Shell Scripting

Week 2

Shell Script

- The ability to enter multiple commands and combine them logically
- Must specify the shell you use in the first line
  - #!/bin/bash
    (# itself can lead comments)
- You can create easiest shell script by listing commands in separate lines

  (shell will process commands in order)

Q: How to write a shell script that can find the logged in user with username “betsy”?

- What commands to use?
- How to write and execute the script?

Example:

```
$ who | grep betsy                    Where is betsy?
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
```

Script:

```
#! /bin/sh
# finduser --- see if user named by betsy is logged in
who | grep betsy
```

Run it:

```
$ chmod +x finduser                   Make it executable
$ ./finduser                          Test it: find betsy
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
```

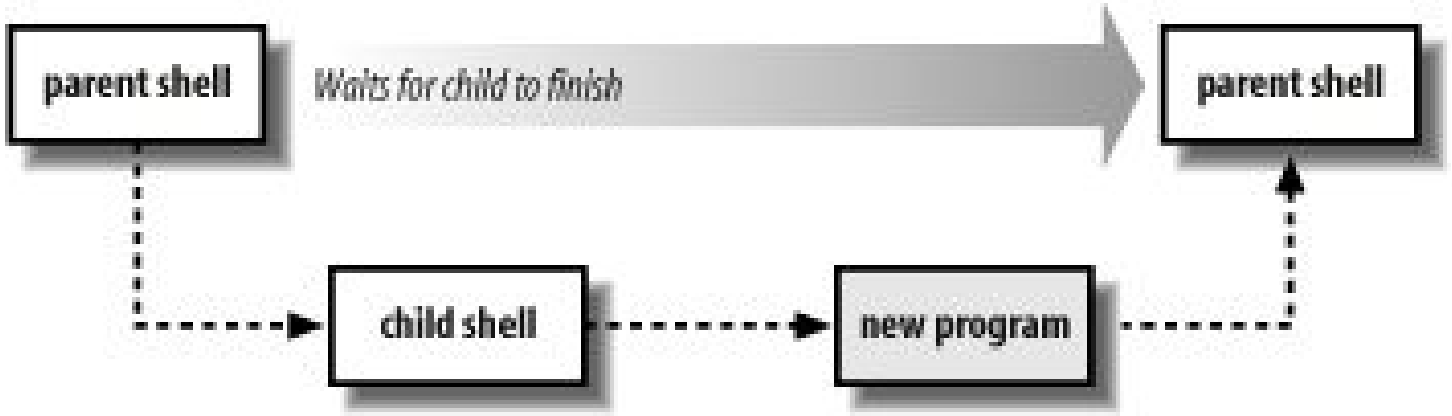
Self-Contained Scripts: The #! First Line

- When the shell runs a program, it asks the kernel to start a new process and run the given program in that process.
- But if there is more than one shell installed on the system, we need a way to tell the kernel which shell to use for a script

  #! /bin/csh –f

  #! /bin/awk –f

  #! /bin/sh
- A shell can initiate a shell different from itself



Variables

- Allows you to temporarily store info and use it later
- Two general types
  - Environment variables
  - User defined variables (UDV)
- Environment variables
  - Created and maintained by Linux itself
  - Track specific system info
  - defined in CAPITAL LETTERS
  - ex: $PATH, $PWD
- User defined variables (UDV)
  - Created and maintained by user
  - defined in lower letters

- When refer a variable value, use dollar sign
  - echo $PWD
- When refer a variable to assign a value to it, do not use dollar sign (no space around =)
  - myvar=helloworld
- export: puts variables into the environment
  - Environment is a list of name-value pairs that is available to every running program
- env: Displays the current environment
- unset: remove variable and functions from the current shell

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.

- PATH
  - list of directories in which commands are found
  - echo $PATH
  - export $PATH = /usr/local/bin : $PATH

    (Prepend a new dic path in $PATH)
- ?
  - Show exit status of previous command
  - 0 means exit normally
  - Otherwise exit with some errors
- $IFS
  - Define a list of field separators

Linux Exit Status Codes

Code	Description
0	Successful completion of the command
1	General unknown error
2	Misuse of shell command
126	The command can't execute
127	Command not found
128	Invalid exit argument
128+x	Fatal error with Linux signal x
130	Command terminated with Ctl-C
255	Exit status out of range

Quote

- Single quote '
  - Literal meaning of everything within ''
  - echo '$PATH'
- Double quote "
  - Literal meaning of everything except $ \ `
  - echo "the current directory is $PWD"
- The backtick `
  - Execute the command
  - Allow you to assign the output of a shell command to a variable
  - testing `date`

Q: How to write a shell script that can find the logged in user with username “edison”?

- Can I generalize the script to find any user name?

