Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Администрирование информационных систем»

Профиль: «Разработка программно-информационных систем»

Семестр 5

ОТЧЕТ

по лабораторной работе №7

Тема: «Основы написания скриптов на bash»

Выполнила: студент группы РИС-22-1б

Поважный В.Е. \_\_\_\_\_\_\_\_\_\_\_

Проверил: ст. преподаватель кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2024

**Цель работы**

Получить начальные практические навыки написания сценариев оболочки.

# Теоретические сведения

* 1. **Группирование команд. Скрипты**

Командная оболочка BASH позволяет группировать несколько команд, выполняющих определённое действие, в функции или специальные файлы, называемые скриптами.

*Скрипт* – это обычный текстовый файл, содержащий команды оболочке.

Такой файл может быть запущен на исполнение следующим образом:

# $bash имя\_файла

Другими словами, для выполнения скрипта необходимо запустить командную оболочку, передав ей в качестве параметра имя соответствующего файла.

Другой вариант запуска скрипта – просто указать его имя в командной оболочке (т.е. сделав из него некий вид программы). Для этого надо в параметрах доступа определить файл как исполняемый, и в первых строках этого файла явно указать оболочку, для которой предназначен этот скрипт, следующим образом:

# #!оболочка

В общем случае символ «**#**» в скрипте означает комментарий, требующий игнорировать строку. Однако если он является первым символом файла и за ним следует символ «**!**», то это означает «магическую комбинацию», за которой указывается путь к файлу, использующемуся в качестве интерпретатора скрипта (например, /bin/bash, /bin/perl, /bin/sh и т.д). Встретив такую комбинацию символов, командная оболочка запустит соответствующий файл и передаст ему его имя в качестве аргумента.

Скрипт, в свою очередь, может содержать все конструкции, описанные ранее (в том числе и функции).

Выполнение скрипта происходит построчно. При этом если конструкция включает в себя несколько команд, то они могут располагаться на нескольких строках, и указывать символ «\» в конце неоконченной строки нет необходимости. Например:



X=1

read –p “Введите X = ” X if [ $X 0 ] ; then

отрицательное число”

else

положительное число”

fi

Любому скрипту, точно так же, как и функции, могут быть переданы аргументы (так, как это делается при запуске любой команды). Передаются аргументы в виде специальных переменных (таблица 1).

Таблица 1. Специальные переменные, используемые в скриптах.

|  |  |
| --- | --- |
| **Переменная** | **Описание** |
| $0 | Имя выполняемой команды. Для скрипта – это путь, указанный при его вызове. Для функции – имя оболочки. |
| $n | Переменные, соответствующие аргументам, заданным при вызове сценария. Здесь n – десятичное число, соответствующее номеру аргумента. (Первый аргумент – $1, второй – $2 и т.д.). |
| $# | Число аргументов, указанных при вызове сценария. |
| $\* | Строка аргументов, заключённая в двойные кавычки. |
| $@ | Все аргументы, каждый заключён в двойные кавычки. |
| $? | Статус завершения последней выполненной команды. |
| $; | Номер процесса, соответствующего текущей оболочке. |
| $! | Номер процесса, соответствующий команде, запущенной в фоновом режиме. |

Последовательно просмотреть аргументы командной строки можно, используя следующую конструкцию:

#!/bin/bash

while [ -n “$1” ] ; do

echo “Имеется аргумент – ”$1 shift

done

Аргументы просматриваются в цикле **while**, условием выполнения которого является результат команды **test** с параметрами, требующими, чтобы длина строки в аргументе **$1** была отлична от нуля. В строку подставляется значение первого аргумента. Если аргумент не указан, то переменная **$1** имеет пустое значение, соответственно строка будет иметь нулевую длину, и цикл сразу же прекратится. Если аргумент указан, то он выводится в теле цикла, и затем выполняется команда **shift**, которая изменяет переменные **$n**, сдвигая их на одну влево (т.е. **$1=$2, $2=$3** и т.д.). Значение первого аргумента при этом теряется. Когда сдвигается последний аргумент, то переменной **$1** присваивается пустое значение (так как следующей за ней переменной не существует). В качестве параметра команды **shift** можно указать, на сколько позиций требуется сдвинуть строку аргументов. Например, **shift** 2 приведёт к следующему изменению: **$1=$3, $2=$4** и т.п.

