPad-E470:~$ echo 'Hello World!' # Это тоже комментарий
Hello World!
maryia@ThinkPad-E470:~$
```

Первая строка в сценарии смотрится несколько необычно. Она похожа на комментарий, потому что начинается с символа #, но выглядит какой-то уж слишком специальной, чтобы быть комментарием. Последовательность символов #! — это на самом деле специальная конструкция, которая называется *shebang* (произносится как «ше-банг») и сообщает системе имя интерпретатора, который должен использоваться для выполнения следующего за ним текста сценария. Каждый сценарий командной оболочки должен включать это определение в первой строке.

Сохраните файл сценария из примера выше с именем *hellow*.

Разрешения на выполнение

Далее сделаем сценарий исполняемым при помощи команды `chmod`. Просмотрим текущие права доступа

```
$ ls -l hellow
```

```
-rw-r--r-- 1 maryia maryia 79 вер 10 08:24 hellow
```

```
maryia@ThinkPad-E470:~$ ls -l hellow
-rw-r--r-- 1 maryia maryia 79 вер 10 08:24 hellow
```

Существует два распространенных набора разрешений для сценариев: **755** — для сценариев, которые должны быть доступны для выполнения всем, и **700** — для сценариев, которые могут выполняться только владельцами.

Добавим права на выполнение, используя команду `chmod`:

```
$ chmod 755 hellow
```

```
$ ls -l hellow
```

```
-rwxr-xr-x 1 maryia maryia 79 вер 10 08:24 hellow
```

```
maryia@ThinkPad-E470:~$ chmod 755 hellow
maryia@ThinkPad-E470:~$ ls -l hellow
-rwxr-xr-x 1 maryia maryia 79 вер 10 08:24 hellow
maryia@ThinkPad-E470:~$
```

Обратите внимание, что сценарии необходимо сделать доступными для чтения, чтобы их можно было выполнить.

Местоположение файла сценария

После установки разрешений попробуем запустить сценарий:

```
$ ./hellow
```

```
maryia@ThinkPad-E470:~$ ./hellow  
Hello World!  
maryia@ThinkPad-E470:~$
```

Но чтобы это сделать, необходимо добавить явный путь перед его именем. В противном случае мы получим следующее сообщение:

```
$ hellow
```

```
bash: hellow: команда не найдена
```

В чем причина? Чем наш сценарий отличается от других программ? Как оказывается, ничем. У нас замечательный сценарий. Его проблема — местоположение.

Напомним, что система просматривает каталоги по списку всякий раз, когда требуется найти исполняемую программу, если путь к ней не указан явно. Именно так система выполняет программу `/bin/ls`, если мы вводим `ls` в командной строке. Каталог `/bin` — один из каталогов, которые система просматривает автоматически. Список каталогов хранится в переменной окружения `PATH`. Она содержит список каталогов, перечисленных через двоеточие. Увидеть, что содержится в `PATH`, можно с помощью команды:

```
$ echo $PATH
```

```
/home/maryia/bin:/home/maryia/.local/share/umake/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/lib/jvm/java-12-oracle/bin:/usr/lib/jvm/java-12-oracle/db/bin
```

Получим список каталогов. Если поместить сценарий в любой из этих каталогов, наша проблема будет решена. Обратите внимание на первый каталог в списке, `/home/maryia/bin`. В большинстве дистрибутивов Linux в переменную `$PATH` включается каталог `bin` в домашнем каталоге пользователя, чтобы дать пользователям возможность выполнять собственные программы.

Если каталога `bin` в домашнем каталоге в списке путей окружения нет, то достаточно создать его, добавить в список путей, поместить сценарий в него и можно будет запускать сценарий как любые другие программы:

```
$ mkdir bin
```

```
$ mv hellow bin
```

```
$ hellow
```

```
Hello World!
```

Если каталог отсутствует в переменной `PATH`, его легко туда добавить, включив следующую строку в файл `.bashrc`:

