Shell Scripts II

1. 05b - Shell Scripts II

1.1 Shell Scripts II

1.2 Unit Overview

Unit Overview

- how to debug ?
- Decision
  - case
- Repetition
  - while, until
  - for
- Functions

1.3 Debug shell Scripts

Debug shell Scripts

- Debugging is troubleshooting errors that may occur during the execution of a program/script

- 2 commands can help to debug:
  - echo

    use explicit output statements to trace execution
  - set

    trace execution path

1.4 Debugging using "set"

Debugging using "set"

- "set" command is a shell built-in command
- has options to allow tracing of execution
  - -v print shell input lines as they are read
  - -x option displays expanded commands and its arguments
- options can turned on or off
  - To turn on the option: **`set -xv`**
  - To turn off the options: **`set +xv`**
- options can also be set via she-bang line

```
#! /bin/bash -xv
```

1.5 Debugging using "set"

Debugging using "set"

1.6 The case Statement

The case Statement

- to make decision that is based on multiple choices

Syntax:

```
case word in
    pattern1) command-list1
    ;;
    pattern2) command-list2
    ;;
    patternN) command-listN
    ;;
esac
```

1.7 case pattern

case pattern

- checked against word for match
- may also contain:

```
*
?
[ … ]
[:class:]
```

- multiple patterns can be listed via:

```
|
```

1.8 Example: case Statement

Example: case Statement

Terminal - student@csci330: ~

File Edit View Terminal Tabs Help

```
student@csci330:~$ cat choice
#! /bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter Q to quit"

read -p "Enter your choice: " reply

case "$reply" in
  Y|YES) echo "Displaying all (really…) files"
         ls -a ;;
  N|NO)  echo "Display all non-hidden files..."
         ls ;;
  Q)     exit 0 ;;

  *) echo "Invalid choice!"; exit 1 ;;
esac
student@csci330:~$
```

