Shell Scripting: Bash

Dr. Hyrum Carroll

August 30, 2016

Background

- Shell scripting: A shell provides a Command Line Interface (CLI) to interact with the system (especially the file system)
- BASH
  - GNU **B**ourne **A**gain **SH**ell
  - Most common shell

Shell Script Examples

- removeFirstLines.sh
- code2html.sh
- jobsManager.sh

Bash's Configuration Files

In HOME directory:

- `.bash_profile`: read and executed by Bash every time you log in to the system
- `.bashrc`: read and executed by Bash every time you start a subshell
- `.bash_logout`: read and executed by Bash every time a login shell exits

If they're missing, Bash defaults to `/etc/profile`.

Bash: Variables

- Variable names can comprise letters numbers and some characters (“_”, etc.)
- No data type
- Assign values to a variable with = and the variable name without a “$”
- Use the variable with a “$”

Example:

```
1 STR="Hello World!"
2 echo $STR
```

Output:
Hello World!

Bash: Variables: Examples

Example:

```
1 STR="Hello World!"
2 echo $STR
```

Output:
Hello World!

Example:

```
1 STR="Hello World!"
2 echo STR
```

Output:
STR

Example:

```
1 COUNTER=9
2 echo $COUNTER
```

Output:
9

Bash: Variables

Concatenation:

```
1  str="Column 1"
2  str="$str Column 2"
3  str="$str Column 3"
4  echo "Header: $str"
```

Output:

Header: Column 1 Column 2
Column 3

Bash: Special Shell Variables

Variable	Meaning
$0	Filename of script
$1 - $9	Positional parameter #1 - #9
${10}	Positional parameter #10
$#	Number of positional parameters
"$*"	All positional parameters (as a single word) *
"$@"	All positional parameters (as separate strings)
$?	Return value
$$	Process ID (PID) of script
$-	Flags passed to script (using set)
$_	Last argument of previous command
$!	Process ID (PID) of last job run in background

* Must be quoted, otherwise it defaults to $@.

Bash: Comments

- Bash ignores everything on a line after a "#"
- "#"s in quotes are NOT comments

Concatenation:

```
1  str="Column 1"
2  #str="$str Column 2"
3  #str="$str Column 3"
4  echo "Header: $str"
```

Output:

Header: Column 1

Example:

```
1  echo "#"
```

Output:

#

Bash: Conditionals

- How to tell the computer to do one thing OR another
- Basic form:
  if `expression` **then** `statement`
  - If `expression` evaluates to TRUE, then execute `statement`, otherwise skip it
- Bash:

```
1    if [ expression ]; then statement; fi
```

Command-line example:

```
1 if [ 1 ]; then echo "Evaluated to true"; fi
```

Output:
Evaluated to true

Bash: Conditionals

- Another form:
  if `expression` **then** `statement1` **else** `statement2`
  - If `expression` evaluates to TRUE, then execute `statement1`, otherwise `statement2`

Command-line example:

```
1 if [ 1 ]; then echo "Evaluated to true"; else echo "
    Evaluated to false"; fi
```

Output:
Evaluated to true

Bash: Conditionals: Test Operators

TEST Operators: Binary Comparison

Arithmetic Operator	String Operator	Meaning
-eq	=	Equal to
	==	Equal to
-ne	!=	Not equal to
-lt	\<	Less than
-le		Less than or equal to
-gt	\>	Greater than
-ge		Greater than or equal to
	-z	String is empty
	-n	String is not empty

Bash: Conditionals: Test Operators

Example:

```
1 counter=1
2 if [ $counter -eq 0 ]; then
3   echo "counter equals 0"
4 else
5   echo "counter is non-zero
      "
6 fi
```

Output:

counter is non-zero

Example:

```
1 counter=0
2 if [ $counter -eq 0 ]; then
3   echo "counter equals 0"
4 else
5   echo "counter is non-zero
      "
6 fi
```