```
$ export PATH="$HOME/bin:$PATH"
```

Команда `export` экспортирует измененную переменную в дочерние среды процессов оболочки. Теперь вы можете запускать ваши скрипты, просто набрав имя исполняемого скрипта без указания полного пути к исполняемому файлу.

Однако это изменение носит временный характер и действует только в текущем сеансе оболочки.

Чтобы сделать изменение постоянным, вам нужно определить переменную `$PATH` в файлах конфигурации оболочки.

В большинстве дистрибутивов Linux при запуске нового сеанса переменные среды считываются из следующих файлов:

- Конфигурационные файлы глобальной оболочки, такие как `/etc/profile`. Используйте этот файл, если вы хотите, чтобы новый каталог был добавлен всем системным пользователям `$PATH`.
- Конфигурационные файлы для отдельных пользовательских оболочек. Например, если вы используете Bash, вы можете установить переменную `$PATH` в файле `~/.bashrc`, а если вы используете Zsh – имя файла `~/.zshrc`.

В этом примере мы установим переменную в файле `~/.bashrc`. Откройте файл в текстовом редакторе и добавьте в конце следующую строку:

```
export PATH="$HOME/bin:$PATH"
```

Чтобы применить изменения в текущем сеансе, нужно заставить командную оболочку повторно прочитать файл `bashrc`, например, так:

```
$ . ~/.bashrc
```

Или так:

```
$ source ~/.bashrc
```

Команда «точка» (`.`) в первом примере является синонимом `source`, встроенной команды, которая читает указанный файл и интерпретирует его как ввод с клавиатуры.

Дополнительные материалы по командной оболочке bash и разработке сценариев опубликованы в [теме 4](#) курса «Системное программирование».

Задания лабораторной работы

Критерии оценивания

Оценка 4

Выполнены все примеры и задания 2.1-2.3. Представлены файлы протоколов команд и меток времени с недочетами, исходный код скриптов. Лабораторная работа сдана с поддержкой в 2 недели.

Оценка 5-6

Выполнены все примеры и задания 2.1-2.3 и задачи 2 и 4 из задания 2.4. Представлен отчет, файлы протоколов и меток времени, код скриптов в git-репозитории. Отчет, файлы протоколов команд и меток времени без ошибок. Лабораторная работа сдана с задержкой в 1 неделю.

Оценка 7-8

Выполнение примеров не требуется. Выполнены задания 2.1-2.5. Представлен отчет, ответы на контрольные вопросы, файлы протоколов и меток времени, исходные коды скриптов в git-репозитории. Отчет, файлы протоколов команд и меток времени могут содержать незначительные ошибки. Лабораторная работа сдана с задержкой в 1 неделю.

Оценка 9

Выполнение примеров не требуется. Выполнены задания 2.1-2.5 на отличном уровне. Представлен отчет, ответы на контрольные вопросы и файлы протоколов и меток времени, исходные коды скриптов в git-репозитории. Отчет, файлы протоколов команд и меток времени не содержат ошибок. Лабораторная работа сдана в срок.

Содержание отчета

1. Цель работы.
2. Вариант задания.
3. Протоколы выполненных действий.
4. Исходные код скриптов

Отчет должен быть опубликован в git-репозитории.

Требования к отчёту

В файле Readme проекта на github должна быть ссылка на отчёт с результатами выполнения упражнений 1.1, 1.2 примеров и 2.1, 2.2 задания для самостоятельной работы и в папке labrabota4 на файл/ы с результатом записи команды script. Отчет опубликовать в репозитории в папке docs или во внешнем хранилище, добавив ссылку на него в файл Readme репозитория. Отчёт должен содержать скриншоты и описания этапов выполнения лабораторной работы с примерами команд.

Перед выполнением упражнений задания для самостоятельной работы включите ведение протокола командой script с журналом меток времени. Протокол назвать по следующему шаблону — taskXФамилия, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами. Журнал меток назвать по следующему шаблону — timelogXФамилия, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами.

1 Примеры для изучения

1.1 Исполнительная среда оболочки

1. Войдите в систему под учетной записью student.
2. Выполните следующие команды, которые используют переменные, full_name и short_name командной оболочки Bash.

```
$ a=879
$ echo "The value of \"a\" is $a."
$ export full_name="Ivan Petrov"
$ echo $full_name
$ export short_name="Ivan"
$ export short_name
$ echo $short_name
```

3. Выполните в командной строке следующие команды, которые иллюстрируют операции над переменными

```
$ export foo=""
$ echo ${foo:-one}
$ echo $foo
$ echo ${foo:=one}
$ echo $foo
$ export foo="this is a test"
$ echo $foo
$ echo ${foo:+bar}
```

4. Выполните в командной строке следующие команды, которые создают переменные массивы.

