Lecture #12 – Bash (Chapter 13 and 14)

- Introduction

  Bash (Bourne again shell)
  Intended to be an upgrade to the original Bourne shell
  Contains many influences from csh and ksh
  Same basic programming interface as Bourne shell

  Originally written by Brian Fox in 1988 (v 0.99), later adopted by Chet Ramey
  Current versions are 3.0 ($ bash –version)

- Initialization Files

  First, run /etc/profile
  Second, run $HOME/.bash_profile if it exists
  If $HOME/.bash_profile does not exist, then $HOME/.bash_login is run
  If $HOME/.bash_login does not exist, then $HOME/.profile is run
  Finally, $HOME/.bashrc is run (name controlled by BASH_ENV variable)

- Shell Options

  Just like Korn shell, Bash support many options via the "set" command.

  ```
  $  set –o allexport                # turn on allexport option
  ```

  Example options:

  ```
  allexport
  emacs
  history
  ignoreeof
  verbose
  vi
  ```

- Prompts

  Bash uses the same prompt variables as Bourne and Korn shell (PS1, PS2)

  Example:

  ```
  $  PS1="$(uname –n) > "
  chargers>
  ```

Bash supports many special escape sequences for the prompt variables:

\h	hostname
\s	name of the shell
\d	date in Weekday Month Day format (i.e. Tue May 26)
\t	time in HH:MM:SS
\u	current username
\w	current working directory

Example:

```
$ PS1=”\u@\h:\w$ “
richj@chargers:/etc$
```

- Command history

HISTSIZE controls how many commands are remembered for history
HISTFILESIZE controls how many commands are stored in history file
HISTFILE controls the name of the history file ($HOME/.bash_history)
HISTIGNORE is a colon separated list to decide which commands are stored in history

You can use arrow keys to access commands from history.
You can use the history command to view command history (just like C-shell and Korn)

fc command:

-e editor	puts history list into editor
-l n-m	lists commands in range from n to m
-n	turns off numbering of history list
-s string	accesses command starting with string

```
$  fc –l
4   ls
5   history
6   pwd

$  fc –l –2
7   pwd
8   fc –l
```

You can also use C-shell style command rexecution (i.e. !!)

- Command line editing

Bash provides 2 built-in editors (vi and emacs) for command line editing

Example:

```
$  set –o vi
```

Works same way as Korn shell

- Variable basics

Same assignment and naming rules as Korn shell

declare builtin:

replaces typeset from Korn shell

-a	treat variable as an array
-f	lists function names and definitions
-F	lists just function names
-i	treat variable as integer
-r	makes the variable read only (can also use readonly)
-x	exports variable to subshells (can also use export)

Example:

```
$  declare name=Tommy
```

- Printf

printf format [argument…]

Example:

```
$  printf "%10.2f %5d\n" 10.5 25
```

- Variable expansion modifiers

${var:-word}	if var is set and non-null, sub its value, otherwise word
${var:=word}	if var is set and non-null, sub its value, otherwise word; make change to var permanently
${var:+word}	if var is set and non-null, sub word, otherwise nothing
${var:?word}	if var is set and non-null, sub its value, otherwise print word & exit
${var:offset}	get substring of value starting at offset
${var:offset:len}	get substring of value, starting at offset, for length characters

Examples:

```
$  fruit=peach
$  echo ${fruit:-plum}
peach
$  echo ${newfruit:-apple}
apple
```

${var%pattern}	matches smallest trailing portion of value and remove it
${var%%pattern}	
${var#pattern}	
${var##pattern}	

- Arithmetic expansion

Bash supports 2 methods for evaluating arithmetic expressions:

```
$[ expression ]
$(( expression ))
```

Examples:

```
$  echo $[ 5 + 4 – 2 ]
7
$  echo $(( 5 + 4 ))
9
```

- Reading user input

```
$  read answer          #  read line and assign it to answer
$  read first last      #  read line and assign 1st word to first and rest to last
$  read                 #  read line and assign it to REPLY
$  read –a arrayname    #  read line into array arrayname
$  read –p prompt       #  print prompt, wait for input, store result in REPLY
```

- Math

```
$   declare –i num
$   num=hello
$   echo $num
0
$  num = 5 + 5
bash: +: command not found
$  num=”4 * 6”
$  echo $num
24

$  x=5
$  let x = x + 1
$  echo $x
6
$  let “x += 1”
$  echo $x
7
```

- Test

Can use both single brackets [ ], double brackets [[ ]], and double paren (( )).

String test:

[string1 = string2]	string1 equals string2 (whitespace required)
[string1==string2]	alternative in Bash 2.x
[string1 != string2]	string1 not equal string2
[string]	string is not null
[-z string]	length of string is 0
[-n string]	length of string is non-zero
[-l string]	length of string (number of characters)

Logical test:

[string1 –a string2]	both string1 and string2 are true
[string1 –o string2]	either string1 or string2 is true
[! string1]	not string1
[[pattern1 && pattern2]]	both patterns are true
[[pattern1 || pattern2]]	either pattern1 or pattern2 is true
[[! pattern]]	not a pattern match

Integer test:

[int1 –eq int2]	int1 equals int2
[int1 –ne int2]	int1 not equal to int2
[int1 –gt int2]	int1 is greater than int2
[int1 –ge int2]	int1 is greater than or equal int2
[int1 –lt int2]	int1 is less than int2
[int1 –le int2]	int1 is less than or equal int2

Examples:

```
$  name=Tom; friend=Joseph
$  [[ $name == [Tt]om ]]
$  echo $?
0
$  [[ $name == [Tt]om && $friend == “Jose” ]]
$  echo $?
1
```

- If

Same structure as Korn shell using [...], [[…]], or ((…))

Syntax:

```
if [ expr ]
then
        Command
elif [ expr ]
then
        command
else
        command
fi
```

Examples:

```
if [ $age –ge 0 –a $age –le 12 ]
then
        echo “a child for sure”
fi

if ((age >= 0 && age <= 12))
then
        echo “a child for sure”
fi
```

- Case

  Syntax:

```
case var in
        value1)
                cmds;;
        value2)
                cmds;;
        *)
                cmds;;
esac
```

  Example:

```
case “$color” in
        [Bb]l??)
                xterm –fg blue –fn terminal &
                ;;
        [Gg]ree*)
                xterm –fg darkgreen –fn terminal &
                ;;
        red | orange)
                xterm –ff “$color” –fn terminal &
                ;;
        *)
                xterm –fn terminal
                ;;
esac
```

- For

  Syntax:

```
for var in word list
do
        cmds
done
```

  Example:

```
for pal in Tom Dick Harry Joe
do
        echo “hi $pal”
done
```

```
for person in $(cat mylist)
do
        mail $person < letter
done
```

- While

  Syntax:

```
while command
do
        cmds
done
```

  Example:

```
num=0

while (( $num < 10 ))
do
        let num+=1
done
```

- Select

  Syntax:

```
select var in word list
do
        cmds
done
```

  Example:

```
PS3="Select a program to execute:"

select program in 'ls –F' pwd date
do
        $program
done
```