Seneca | SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

Unix Scripting

Agenda

- Introduction to Shell Scripting
  - Categories of variables
  - Conditional Statements
  - Loops

Using Logic

The purpose of the if statement is to execute a command or commands based on a condition

The condition is evaluated by a test command, represented below by a pair of square brackets

```
if [ condition ]
then
   command(s)
fi
```

if Statement Example

Test with a condition
Notice the spaces after “[“ and before “]”

```
read password


if [  ”$password” = ”P@ssw0rd!”  ]
then
  echo ”BAD PASSWORD!”
fi
```

Observation: What does the following code do?

```
#!/usr/bin/bash

value=33
if test $value -eq 34
then
  echo “OK”
else
  echo “DIFFERENT”
fi
```

The test Command

- The test command can be used in two ways:
  - As a pair of square brackets: [condition]
  - The test keyword: test condition
- The condition test can result in success (0) or failure (1), unless the negation "not" (!), is used
- The test can compare numbers, strings, and evaluate various file attributes
  - Use = and != to compare strings,
    for example: ["$name" = "Bob"]
  - Use -z and -n to check string length,
    for example: [! -z "$name"]
  - Use -gt, -lt, -eq, -ne, -le, -ge for number,
    for example: [”$salary” -gt 100000]

Observation: : what does the following script do?

-
  ```
  value=34
  if test $value -gt 2
  then
    if test $((value % 2)) -eq 0
    then
      echo “even”
    fi
  fi
  ```

Observation: Try the following code in command prompt

- `x=9`
- `test $x -eq 9`
- `echo  $?`
- What is the output?
- Change x=10, and try the above code again. What is the output?

Observation

What does the following script do?

```
if cd $1
then
   echo "Current directory has been changed to $PWD"
else
   echo "Current directory is $PWD, could not be changed to $1"
fi
```

Observation: what does the following script do?

- Script1:

```
if [[ $1 > $2 || $2 > $3 ]]
then
   echo "Arguments are not in correct sort order"
   exit 1
fi
```

- Script2:

```
if (( $1 > $2 || $1 <= 0 ))
then
   echo "Range of first two arguments is incorrect"
   exit 1
fi
```

The Test Command

- Common file test operations include:
  - -e (file exists)
  - -d (file exists and is a directory)
  - -s (file exists and has a size greater than zero)
  - -w (file exists and write permission is granted)
- Check man test for more details

Observation: : what does the following script do?

```
if [ ! -d "$1" ]
    then
        echo "$1 is not a directory"
        exit 1
fi
```

Activity 2

- Change the following script to read a number from input, and displays whether it is an even number or odd number. Make sure the input number is greater than 0 and less than 1000.
-
  ```
  value=34
  if test $value –gt 2
  then
    if test $((value % 2)) -eq 0
    then
      echo “even”
    fi
  fi
  ```

Activity 3

- Change the following script to accept a number as **argument**, and displays whether it is an even number or odd number. Make sure the input number is greater than 0 and less than 1000.
-
```
value=34
if test $value –gt 2
then
  if test $((value % 2)) –eq 0
  then
    echo “even”
  fi
fi
```

elif control-flow statement

- The elif statement is used to work like a nested if statement. It performs another test, and execute the command(s) if the result is true.

-
  ```
  if test $mark -gt 50
    then
      echo “you pass”
    elif test $mark -eq 50
      then
        echo “you JUST passed”
    else
      echo “sorry, you failed”
  fi
  ```

Extended test Command

- **[[...]]** is a keyword rather than a command, so it is more efficient
- conditions can be combined using && (and) and || (or)
- will work correctly even if an unquoted variable is null
- string comparisons can use > and <, they won't be confused with redirection

Extended test: example

- if [[$1 > $2 || $2 > $3]]
- then
-    echo "Arguments are not in correct sort order"
-    exit 1
- fi

stdin, stdout, stderr

- The three input/output (I/O) connections are called
  - standard input (stdin),
  - standard output (stdout)
  - and standard error (stderr).
- Originally I/O happened via a physically connected system console (input via keyboard, output via monitor), but standard streams abstract this.

