Bash Shell Quick Reference

Remember a process in Unix has standard input (stdin), standard output (stdout), and standard error (stderr) streams for reading and writing.

Special Variables:

`CDPATH`	directories searched by cd command
`ENV`	environment file (set this to ~/.kshrc)
`PATH`	command search path
`PS1`	command prompt
`TERM`	terminal definition (e.g. vt100, xterm)

`*, ?, [a-z]`	filename wildcards (anything, 1 char, one of a-z)
`~, ~user`	your home directory, user's home directory
`cmd &`	execute cmd in background
`cmd1 ; cmd2`	execute cmd1, then cmd2
`cmd1 | cmd2`	pipe the output of cmd1 as input into cmd2
`VAR=value`	set the value of VAR as indicated
`export VAR=value`	export VAR as an environment variable
`VAR=`cmd` or VAR=$(cmd)`	set the value of VAR to the output of cmd
`cmd > file`	redirect output of cmd into file instead of screen
`cmd >> file`	append output of cmd onto file
`cmd < file`	redirect input to cmd from file instead of keyboard
`cmd 2> file`	redirect stderr to file
`cmd > file 2>&1`	redirect stderr to the same file as stdout
`$VAR, ${VAR}`	use the value of VAR
`$(( expr ))`	use the value of the arithmetic expression expr
`(( VAR=expr ))`	assign value of arithmetic expression to VAR
`#text`	comment symbol – ignore all text after #
`. file`	execute contents of file in the current shell
`alias name='cmd'`	let name be an alias for cmd
`cd, cd dir`	change to home directory, change to directory dir
Ctrl-C	kill current job
Ctrl-Z	suspend current job
`jobs`	list current jobs
`bg`	put suspended job into background
`fg`	put suspended job into foreground
`fg %1`	put job 1 from current job list into the foreground
`kill pid`	kill process whose ID is pid
`kill -9 pid`	send signal 9, the Die Now (You Scum), message
`exit`	terminate the current shell
`echo [-n] string`	print string [-n means without newline]
`read VAR`	read input from user into VAR
`set -x`	turn on trace/debug mode (display commands)

Operators:

`!`	Boolean not
`*, /, %`	multiply, divide, modulo (remainder)
`+, -`	add, subtract
`<=, >=`	less than or equal, greater than or equal
`<, >`	less than, greater than
`==, !=`	equal, not equal
`&&`	Boolean and
`||`	Boolean or

User-defined function:

```
function name {      or     name() {
   cmd1                        cmd1
   cmd2                        cmd2
}                           }
```

If function is to receive arguments, they are found in variables named $1, $2, $3, …

Boolean conditions come in multiple forms:

`[[ expression ]]`	evaluate Boolean expression
`test` *condition*	see if condition holds

File conditions:

-a file	file exists
-d file	file is a directory
-f file	file is a regular file
-r file	file is readable
-s file	file has size > 0
-w file	file exists and is writable
-x file	file exists and is executable
f1 –nt f2	file f1 is newer than file f2
f1 –ot f2	file f1 is older than file f2

String conditions:

-n s1	string s1 has length > 0
-z s1	string s1 has length 0

Integer conditions:

n1 –eq n2	n1 = n2
n1 –ge n2	>=
n1 –gt n2	>
n1 –le n2	<=
n1 –lt n2	<
n1 –ne n2	!=

If you don't use **`test`** to evaluate integer expressions, the Bash shell will treat the variables as strings.

`#!/bin/bash` Start shell scripts with this as the first line in the file.

If statement:

```
if condition1
then
   commands1
elif condition2          (Else-if clause can be repeated 0 or more times)
then
   commands2
...
else                     (Else clause is optional)
   commands3
fi
```

Case statement:

```
case value in
   pattern1) cmds1;;
   pattern2) cmds2;;
...
esac
```

Some useful patterns:

`*`	match anything (useful for the last case to catch anything)
`[aA]`	match either ‘a’ or ‘A’
`[a-zA-Z]`	match any letter
`[0-9]`	match any digit

For loop:

```
for VAR in list
do
     commands
done
```

While loop:

```
while condition
do
     commands
done
```

Note: using a condition of “:” (a colon by itself) means “execute forever.” You might do this if one of the commands inside the loop is responsible for exiting.

Sample shell script:

```
#!/bin/bash
function get_enter {
  echo ""
  echo -n "Press ENTER to continue. "
  read
}
function help {
  clear
  echo "This is a menu.  It doesn't do much."
  echo "Perhaps you should edit this file."
  get_enter
}

B=`tput bold`      # Start bold mode
R=`tput rev`       # Start reverse video mode
E=`tput sgr0`      # Reset to normal mode

while :
do
  clear
  echo ""
  echo "              $R$B WELCOME TO `hostname` $E"
  echo ""
  echo "                ${B}1.$E  Choice one."
  echo "                ${B}2.$E  Choice two."
  echo "                ${B}3.$E  Choice three."
  echo "                ${B}4.$E  Choice four."
  echo ""
  echo -n "               Enter a selection: "

  read CHOICE
  case $CHOICE in
    1) help ;;
    2) echo "Choice 2."
       get_enter ;;
    3) echo "Choice 3."
       get_enter ;;
    4) echo "You chose 4.  Good-bye."
       exit ;;
    *) echo ""
       echo "$CHOICE is not a valid choice."
       sleep 2
       ;;
  esac
done
```