1.9 The while Loop

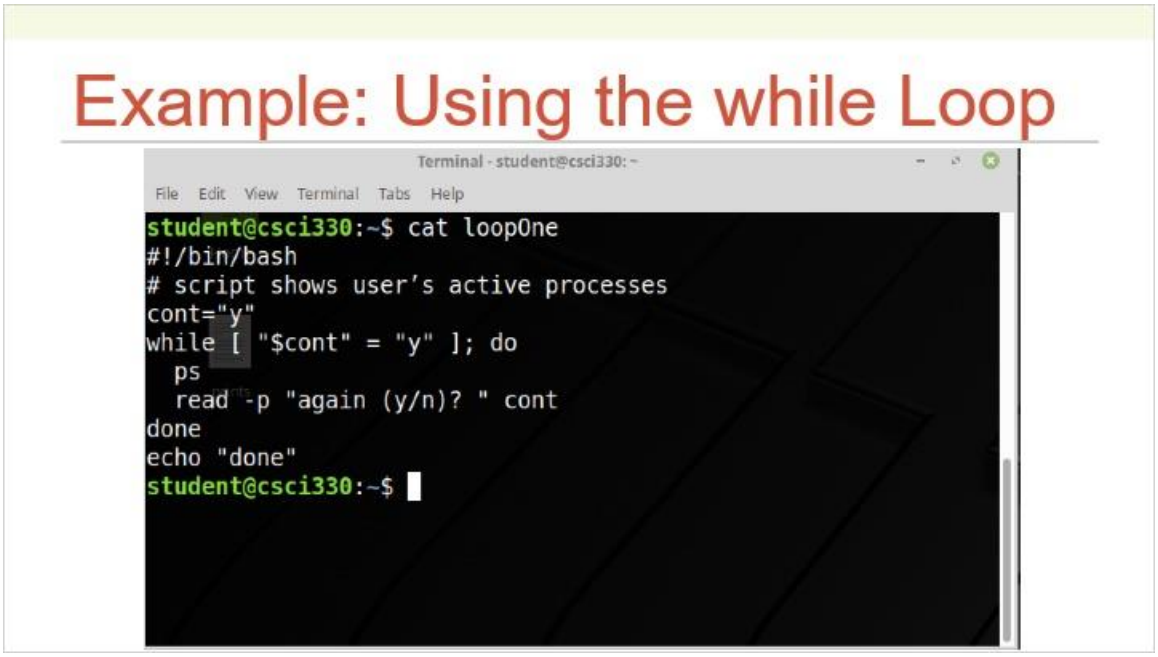
The while Loop

executes “command-list” as long as “test-command” evaluates successfully

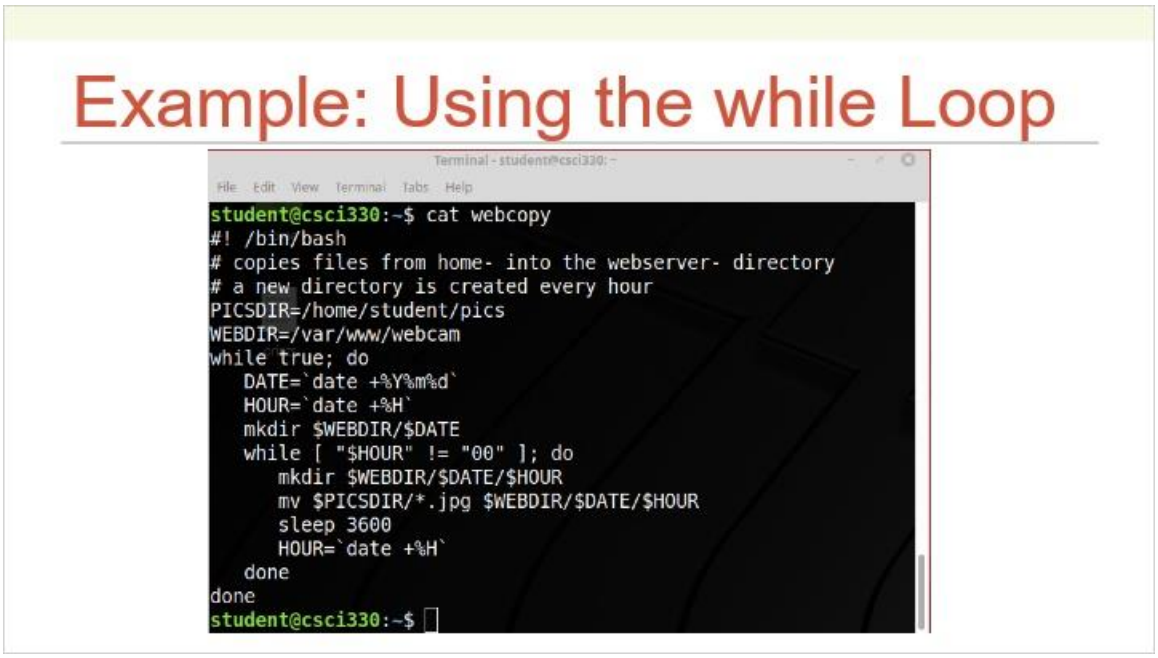
Syntax:

```
while test-command
do
   command-list
done
```

1.10 Example: Using the while Loop



1.11 Example: Using the while Loop



1.12 The until Loop

The until Loop

executes “command-list” as long as
“test-command” does not evaluate successfully

Syntax:

```
until test-command
do
   command-list
done
```

1.13 Example: Using the until Loop

Example: Using the until Loop

```
#!/bin/bash
# script shows user's active processes
stop="n"
until [ "$stop" = "y" ]; do
  ps
  read -p "done (y/n)? " stop
done
echo "done"
```

1.14 The for Loop

The for Loop

executes “commands” as many times as the number of words in the “word-list”

Syntax:

```
for variable in word-list
do
   commands
done
```

1.15 Example 1: for loop

Example 1: for loop

Terminal - student@csci330: ~

File Edit View Terminal Tabs Help

```
student@csci330:~$ cat looptwo
#!/bin/bash

for index in 7 6 5 4 3 2
do
   echo $index
done
student@csci330:~$
```