```
$ ARR[1]=one
$ ARR[2]=two
$ ARR[3]=three
$ ARR1=(zero one two three)
```

5. Выполните следующую команду, которая выводит на консоль значения элементов массива.

```
$ echo ${ARR[0]} ${ARR[1]}
```

6. Выполните в командной строке следующие команды для удаления элемента ARR[1] массива ARR и для удаления всего массива ARR:

```
$ unset ARR[1]
$ unset ARR
```

7. Создайте псевдоним для команды clear, выполнив следующую команду:

```
$ alias c='clear'
```

Просмотрите созданные псевдонимы.

```
$ alias
```

Примените псевдоним команды

```
$ c
```

8. Этот псевдоним будет утерян, если вы выйдете из командной оболочки, а затем войдете в нее вновь. Чтобы обеспечить сохранность псевдонима при каждом входе в командную оболочку пользователя student, нужно выполнить следующие действия.

Откройте файл `.bashrc` в текстовом редакторе, например `vim`

```
$ vim .bashrc
```

Найдите в файле `bashrc` строку, которая содержит текст:

```
# User specific aliases and functions.
```

Добавьте после этой строки следующую строку: `alias`

```
c='clear'
```

Сохраните файл и выйдите из текстового редактора.

9. Чтобы проверить сохранение псевдонима, сначала выйдите из командной оболочки, а затем снова войдите в неё под именем `student` и выполните команду

```
$ alias
```

```
$ c
```

10. Отобразите в терминале текущее значение вашей строки приглашения, выполнив команду

```
$ echo $PS1
```

11. Измените вашу строку приглашения, выполнив следующую команду:

```
$ PS1='Linux ->'
```

12. Восстановите традиционную строку приглашения, выполнив следующую команду. Строка корректно восстановится в терминале, в приложении `xterm` — без цветового оформления:

```
$ PS1='\u@\h:\w\a\]$'
```

1.2 Исполнение команд

1. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором фигурных скобок. В некоторых версиях командного интерпретатора приведенная ниже команда может быть выполнена с опцией `-E`, чтобы расширение фигурных скобок было выполнено:

```
$ echo sp{el,il,al, oo}l
```

```
spell spill spall spool
```

2. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором символа тильда на домашний каталог пользователя.

```
$ cat ~/message.txt
```

3. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором переменной.

```
$ echo $SHELL /bin/bash
```

4. Выполните следующую команду, которая иллюстрирует расширение командным интерпретатором команд.

```
$ echo $(date)
```

5. Объявите переменные `x` и `y` и задайте им некоторые значения. Выполните затем команду, которая иллюстрирует расширение командным интерпретатором арифметического выражения.

```
$ x=15
```

```
$ y=123
```

```
$ echo Area: [$x * $y]
```

6. Выполните следующую команду, которая иллюстрирует запрещение расширения символа, который следует за символом обратный слэш.

```
$ echo Your cost: \$5,00
```

7. Выполните следующую команду, которая иллюстрирует запрещение расширения символов, заключённых в одинарные кавычки.

```
$ echo '$date'
```

```
$ date
```

1.3 Сценарии Bash

1. Создайте текстовый файл *ascript.sh*, который содержит следующий текст.

```
#!/bin/bash
```

```
# This script displays some information about your environment
```

```
echo "Hello."
```

```
echo "The date and time are $(date)" echo
```

```
"Your working directory is: $(pwd)"
```

2. Сохраните файл в текущем каталоге.

3. Сделайте файл *ascript.sh* исполняемым, выполнив команду

```
# chmod u+x ascript.sh
```

4. Запустите сценарий из командной строки, выполнив команду

```
# ./ascript.sh
```

5. Создайте и запустите следующий сценарий, который представляет бесконечный процесс, выводящий значение счетчика каждую секунду и завершающий свою работу при нажатии клавиш <Ctrl+c>.

```
#!/bin/bash
```

```
# trap test
```

```
trap 'echo you hit Ctrl+c; exit' SIGINT
```

```
echo "This is a test script for using trap"
```

```
count=0 while [ $count -le 10 ]
```

```
do
```

```
sleep 1
```

```
count=$((exp $count + 1))
```

```
echo "Проход цикла #$count"
```

```
done
```

1.4 Программирование Bash

1. Создайте и выполните следующий скрипт, который поясняет использование команды *read* для ввода данных с терминала.

