Shell Scripting

Creating a basic script

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

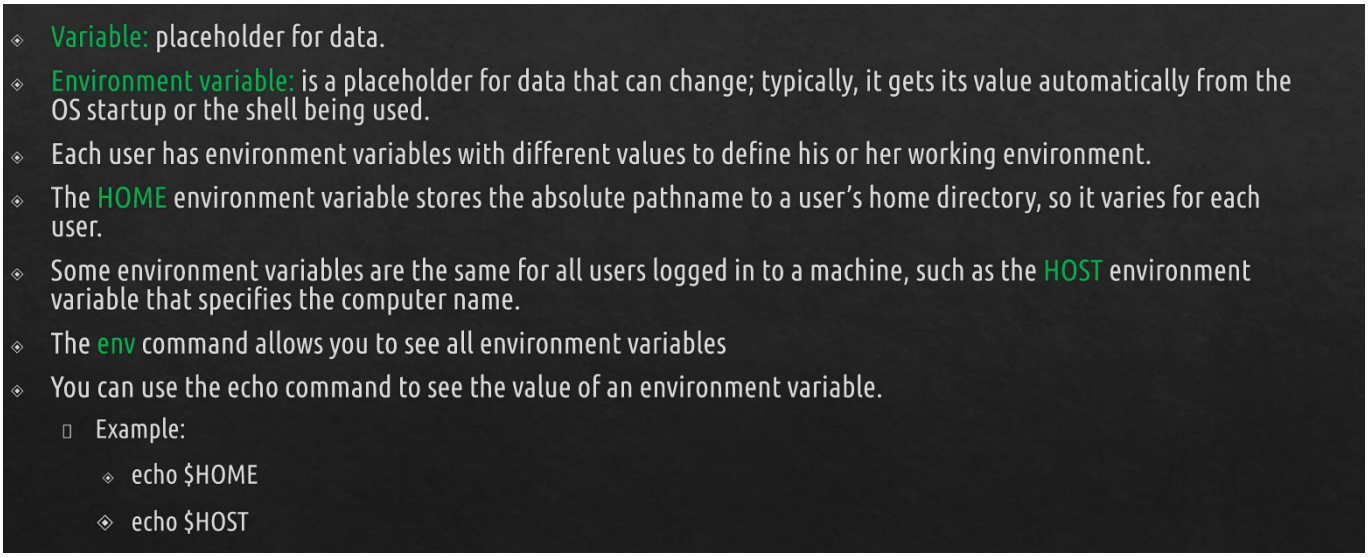
```
#!/bin/bash
echo " to display info about your Linux system"
uname -a
```

> echo -n **does not output a new line**.

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

Working with variables

Shell scripting | Variables



- A **shell variable** is similar to an environment variable, but its value is usually assigned in a shell script.
- Shell variables can be created in **two ways**:

Direct Assignment:

> color = blue

The Prompt MEthod:

