UNIX Commands

Some common UNIX commands to research and become proficient with are:

Common Shell Features

Meaning or Action	Symbol or Command
Redirect output	>
Redirect input	<
Append to file	>>
"Here" document (redirect input)	<<
Pipe output	|
Run process in background	&
Separate commands on the same line	;
Home directory symbol	~
Match any character(s) in filename	*
Match single character in filename	?
Match any characters enclosed	[ ]
Execute in subshell	( )
Expand letters in a list[1]	{ }
Substitute output of enclosed command	` `
Partial quote (allows variable and command expansion)	" "
Full quote (no expansion)	' '
Quote following character	\
Use value for variable	$*var*
Process id (PID)	$$
Command name	$0
*n*th argument where *n* from 0 to 9 (0 < *n* < 9)	$n
All arguments as a simple word	$*
All arguments as a simple word	#
Background execution	bg
Break from loop statements	break

Meaning or Action	Symbol or Command
Change directories	`cd`
Resume a program loop	`continue`
Display output	`echo`
Evaluate arguments	`eval`
Execute a new shell	`exec`
Foreground execution	`fg`
List previous commands	`history`
Show active jobs	`jobs`
Terminate running jobs	`kill`
Shift positional parameters	`shift`
Suspend a foreground job	`suspend`
Time a command	`time`
Set or list file permissions	`umask`
Erase variable or function definitions	`unset`
Wait for a background job to finish	`wait`

1 Brace expansion is a compile-time feature in the Korn shell. Commercial versions do not typically have this feature, but if you compile form source code, it is included by default.

Differing Shell Features

Meaning or Action	bash	ksh	tcsh
Prompt	`$`	`$`	`%`
Force redirection	`>|`	`>|`	`>!`
Force append			`>>!`
Combine stdout and stderr	`>file 2>&1`	`>file 2>&1`	`>&file`
Combine stdout and stderr	`>&file`		`>&file`
Substitute output of enclosed command (preferred form)	`$( )`	`&( )`	
Home directory	`$HOME`	`$HOME`	`$home`
Variable assignment	*var=value*	*var=value*	`set` *var=value*
Set environment variable	`export` *var=val*	`export` *var=val*	`setenv` *var val*
More than nine (9) args can be referenced	`${`*nn*`}`	`${`*nn*`}`	
Number of arguments	`$#`	`$#`	`#$argv`
All args as separate words	`"$@"`	`"$@"`	
Exit status	`$?`	`$?`	`$status`
Last background PID	`$!`	`$!`	
Current options	`$-`	`$-`	
Read commands in *file*	`.` *file*	`.` *file*	`source` *file*
Name *x* stands for *y*	`alias` *x=y*	`alias` *x=y*	`alias` *x y*
Choose alternatives	`case`	`case`	`switch/case`
Switch directories	`cd ~-`	`cd ~-`	`popd/pushd`
Switch directories	`popd/pushd`		`popd/pushd`
End a loop statement	`done`	`done`	`end`
End case or switch	`esac`	`esac`	`endsw`
Exit with a status	`exit [`*n*`]`	`exit [`*n*`]`	`exit [(`*expr*`)]`

Meaning or Action	bash	ksh	tcsh
Loop through values	`for/do`	`for/do`	`foreach`
Ignore echo escapes	`echo -E`	`print -r`	`glob`
Display hashed commands (tracked aliases)	`hash`	`alias -t`	`hashstat`
Remember command locations	`hash` *cmds*	`alias -t` *cmds*	`rehash`
Forget command locations	`hash -r`	`PATH=$PATH`	`unhash`
Redo previous command	`fc -s`	`r`	`!!`
Redo command that starts with *str*	`fc -s` *str*	`r` *str*	`!`*str*
Edit command, then execute	`fc -sx=y[`*cmd*`]`	`rx=y[`*cmd*`]`	`!`*cmd*`:s/`*x*`/`*y*
Sample if statement	`if ((i==5))`	`if ((i==5))`	`if ($i==5)`
Set resource limits	`ulimit`	`ulimit`	`limit`
Print working directory	`pwd`	`pwd`	`dirs`
Read from standard input	`read`	`read`	`$<`
Ignore interrupts	`trap INTR`	`trap INTR`	`onintr`
Begin until loop	`until/do`	`until/do`	
Begin while loop	`while/do`	`while/do`	`while`

