Story

The Three Bricklayers

Introduction to Shell Programming

- Shell programming is one of the most powerful features on any UNIX system
- If you cannot find an existing utility to accomplish a task, you can build one using a shell script

Shell Program Structure

- A shell program contains high-level programming language features:
  - Variables for storing data
  - Decision-making control (e.g. if and case statements)
  - Looping abilities (e.g. for and while loops)
  - Function calls for modularity
- A shell program can also contain:
  - UNIX commands
  - Pattern editing utilities (e.g. grep, sed, awk)

Your Shell Programming Library

- Naming of shell programs and their output
  - Give a meaningful name
  - Program name example: findfile.csh
  - Do not use: script1, script2
  - Do not use UNIX command names
- Repository for shell programs
  - If you develop numerous shell programs, place them in a directory (e.g. bin or shellprogs)
  - Update your path to include the directory name where your shell programs are located

Steps to Create Shell Programs

- Specify shell to execute program
  - Script must begin with #! (pronounced "shebang") to identify shell to be executed

Examples:

```
#! /bin/sh          (defaults to bash, but be explicit)
#! /bin/bash
#! /bin/csh
#! /usr/bin/tcsh
```

- Make the shell program executable
  - Use the "chmod" command to make the program/script file executable

Chmod

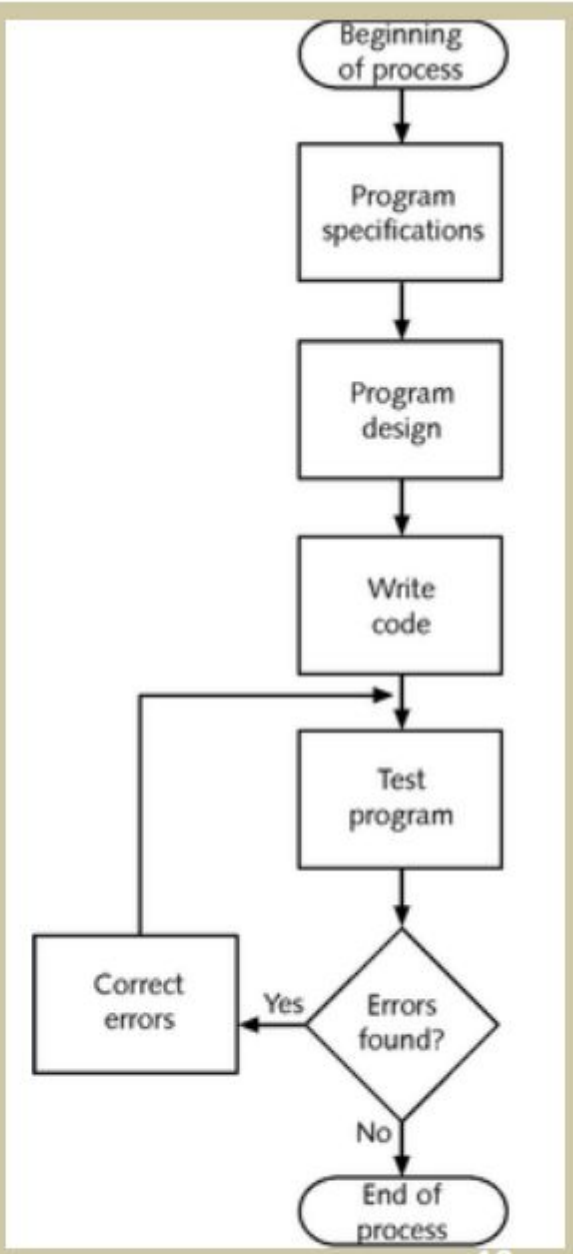
```
chmod +rwx myfile

chmod u=rwx,g=rx,o=r myfile
chmod 754 myfile

4 stands for "read"
2 stands for "write"
1 stands for "execute"
0 stands for "no permission"

7 = 4 + 2 + 1 (read, write, execute)
5 = 4 + 0 + 1 (read, no write, execute)
4 = 4 + 0 + 0 (read, no write, no execute)
```

Formatting Shell Programs

- Formatting of shell programs
  - Indent areas (3 or 4 spaces) of programs to indicate that commands are part of a group
  - To break up long lines, place a \ at the end of one line and continue the command on the next line
- Comments
  - Start comment lines with a pound sign (#)
  - Include comments to describe sections of your program
  - Help you understand your program when you look at it later

Steps of Programming

- Guidelines:
  - use good names for
    - script
    - variables
  - use comments
    - lines start with #
  - use indentation to reflect logic and nesting



Example: "HELLO" Script

```
#! /bin/bash
echo "Hello $USER"
echo "This machine is `uname -n`"
echo "The calendar for this month is:"
cal
echo "You are running these processes:"
ps
```

Variables

- 2 types of variables
  - Environment
    - valid for complete login session
    - upper case by convention
  - Shell
    - valid for each shell invocation
    - lower case
- To display value

  echo $variable

Predefined Shell Variables

Shell Variable	Description
PWD	The most recent current working directory.
OLDPWD	The previous working directory.
BASH	The full path name used of the bash shell.
RANDOM	Generates a random integer between 0 and 32,767
HOSTNAME	The current hostname of the system.
PATH	A list of directories to search of commands.
HOME	The home directory of the current user.
PS1	The primary prompt (also PS2, PS3, PS4).

User-defined Shell Variables

- Syntax:

```
varname=value
```

Example:

```
rate=7.65
echo "Rate today is: $rate"
```

- Use double quotes if the value of a variable contains white spaces

Example:

```
name="Thomas William Flowers"
```

Numeric variables

- Syntax:

```
let varname=value
```

- Can be used for simple arithmetic:

```
let count=1
let count=$count+20
let count+=1
```

Variable cheatsheet

```
a=hello                 # Bash is untyped
a=2

a=”hello world”         # string
let a=2                 # number

a=$((1+2))              # arithmetic expansion substitutes output of operation
                        #CANNOT be used in conditional statements

while ((...)) ; do      # arithmetic evaluation works for conditionals
                        # CANNOT be used in variable assignments

a=$(ls -l)              # command substitution
a=$(expr 1 + 2)         # command substitution; expr is a command-line calculator
                        # output is same as $((1+2))
a=`ls -l`               # command substitution
echo “$a”               # quotes preserve white space
                        # same as surrounding a string with quotes

a=$[1+2]                # old syntax for arithmetic expansion; deprecated
[ $a -eq $b ]           # test command; note the spaces
```

Variables commands

- To delete both local and environment variables

  `unset varname`

- To prohibit change

  `readonly varname`

- list all shell variables (including exported)

  `set`

Shell Scripting Tutorials

Bash basics:

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

Advanced:

http://tldp.org/LDP/abs/html