```
#!/bin/bash
```

```
echo -n "Enter your name and press [ENTER]: "
```

```
read var_name
```

```
echo "Your name is: $var_name"
```

2. Выполните в командной строке следующие команды, которые иллюстрируют использование команды *printf* для форматированного вывода данных.

```
# printf "%d\n" 5
```

```
# printf "Hello, $USER.\n\n"
```

```
# distance=15
# printf "Distance is %5d km\n" $distance
# printf "%d\n"
# printf "0x%X\n " 15
# printf "0x%X\n " 22
```

3. Создайте и выполните следующий скрипт, который иллюстрирует использование условной инструкции if-then-else, сравнивая значения двух переменных.

```
#!/bin/bash
```

```
V1="foo"
```

```
V2="bar"
```

```
if [ "$V1" = "$V2" ]; then
    echo true
else
    echo false
fi
```

4. Создайте и выполните следующий скрипт, который объясняет использование условной инструкции case для выбора нужного варианта значения переменной.

```
#!/bin/bash
```

```
printf "Which Linux distribution do you know?\n\n"
```

```
read DISTR
```

```
case $DISTR
in ubuntu) echo "I know ubuntu." ;;
centos|rhel) echo "I know it too.>";;
windows) echo "Very funny." ;;
*) echo "I don't know it." ;;
esac
```

5. Создайте и выполните три следующих скрипта, которые иллюстрируют использование инструкции цикла for для числовых и символьных значений, а также показывает, как программируются бесконечные циклы for.

```
# script1.sh
```

```
#!/bin/bash
```

```
for (( c=1; c<=3; c++ ))
do echo "$c iteration"
done
```

```
# script2.sh
```

```
#!/bin/bash
```

```
# Loop through a set of strings:
```

```
for m in 'Apple Sony Panasonic "Hewlett Packard" Nokia'
do echo "Manufacturer is:" $m
done
```

```
# script3.sh
#!/bin/bash
for (( ; ; ))
do echo "infinite loops [hit CTRL+C to stop]"
done
```

6. Создайте и выполните следующий скрипт, который иллюстрирует использование инструкций циклов while и until.

```
#!/bin/bash
# Цикл while
echo "Цикл while"
x=1

while [ $x -le 4 ]
do
echo "x = $x"
x=$(( $x + 1 ))
done

# Цикл until
echo -e "\nЦикл until"
x=5
until [ $x -le 3 ]
do
echo "x = $x"
x=$(( $x - 1 ))
done
```

7. Создайте и выполните следующий скрипт, который иллюстрирует использование команды принудительного выхода из цикла continue.

```
#!/bin/bash
for myloop in 1 2 3 4 5

do

if [ "$myloop" -eq 3 ]; then
echo -e "To skip iteration No. 3\n"
continue
# Skip rest of this particular loop iteration

fi

echo -n -e "Iteration #$myloop \n"

done
```

8. Создайте и выполните следующий скрипт, который иллюстрирует использование команды принудительного завершения цикла break.

```
#!/bin/bash
```

```

for myloop in 1 2 3 4 5
do
echo -n -e "Iteration #${myloop}\n"
if [ "$myloop" -eq 3 ]; then
echo -e "\nTo break loop after 3 iterations\n"
break
# This line will break out of the loop
fi
done

```

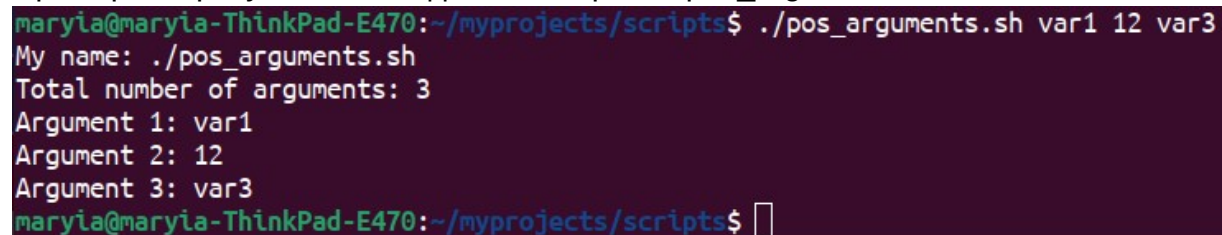
9. Создайте и выполните следующий скрипт, который выводит свое имя файла, количество аргументов и значения первых трёх аргументов, иллюстрируя тем самым обработку аргументов, которые передаются скрипту при вызове. При вызове этого скрипта передайте ему три произвольных аргумента.