Standard Input and Standard Output

- Standard input (stdin) is a term which describes from where a command receives input
- Standard output (stdout) describes where a command sends it's output
- For most commands the default standard input and output are your terminal's keyboard and screen
- Standard input can be redirected from a file or piped from another command
- Most commands also accept a filename argument, which is internally redirected to standard input
- Standard output can be redirected to a file or piped to another command

Standard Input Redirection

```
command < filename
```

- Example:
  ```
  cat < cars
  ```
- Used for commands which do not accept a filename as an argument

Standard Output Redirection

```
command > filename
```

- Redirects a command's standard output to a file
- Stdout redirection is represented by the **>** symbol

Example:

`ls > ls.txt` - will save output from the ls command into a file called ls.txt

- If the file exists already its content will be replaced
- To append (add) to a file, the **>>** symbol can be used

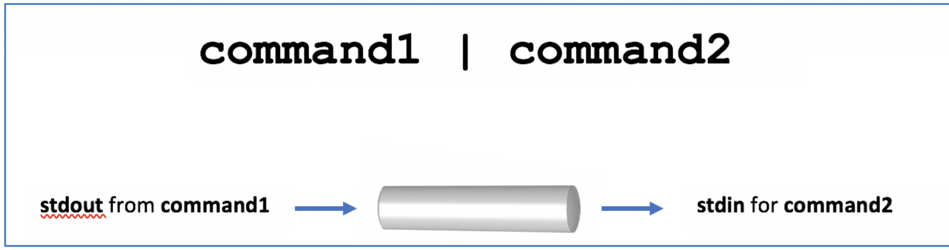
Inter-process communication

- Commands can send their standard output directly to standard input of other commands
- A few simple commands can be combined to form a more powerful command line
- No temporary files are necessary
- This is achieved by using pipes and tees

Pipes

- Pipes are represented by |
- Many commands can be "piped" together, filter commands are especially useful
  - Each filter processes the initial input based on it's design
  - Filters must be chained in a specific order, depending on what you wish to accomplish
- Example piping use:

```
ls -al | more
```



```
command1 | command2
```

stdout from command1 → → stdin for command2

"Here" documents

- The << symbol indicates a "here" document

Example:

```
sort << EOF
word
name
car
EOF
```

- Anything between EOF…EOF is sent to the standard input of a utility
- You can use some other string instead of "EOF"
- This is especially useful for embedding a small file within a shell script

Let’s learn StdErro

- What does the following command do?
  - `cat nofile`
- How to redirect the error output to a file?

Standard Error

- In addition to standard input and standard output UNIX commands have standard error, where error messages are sent
- By default error messages are sent to the terminal
- Standard error can be redirected by using the **`2>`** or **`2>>`** redirection operators
- To redirect both standard output and the standard error to the same destination, use **`>&`**
  - `wc nofile file1 >& wordcount`

Example

-
  ```
  cat nofile 2> file1
  ```
  - the standard error from the above example could be redirected from appearing on the display screen to being written to a file named *file1* .

What does the following command do?

- `cat nofile1 2> /dev/null`
  - Error messages can be discarded so that they neither appear on the screen nor are written to any file by redirecting them to a special file called ***/dev/null***

/dev/null   file

- The /dev/null file (sometimes called the bit bucket or black hole) is a special system file that discards all data written into it
  - Useful to discard unwanted command output, for example:
    `find / -name "tempfile" 2> /dev/null`
- Also, /dev/null can provide null data (EOF only) to processes reading from it
  - Useful to purge (empty) files etc, for example: **`cat /dev/null > ~/.bashrc`**

Activity 4

- Run the following script and explain how it works (explain line by line)

```
– echo -n "Please enter an integer: " > /dev/tty
– read number
– if [ -z "$number" ] || echo $number | grep "[^0-9]" > /dev/null
– then
–    echo "Sorry, '$number' is not a valid integer" >&2
– else
–    echo "Thank you!"
– fi
```

Review

- Check BB->Week2 for a quick review about the filtering commands in Unix:
  - cut
  - head and Tail
  - grep
  - sort
  - wc