Output:

counter equals 0

Bash: Conditionals: Test Operators

Example:

```
1 counter="1"
2 if [ $counter == "0" ];
      then
3   echo "counter equals 0"
4 else
5   echo "counter is non-zero
        "
6 fi
```

Output:

counter is non-zero

Example:

```
1 counter="0"
2 if [ $counter == "0" ];
      then
3   echo "counter equals 0"
4 else
5   echo "counter is non-zero
        "
6 fi
```

Output:

counter equals 0

Bash: Conditionals: Test Operators

Example:

```
1 counter="1"
2 if [ -z "$counter" ]; then
3   echo "counter is an empty
        string"
4 else
5   echo "counter: $counter"
6 fi
```

Output:

counter: 1

Example:

```
1 counter=""
2 if [ -z "$counter" ]; then
3   echo "counter is an empty
        string"
4 else
5   echo "counter is non-zero
      "
6 fi
```

Output:

counter is an empty string

Bash: Loops

For Loops

For loops:

```
1 for i in "1" "2" "3"; do
2     echo $i
3 done
```

Output:

1
2
3

```
1 for i in $( ls ); do
2     echo "File: $i"
3 done
```

File: Makefile
File: lecture05.tex
File: loops.sh

```
1 for i in `ls`; do
2     echo "File: $i"
3 done
```

File: Makefile
File: lecture05.tex
File: loops.sh

Bash: Loops

While Loops

While loop:

```
1 counter=0
2 while [ $counter -lt 3 ]; do
3     echo "The counter is $counter"
4     counter=$(( counter + 1))
5 done
```

Output:

The counter is 0

The counter is 1

The counter is 2

Bash: Redirection

3 file descriptors: (std=standard)

1. `stdin`
2. `stdout`
3. `stderr`

1 'represents' `stdout`, 2 `stderr`

Bash: Redirection

You can redirect ...

1. `stdout` to a file

Examples:

1. `ls -l > ls-l.txt`

Bash: Redirection

You can redirect ...

1. `stdout` to a file
2. `stderr` to a file

Examples:

1. `ls -l > ls-l.txt`
2. `cat data*.txt 2> catError.txt`

Bash: Redirection

You can redirect ...

1. `stdout` to a file
2. `stderr` to a file
3. `stdout` to a file and `stderr` to a file

Examples:

1. `ls -l > ls-l.txt`
2. `cat data*.txt 2> catError.txt`
3. `cat data*.txt > allData.txt 2> catError.txt`

Bash: Redirection

You can redirect ...

1. `stdout` to a file
2. `stderr` to a file
3. `stdout` to a file and `stderr` to a file
4. `stdout` to `stderr`

Examples:

1. `ls -l > ls-l.txt`
2. `cat data*.txt 2> catError.txt`
3. `cat data*.txt > allData.txt 2> catError.txt`
4. `ls -s 1>&2`

Bash: Redirection

You can redirect ...

1. `stdout` to a file
2. `stderr` to a file
3. `stdout` to a file and `stderr` to a file
4. `stdout` to `stderr`
5. `stderr` to `stdout`

Examples:

1. `ls -l > ls-l.txt`
2. `cat data*.txt 2> catError.txt`
3. `cat data*.txt > allData.txt 2> catError.txt`
4. `ls -s 1>&2`
5. `ls -s 2>&1`

Bash: Redirection

You can redirect ...

1. `stdout` to a file
2. `stderr` to a file
3. `stdout` to a file and `stderr` to a file
4. `stdout` to `stderr`
5. `stderr` to `stdout`
6. `stderr` and `stdout` to a single file

Examples:

1. `ls -l > ls-l.txt`
2. `cat data*.txt 2> catError.txt`
3. `cat data*.txt > allData.txt 2> catError.txt`
4. `ls -s 1>&2`
5. `ls -s 2>&1`
6. `cat data*.txt &> allData_andErrors.txt`