```

#!/bin/bash
echo "My name: $0"
echo "Total number of arguments: $# "
echo "Argument 1: $1"
echo "Argument 2: $2"
echo "Argument 3: $3"

```

Примерный результат вывода, если скрипта pos_arguments.sh



```

maryia@maryia-ThinkPad-E470:~/myprojects/scripts$ ./pos_arguments.sh var1 12 var3
My name: ./pos_arguments.sh
Total number of arguments: 3
Argument 1: var1
Argument 2: 12
Argument 3: var3
maryia@maryia-ThinkPad-E470:~/myprojects/scripts$ 

```

10. Создайте и выполните следующий скрипт, который выводит все свои аргументы, иллюстрируя тем самым обработку произвольного количества аргументов. Вызовите этот скрипт, передав ему произвольное количество аргументов.

```

#!/bin/bash

numargs=$#

for ((i=1 ; i <= numargs ; i++))
do
echo "$1"
shift
done

```

Примерный вывод представлен на скриншоте:

```
maryia@maryia-ThinkPad-E470:~/myprojects/scripts$ ./pos_arguments2.sh 12 43 324 dd 22 ss 292 fds ss
12
43
324
dd
22
ss
292
fds
ss
maryia@maryia-ThinkPad-E470:~/myprojects/scripts$
```

11. Объявите и вызовите функцию `mcd`, которая создает каталог и делает его текущим. В результате применения функции будет создан каталог `temp` и выполнен переход в него:

```
$ function mcd() { mkdir $1 && cd $1; }
$ mcd temp
```

2 Задания для самостоятельной работы

2.1 Исполнительная среда оболочки

1. Войдите в систему под своей учетной записью.
2. Создайте переменную `temp`, присвойте ей значение `old` и выведите значение этой переменной на терминал.
3. Присвойте переменной `temp` значение `new` и выведите значение этой переменной на терминал.
4. Добавьте к значению переменной `temp` значение `and new` и выведите значение этой переменной на терминал.
5. Удалите переменную `temp`.
6. Создайте переменную-массив `vect` и присвойте элементам массива значения `x`, `y` и `z`. Выведите значения элементов массива `vect` на терминал.
7. Замените значение элемента `z` на значение `w`. Выведите значения элементов массива `vect` на терминал.
8. Удалите массив-переменную `vect`.
9. Создайте псевдоним `lr`, который вызывает команду `ls` со следующими возможностями:
 - псевдоним перечисляет файлы в длинном формате;
 - псевдоним перечисляет файлы, которые начинаются с точки;
 - псевдоним классифицирует файлы, присоединяя индикатор типа файла к именам файлов;
 - псевдоним перечисляет файлы в порядке времени их модификации.
10. Используя команду `notify-send`, отправить текущее время и календарь на текущий месяц в качестве уведомления в сеансе графического интерфейса.

2.2 Исполнение команд

1. Используя расширение командным интерпретатором фигурных скобок, создайте одной командой в текущем каталоге следующие файлы:

