Shell Scripting Part II

Tyler Hutcheon

Shell Scripting

Objectives

- Advanced Scripting Techniques
- Use Shell Functions
- Useful Commands in Shell Scripts

Advanced Scripting Techniques

Advanced Scripting Techniques

Functions

- There is often an need to perform a task multiple times, and instead of writing the same code multiple times ***functions*** are used

- Shell functions act like modules make an entire script section available with a single name. (Think of it like a script within a script)

- Shell functions are normally defined at the beginning of a script
  - Shell scripts are executed top to bottom.
  - The commands in the function aren't run yet, however, the section is learned for later use.
  - You cannot run a function before it's learned.

- You can store several functions in a file and include this file whenever the functions are needed

Advanced Scripting Techniques

Functions

- Basic Syntax of a function

```
functionname ()  {
    command1
    command2
    …
}
```

- Alternative syntax is using the “function” command:

```
function functionname  {
    command1
    command2
    …
}
```

Advanced Scripting Techniques

Functions

```
# mcd: mkdir + cd; creates a new directory and
# changes into that new directory right away
mcd (){
    mkdir "$1"
    cd "$1"
}
```

- After creating this function, it can be called in a shell script like so:

```
mcd /VAS
```

Advanced Scripting Techniques

Functions

- Functions are scripts within scripts.
  - They get “command line”-like paramters, $1, $2, $3, etc.
  - They also must return an exit code
- The error code (or “exit code” or “status code” or “return code”) is a number:
  - 0 = Success
  - 1-255 = Failure (use different numbers for different errors)
- To return an exit code from a function, use **return:**

  return 0
- Like “exit” in the main script, this will immediately stop execution of the function and return to where it was in the main script.
  - If you do not run **return**, the error code of the last run command is used instead. (Same is true for not using **exit** in the main script).

Advanced Scripting Techniques

Example 11 – userdel1.sh

Advanced Scripting Techniques

Read Options with getopts

- **getopts** is a shell built-in command

- With **getopts** you can extract the options supplied to a script on the command line

- The shell interprets command-line arguments as command options only if they are prefixed with a "**-**"

- This makes it possible to place options in different positions on the command line and to supply then in any order
  - **myscript.sh -d -j -p *.txt -u texts/** is the same as **myscript.sh -djp *.txt -u texts/**

Advanced Scripting Techniques

Read Options with getopts

- The syntax is **getopts options variable**

- If a parameter is expected for the option (such as **-m maxvalue**), the corresponding option must be followed by a "**:**" in the string (as in **getopts m:**)

- The option string is followed by a variable to which all the specified command-line options are assigned as a list

- The command is usually used in a **while** loop together with **case** to define which command to execute for an option

Advanced Scripting Techniques

Read Options with getopts

- Example usage of **getopts** in a **while** loop

```
while getopts ab:c variable
 do
      case $variable in
      a ) echo "The option -a was used." ;;
      b ) echo "The option -b was used and the value is $OPTARG" ;;
      c ) echo “The option -c was used." ;;
      esac
 done
```

- If the option **-b *value*** is used, the value is assigned to **$OPTARG** for that loop only. You will need to save it in another variable or act on it before the loop finishes.
- If you have “mandatory” options, you will need to give the user an error when they are not set.

./script.sh -acb five

./script.sh -a

./script.sh -a -b five -c

./script.sh -b five -ac

Advanced Scripting Techniques

Example 12 – userdel2.sh

Useful Commands in Shell Scripts

Useful Commands in Shell Scripts

The cat command

- In interactive use, it is mostly run with a file name as an argument

```
cat /etc/passwd
```

- When combined with here operator << it can output several lines of text from a script

```
cat << EOM
> Insert Multi-
> Line Message
> Here
> EOM
Insert Multi-
Line Message
Here
```

Useful Commands in Shell Scripts

The cut command

- Most data and output in Linux is machine parseable. If you are dealing with data that has columns, like theoutput of “ls -l”, this command can be used to single out columns, hide columns.

- It is applied to each line of text from a file or standard input

- **cut -f** cuts out text fields

- **cut -c** works with the specified characters

- **cut -d** defines the delimiter
  - The default delimiter is **tab**

Useful Commands in Shell Scripts

The date command

- The command obtains the system date
- You can format the date easily

```
date -I
2004-09-03

date +%m-%d %H:%M
09-03 14:19

date +%D, %r
09/03/02, 02:19:58 PM

date +%d.%m.%y
03.09.02
```

Useful Commands in Shell Scripts

The echo command

- Exists as both a shell built-in command and an external command
- Used to print text lines to standard output
  - Inserts a line break after each line
- Syntax is **echo "please enter the first number:"**
- The **-e** option accepts a number of additional options
  - **\a** - Outputs an alert (sounds bell)
    - Does not always work depending on your terminal
  - **\c** - No new line at end of output
  - **\n** - Add a new line

Useful Commands in Shell Scripts

The grep and egrep commands

- Commands used to search a file for certain patterns
  - **egrep** is extended **grep**
- The syntax is **grep *searchpattern filename ...***
- Prints lines that match the search pattern
  - If specify several files will print line number and file name
- Can use options to specify
  - Print only the line numbers
  - Print matching lines together with leading and trailing context lines

Useful Commands in Shell Scripts

Command Variations

- Linux and UNIX commands are often identical, however, many commands support different options. If you are building a script to run on both Linux and UNIX systems, you will want to use the POSIX-supported parameters where possible. This means that a lot of the easy or convenient parameters added in for Linux are not available.
  - POSIX = Portable Operating System Interface for UNIX
  - POSIX is a programming API built for cross-platform compatability.
- See the "**man 1p**" pages.