> echo "Enter a color:";read color

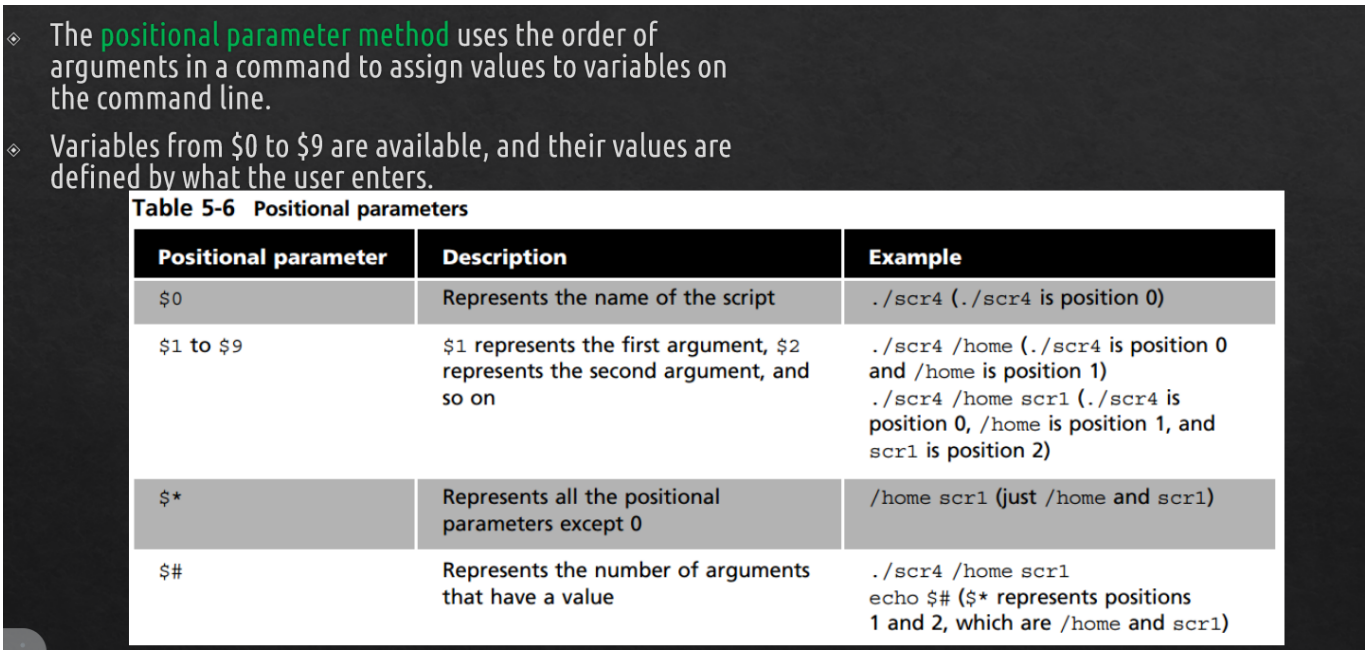
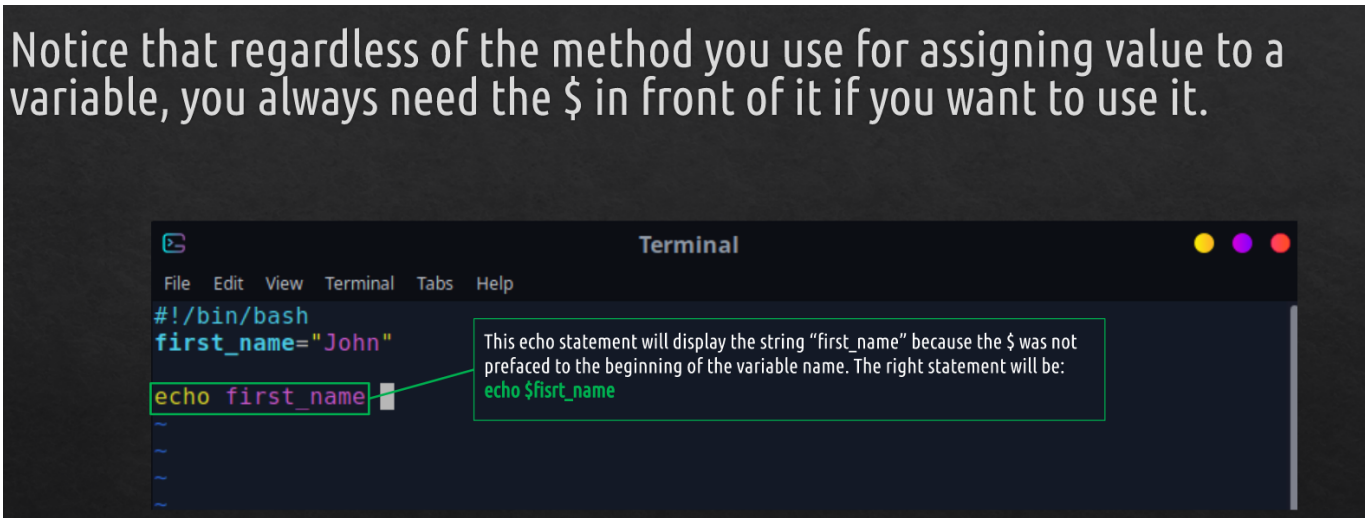


Table 5-6 Positional parameters

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`)



- You can use **curly braces** to reference a variable's value: **${variable_name}**
- It is useful if you want to **append a string to a variable**.
- **Command Substitution:** allows the output of a command to replace the command itself.

Can be done in **TWO WAYS:**

```
$(command)
`command`
```

Shell scripting | Exit Status Codes

Exit status code: a number sent to the shell when you run a command.

Type:

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

- Successful commands usually return the **code 0**, and failures return a value **greater than 0**.
- to see an exit status use the **$? variable**.

Using Structured Commands

if-then-Else statements, nesting if statements, test, compound testing, and case statements