1.16 Example 2: Using the for loop

Example 2: Using the for loop

Terminal - student@csci330: ~

File Edit View Terminal Tabs Help

```
student@csci330:~$ cat tmpavg
#!/bin/bash
# compute average weekly temperature
TempTotal=0
for day in 1 2 3 4 5 6 7
do
   read -p "Enter temp for $day: " Temp
   TempTotal=$((TempTotal+Temp))
done
AvgTemp=$((TempTotal/7))
echo "Average temperature: " $AvgTemp
student@csci330:~$
```

1.17 Example 3: Using the for loop

Example 3: Using the for loop

```
#!/bin/bash
# compute average weekly temperature
TempTotal=0
for day in `seq 7`
do
   read -p "Enter temp for $day: " Temp
   TempTotal=$((TempTotal+Temp))
done
AvgTemp=$((TempTotal/7))
echo "Average temperature: " $AvgTemp
```

1.18 Example 4: Using the for Loop

Example 4: Using the for Loop

```
#!/bin/bash
# compute average weekly temperature
TempTotal=0
for day in Mon Tue Wed Thu Fri Sat Sun
do
   read -p "Enter temp for $day: " Temp
   TempTotal=$((TempTotal+Temp))
done
AvgTemp=$((TempTotal/7))
echo "Average temperature: " $AvgTemp
```

1.19 Example 5: Using the for Loop

Example 5: Using the for Loop

```
#!/bin/bash
# compute average weekly temperature
TempTotal=0
for day in `cat day-file`
do
   read -p "Enter temp for $day: " Temp
   TempTotal=$((TempTotal+Temp))
done
AvgTemp=$((TempTotal/7))
echo "Average temperature: " $AvgTemp
```

1.20 looping over arguments

looping over arguments

- simplest form will iterate over all command line arguments:

```
#! /bin/bash
for parm
do
        echo $parm
done
```

1.21 break and continue

break and continue

- interrupt for, while or until loop
- break statement
  - terminate execution of the loop
  - transfers control to the statement AFTER the done statement
- continue statement
  - skips the rest of the current iteration
  - continues execution of the loop

1.22 Shell Functions

Shell Functions

- must be defined before they can be referenced
- usually placed at the beginning of the script

Syntax:

```
function-name () {
   statements
}
```

1.23 Example: function

Example: function

Terminal - student@csci330: ~

File Edit View Terminal Tabs Help

```
student@csci330:~$ cat funtest
#!/bin/bash

funky () {
        # This is a simple function
        echo "This is a funky function"
}

# declaration must precede call:

funky
student@csci330:~$
```

1.24 Function parameters

Function parameters

- Need not be declared
- Arguments provided via function call are accessible inside function as $1, $2, $3, …

$# reflects number of parameters

$0 still contains name of script
(not name of function)

1.25 Example: function with parameters

Example: function with parameters

Terminal - student@csci330: ~

File Edit View Terminal Tabs Help

```
student@csci330:~$ cat check
#! /bin/bash
checkfile() {
   for file
   do
      if [ -f "$file" ]; then
         echo "$file is a file"
      else
         if [ -d "$file" ]; then
            echo "$file is a directory"
         fi
      fi
   done
}
checkfile . funtest
student@csci330:~$
```

1.26 Local Variables in Functions

Local Variables in Functions

- Variables defined within functions are global, i.e. their values are known throughout the entire script
- Keyword “local” inside a function defines variables that are “local” to that function, i.e. not visible outside

1.27 Example: function

Example: function

```
#! /bin/bash

global="pretty good variable"

foo () {
    local inside="not so good variable"
    echo $global
    echo $inside
    global="better variable"
}

echo $global
foo
echo $global
echo $inside
```

1.28 return from function

return from function

Syntax:

```
return [status]
```

- ends execution of function
- optional numeric argument sets return status
  - default is "`return 0`"

1.29 return example

return example

```
#! /bin/bash
testfile() {
  if [ $# -gt 0 ]; then
     if [ ! -r $1 ]; then
        return 1
     fi
  fi
}
if testfile funtest; then
 echo "funtest is readable"
fi
```

1.30 Unit Summary

Unit Summary

- Debugging
- Decision
  - case
- Repetition
  - while, until
  - for
- Functions