# Переменные

Кроме переменных среды окружения командная оболочка позволяет во время своего выполнения хранить данные в виде собственных переменных и даже массивов. Значения этих переменных используются только самой оболочкой и, в отличие от переменных среды окружения, недоступны запускаемым из неё программам.

Значение переменной присваивается следующим образом: переменная=значение (т.е. без процедуры экспортирования). Например, **X=1**, или **X=а**, или **X=”f”** и т.п.

**Обратите внимание, что до и после знака «=» нет пробелов!!!** Если поставить пробелы, например, так x = 1, то командная оболочка будет считать, что введена команда x, и она имеет два параметра: = и 1.

Если в командной оболочке создать переменную с тем же именем, что и переменная среды окружения, то в командной оболочке будет использоваться значение этой переменной, а в запускаемых программах – старое значение переменной среды окружения. Получить список всех переменных можно с использованием команды **declare**.

Удалить значение переменной или массива можно также с использованием команды **unset**.

# Подстановка переменных, команд и арифметических выражений

Командная оболочка BASH позволяет формировать команды с использованием значений переменных, результатов работы других команд и т.п. Такое формирование называется подстановкой. Т.е. в команду

«подставляется» что-либо (переменная, вывод другой команды и т.п.). Для подстановки используется либо символ «**$**», либо выражение, заключённое в обратные апострофы **(`выражение`).**

Если в тексте команды встречается символ «**$**», то следующий за ним текст до пробела или конца команды интерпретируется как имя переменной, значение которой подставляется в текст команды. Например:

[student@wp1 student]$FRUIT=Апельсин<Enter> [student@wp1 student]$echo “Фрукт ”$FRUIT<Enter>

Фрукт Апельсин [student@wp1 student]$\_

В примере создаётся одна переменная командной оболочки (**FRUIT**), которой присваивается значение «Апельсин». Затем выполняется команда **echo**, которая должна выдать на экран текст, указанный ей в качестве параметра. В данном случае параметром является строка **“Фрукт “$FRUIT**, в которой присутствует символ **«$»**. Поэтому прежде чем выполнить команду **echo**, командная оболочка «подставит» в её аргумент значение переменной **FRUIT**, сформировав тем самым текст **“Фрукт Апельсин”**.

Обратите внимание, что при присваивании переменной значения её имя указывается без знака доллара. Т.е. если Вы напишите **$FRUIT=apple**, командная оболочка выдаст ошибку: **-bash Апельсин=apple:command not found.** Так как прежде чем выполнить команду, командная оболочка подставит в неё значение переменной **FRUIT**, а затем только попытается её выполнить. И если Вы напишите **echo FRUIT**, то на экран будет выведено слово **FRUIT**, а не значение переменной с таким именем.

Подстановку можно использовать также и в случае, если требуется в команде использовать то, что некоторая программа помещает в поток вывода. В этом случае программа заключается в **обратные апострофы**. Прежде чем выполнить команду, командная оболочка выполнит программу, заключённую в обратные апострофы, затем всё, что она поместит в поток вывода, будет подставлено в командную строку, и только затем команда будет выполнена.

Например:

[student@wp1 student]$DATE=`date`<Enter> [student@wp1 student]$echo $DATE

Ср. нояб. 30 13:32:23 NOVT 2011

[student@wp1 student]$\_

Переменной **DATE** присваивается текст, который должна была вывести на экран команда **date**, т.е. текущую системную дату. Затем на экран выводится значение переменной **DATE**.

Командная оболочка позволяет выполнять арифметические операции. Для этого выражение, которое необходимо **интерпретировать как арифметическое**, заключается в двойные круглые скобки, и перед ним ставится знак доллара. Например, в результате выполнения команды:

[student@wp1 student]$foo=$((((5+3\*2)–4)/2))<Enter> [student@wp1 student]$echo $foo

3

[student@wp1 student]$\_

переменной foo будет присвоено значение, равное 3.