```
report_1  
report_2  
report_3  
request_1  
request_2  
request_3
```

Проверьте содержимое текущего каталога.

2. Создайте в текущем каталоге подкаталог `report`.
3. Используя расширения командный интерпретатором, переместите одной командой файлы `report_1`, `report_2` и `report_3` в созданный каталог `report`.
4. Используя расширения командный интерпретатором, удалите одной командой файлы `request_1`, `request_2` и `request_3` из текущего каталога.
5. Выполните в команде `echo` команду, которая выводит на терминал содержимое каталога `report`.

2.3 Программирование сценариев Bash: общие задания

1. Создайте и запустите сценарий, который представляет бесконечный процесс, выводящий значение счетчика каждые две секунды, а через 10 секунд сбрасывающий его значение в 0, и завершающий свою работу при нажатии клавиш `<Ctrl+z>`.
2. Напишите скрипт, который выводит список всех, подключённых к системе сетевых интерфейсов. Использовать цикл и команду `ls`.
3. Напишите скрипт, который создаёт файл с заданным именем и устанавливает для него заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию.
4. Написать скрипт, который будет выводить все файлы из указанной директории, которые имеют права доступа 755. Для хранения списка файлов использовать массив.

2.4 Программирование сценариев Bash: индивидуальные задания

Во всех заданиях должен быть контроль ошибок (если к какому-либо каталогу нет доступа, необходимо вывести соответствующее сообщение и продолжить выполнение). Вывод сообщений об ошибках должен производиться в стандартный поток вывода сообщений об ошибках (`stderr`) в следующем виде:

Имя_модуля: текст_сообщения.

Пример: `./script1 : error open file: 1.txt.`

Имя модуля, имя файла берутся из аргументов командной строки.

Варианты

Для задач 2-3 проверить отложенный запуск скрипта в окне, созданном мультиплексором, например `screen` или `tmux`, отсоединиться от терминала, созданного мультиплексором, подключиться к терминалу и проверить результаты работы скрипта.

<https://medium.com/swlh/how-to-use-screen-on-linux-to-detach-and-reattach-your-terminal-2f52755ff45e>

<https://habr.com/ru/companies/otus/articles/646037/>

<https://losst.pro/shpargalka-po-tmux>

<https://medium.com/codex/start-using-tmux-3d6ce41ed138>

Номер варианта получить у преподавателя (см. таблицу посещаемости).

Вариант 1.

Задача 1. Разработать скрипт, который:

- запрашивает имя пользователя
 - если указанный пользователь не зарегистрирован в системе, то выводит сообщение об этом и повторно запрашивает имя пользователя
- если указанный пользователь зарегистрирован в системе, то выводит его уникальный идентификатор (UID) и имена групп, в которые входит этот пользователь, разделенные пробелом
 - основная (первичная) группа должна быть указана отдельно

Задача 2. Напишите скрипт, который создает файл с заданным именем и устанавливает для него заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию. Использовать команду `umask`.

Задача 3. Написать скрипт для поиска файлов заданного размера в заданном каталоге и во всех его подкаталогах (имя каталога задаётся пользователем в качестве третьего аргумента командной строки). Диапазон (мин.— макс.) размеров файлов задаётся пользователем в качестве первого и второго аргумента командной строки. Проверить работу программы для каталога `/usr` и диапазона (мин.— макс.) 1000 1510.

Задача 4. Написать скрипт, который генерирует уникальное название для каждого созданного файла, и выводит на экран список из 10 случайных имен. Имя файла должно состоять из префикса „fileN“, где N принимает значения от 1 до 10, и суффикса, генерируемого случайно, например утилитой `md5sum`. Например, результат вывода может быть таким:

Имя файла1 = "file1.7a3e710af8c0f94798a666313fd502a1"

Имя файла2 = "file2.885fd9f36289275d0af397dc1cded003"

Вариант 2.

Задача 1. Разработать скрипт работы с архивами, который:

- запрашивает тип действия: разархивировать или заархивировать

- для архивации: запрашивает каталог для архивации и имя архива, создаёт архив с этим именем
- для распаковки: спрашивает имя файла с архивом и распаковывает его

Для выполнения задания используйте утилиту `tar`.

Задача 2. Напишите скрипт, который создает файл с заданным именем и устанавливает для него заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию.

Задача 3. Написать скрипт с использованием цикла `for`, выводящий на консоль размеры и права доступа для всех файлов в заданном каталоге и во всех его подкаталогах (имя каталога задается пользователем в качестве первого аргумента командной строки). Проверить работу программы для каталога `/usr`.

