This notes Includes

- Writing your first script
- Working with variables
- Working with functions
- Flow control: if/else

Before you start ...

- Before you start shell scripting it is important that you understand the following concepts:
- Source code vs machine code:
  - Source code is the human readable code written in a programming language like Python.
  - machine code consists of binary 1s and 0s and is the language a computer's CPU understands.
- Compiler: a program used for converting a complete source code into machine code.
- Interpreter: a program that converts source code into machine code line by line.
- Compiled program vs Interpreted program:
  - a compiled program is a binary file produced by a compiler.
  - an interpreted program is a program that requires an interpreter to interpret and run each instruction.

2

Creating a basic script

- Start vim, enable line numbers, and enter insert mode.
- Type:

```
#!/bin/bash
echo "This is a script that displays information about your Linux system"
uname -a
```

- Save the file and name it "script1.sh"
- Type: chmod u+x script1.sh to make the file executable.
- To run the script type: ./script1.sh

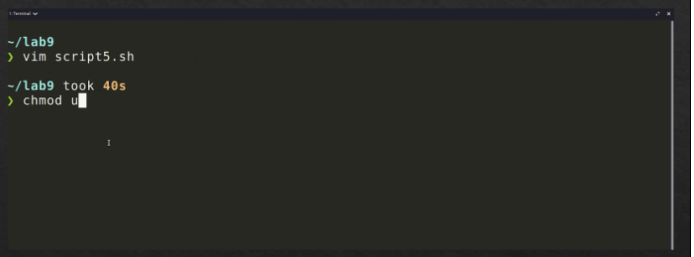
4

Shell scripting | Displaying text

- To display a line of text use the echo command:
- Example:
  - `echo “This is a message”`
  - `echo “This is another message” “and another message”`
  - `echo -n “this is, again, another message”`
  - `echo “this is the last message”`
- echo -n does not output a new line.

Practice:

- open script1.sh in vim and type the echo examples.



Shell scripting | Variables

- Variable: placeholder for data.
- Environment variable: is a placeholder for data that can change; typically, it gets its value automatically from the OS startup or the shell being used.
- Each user has environment variables with different values to define his or her working environment.
- The HOME environment variable stores the absolute pathname to a user’s home directory, so it varies for each user.
- Some environment variables are the same for all users logged in to a machine, such as the HOST environment variable that specifies the computer name.
- The env command allows you to see all environment variables
- You can use the echo command to see the value of an environment variable.
  - Example:
    - echo $HOME
    - echo $HOST

