Shell Scripts

Matt Warner

Contents

1 Overview

Shell scripts can do what can be done on command line

Shell scripts simplify recurring tasks. If you cannot find an existing utility to accomplish a task, you can build one using a shell script

Note:-

Much of UNIX administration and house keeping is done via shell scripts

2 Shell Script features

- Variables for storing data
- Decision-making control (e.g. if and case statements)
- Looping abilities (e.g. for and while loops)
- Functions for modularity
- Any UNIX command

  file manipulation: cat, cp, mv, ls, wc, tr, ...

  utilities: grep, sed, awk, ...
- Comments: lines starting with #

3 The basics

First line is always shebang

```
#! /bin/bash
```

To run shell scripts

```
bash script
```

Or, make executable

```
chmod +x script
./script
```

3.1 Simple Script

```
#! /bin/bash
date > usage-status
ls -l >> usage-status
du -s * >> usage-status
```

4 Bash Shell Programming Features

- Variables

  string, number, array
- input/output

  echo, printf

  command line args, read from user

- Decision

  conditional execution, if-then-else, case

- Repetition

  while, until, for

- Functions

5 User-defined shell variables

Syntax:

```
varname=value
```

Example:

```
rate=moderate
echo "Rate today: $rate"
```

Note:-

use double quotes if value of variable contains white spaces

Example:

name=“Thomas William Flowers”

6 Output via echo command

- Simplest form of writing to standard output

Syntax:

```
echo [-ne] arguments
```

-n suppresses trailing newline

-e enables escape sequences:

\t horizontal tab
\b backspace
\a alert
\n newline

6.1 Examples: shell scripts with output

```
#! /bin/bash
echo "You are running these processes:"
ps
```

```
#! /bin/bash
echo -ne "Dear $USER:\nWhat's up this month:"
cal
```

7 Command line arguments

- Use arguments to modify script behavior
- command line arguments become positional parameters to shell script
- positional paramters are numbered variables

  $1, $2, $3 . . .

7.1 Meanings

$1 first parameter

$2 second parameter

${10} 10th parameter (prevents "$1" misunderstanding)

$0 name of the script

$* all positional parameters

$# the number of arguments

7.2 Example: Command Line Arguments

```
#! /bin/bash
# Usage: greetings name1 name2

echo $0 to you $1 $2
echo Today is `date`
echo Good Bye $1
```

Make sure to protect complete argument

```
#! /bin/bash
# counts lines in directory listing

ls -l "$1" | wc -l
```

If we had a bash script as such:

```
#! /bin/bash
ls -l $1 | wc -l
```

And had a file called "file example"
We would not be able to use this file as a parameter, since our argument is not protected.

8 Arithmetic expressions

Syntax:

```
$((expression))
```

This can be used for simple arithmetic

```
count=1
count=$((count+20))
echo $count
```

9 Array variables

Syntax:

```
varname=(list of words)
```

Accessed via index:

```
${varname[index]}
${varname[0]}     first word in array
${varname[*]}     all words in array
```

9.1 Using array variables

Examples

```
ml=(mary ann bruce linda dara)
echo $ml
**prints mary**

echo ${ml[*]}
**prints mary ann bruce linda dara**

echo ${ml[2]}
**prints bruce**

ml[2]=john
echo ${ml[*]}
**prints mary ann john linda dara**
```

10 Output: printf command

Syntax:

```
printf format[arguments]
```

Writes formatted arguments to standard output under the control of "format"

Format string may contain:

- plain characters: printed to output
- escape characters: e.g. \t, \n, \a ...
- format specifiers: prints next successive arguement

10.1 printf format specifiers

```
%d  number
```

also

```
%10d    10 chars wide
%-10d   left justified

%s  string
```

also

```
%20s    20 chars wide
%-20s   left justifed
```

```
printf "random number\n"

printf "random number %d\n" 12

printf "random number %d\n" $RANDOM

printf "random number %10d\n" $RANDOM

printf "rando number %-10d %s\n" $RANDOM $USER
```

11 User input: read command

Syntax:

`read [-p "prompt"] varname [more vars]`

words entered by user are assigned to

varname and "more vars"

Last variable gets rest of input file

11.1 Example: Accepting User input

```
#! /bin/bash
read -p "enter your name: "  first last

echo "First name: $first""
echo "Last name: $last"
```