Lecture # 6 – The Interactive Bourne Shell (Chapter 7)

- History

Bourne shell was the first shell to become a part of UNIX
Rarely used today, but it is the foundation for many other shells (ksh, bash, posix, zsh)

Korn shell came out with System V and contained many additions and improvements
Bash is the GNU/Linux alternative to Bourne shell and Korn shell

- Assignment statements

```
VARIABLE=value
```

Note: There are no spaces on either side of the equals sign!
Note: If value contains white space, you must enclose the value in quotes

Examples:

```
TODAY=Monday
```

- The initialization files

On login, Bourne shell runs the commands in /etc/profile followed by $HOME/.profile.

/etc/profile contains system wide settings (i.e. basic path setup, etc.)
$HOME/.profile contains settings for your account

- The prompts

Bourne shell provides two prompt – the primary prompt ($) and the secondary prompt (>)
The primary prompt is controlled by the variable PS1
The secondary prompt is controlled by the variable PS2

```
$  PS1=”`hostname`>”
```

- The search path

The path is an ordered list of directories that the shell uses to locate executable commands.

```
$  echo $PATH
/usr/bin:/usr/local/bin:.

$  PATH=$HOME:/usr/bin:/usr/local/bin:.
$  export PATH
```

- The dot command

  Under normal circumstances, commands entered at the command line are executed in a separate process (a copy of the current shell).

  The dot command is used to run a command in the current shell (i.e. allows you to change variables in the current shell).

  Re-execute the .profile script:
  ```
  $  .  .profile
  ```

  What happens if we do not use the dot command to execute the .profile script?

- The exit status

  When a command terminates, it returns an exit status to the parent process.
  A zero (0) exit status indicates success, and a non-zero status indicates failure.
  The shell variable ($?) contains the value of the exit status for the most recent command.

  ```
  $  grep testing myfile
  $  echo $?
  1
  ```

- Command grouping and separation

  Commands can be separated by a newline or “;”
  Examples:

  ```
  $ a
  $ b
  $ c
  ```

  is the same as $ a ; b ; c (Spaces only included to help readability)

  You can use the back slash (‘\’) to continue long lines
  Recall the pipe symbol (‘|’) from chapter 5

  Background (&) returns the prompt immediately while running your command in the background.

  Subshells are started with parentheses ().
  For example, (a ; b) & c creates a subshell to run a & b in the background, and runs c in the foreground.

- Filename substitution

  When evaluating the command line, the shell uses meta-characters to abbreviate filenames or pathnames that match a certain set of characters.

  File substitution meta-characters:

*	matches zero or more characters
?	matches exactly one character
[abc]	matches one character in the set (i.e. a or b or c)
[a-z]	matches one character in the range a to z
[!a-z]	matches one character not in the range a to z
\	quotes the next character

  List all files in the current directory that start with “f”:
  ```
  $  ls f*
  ```

  Display all files that end with “.txt”:
  ```
  $  cat *.txt
  ```

  List all files that are named “file” followed by 2 characters:
  ```
  $  ls file??
  ```

  List all files like a1, a2, and a3:
  ```
  $  ls a[123]
  ```

- Redirecting standard error

  Stderr is redirected with “2>” command

  For example, if the file y exists but x does not:

  ```
  $ cat y
  This is y.

  $ cat x y
  cat: x: No such file or directory
  This is y.

  $ cat x y > hold
  cat: x: No such file or directory
  $ cat hold
  This is y.

  $ cat x y 1>hold1 2>hold2
  $ cat hold1
  ```

```
This is y.
$ cat hold2
cat: x: No such file or directory
```

Can use “dup” command to send stderr and stdout to same place (2>&1)

```
$ cat x y > hold 2>&1
$ cat hold
cat: x: No such file or directory
This is y.
```

- Variables

  User-created versus shell variables (PATH, HOME, etc)
  You can change user variables at any time; can be made readonly; can be exported

- User-created variables

  Variable can be any combination of letters and digits as long as first character is a letter.