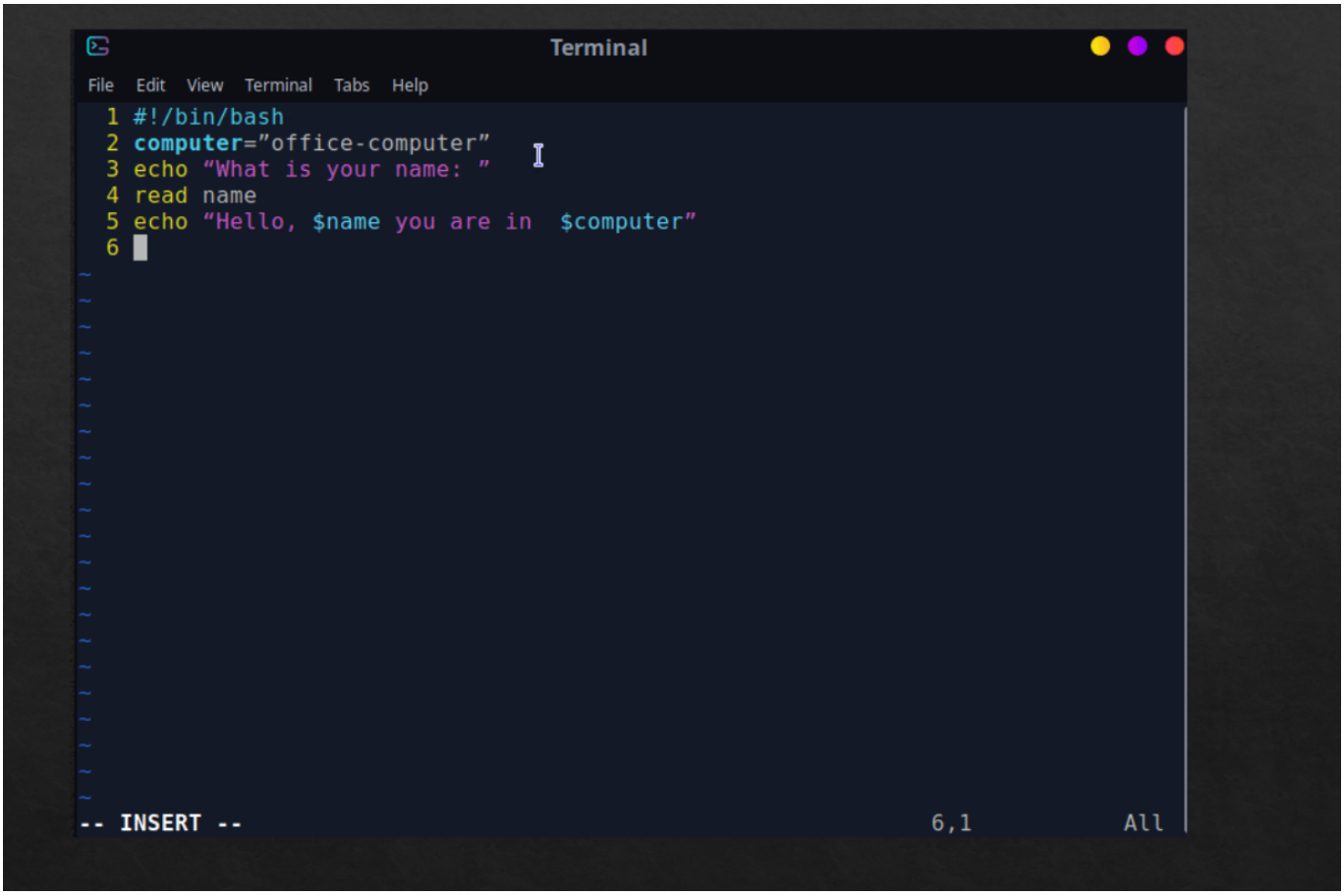
Shell scripting | Variables

- **Practice:**
- Open script1.sh in vim and enter the following text

  echo $HOME

  echo $SHELL

  echo $USER
- Save the file and close vim
- $HOME = The home directory of the current user.
- $SHELL = Set path to login shell.
- $USER = user name



Shell scripting | Variables

- The positional parameter method uses the order of arguments in a command to assign values to variables on the command line.
- Variables from $0 to $9 are available, and their values are defined by what the user enters.

Table 5-6 Positional parameters

Positional parameter	Description	Example
`$0`	Represents the name of the script	`./scr4` (`./scr4` is position 0)
`$1` to `$9`	`$1` represents the first argument, `$2` represents the second argument, and so on	`./scr4 /home` (`./scr4` is position 0 and `/home` is position 1) `./scr4 /home scr1` (`./scr4` is position 0, `/home` is position 1, and `scr1` is position 2)
`$*`	Represents all the positional parameters except 0	`/home scr1` (just `/home` and `scr1`)
`$#`	Represents the number of arguments that have a value	`./scr4 /home scr1` `echo $#` (`$*` represents positions 1 and 2, which are `/home` and `scr1`)

Shell scripting | Exit Status Codes

- Exit status code: a number sent to the shell when you run a command.
- These codes differ, depending on the Linux distribution or the shell.
- Successful commands usually return the code 0, and failures return a value greater than 0.
- to see an exit status use the $? variable.

- **Practice:**
- Create file named script3.sh and start vim.
- Enable line numbers and enter insert mode.
- type:

```
#!/bin/bash
cd baddir
echo $?
```

- Save the file and exit vim
- Make the script executable with: chmod u+x script3.sh
- Run the script with: ./script3.sh

Shell scripting | conditions

- The if statement is used to carry out certain commands based on testing a condition and the exit status of the command.
- If statements are used to control how the script will execute.
- For instance, you might want a portion of the script to run if the user is in the Marketing Department and have another portion run if the user is in Human Resources.
- Or you may want to run another command if the a given command executes successfully.
- if statement—Starts the condition being tested
- then statement—Starts the portion of code specifying what to do if the condition evaluates to true
- else statement—Starts the portion of code specifying what to do if the condition evaluates to false

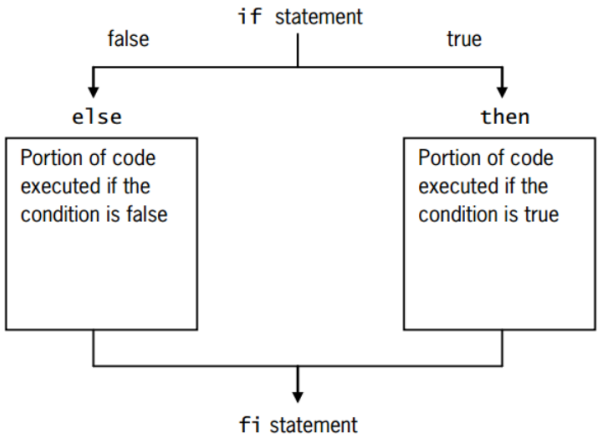


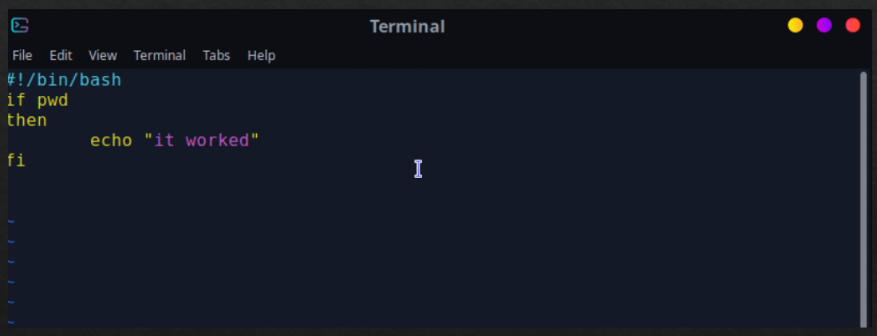
Figure 5-3 A flowchart of the `if` statement

© Cengage Learning 2013

Here is an example

if the pwd command executes successfully, the string "it worked" will be displayed

```
#!/bin/bash
if pwd
then
        echo "it worked"
fi
```



Here is a more interesting example!