# Получение информации от пользователя

При необходимости командная оболочка позволяет сформировать значение некоторой переменной, «спросив» его у пользователя. Для этого используется команда **read**, которой в качестве аргумента передаётся имя требуемой переменной.

[student@wp1 student]$read CHOICE Привет !!!<Enter>

[student@wp1 student]$echo “Вы ввели ”${CHOICE} Привет !!!

[student@wp1 student]$\_

Чтобы указать, какое приглашение должно быть выведено в строке для ввода, можно использовать параметр **-p**. Например:

# $ read -p “Введите X” X.

Время ожидания (в секундах) ввода задаётся или при помощи переменной **TMOUT**, или при помощи параметра **-t**. Если переменная **TMOUT** не определена или её значение равно 0, и не указан параметр **-t**, то время ожидания считается бесконечным. Обратите внимание, что значение переменной **TMOUT** также влияет на время ожидания командной оболочкой очередной команды!!!

Используя параметр -s, можно запретить отображение вводимых символов на экране. Это удобно, например, при вводе паролей.

# Условный оператор if-fi

Выражение **if** записывается следующим образом:

$if выражение1 ; then \

>выражение2 ; \

>elif выражение3 ; then \

>выражение4 ; \

>else \

>выражение5 ; \

>fi<Enter>

В приведённой выше команде **if** сначала выполняется **выражение1**. Если код завершения **выражения1** равен 0 (что интерпретируется как его истинность), то выполняется **выражение2**, и команда **if** заканчивается. В противном случае выполняется **выражение3**, и проверяется код его завершения. Если **выражение3** возвращает значение, равное 0, то выполняется **выражение4** и команда **if**. Если **выражение3** возвращает ненулевое значение, то выполняется **выражение5**. Наличие операторов **elif** и else необязательно. В блоке **if-fi** может содержаться несколько **elif**.

Часто в блоке **if-fi** в качестве выражений, результаты которых проверяются, используется команда **test**, которая имеет две формы записи: **test параметры** или **[ параметры ]**. После интерпретации параметров (таблица 2) как логического выражения команда **test** возвращает значение 0 – истина либо 1 – ложь.

Если команда не помещается в командную строку, то её можно продолжить на следующей строке, для чего текущую строку надо завершить символом **«\»** (обратный слеш) и нажать **Enter**. Командная оболочка после нажатия **Enter** определит, что последним символом в команде был символ

«обратный слеш», и будет ожидать продолжения команды на новой строке. Точно так же можно завершить вторую, третью и последующие строки.

Таблица 2. Специальные переменные, используемые в скриптах.

|  |  |
| --- | --- |
| **Выражение** | **Значение** |
| -d файл | существует ли *файл* и является ли он каталогом? |
| -e файл | существует ли указанный *файл*? |
| -f файл | существует ли *файл* и является ли он обычным файлом? |
| -L файл | существует ли *файл* и символьная ли он ссылка? |
| -r файл | существует ли *файл* и разрешено ли его чтение? |
| -s файл | существует ли *файл* и имеет ли он нулевой размер? |
| -w файл | существует ли *файл* и разрешена ли в него запись? |
| -x файл | существует ли *файл* и является ли он исполняемым? |
| -O файл | существует ли *файл* и принадлежит ли он текущему поль-  зователю? |
| файл1 -nt файл2 | был ли *файл1* последний раз модифицирован позже, чем  *файл2*? |
| -z строка | указанная строка имеет нулевую длину? |
| -n строка | указанная строка имеет ненулевую длину? |
| стр1 == стр2 | указанные строки совпадают? |
| ! выражение | указанное выражение *false*?(содержит не нуль) |
| выр1 -a выр2 | логическое И двух выражений |
| выр1 -o выр2 | логическое ИЛИ двух выражений |
| выр1 -eq выр2 | *выр1* равно *выр2*? |
| выр1 -ne выр2 | *выр1* неравно *выр2*? |
| выр1 -lt выр2 | *выр1* меньше (в арифметическом смысле) *выр2*? |
| выр1 -le выр2 | *выр1* меньше либо равно *выр2*? |
| выр1 -gt выр2 | *выр1* больше *выр2*? |
| выр1 -ge выр2 | *выр1* больше либо равно *выр2*? |