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

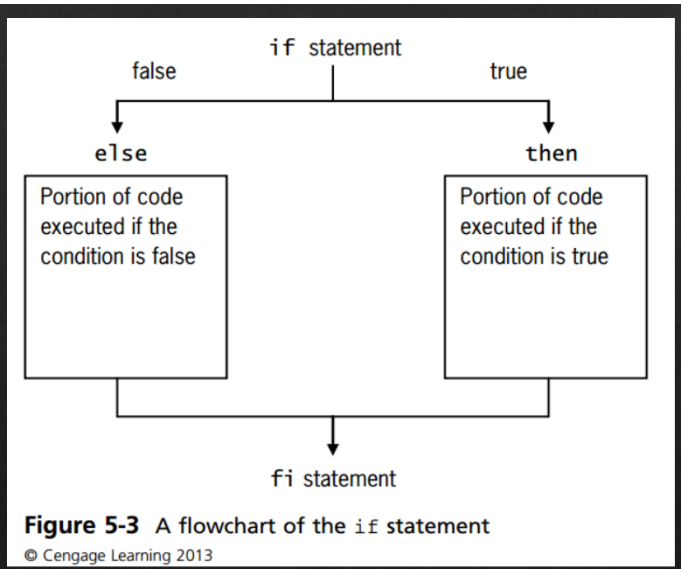
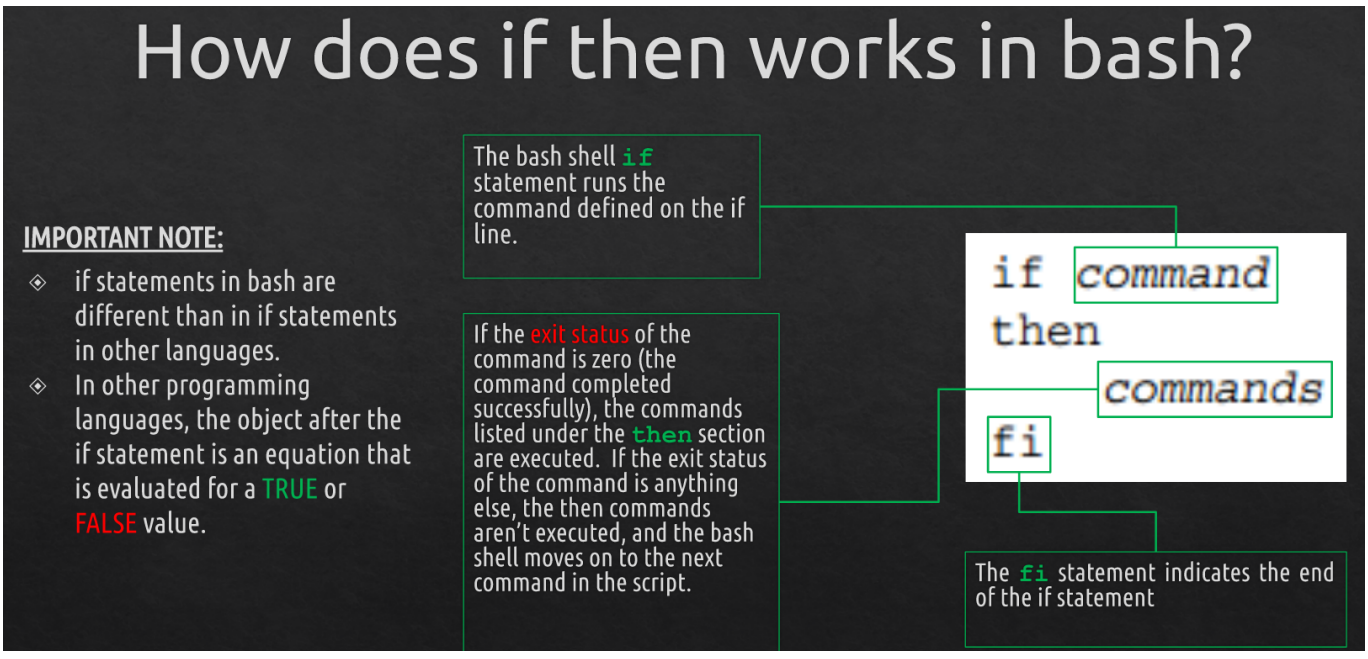