Задача 4. Написать скрипт, который при вызове делает фото с веб-камеры утилитой `fswebcam` и сохраняет в заданную папку, которую задать в скрипте. В скрипт добавить проверку, если утилита `fswebcam` не установлена, то установить ее.

Вариант 3.

Задача 1. Разработать скрипт, который:

- запрашивает имя пользователя
 - если указанный пользователь не зарегистрирован в системе, то выводит сообщение об этом и повторно запрашивает имя пользователя
- если указанный пользователь зарегистрирован в системе, то выводит его уникальный идентификатор (UID) и имена групп, в которые входит этот пользователь, разделенные пробелом
 - основная (первичная) группа должна быть указана отдельно

Задача 2. Напишите скрипт, который запрашивает номер зачетной книжки и переводите его в восьмеричную систему счисления и устанавливает для существующего файла заданную маску разрешений на доступ. После завершения работы скрипта, маска разрешений на доступ должна быть установлена в значение, используемое по умолчанию.

Задача 3. Написать скрипт для поиска заданной пользователем строки во всех файлах заданного каталога и во всех его подкаталогах (строка и имя каталога задаются пользователем в качестве первого и второго аргумента командной строки). На консоль выводятся полный путь и имена файлов, в содержимом которых присутствует заданная строка, и их размер. Если к какому-либо каталогу нет доступа, необходимо вывести соответствующее сообщение и продолжить выполнение. Проверить работу программы для каталога `/usr` и строки «`stdio.h`».

Задача 4. Написать скрипт, который генерирует уникальное название для каждого созданного файла, и выводит на экран список из 10 случайных имен. Имя файла должно состоять из префикса „fileN“, где N принимает значения от 1 до 10, и суффикса, генерируемого случайно, например утилитой `md5sum`. Например, результат вывода может быть таким:

Имя файла1 = "file1.7a3e710af8c0f94798a666313fd502a1"

Имя файла2 = "file2.885fd9f36289275d0af397dc1cded003"

Вариант 4.

Задача 1. Разработать скрипт работы с архивами, который:

- запрашивает тип действия: разархивировать или заархивировать
- для архивации: запрашивает каталог для архивации и имя архива, создаёт архив с этим именем
- для распаковки: спрашивает имя файла с архивом и распаковывает его

Для выполнения задания используйте утилиту `tar`.

Задача 2. Напишите скрипт, который выполняет следующие действия:

- Выводит на терминал меню, которое предлагает выбор из следующих действий:
 - удалить файл;
 - переименовать файл;
 - переместить файл;
 - создать файл.
- запрашивает имена файлов, для выбранного действия; выполняет выбранное действие.

Задача 3. Написать скрипт поиска одинаковых по их содержимому файлов в двух каталогах, например `Dir1` и `Dir2`. Пользователь задаёт имена `Dir1` и `Dir2` в качестве первого и второго аргумента командной строки. В результате работы программы файлы, имеющиеся в `Dir1`, сравниваются с файлами в `Dir2` по их содержимому. На экран выводятся число просмотренных файлов и результаты сравнения. Проверить работу программы для каталога `/usr` (`Dir1`) и любого каталога в каталоге `/home` (`Dir2`).

Задача 4. Написать скрипт, который при вызове делает фото с веб-камеры утилитой `fswebcam` и сохраняет в заданную папку, которую задать в скрипте. В скрипт добавить проверку, если утилита `fswebcam` не установлена, то установить ее.

Вариант 5.

Задача 1. Разработать скрипт, который:

- запрашивает имя пользователя
 - если указанный пользователь не зарегистрирован в системе, то выводит сообщение об этом и повторно запрашивает имя пользователя
- если указанный пользователь зарегистрирован в системе, то выводит его уникальный идентификатор (UID) и имена групп, в которые входит этот пользователь, разделенные пробелом

- основная (первичная) группа должна быть указана отдельно

Задача 2. Напишите сценарий, проверяющий имя текущего каталога и выводящий сообщение об ошибке, если оно короче пяти символов. Если больше или равно пяти символам, то выводящий посекундно в цикле имена файлов текущего каталога и их порядковый номер.

Задача 3.

Написать скрипт, находящий в заданном каталоге и во всех его подкаталогах все файлы, владельцем которых является заданный пользователь. Имя владельца и каталог задаются пользователем в качестве первого и второго аргумента командной строки. Скрипт выводит результаты в файл (третий аргумент командной строки) в виде полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов. Проверить работу программы для каталога /usr пользователь root.

Задача 4. Написать скрипт, который генерирует уникальное название для каждого созданного файла, и выводит на экран список из 10 случайных имен. Имя файла должно состоять из префикса „fileN“, где N принимает значения от 1 до 10, и суффикса, генерируемого случайно, например утилитой mcookie. Например, результат вывода может быть таким:

