

Bash Programming

Kameswari Chebrolu



Introduction

- Command-line `bash` vs `bash script`
- Scripting: take a set of commands you'd normally run by hand and put them in a file. Why?
 - Automate tasks to make life easier!
 - Can run them repeatedly with one command!

Example

- A sequence of commands
- Invoke a script using `bash`
- Invoke a script as a hash-bang executable
 - `#!/usr/bin/env bash` (env will find the path, better!) or
 - `#!/bin/bash` (absolute path)
 - Above is called shebang
 - A line of code that tells the shell what program to use
 - `#` is used for commenting, hence ignored during execution
 - Want to use python, replace bash with python
 - Need to make the file executable via `chmod`

Variables

- Debug: use echo to print statements
- No typing, no need for declaration
 - Bash variables are character strings, but, depending on context permits integer operations
- Defined using the assign operator "="
 - e.g. s=hello
 - No space before or after!
 - Variable name can include alphabets, digits, and underscore; can be started with alphabets and underscore only; case sensitive
 - Value can be used with the expression \$s

Few things to note

- Special character need escaping via single quotes ('...') or back slash (\)
 - \" or \' or \\ or \\$
- Single and double quotes: Help group arguments with spaces
 - Single quotes (') preserve the literal value of each character
 - Double quotes (") preserve the literal value of all characters with the exception of \$, `, \

Environment Variables

- Variables in your system that describe your environment!
 - SHELL: what shell you're running
 - USER: username of the current user
 - PWD: present working directory
 - PATH: specifies the directories to be searched to find a command
 - etc
- “env” command shows the list

Arithmetic Expressions

- Some support but not great
- `let` is a built-in command in Linux systems used for evaluating arithmetic expressions
 - `let "x = 1" or let "a = a + 1"`
 - `let "var1 = 5" "var2 = 10" "var3=var1+var2"; echo $var3`
- `$((expression))`, `#[expression]`, declare `i` and `bc` are a few other options

If-then-else

Syntax:

- ... represent conditions
 - Conditions can be commands as well
 - Every command you run in shell returns a number
 - No error: returns 0
 - Error: usually 1 or -1 or some other number corresponds to the type of error that occurred
- Chained elif and the else parts are optional

```
if ...  
then  
    some statements  
elif ...  
    some statements  
else  
    some statements  
fi
```


Comparisons

-lt <

-gt >

-le <=

-ge >=

-eq ==

-ne !=

Loops

- Multiple types of loops: for, while, and until
- ... list of things (need not be numeric)
 - item is the loop variable which iterates through each item in the list
- while executes a piece of code if the control expression is true and only stops when it is false (or a explicit break is found)
- until loop is almost equal to the while loop, except that the code is executed while the control expression evaluates to false!

```
for item in [LIST]
do
  [COMMANDS]
done
```

```
while [CONDITION]
do
  [COMMANDS]
done
```

```
until [CONDITION]
do
  [COMMANDS]
done
```

Command Line Arguments and other Special Shell Variables

`$0` Name of the current shell script

`$1-$9` Positional parameters 1 through 9

`$#` The number of positional parameters

`$*` All positional parameters, “`$*`” is one string

`$@` All positional parameters, “`$@`” is a set of strings

`$?` Return status of most recently executed command

`$$` Process id of current process

Functions

- In Bash, functions emulate the way commands work
 - `$*`, `$#`, `$1`, `$2` ... (no explicit arguments, much like in command line arguments)
 - Do not return values in usual way
 - Any value sent back must be an integer which acts like the exit code of an executable

Local vs Global Variables

- Environment variables are global; inherited by any child shells or processes
 - Shell variables are only present in the shell in which they were defined
 - Shell variable can be made an environment variable by using export command

Arrays

- Arrays declared via `-a` command
- To access all elements in the array we can use `@`
- To get the number of the elements in array we can use `#`
- Use “unset” to delete elements

Read from a file

See relevant file

References

1. <https://linuxconfig.org/bash-scripting-tutorial>
(a good beginner's guide)
2. <https://www.javatpoint.com/bash> (another guide)
3. <https://tldp.org/LDP/abs/html/> (more advanced)