Command Line Parameters

- Allow user to pass data to script before execution
- Positional parameters
  - represent all parameters entered in a command line
- $0: the name of the program

  $1: the first parameter

  ${10}: the 10th parameter

Example:

```
$ who | grep betsy                    Where is betsy?
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
```

Script:

```
#! /bin/sh
# finduser --- see if user named by first argument is logged in
who | grep $1
```

Run it:

```
$ chmod +x finduser                   Make it executable
$ ./finduser betsy                    Test it: find betsy
betsy pts/3 Dec 27 11:07 (flags-r-us.example.com)
$ ./finduser benjamin                 Now look for Ben
 benjamin dtlocal Dec 27 17:55 (kites.example.com)
```

Structured commands

- Alter the flow of operations based conditions
- If statement
- For statement
- While loops
- Case statement
- Break statement
- Continue statement

IF-THEN Statement

- ```
  if command
  then
       commands
  fi
  ```
- If the exit status of command is zero (complete successfully), the command listed under then *then* section are executed
- ```
  #!/bin/bash
  # testing the if statement
  if date
  then
     echo "it worked"
  fi
  ```

More IF Statement

```
if command
then
    commands
else
    commands
fi
```

```
if command1
then
     command set 1
elif command2
then
    command set 2
elif command3
then
    command set 3
elif command4
then
    command set 4
fi
```

Test command

- The ability to evaluate any condition other than the exit code of a status
  (i.e. evaluate true/false)
-
```
if test condition
then
    commands
fi
```

```
if [ condition ]
then
    commands
fi
```

- If the condition listed in the test command is true, the test command exits with 0

Test command

- Three classes of conditions
  - Numeric comparisons
  - String comparisons
  - File comparisons

```
if test condition
then
    commands
fi
```

```
if [ condition ]
then
    commands
fi
```

- Numeric comparisons
- Evaluate both numbers and variables
- Ex: $var -eq 1
  $var1 -ge $var2

The test Numeric Comparisons

Comparison	Description
n1 -eq *n2*	Check if *n1* is equal to *n2*.
n1 -ge *n2*	Check if *n1* is greater than or equal to *n2*.
n1 -gt *n2*	Check if *n1* is greater than *n2*.
n1 -le *n2*	Check if *n1* is less than or equal to *n2*.
n1 -lt *n2*	Check if *n1* is less than *n2*.
n1 -ne *n2*	Check if *n1* is not equal to *n2*.

- String comparisons
- The greater-than and less-than symbols must be escaped (otherwise will be interpreted as redirection)
- The greater-than and less-than order is not the same as sort (ASCII vs. locale language)
- Ex: $USER = $testuser

The test Command String Comparisons

Comparison	Description
str1 = *str2*	Check if *str1* is the same as string *str2*.
str1 != *str2*	Check if *str1* is not the same as *str2*.
str1 < *str2*	Check if *str1* is less than *str2*.
str1 > *str2*	Check if *str1* is greater than *str2*.
-n *str1*	Check if *str1* has a length greater than zero.
-z *str1*	Check if *str1* has a length of zero.

- File comparisons
  - Test the status of files and directories in linux file system

The test Command File Comparisons

Comparison	Description
`-d file`	Check if *file* exists and is a directory.
`-e file`	Checks if *file* exists.
`-f file`	Checks if *file* exists and is a file.
`-r file`	Checks if *file* exists and is readable.
`-s file`	Checks if *file* exists and is not empty.
`-w file`	Checks if *file* exists and is writable.
`-x file`	Checks if *file* exists and is executable.
`-O file`	Checks if *file* exists and is owned by the current user.
`-G file`	Checks if *file* exists and the default group is the same as the current user.
file1 `-nt` *file2*	Checks if *file1* is newer than *file2*.
file1 `-ot` *file2*	Checks if *file1* is older than *file2*.

FOR Statement

```
for var in list
do
   commands
done
```

```
for (( i=1; i <= 10; i++ ))
do
   echo "The next number is $i"
done
```

- For each value in the list, do a set of operations on it
- example

```
for test in Alabama Alaska Arizona
do
   echo The next state is $test
done
```

- Q: how does computer know how to split the list?

Q: how does computer know how to split the list?

- $IFS
  - Internal field separator
  - Define a list of characters the bash shell uses as field separators
  - Default values: space, tab, newline
  - Can change value of $IFS to split list in different ways
  - Better store original values and restore later

```
IFS.OLD=$IFS
IFS=$'\n'
<use the new IFS value in code>
IFS=$IFS.OLD
```

Other loop statements

```
while test command
do
  other commands
done
```

```
until test commands
do
    other commands
done
```

- In while loop, commands in the loop are executed as long as the exit code of test command is 0
- In until loop, as long as the exit status of the test command is non-zero, commands listed in the loop are executed

Homework: find duplicate files

- Input argument: the path of directory
- Usage: ./sameln [directory name]
- Output: a list of regular files immediately under the given directory which have duplicates
- First line: #!/bin/bash

Homework: find duplicate files

- Some tips
  - Only consider files immediately under given directory
    (hints: find -maxdepth 1 -type f)
  - For duplicates, keep the one whose name is lexicographically first, replace other files with hard links to the first one
    (hints: use ln command)
  - Don't forget hidden files that begin with . !
    (hints: ls -a [directory] | grep '^\.')
  - Ignore non-regular and not readable files
  - File names may contain special characters (e.g. space, *, -)

Scripting Languages VS Compiled Languages

- Compiled Languages
  - Programs are translated from human-readable code to machine-readable code by compiler
  - Efficient
  - Ex: C/C++, Java
- Scripting languages
  - rely on source-code all the time
  - Interpreter reads program, translates it into internal form, and execute programs on the fly
  - Inefficient (translation on the fly)
  - Ex: Python, Ruby, PHP, Perl