Introduction to Bash Shell

What is Shell?

- The shell is a command interpreter.
- It is the layer between the operating system kernel and the user.

Some Special characters used in shell scripts

- #:Comments
- ~:home directory

Invoking the script

- The first line must be “#!/bin/bash”.
  - setup the shell path
- **chmod u+x scriptname** (gives only the script owner execute permission)
- ./scripname

Some Internal Commands and Builtins

- getopts:
  - parses command line arguments passed to the script.
- exit:
  - Unconditionally terminates a script
- set:
  - changes the value of internal script variables.
- read:
  - **Reads" the value of a variable from stdin**
  - **also "read" its variable value from a file redirected to stdin**
- wait:
  - Stop script execution until all jobs running in background have terminated

Some Internal Commands and Builtins (cont.)

- grep:
  - grep pattern file
    - search the files file, etc. for occurrences of *pattern*
- expr:
  - evaluates the arguments according to the operation given
    - **y=`expr $y + 1`** (same as **y=$(($y+1))**

I/O Redirection

- >: Redirect stdout to a file, Creates the file if not present, otherwise overwrites it
- < : Accept input from a file.
- >>: Creates the file if not present, otherwise appends to it.
- <<:
  - Forces the input to a command to be the shell's input, which until there is a line that contains only *label*.
  - cat >> mshfile << .
- |:pipe, similar to ">",

if

```
if [ condition ] then
  command1
elif   # Same as else if
 then
          command1
 else
  default-command
fi
```

case

```
x=5
case $x in
  0) echo "Value of x is 0."
    ;
  5) echo "Value of x is 5."
    ;
  9) echo "Value of x is 9."
    ;
  *) echo "Unrecognized value."
esac
done
```

Loops

- for [arg] in [list];
  do
  command
  done
- while [condition];
  do
  command...
  done

Loops (cont.)

- break, continue
  - **break** command terminates the loop
  - **continue** causes a jump to the next iteration of the loop

Introduction to Variables

- $: variable substitution
  - If **variable1** is the name of a variable, then **$variable1** is a reference to its *value*.

Pattern Matching

- ${variable#pattern}
- ${variable##pattern}
- ${variable%pattern}
- ${variable%%pattern}

Examples of Pattern Matching

```
x=/home/cam/book/long.file.name
echo ${x#/*/}
echo ${x##/*/}
echo ${x%.*}
echo ${x%%.*}
```

cam/book/long.file.name
long.file.name
/home/cam/book/long.file
/home/cam/book/long

Aliases

- avoiding typing a long command sequence
- Ex: alias lm="ls -l | more"

Array

- Declare:
  - declare -a array_name
- To dereference (find the contents of) an array variable, use *curly bracket* notation, that is, **${ array[xx]}**
- refers to *all* the elements of the array
  - *${array_name[@]}* or *${array_name[*]}*
- get a count of the number of elements in an array
  - *${#array_name[@]}* or *${#array_name[*]}*

Functions

- Type
  - *function function-name* {
    command...
    }
  - *function-name* () {
    command...
    }
- Local variables in function:
  - Declare: local var_name
- functions may have arguments
  - function-name $arg1 $arg2

Positional Parameters

- $1, $2, $3 …..
- $0 is the name of the script.
- The variable $# holds the number of positional parameter.

Positional Parameters in Functions

- $1, $2, $3….
- Not from $0

Files

- /etc/profile
  - systemwide defaults, mostly setting the environment
- /etc/bashrc
  - systemwide functions and and aliases for Bash
- $HOME/.bash_profile
  - user-specific Bash environmental default settings, found in each user's home directory
- $HOME/.bashrc
  - user-specific Bash init file, found in each user's home directory

Debugging

- The Bash shell contains no debugger, nor even any debugging-specific commands or constructs.
- The simplest debugging aid is the output statement, **echo**.
- Set option
  - -n: Don’t run command; check for syntax error only
  - -v: Echo commands before running them
  - -x: Echo commands after command-line processing