В следующем примере проверяется, существует ли каталог

**$HOME/bin**, и, если он существует, то он добавляется к переменной

# PATH.

$if [ -d $HOME/bin ] ; then \

>PATH="$PATH:$HOME/bin" ; fi

1. **Оператор множественного выбора case-esac**

Блок **case-esac** аналогичен оператору **if-fi** со множеством **elfi** и предназначен для проверки одной переменной на несколько возможных значений. Блок **case-esac** записывается следующим образом:

$case значение in \

>шаблон1) \

>список команд1 ;;

>шаблон2) \

>список команд2 ;;

>esac<Enter>

В данном случае значение – это строка символов, сравниваемая с шаблоном до тех пор, пока она не совпадёт с ним. Список команд, следующий за шаблоном, которому удовлетворяет значение, запускается на выполнение. За списком следует команда **«;;»**, которая завершает работу блока **case-esac**.

Если значение не удовлетворяет ни одному из шаблонов, выражение **case** завершается. Если необходимо выполнить какие-то действия по умолчанию, следует включить в выражение шаблон **«\*»**, которому удовлетворяет любое значение.

В выражении **case-esac** должен присутствовать, по крайней мере, один шаблон. Максимальное число шаблонов не ограничено.

Шаблон формируется по правилам, аналогичным именам файлов и каталогов (с учётом символов расширения), а также используется оператор дизъюнкции **«|»** (операция ИЛИ). Ниже приведён пример использования блока **case-esac**.

$case “$TERM” in \

>\*term) \

>TERM = xterm \

>;; \

>network | dialup | unknown | vt[0-9]\*) \

>TERM=vt100 \

>;; \

>esac

# Цикл for

Цикл **for** предназначен для выполнения определённых действий над несколькими данными. Формат записи цикла следующий:

$for имя\_переменной in список\_значений ; \

>do \

>команда1 ; команда2 ...\

>done

В цикле **for** переменной с указанным именем последовательно присваиваются все значения из **списка\_значений**, и для каждого из этих значений выполняется **список\_команд**. Значения в **списке\_значений** перечисляются через пробел. Например, следующая команда выдаст на экран десять строк: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10:

$for i in 1 2 3 4 5 6 7 8 9 10 ; \

>do \

>echo $i ; \

>done

# Цикл while

Если необходимо выполнять какие-либо действия до тех пор, пока некоторое выражение истинно, то следует использовать цикл **while**, который записывается следующим образом:

$while выражение ; \

>do \

>список\_команд ; \

>done

На каждой итерации этого цикла выполняется **выражение** и до тех пор, пока оно возвращает значение 0, выполняется **список\_команд**. Если в качестве выражения указать команду **/bin/true**, то цикл будет бесконечным и завершить его можно только, используя оператор **break**. Пропустить какую- либо часть цикла и перейти на следующую его итерацию можно, используя в **списке\_команд** команду **continue**.

Например, вывод последовательности цифр от 1 до 9 можно организовать следующим образом:

$x=1

$while [ $x -lt 10 ] ; \

>do \

>echo $x ; \

>x=$(( $x+1 )) ; \

>done

# Порядок выполнения лабораторной работы

1. Создайте скрипт, находящий сумму двух переданных ему аргементов. Выведите результат сложения и сообщение о том, меньше ли результат нуля, либо больше, либо равен нулю.
2. Создайте скрипт, принимающий в качестве агрумента **строку**, и создающий архив, включающий в себя все файлы из домашней директории пользователя с расширением **.txt**. Имя итогового архива должно состоять из переданного аргумента и отметки времени **строка\_час\_минута\_секунда**.

