Bash Scripting Basics

Hydroplane

https://github.com/cdhop/bash_basics

Introduction

Bash Scripting Basics

- The GNU Bourne-Again Shell
- Project Started in 1989 (Current version: 4.4)
- Default shell for MacOS and many Linux/BSD distributions
- Available for Windows 10

Bash Scripting Basics

- Command Line Editing
- Command History
- Command Substitution (POSIX)
- Command Line Completion
- Piping and Redirection
- Job Control

- Built-in Commands are implemented inside Bash (Examples Include: cd, pwd, type, test, alias, etc)
- External Commands are binaries and other scripts (Examples Include: ls, mv, vi, grep, sed, awk, etc)
- Use ‘type’ to find out if a command is ‘Built-in’ or ‘External’

- For Built-in Commands use ‘help’
- For External Commands use ‘man’
- For meta-searches use ‘apropos’
- Frequently you can find documentation in the /usr/share/doc directory
- The Linux Documentation Project (tldp.org)

The ‘File’ is the ultimate abstraction in Unix (or Unix derived) systems. This includes:

- **Normal Files**
- **Devices**
- **Processes**

Everything is a File

Every program has at least three files associated with it:

- **In (0)**
- **Out (1)**
- **Error (2)**

- Piping (AKA: Inter-process communication) is used to chain commands together
- Redirection is used to change input, output, and errors from/to files instead of the terminal

Bash Scripting Basics

- Generally, there are three file permissions (Read, Write, and Executable)
- Generally, there are three entities that permissions can be assigned to (Owner, Group, and Others)
- Permissions are usually represented in Octal (Base 8)
- 4 (Read), 2 (Write), 1 (Executable)

File Permissions

Bash Scripting Basics

- Bash scripts are executable text files
- Bash scripts start with the line ‘#!/bin/bash’
- You can make a file executable with the ‘chmod +x file.sh’ command
- If a file is not in your PATH, then you must specify the location of the file to execute it (for example: ./file.sh)

Storing and Running Scripts

Variables and Parameters

Bash Scripting Basics

- **Definition/Assignment: FOO=bar**
- **Use: echo $FOO**

Defining and Using Variables

- Used to obtain input from a user during the execution of a script
- Example: read FOO; echo $FOO

The ‘read’ Command

- A Bash script is executed in a ‘subshell'
- Usually, variables are limited in scope to the shell they were created
- If you want make a variable available to a ‘subshell’ then ‘export’ it (example: export FOO=bar)

Bash Scripting Basics

- You can include other scripts/files in your scripts (similar to include/import/require statements in other languages)
- Example: source file_to_be_included

Sourcing

- **Double Quotations allow the shell to interpret expressions/variables within them (Example: “Hello, $NAME”)**
  OUTPUT: Hello, Bob
- **Single Quotations do not allow the shell to interpret expressions/varables within them, and must be taken literally**
  (Example: ‘Hello, $NAME’)
  OUTPUT: Hello, $NAME

- Scripts can access arguments/parameters passed to them when they are invoked
- The first argument is accessed using $1, the second argument using $2, and so forth
- The collection of arguments can be accessed using $*

- You can substitute the output of a command within a script
  Example: FOO=$(whoami)
- You may encounter the old form of command substitution (using the grave/backtick characters)
  Example: FOO=`whoami`

Transforming Input

Bash Scripting Basics

- Used to manipulate and/or expand variables
- Also known as: Parameter Substitution
- Uses: default parameters, substrings, string length, etc

Substitution/Pattern Matching Operators

- Internal (Integer) calculations can be performed using (()) and let
- More involved calculations can be performed using the external command ‘bc’

Essential External Commands

- String searching algorithms
- Think of them as advanced find/replace
- Tutorial: regexone.com
- Presentation: DC801 Presents: Regular Expression - Globby March 2017

Bash Scripting Basics

- Used to perform regular expression searches
- Derived from the ‘ed’ command: g/re/p
- Globally search a Regular Expression and Print
- The GNU version (usually found in Linux) will differ from the BSD version

Bash Scripting Basics

- Used to manipulate and order textual data
- Cut allows you to ‘cut out selected portions of each line of a file’
- Sort allows you to ‘sort lines of text files’
- Frequently used together to process output from other commands

CUT and SORT

Bash Scripting Basics

- Used to edit streams of textual data
- Can do inline search and replace
- Frequently used in conjunction with other commands

SED

- Used for more advanced text processing
- Extremely terse programming language
- Helped inspire Perl

Bash Scripting Basics

- **(Tr)anslates ‘the standard input to the standard output with the substitution or deletion of selected characters’**

TR

Flow Control

Bash Scripting Basics

- Frequently used to evaluate expression(s) that determine the outcome of flow control structures
- Internal command that returns 0 for True and 1 for False
- Can be used with && and || to create single-line if-then-else statements

Test

Bash Scripting Basics

- Provides basic flow control functionality to Bash
- If its exit status is zero (successful), then the ‘then COMMANDS’ is executed
- Otherwise, each ‘elif COMMANDS’ list is executed in turn. If their exit status is zero, then their 'then COMMANDS’ are executed.
- Finally, if the 'if' command’s exit status is not zero, then the ‘else COMMANDS’ are executed.
- ‘fi’ is used to close the if flow control structure

- Used to selectively execute commands based upon word matching patterns.
- Each matching conditions commands are terminated by a ‘;;’
- The ‘esac’ command terminates the ‘case’ flow control structure

- Used to execute a sequence of commands for each member in a list of items
- The commands for a ‘for' loop start after the word ‘do’, and end before the word ‘done’

- The ‘while’ command executes a loop as long as its argument evaluates to zero
- The ‘until’ command executes a loop as long as its argument evaluates to NOT zero

Some Advanced Stuff

- You can use the ‘getopts’ internal command to parse positional parameters (for example: -d)
- Usually options are used to modify the behavior of a program/script

Options

- Functions are blocks of code that can be invoked inside scripts
- Arguments to the function can be accessed using the $0 .. $n variables

- Arrays are variables that contain multiple values
  Example: FOO=(one two three)
- Arrays can be indexed by position
  Example: echo ${FOO[0]}
- Arrays that are not indexed return the value of their first element

- Menu interfaces can be created using the ‘select’ internal command
- Menu interfaces frequently use ‘case’ statements
- Signals can (and usually are) ‘trap’ed to prevent users from escaping from the menu interface.

Bash Scripting Basics

- Use the ‘bash -v script_name’ command to get verbose details
- Use the ‘bash -n script_name’ command to check for syntax errors
- Use the ‘bash -x script_name’ command to get debug details for the entire script
- Use ‘trap DEBUG’ inside your scripts to get debug details for specific parts of your script

Interesting Examples

Questions?

Go away or I will replace you with a very small shell script.