Bash: Concatenation

- Use the >> operator
- Example: `echo "End of file list" >> ls-l.txt`

Bash: Pipes

- Pipes ("|") let you use the `stdout` of a program as the `stdin` of another program
- Allows for "chaining" together processes
- Somewhat similar to ">"
- Example: `du -a | sort -n`
- Example: `cat *.txt | sort -n | uniq > unique.txt`
  - Removes duplicate lines (by first sorting the lines in the *.txt files, then calling `uniq`)

Bash: Aliasing Commands

- In `.bash_profile` or `.bashrc`
- Syntax:
  - `alias name=command`
- Examples:
  - `alias ls='ls --color=auto -p -a'`
  - `alias diffsides='diff --side-by-side -W 170'`
  - `alias sshranger="ssh hcarroll@ranger0.cs.mtsu.edu"`
- Type `alias` to see all active aliases
- Type `unalias` to disable all aliases

Dealing with Spaces

Solutions:

1. Backslash escape sequence
2. Single/double quotes with spaces and variables

Examples:

1. `touch Not\ a\ unix\ friendly\ filename.txt`
2. `touch "Not a unix friendly filename.txt"`

BASH: Command Prompt

BASH: Command Prompt

Set by manipulating the PS1 variable. A simple prompt is:

```
PS1="\h:\w \u\$"
```

Backslash-escape special characters for prompts:

```
\a      an ASCII bell character (07)
\d      the date in "Weekday Month Date" format
        (e.g., "Tue May 26")
\e      an ASCII escape character (033)
\h      the hostname up to the first '.'
\H      the hostname
\j      the number of jobs currently managed by the shell
\l      the basename of the shell's terminal device name
\n      newline
\r      carriage return
\s      the name of the shell, the basename of $0
        (the portion following the final slash)
\t      the current time in 24-hour HH:MM:SS format
```

BASH: Command Prompt

```
\T      the current time in 12-hour HH:MM:SS format
\@      the current time in 12-hour am/pm format
\u      the username of the current user
\v      the version of bash (e.g., 2.00)
\V      the release of bash, version + patchlevel
        (e.g., 2.00.0)
\w      the current working directory
\W      the basename of the current working direcory
\!      the history number of this command
\#      the command number of this command
\$      if the effective UID is 0, a #, otherwise a $
\nnn    the character corresponding to the octal number nnn
\\      a backslash
\[      begin a sequence of non-printing characters,
        which could be used to embed a terminal control
        sequence into the prompt
\]      end a sequence of non-printing characters
```

BASH: Command Prompt: Colors

Black	0;30	Dark Gray	1;30
Blue	0;34	Light Blue	1;34
Green	0;32	Light Green	1;32
Cyan	0;36	Light Cyan	1;36
Red	0;31	Light Red	1;31
Purple	0;35	Light Purple	1;35
Brown	0;33	Yellow	1;33
Light Gray	0;37	White	1;37

```
PS1="\[\033[1;34m\][\$(date +%H%M)][\u@\h:\w]$\[\033[0m\] "
```

Turns the text blue, displays the time in brackets, displays the user name, host, and current directory enclosed in brackets. The part following the $ returns the color to the previous foreground color.

BASH: Command Prompt: Colors

This one sets up a prompt like this: `[user@host] directory $`:

```
PS1="\[\033[1;30m\][\[\033[1;34m\]\u\[\033[1;30m\]@
\[\033[0;35m\]\h\[\033[1;30m\]] \[\033[0;37m\]\W
 \[\033[1;30m\]\$\[\033[0m\] "
```

Sets the color for the characters that follow it. Below `1;30` will set them to Dark Gray.
Look at the table above
Sets the colors back to how they were originally.

```
\[\033[1;30m\]
\u \h \W \$
\[\033[0m\]
```

Bash: Resources and Sources

Resources:

- https://cs.mtsu.edu/ untch/4900/

Sources:

- Getting Started with BASH: A Bash Tutorial (`http://www.hypexr.org/bash_tutorial.php`)
- BASH Programming - Introduction HOW-TO by Mike G (`http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`)
- Advanced Bash-Scripting Guide: An in-depth exploration of the art of shell scripting by Mendel Cooper (`http://www.tldp.org/LDP/abs/html/`)