**Ход работы**

1. Сложение двух аргументов и проверка результата:

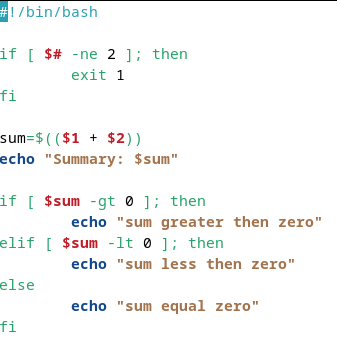


Рис. 1 – код скрипта



Рис. 1 – Сложение двух аргументов и проверка результата

1. Архивация файлов .txt из домашней директории:

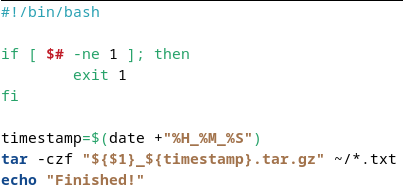


Рис. 3 – код скрипта

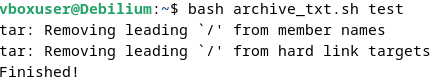


Рис. 4 – Архивация файлов

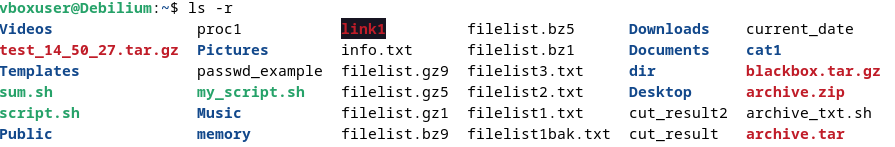


Рис. 5 – архив

**Ответы на контрольные вопросы**

**1. Что такое скрипт?**

Скрипт — это текстовый файл, содержащий последовательность команд, которые выполняются одна за другой в заданном порядке. Чаще всего скрипты создаются для автоматизации задач в операционной системе. Они могут выполнять разнообразные функции: запускать программы, обрабатывать файлы, производить вычисления и т.д.

**Пример простого скрипта:**

#!/bin/bash

echo "Hello, World!"

Файл с таким содержимым, сохранённый с расширением .sh, можно выполнить, чтобы вывести "Hello, World!" в терминале.

**2. Что означает символ \, введённый в командной строке перед нажатием Enter?**

Символ \ в командной строке используется для продолжения команды на следующей строке. Это удобно для длинных команд, чтобы улучшить читаемость или добавить параметры. Когда \ используется в конце строки, оболочка интерпретирует следующую строку как продолжение текущей команды.

**Пример:**

echo "This is a very long sentence that \

continues on the next line."

**3. Команда read**

Команда read используется для считывания пользовательского ввода и сохранения его в переменной. Это полезно при написании скриптов, которые требуют ввода данных.

**Пример:**

#!/bin/bash

echo "Введите ваше имя:"

read name

echo "Привет, $name!"

В этом скрипте read name ждёт, пока пользователь введёт имя, и затем сохраняет его в переменной $name.

**4. Условный оператор if-fi. Команда test**

Оператор if-fi используется для выполнения команд на основе условий. Он проверяет логические выражения или результаты команд, и если условие истинно, выполняет соответствующий блок кода.

**Пример:**

#!/bin/bash

read -p "Введите число: " num

if [ $num -gt 10 ]; then

echo "Число больше 10"

else

echo "Число 10 или меньше"

fi

* **Команда test** проверяет условия, такие как сравнение чисел, строк, существование файлов и т.д. В примере выше [ $num -gt 10 ] — это эквивалент команды test $num -gt 10.

**5. Циклические конструкции**

Циклы позволяют выполнять повторяющиеся действия. Основные циклы в Bash:

* **Цикл for** используется для перебора элементов, как в списке:

for i in 1 2 3 4 5; do

echo "Число: $i"

done

* **Цикл while** выполняется, пока условие истинно:

count=0

while [ $count -lt 5 ]; do

echo "Счётчик: $count"

count=$((count + 1))

done

* **Цикл until** выполняется, пока условие не станет истинным:

count=0

until [ $count -ge 5 ]; do

echo "Счётчик: $count"

count=$((count + 1))

done