Shell Scripting

and

Regular Expressions

CS 35L

Slide Set 2.2

Winter 2020 - Lab 1

If statements

- If statements use the test command or []
- man test
  - to see the expressions that you can create
  - e.g:

```
#!/bin/bash
if test "$#" -ne 2; then        #if number of args not equal to 2
then...
    echo "Illegal number of parameters"
else
    if [ $1 -gt $2 ]; then      #if arg $1  >= arg $2 ...
        echo "1st argument is greater than 2nd"
    else
        echo "not possible"
    fi
fi
```

If statements

- Use: -eq, -lt, -gt, … for integers
- Use ==, >, <, !=, … for strings

- Reference: http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html

```
#!/bin/bash
if test "$#" -ne 2; then          #if number of args not equal to 2
then...
    echo "Illegal number of parameters"
else
    if [ $1 -gt $2 ]; then        #if arg $1  >= arg $2 ...
        echo "1st argument is greater than 2nd"
    else
        echo "not possible"
    fi
fi
```

If statements

```
if condition
then
     statements-if-true-1
elif condition
then
     statements-if-true-2
else
  statements-if-all-else-fails ]
fi
```

Example

```
if grep pattern myfile > /dev/null
then
  ... Pattern is there
else
  ... Pattern is not there
fi
```

case Statement

```
case $1 in
-r)
   ... Code for -r option (recursive)
   ;;
-d | --directory) # long option allowed
   ... Code for -d option
   ;;
*) #all strings (default)
   echo $1: unknown option >&2
   exit 1 # ;; is good form before `esac', but not required
esac
```

for Loops

Generally follows form of foreach loop, for example:

```
for i in atlbrochure*.xml
do
    echo $i
    mv $i $i.old
done
```

while and until loops

Standard syntax for while loops:

```
while condition
do
  statements
done
```

Also supports negation of condition (stop when condition is true):

```
until condition
do
  statements
done
```

break and continue

- Pretty much the same as in C/C++

Functions

- Semantically, calling a function is very similar to invoking another bash script.
- Must be defined before they can be used
- Can be done either at the top of a script or by having them in a separate file and source them with the “dot” (.) command.

Example

```
func_name ( ) {
    while condition
    do
        echo $1   #$1 is first argument passed to function
    done
}
```

Functions are invoked the same way a command is:

```
func_name arg1     #function call
```

Accessing Shell Script Arguments

- Positional parameters represent a shell script's command line arguments
- For historical reasons, enclose the number in braces if greater than 9

```
#! /bin/bash
#test script
echo first arg is $1
echo tenth arg is ${10}
echo all args is $@

> ./argtest 1 2 3 4 5 6 7 8 9 10
```

Function Return

The return command serves the same function as exit and works the same way:

```
answer_the_question ( ) {
    ...
    return 42
}
```

Exit: Return value

Check exit status of last command that ran with `echo $?`

Value	Meaning
0	Command exited successfully
>0	Failure during redirection
1-125	Command exited unsuccessfully. The meanings
126	Command found, but file was not executable
127	Command not found
>128	Command died due to receiving a signal

Quotes

Assigned variable: `world=42`

- Three kinds of quotes
  - Backticks: ``
    - `echo \`ls $world\` -> <result of ls 42>`
    - Same as: `echo $(ls $world)`
  - Double quotes: “”
    - `echo “ls $world” -> ls 42`
  - Single quotes: ‘’
    - `echo ‘ls $world’ -> ls $world`

Simple Execution Tracing

- To get shell to print out each command as it's execute, precede it with "+"
- You can turn execution tracing within a script by using:

  `set -x`: to turn it on

  `set +x`: to turn it off

POSIX Built-in Shell Variables

Variable	Meaning
#	Number of arguments given to current process.
@	Command-line arguments to current process. Inside double quotes, expands to individual arguments.
*	Command-line arguments to current process. Inside double quotes, expands to a single argument.
- (hyphen)	Options given to shell on invocation.
?	Exit status of previous command.
$	Process ID of shell process.
0 (zero)	The name of the shell program.
!	Process ID of last background command. Use this to save process ID numbers for later use with the *wait* command.
ENV	Used only by interactive shells upon invocation; the value of $ENV is parameter-expanded. The result should be a full pathname for a file to be read and executed at startup. This is an XSI requirement.
HOME	Home (login) directory.
IFS	Internal field separator; i.e., the list of characters that act as word separators. Normally set to space, tab, and newline.
LANG	Default name of current locale; overridden by the other LC_* variables.
LC_ALL	Name of current locale; overrides LANG and the other LC_* variables.
LC_COLLATE	Name of current locale for character collation (sorting) purposes.
LC_CTYPE	Name of current locale for character class determination during pattern matching.
LC_MESSAGES	Name of current language for output messages.
LINENO	Line number in script or function of the line that just ran.
NLSPATH	The location of message catalogs for messages in the language given by $LC_MESSAGES (XSI).
PATH	Search path for commands.
PPID	Process ID of parent process.
PS1	Primary command prompt string. Default is "$ ".
PS2	Prompt string for line continuations. Default is "> ".
PS4	Prompt string for execution tracing with set -x. Default is "+ ".
PWD	Current working directory.

$IFS (Internal Field Seperator)

- This variable determines how Bash recognizes fields, or word boundaries, when it interprets character strings.
- $IFS defaults to whitespace (space, tab, and newline), but may be changed
- **echo "$IFS"** (With $IFS set to default, a blank line displays.)
- More details:
  - http://tldp.org/LDP/abs/html/internalvariables.html

Hints for Lab

- Step1: Use sed to remove ?, <u>, </u>,
- Step 2: Use sed to remove \n from English lines that wrap to a new line (do it twice)
  - Look for a pattern where character before end of line is not >
  - Append next line to current line ({N; ….})
  - Substitute \n with empty space
- Step 3: Use grep to find all lines with <td> tags
- Step 4: Delete HTML stuff at top of file (using sed)
- Step 5: Delete English lines (sed)
- Step 6: Remove all tags (sed)
- Step 7: Treat ASCII ` to ASCII ‘ (tr)
- Step 8: Make everything lowercase (tr)
- Step 9: Replace comma, space, dash with newline (tr)
- Step 10: Delete all leftover empty lines (sed)
- Step 11: Remove all non-Hawaiian letters (sed)
- Step 12: Sort list and remove duplicates (sort -u)