Figure 5-3 A flowchart of the `if` statement

© Cengage Learning 2013

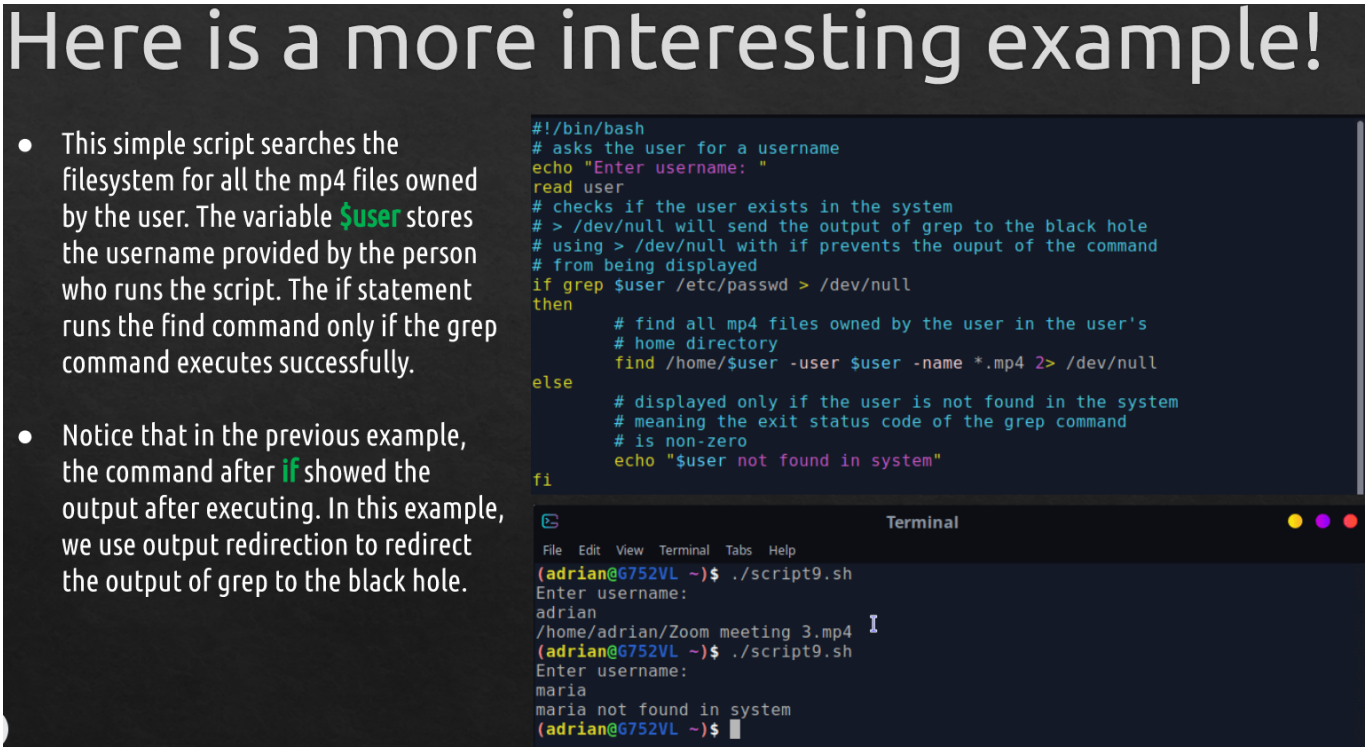


Example

```
#!/bin/bash
if pwd
then
```

```
        echo "it worked"
fi
```

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



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`

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]

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]

A **case statement** uses one variable to specify multiple values and matches a portion of the script to each value.

```
#!/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

- Looping is used to perform a set of commands repeatedly. In the menu script, the user is given a list of options to choose from, and after a selection is made, the script ends.
- Shell scripting support different types of loops:
  - while loop
  - until loop
  - for loop

```
while [ condition ]
do
        command1
        command2
        commandN
done
```

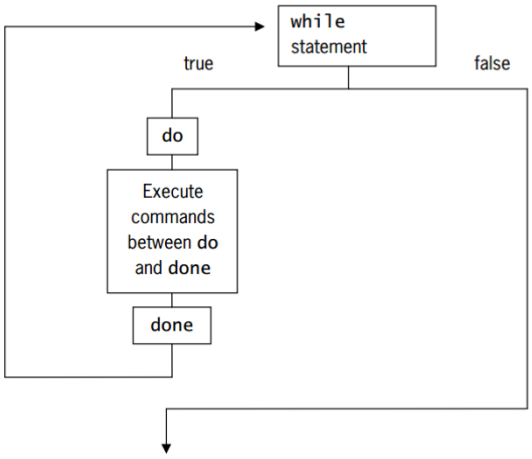


Figure 5-4 A `while` loop
© Cengage Learning 2013

- An until loop repeats the commands between do and done as long as the tested condition is false (exit status code is greater than 0)—in other words, until the condition is true.

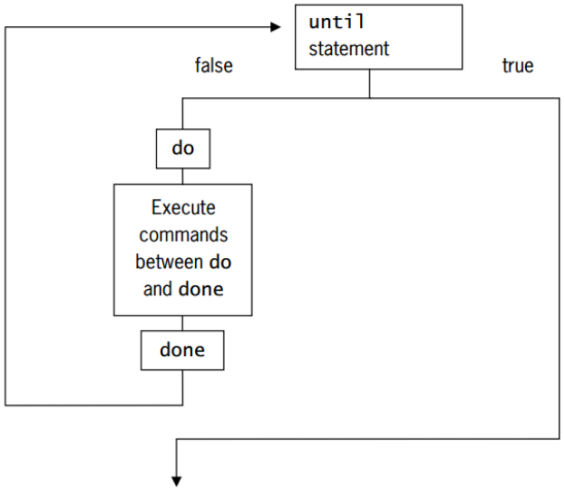


Figure 5-5 An `until` loop
© Cengage Learning 2013

- 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
```