  VARNAME=value (No spaces before or after the equals sign, quotes around values if it contains imbedded spaces)
  When using the variable, you use $VARNAME.

```
$  COUNT=10
$  echo COUNT
COUNT
$  echo $COUNT
10
```

- Quoting variables:

```
$  echo $COUNT
10
$  echo “$COUNT”
10
$  echo ‘$COUNT’
$COUNT
$  echo \$COUNT
$COUNT
```

- Removing variables:

```
        VARNAME=
OR
        unset VARNAME
```

- Readonly command

  The readonly command allows us to protect a variable from being changed

```
$  person=jenny
$  echo $person
jenny
$  readonly person
$  person=Helen
person: is read only
```

  If you use the readonly command with any arguments, then it displays a list of readonly variables.

- Export command

  Variables are normally only accessible in the current shell.
  The export command allows access to a variable in child shells.

  Export is call by value, each child receives a copy of the variable, and cannot affect the parent.

```
$  cat extest1
cheese=American
echo “extest1 1: $cheese”
subtest
echo “extest1 2: $cheese”
$  cat subtest
echo “subtest 1: $cheese”
cheese=swiss
echo “subtest 2: $cheese”
$ extest1
extest1 1: american
subtest 1:
subtest 2: swiss
extest1 2: american
```

  Notice that subtest did not get the initial value of cheese, and extest1 was not affected by subtest’s reassignment of cheese to swiss.

```
$  cat extest2
export cheese
cheese=American
echo “extest2 1: $cheese”
subtest
echo “extest2 2: $cheese”
```

```
$   extest2
extest2 1: american
subtest 1: american
subtest 2: swiss
extest2 2: american
```

Notice this time that subtest has the initial value of cheese (american), but extest2 is unaffected by subtest changing the value to swiss.

- Shell Variables

  HOME = your home directory
  cd without any arguments goes to the directory pointed to by this variable

  PATH = ordered list of directories to search for an executable
  Directories are separated by a colon

  MAIL = name of the file that stores your email (normally /var/mail/username)

  PS1 = primary shell prompt string (example: PS1="`hostname`: ")
  PS2 = secondary prompt string (normally '>') used for line continuation

  TZ = time zone (example: CST6CDT)

- Readonly shell variables

  Name of the calling program ($0)

  Command line arguments:

    Positional command line arguments ($1, $2, $3, …, $9)
    $* = All arguments

    $@ same as $* but when you put "" around them
      ($* puts one set around all args, but $@ puts one set around each arg)

    $# = number of arguments

  The shift command promotes each command line arg by one (i.e. $2 becomes $1, etc)

  The set command can be used to set $1-$9.

  For example, "set this is it" causes $1= this, $2 = is, $3 = it

```
$  cat dateset
set `date`
```

```
echo $*
echo
echo $2, $3, $6
$  dateset
Fri Jun 17 23:04:13 PDT 1994
Jun 17, 1994
```

- Command Substitution

  Command substitution involves running a command and substituting the output in place of the quoted command.

  Often, we would like to save the results of a command in a variable.
  Back quotes (`) allow us to do this.

```
$  DIR=`pwd`
```

- Here Documents

  Allows you to redirect input to a shell script from within the shell itself.
  Literally means “the document is here”
  Symbol is “<<” followed by an ending delimiter that must be on a line by itself.

```
$  cat birthday
grep –I “$1” <<ENDING
Alex    June 22
Helen   March 13
Rich    Jan 1
ENDING

$  birthday Rich
Rich    Jan 1
```

  This is extremely useful when imbedding things like longer awk scripts into a shell script.

- Exec Command

  Exec runs another command (script or executable) in place of the current shell, and does not return.

```
$  cat exec_demo
who
exec date
echo “Never reached”

$  exec_demo
```

<who output>
<date output>

Another use of exec is to redirect stdin, stdout, or stderr from within a script.
Example: exec >outfile 2>errfile < infile

- Trap Command

The trap command can be used to catch certain signals.
There are many conditions which generate signals (intr character, terminal disconnect)

Hang up	1
Interrupt	2
Quit	3
Kill	9
Software Term	15
Stop	18

trap ‘commands’ signal numbers

Example: trap ‘echo PROGRAM INTERRUPTED; exit 1’ 2

- Functions

```
function-name()
{
        commands
}
```

$1, $2, … $9 are arguments to function

Example:

```
        welcome()
        {
                echo “Hi $1 and $2”;
        }

        welcome tom joe
```