- This simple script searches the filesystem for all the mp4 files owned by the user. The variable **$user** stores the username provided by the person who runs the script. The if statement runs the find command only if the grep command executes successfully.
- Notice that in the previous example, the command after **if** showed the output after executing. In this example, we use output redirection to redirect the output of grep to the black hole.

```
#!/bin/bash
# asks the user for a username
echo "Enter username: "
read user
# checks if the user exists in the system
# > /dev/null will send the output of grep to the black hole
# using > /dev/null with if prevents the ouput of the command
# from being displayed
if grep $user /etc/passwd > /dev/null
then
        # find all mp4 files owned by the user in the user's
        # home directory
        find /home/$user -user $user -name *.mp4 2> /dev/null
else
        # displayed only if the user is not found in the system
        # meaning the exit status code of the grep command
        # is non-zero
        echo "$user not found in system"
fi
```

```
Terminal
File  Edit  View  Terminal  Tabs  Help
(adrian@G752VL ~)$ ./script9.sh
Enter username:
```

Shell scripting | Comparison operators

Table 5-7 File attribute operators in the BASH shell

File attribute operator	Description
-a	Checks whether the file exists
-d	Checks whether the file is a directory
-f	Checks whether the file is a regular file
-r	Checks whether the user has read permission for the file
-s	Checks whether the file contains data
-w	Checks whether the user has write permission for the file
-x	Checks whether the user has execute permission for the file
-O	Checks whether the user is the owner of the file
-G	Checks whether the user belongs to the group owner of the file
file1 -nt file2	Checks whether file1 is newer than file2
file1 -ot file2	Checks whether file1 is older than file2

Shell scripting | Comparison operators example

```
#!/bin/bash
echo "enter a file name: "; read file
# if the file exists the ls command will run
if [ -a $file ]
then
        ls -l $file
fi
# if the file is a directory the tar command will run
if [ -d $file ]
then
        tar cf ${file}.tar $file
fi
```

```
#!/bin/bash
echo "enter a file name: "; read file
# if the file exists the ls command will run
if [ -a $file ]
then
        ls -l $file
fi
# if the file is a directory the tar command will run
if [ -d $file ]
then
        tar cf ${file}.tar $file
fi
```

Shell scripting | Comparison operators

Numeric Comparison		
Comparison	Description	Example
n1 -eq n2	Checks if n1 is equal to n2	If [$n1 -eq $n2]
n1 -ge n2	Checks if n1 is greater than or equal to n2	If [$n1 -ge $n2]
n1 -gt n2	Checks if n1 is greater than n2	If [$n1 -gt $n2]
n1 -le n2	Checks if n1 is less than or equal to n2	If [$n1 -le $n2]
n1 -lt n2	Checks if n1 is less than n2	If [$n1 -lt $n2]
n1 -ne n2	Checks if n1 is not equal to n2	If [$n1 -ne $n2]

Shell scripting | Comparison operators

String Comparison

Comparison	Description	Example
str1 = str2	Checks if str1 is the same as string str2	If [$str1 = $str2]
str1 != str2	Checks if str1 is not the same as str2	If [$str1 != $str2]
str1 < str2	Checks if str1 is less than str2	If [$str1 < $str2]
str1 \> str2	Checks if str1 is greater than str2	If [$str1 > $str2]
-n str1	Checks if str1 has a length greater than zero	If [$str1 -n]
-z str1	Checks if str1 has a length of zero	If [$str1 -z]

Terminal - adrian@adrian-GL:~

File Edit View Terminal Tabs Help

```
#!/bin/bash
clear
echo "--------------------------------------------------"
echo -e "\twhich command would you like to run?"
echo "1) ls"
echo "2) ls -a"
echo "3) ls -l"
echo "4) ls -la"
echo "5) ls -laR"
echo "--------------------------------------------------"
read answer

case $answer in
        1)
                ls
                ;;
        2)
                ls -a
                ;;
        3)
                ls -l
                ;;
        4)
                ls -la
                ;;
        5)
                ls -laR
                ;;
        *)
                exit
                ;;
esac

~
                                                11,1          All
```

Shell scripting | looping

- A for loop repeats the commands between do and done a specified number of times. Each time the script carries out the commands in the loop, a new value is given to a variable.

```
for VARIABLE in 1 2 3 4 5 .. N
do
        command1
        command2
        commandN
done
```

```
for VARIABLE in file1 file2 file3
do
        command1 on $VARIABLE
        command2
        commandN
done
```