Имя файла1 = "file1.7a3e710af8c0f94798a666313fd502a1"

Имя файла2 = "file2.885fd9f36289275d0af397dc1cded003"

Вариант 6.

Задача 1. Разработать скрипт работы с архивами, который:

- запрашивает тип действия: разархивировать или заархивировать
- для архивации: запрашивает каталог для архивации и имя архива, создаёт архив с этим именем
- для распаковки: спрашивает имя файла с архивом и распаковывает его

Для выполнения задания используйте утилиту tar.

Задача 2. Требуется проверить, является ли файл обычным или он является каталогом. Если это обычный файл, то сценарий должен выводить имя файла и его размер. В случае, если размер файла превышает килобайт, то размер должен выводиться в килобайтах. Если размер превышает мегабайт — в мегабайтах.

Задача 3. Написать скрипт, находящий в заданном каталоге и во всех его подкаталогах все файлы, заданного размера в заданном каталоге (имя каталога задаётся пользователем в качестве первого аргумента командной строки). Диапазон (мин.— макс.) размеров файлов задаётся пользователем в качестве второго и третьего аргументов командной строки. Скрипт выводит результаты поиска в файл (четвертый аргумент командной строки) в виде: полный путь, имя файла, его размер. На консоль выводится общее число просмотренных файлов. Проверить работу программы для каталога /usr и диапазона (мин.— макс.) 1000 1010.

Задача 4. Написать скрипт, который при вызове делает фото с веб-камеры утилитой `fswebcam` и сохраняет в заданную папку, которую задать в скрипте. В скрипт добавить проверку, если утилита `fswebcam` не установлена, то установить ее.

Вариант 7.

Задача 1. Разработать скрипт, который:

- запрашивает имя пользователя
 - если указанный пользователь не зарегистрирован в системе, то выводит сообщение об этом и повторно запрашивает имя пользователя
- если указанный пользователь зарегистрирован в системе, то выводит его уникальный идентификатор (UID) и имена групп, в которые входит этот пользователь, разделенные пробелом
 - основная (первичная) группа должна быть указана отдельно

Задача 2. Напишите сценарий, который генерирует тысячу файлов `1.txt` `1000.txt`, и в каждый файл записывает подряд 100 чисел N , где N = порядковый номер файла. Затем скрипт должен соединить в один файл все файлы с четными номерами (`even.txt`) и в другой файл — все файлы с нечетными номерами (`odd.txt`).

Задача 3.

Написать скрипт, подсчитывающий суммарный размер файлов в заданном каталоге и во всех его подкаталогах (имя каталога задаётся пользователем в качестве аргумента командной строки). Скрипт выводит результаты подсчёта в файл (второй аргумент командной строки) в виде: каталог (полный путь), суммарный размер файлов, число просмотренных файлов. Проверить работу программы для каталога `/usr`.

Задача 4. Написать скрипт, который генерирует уникальное название для каждого созданного файла, и выводит на экран список из 10 случайных имен. Имя файла должно состоять из префикса „fileN“, где N принимает значения от 1 до 10, и суффикса, генерируемого случайно, например утилитой `md5sum`. Например, результат вывода может быть таким:

Имя файла1 = "file1.7a3e710af8c0f94798a666313fd502a1"

Имя файла2 = "file2.885fd9f36289275d0af397dc1cded003"

2.5 Программирование сценариев Bash: творческое задание

1. Сформировать группу из 2-х человек и выполнить задание с реализацией подключения по `ssh` и запуска. Для автоматического входа на удаленный компьютер сгенерировать и использовать ключи. Необходимо разбиться на пары (дракон-воин) сначала опробовать модельное сражение (вручную вводя команды), затем реализовать скрипты.
2. Необходимо создать скрипт воина (`warrior.sh`), который запрашивает `ip`-адрес компьютера замка (`castle`), заходит на него по протоколу `ssh`, и должен забрать с этого компьютера на свой при помощи команды `scp` похищенный герб (`gerb.gif`). Воину противостоит дракон (`dragon.sh`), который в начале работы прячет герб

(gerb.gif) в случайное место файловой системы своего компьютера, а далее осуществляет охрану замка: с некоторой периодичностью просматривает список процессов и, если обнаруживает воина, то убивает его (убивать службу sshd запрещается).

3. Сражаться разрешается при помощи сигналов 2 (INT) и 15 (TERM). Защищаться можно при помощи ловушек (trap).
4. Воин также может (и должен) убить дракона.
5. Если останется время, сразиться с противниками из другой группы.

Задание засчитывается при предоставлении результатов модельного сражения (ручной ввод команд), результатов выполнения скриптов и результатов сражения с другой командой.