Variable Expansion Syntax for Bourne & BASH Shells

Function	Example
Simple usage	`$PARAMETER` `${PARAMETER}`
Indirection	`${!PARAMETER}`

Case modification	`${PARAMETER^}`
	`${PARAMETER^^}`
	`${PARAMETER,}`
	`${PARAMETER,,}`
	`${PARAMETER~}`
	`${PARAMETER~~}`

Variable name expansion	`${!PREFIX*}` `${!PREFIX@}`

Substring removal (also used for filename manipulation)	`${PARAMETER#PATTERN}`
	`${PARAMETER##PATTERN}`
	`${PARAMETER%PATTERN}`
	`${PARAMETER%%PATTERN}`

Search and replace	`${PARAMETER/PATTERN/STRING}`
	`${PARAMETER//PATTERN/STRING}`
	`${PARAMETER/PATTERN}`
	`${PARAMETER//PATTERN}`

String length	`${#PARAMETER}`
Substring expansion	`${PARAMETER:OFFSET}` `${PARAMETER:OFFSET:LENGTH}`
Use a default value	`${PARAMETER:-WORD}` `${PARAMETER-WORD}`
Use an alternate value	`${PARAMETER:+WORD}` `${PARAMETER+WORD}`
Display error if null or unset	`${PARAMETER:?WORD}` `${PARAMETER?WORD}`

BASH Conditional Expressions

BASH Conditional Expression Alternative Syntax is `[[ expression ]]`

Return a status of 0 or 1 depending on the evaluation of the conditional expression. The command is a variant of the test command where words splitting and pathname expansion don't occur. However, tilde expansion, parameter and variable expansion, arithmetic expansion, command substitution, process substitution, and quote removal are still performed by the shell.

The testing features basically are the same with some additions and extensions.

Operation	Description
`( expression )`	Used to group expressions, to influence precedence of operators
`expression1 && expression2`	TRUE if expression1 and expression2 are true (do not use -a)
`expression1 || expression 2`	TRUE if expression1 or expression2 is true (do not use -o)
`STRING == PATTERN`	STRING is checked against the pattern PATTERN (expanded pattern as if extglob was enabled)- true on a match
`STRING = PATTERN`	equivalent to the == operator
`STRING != PATTERN`	STRING is checked against the pattern PATTERN (expanded pattern as if extglob was enabled) - true on no match
`STRING =~ ERE`	STRING is checked against the extended regular expression ERE - true on a match

BASH Loop and the Shift Command

shift[n] The positional parameters from n+1 and up are renamed to $1, $2. Parameters represented by $# down to $#-n+1 are unset. n must be a non-negative number less than or equal to $#. If n is 0 or greater than $#, no parameters are changed. If n is not given, it is assumed to be 1. The return status is zero if and only if n is between 0 and $#.

Example

Consider the following script that adds two number and print the result on the screen:

```
#! /bin/bash

echo `expr $1 + $2`
```

What if the script was to be changed to add an arbitrary number of integers? Using the positional arguments $1, $2, …, will no longer be an option because you don't know how many numbers would be passed to the script ahead of time. A more elegant approach consists in using the shift command, the $# variable, and the first positional argument only. Take a moment to figure out the solution on your own before reading the solution provided:

```
#! /bin/bash

sum=0

while [ $# -gt 0 ]

do

      sum=$(expr $sum + $1)

      shift

done

echo $sum
```

I/O Redirection and File Descriptors

Each open file gets assigned a file descriptor by the kernel. The file descriptor for stdin, stdout, and stderr are respectively 0, 1 and 2. You can assign file descriptors to additional files using numbers from 3 and above. The table below shows usages of file descriptors in I/O redirection. Those I/O redirections directives can be used in two different ways:

- When placed after any command except the "exec" command, redirection is just for the current command-line. All opened files are closed after the command.
- When placed after an "exec" command, redirection is forced to continue for the all the following command lines until the redirection is cancelled or undone with another "exec" command.