- cat, grep, head, tail and other similar commands have a variation to deal gzipped files.
  - In Linux, this is either the normal commands (cat, grep, head tail), which will detect if the input file is gzipped, or,
  - In Linux, zcat, zgrep, zhead, ztail, etc.
  - In Solaris, gzcat, gzgrep, gzhead, gztail, etc.

Useful Commands in Shell Scripts

The sed command

- This command is a stream editor and is used from the command line rather than interactively
- It performs text transformations on a line-by-line basis
- Can be specified either directly on the command line or in a special command script loaded by the program on execution
- The output normally goes to **stdout**, but can go to a file
- The syntax is **sed editing-command filename**

Useful Commands in Shell Scripts

The sed command

- The editing commands are single-character arguments
  - **d** - Delete
  - **s** - Substitute or replace
  - **p** - Output line
  - **a** - Append after

Useful Commands in Shell Scripts

The sed command

- There are options to influence the overall behavior

  -n, --quiet, --silent

- By default it prints all lines on **stdout** as processed
- This suppresses so only prints lines for which **p** edit command has been given explicitly

  -e command1 -e command2 ...

- This option is necessary when specify two or more editing commands
- Must be inserted before each additional editing command

  -f filename

- Use this option to specify a script file where it should get it's editing commands

Useful Commands in Shell Scripts

The sed command

- For some commands you need to specify the exact line or lines that should be processed
  - "**$**" stands for the last line

- **sed -n '1,9p'** ***somefile***
  - Prints lines 1 through 9 on **stdout**

- **sed '10,$d'** ***somefile***
  - Deletes everything from line 10 to the end of the file
  - Prints the first 9 lines on **stdout**

Useful Commands in Shell Scripts

The sed command

- Some examples of using the **s** command
  - **sed 's/a/b/' ~/testdata.txt**
  - Replaces the first letter “a” in each line with a “b”
  - **sed 's/a/b/g' ~/testdata.txt**
  - Replaces all “a” letters in all lines (global) with a “b”
  - **sed 's/a/b/2' ~/testdata.txt**
  - Replaces only the second “a” in each line with a “b”
  - **sed -n 's/\([aeiou]\)/\1\1/igp'**
  - Replace all single vowels with double vowels
  - **i** ignores case, **g** replaces globally, **p** prints all processed lines
  - **\1** references the matched pattern

Useful Commands in Shell Scripts

The test command

- This command exists as a built-in and an external command
- It is used to compare values and to check for files and their properties
  - Whether a file exists, it is executable, etc
  - If the test condition is **true** the exit status is zero (**0**)
  - If the test condition is **not true** the exit status is one (**1**)
- This command is used mainly to declare conditions to influence the operation of loops, branches, and other statements
- The syntax is **test condition**

Useful Commands in Shell Scripts

The test command

- Testing for file existence
  - **-e** - File exists
  - **-f** - File exists and is a regular file
  - **-d** - File exists and is a directory
  - **-x** - File exists and is an executable

- Comparing two files
  - **-nt** - Newer than
  - **-ot** - Older than
  - **-ef** - Refers to same inode (hard link)

Useful Commands in Shell Scripts

The test command

- Comparing two ***integers***
  - **-eq** - Equal
  - **-ne** - Not equal
  - **-gt** - Greater than
  - **-lt** - Less than
  - **-ge** - Greater than or equal
  - **-le** - Less than or equal

```
test $NUM -ge 5
```

Useful Commands in Shell Scripts

The test command

- Testing ***strings***

test -z *string*

- Exit status is 0 (true) if string has zero length (is empty)

test *string*

- Exit status is 0 (true) if string has nonzero length (is not empty)

test *string1 = string2*

- Exit status is 0 (true) if the strings are equal

test *string1 != string2*

- Exit status is 0 (true) if the strings are not equal

Useful Commands in Shell Scripts

The test command

- Combines tests

test ! *condition*

- Exit status is 0 (true) if the condition is not true

test *condition1* -a *condition2*

- Exit status is 0 (true) if both conditions are true

Useful Commands in Shell Scripts

The tr command

- This command is used to translate (replace) or delete characters
- It reads from standard input and prints the results to standard output
- You can replace regular characters, sequences of characters, and special characters like \t (tab) or \r (return)
- The syntax is **tr *set1 set2***
  - The characters in ***set1*** are replaced by the characters in ***set2***

Useful Commands in Shell Scripts

The tr command

- All lower case characters in the file will be changed to upper case and printed to standard out
  - **cat *text-file* | tr a-z A-Z**

- This will delete characters from the first set
  - **tr -d *set1***

- This deletes the percent sign from the original value of VAR and the result is assigned a new value to the same variable
  - **VAR='echo $VAR | tr -d %'**

- This will replace a set of characters with a single character
  - **tr -s *set1 char***

Where to go from here?

Useful Resources

- MAN Pages!

- The Advanced Bash-Scripting Guide
  http://tldp.org/LDP/abs/html/