Operation	Description
`n> filename`	Opens filename for writing and assigns it the file descriptor n, which is created if it doesn't exist. When n is 1 or omitted, stdout is used for redirection.
`n< filename`	Opens filename for reading and assigns it the file descriptor n. When n is 0 or omitted, stdin is used for the redirection.
`n<> filename`	Opens filename for reading and writing and assigns it the file descriptor n. The file descriptor n and filename are created if they don't exist.
`1>filename`	Redirects stdout to filename.
`1>> filename`	Redirects and appends stdout to filename.
`2>filename`	Redirects stderr to filename.
`2>>filename`	Redirects and appends stderr to filename.
`&>filename`	Redirects both stdout and stderr to filename.
`i>&j`	Redirects file descriptor i to j.
`1>&j (>&j)`	Redirects stdout to file descriptor j.
`<&j`	Redirects stdin to fd j.
`n<&-`	Closes input file descriptor n, which was open for reading.
`0<&- (<&-)`	Closes file descriptor stdin (fd 0).
`n>&-`	Closes output file descriptor n, which was open for writing.
`1>&- (>&-)`	Closes stdout (fd 1)

Example 1:

For this example, the file echo_output.txt is opened for reading and writing, and then used to redirect the standard output. The string "Helloworld" is written directly inside echo_output.txt and nothing appears on the screen.

```
$ echo "Helloworld" 3<> echo_output.txt 1>&3
```

Example 2

(reference: http://www.faqs.org/docs/abs/HTML/io-redirection.html):

$echo 1234567890 > output.txt	**# write the string "1234567890" to file output.txt**
$ exec 3<> output.txt	**# open "output.txt" and assign fd 3 to it**
$ read –n 4 <&3	**# read 4 characters from open file with file descriptor (fd) 3. File cursor/pointer moves 4 characters forward.**
$ echo –n . >&3	**# write a decimal point at the file pointer position**
$ exec 3>&-	**# close fd 3**
$ cat output.txt	**# output on the screen will be: 1234.67890**

Regular Expressions and GREP

Code	Description
.	Matches any single character.
(pattern-list)?	The preceding item is optional and will be matched, at most, once.
(pattern-list)*	The preceding item will be matched zero or more times.
(pattern-list)+	The preceding item will be matched one or more times.
(pattern-list){N}	The preceding item is matched exactly N times.
(pattern-list){N,}	The preceding item is matched N or more times.
(pattern-list){N,M}	The preceding item is matched at least N times, but not more than M times.

Recommended Readings

- **UNIX and Linux System Administration Handbook, 4th Edition**
  - Section One: Basic Administration > 2. Scripting and the Shell > *Read Shell Basics, bash scripting, and Regular Expressions*
  - Section Three: Bunch O' Stuff > 31. Serial Devices and Terminals > Configuration of terminals > *Read "Special Characters and the Terminal Driver" (31.9: Special Characters and the Terminal Driver), "stty: set terminal options" (31.10: Stty: Set Terminal Options), and "Terminal unwedging" (31.12: Terminal Unwedging)*
- **UNIX in a Nutshell, 4th Edition**
  - Part I: Commands and Shells > *Read Chapter 3.The Unix Shell: An Overview, Chapter 4. The Bash and Korn Shells, andChapter 5. tcsh: An Extended C Shell*
  - Part II Text Editing and Processing read Chapter 7: Pattern Matching, *Read Chapter 9. The vi, ex, and vim Editors*

Recommended Internet Sites

- Linux Documentation Project: https://web.archive.org/web/20160726214403/http://www.tldp.org/
- Advanced Bash Scripting Guide: https://web.archive.org/web/20160726214449/http://www.tldp.org/LDP/abs/html/
- Tar Command Tutorial: https://web.archive.org/web/20160809111628/http://www.thegeekstuff.com/2010/04/unix-tar-command-examples

Please contact the Course Coordinators if you are unable to access any of the Recommended Internet Sites.