[CEG 233: Linux and Windows]

BASH and PowerShell Quick Reference

bash	**PowerShell**	**Description**
	Scripting Basics	
Put a "shebang" at the beginning of the file: `#!/bin/bash` Change permissions on script file to allow execution.	Give the file a ps1 extension. For downloaded scripts, unblock the file under file properties in Windows Explorer.	Steps for making scripting files run. In PowerShell, the first time you do scripting, you will need to set the appropriate security settings: run PowerShell as administrator and type `set-executionpolicy remotesigned`.
`source (or) .`	`.`	shell built-in: execute the commands in a file
`echo` *`String`*	`echo` *`String`* (or) `Write-Host` *`String`*	Prints *String* to the screen. In PowerShell, `Write-Host` forces the output to the screen instead of being a return value.
`var``=0` (No spaces around =)	`$`*`var`* `= 0`	Creates a variable *$var*. In BASH, do not put whitespace around the equals sign, and do not use a $ in the variable assignment.
`let` *`var`*`=$`*`var`*`+5` (or) *`var`*`=$(( $`*`var`* `+ 5 ))`	`$`*`var`* `+= 5`	Add 5 to $*var*
`# comment`	*`# comment`*	A comment
	Strings	
`= !=`	`-eq -ne -ceq -cne`	String comparisons. In BASH, be sure the strings litereals are in quotes.

`"" | gm`		Get a list of non-static string members
`[string] | gm -static`		Get a list of static string members
`${`*`string#text_to_remove`*`}`	*`string`*`.TrimStart("`*`characters`*`")`	Removes the specified characters/text from the beginning of the string.
`${`*`string%text_to_remove`*`}`	*`string`*`.TrimEnd("`*`characters`*`")`	Removes the specified characters/text from the end of the string. Suppose $fnm == helloThere.txt; then ${fnm%.???} is helloThere
Pattern Matching		
`grep`	`select-string`	print lines matching a pattern
`sed`	`-replace`	performs string transformations
Booleans and Conditions		
`true  false`	`$true  $false`	Boolean literals
`-lt -gt -le -ge -eq -ne`	`-lt -gt -le -ge -eq -ne`	Arithmetic relational operators
	`-like`	True if a string matches a wildcard pattern
	`-match`	True if a string matches a regular expressions
	`Where-Object { `*`condition`*` }`	Used to filter input by a condition. Remember that $_ refers to the current object being tested.
`-z $`*`var`*	`$`*`var`*` -eq $null`	True if $*var* is null
`-n $`*`var`*	`$`*`var`*` -ne $null`	True if $*var* is not null (contains one or more characters)
`-o -a`	`-or -and`	Logical *or* and *and*

`-e` *file*	`Test-Path` *file*	True if *file* exists.
`! -e` *file*	`! (Test-Path` *file*`)`	True if *file* does not exist.
`-d` *file*	*file*`.PSISContainer`	True if *file* is a directory. In PowerShell, if *file* is not a file variable, be sure to get the file object first with `gi`.
`-s` *file*		True if *file* exists and has a size greater than zero.
file1 `-nt` *file2*		True if *file1* is newer (according to modification date) than *file2*
file1 `-ot` *file2*		True if *file1* is older (according to modification date) than *file2*
Control Structures		

`if [` *condition* `]`
`then`
`  ` *codeblock*
`fi`
	`if (`*condition*`) {`
	`  ` *codeblock*
	`}`
		If statement. In BASH, be sure to leave a space between the condition and the bracket.

`if [` *condition* `]`
`then`
`  ` *codeblock*
`elif [` *condition* `]`
`then`
`  ` *codeblock*
`else`
`  ` *codeblock*
`fi`
	`if (`*condition*`) {`
	`  ` *codeblock*
	`}`
	`elseif (`*condition*`) {`
	`  ` *codeblock*
	`}`
	`else {`
	`  ` *codeblock*
	`}`
		If - else if - else statement

`var=0`
`while [ $var -lt 10 ]`
`do`
`  echo $var`
`  var=$(( $var + 1 ))`
`done`
	`$var = 0`
	`while ($var -lt 10) {`
	`  echo $var`
	`  $var++`
	`}`
		Prints numbers 0 through 9.

`for ((i=0; i < 10; i++))`
`do`
`  echo $i`
`done`
	`for ($i=0;$i -lt 10; $i++)`
	`{`
	`  echo $i`
	`}`
		Prints numbers 0 through 9.

`for` *var* `in $`*array*
`do`
`  ` *codeblock*
	`foreach ($`*var* `in $`*array*`)`
	`{`
	`  ` *codeblock*
		For each loop

`done`	`}`	
`continue  break`	`continue  break`	Loop controls: continue stops current loop iteration and begins the next; break exits the loop currently being executed.
`basename` *file*	*file*`.name`	The name of *file* without the path. In PowerShell, remember to first get the file object.
`dirname` *file*	*file*`.directoryname`	The name directory *file* is in. In PowerShell, remember to first get the file object.
`stat -c%s $`*file* (or) `$(stat -c%s $`*file*`)`	*file*`.length`	The number of bytes in *file*. In PowerShell, remember to first get the file object.
	file`.LastWriteTime`	The last modified time for *file*. Remember to first get the file object.
files`=\`ls\`` (or) *files*`=$(ls)` (or) *files*`=*`	`$`*files* `= Get-Item *`	Store a list of the files in the current working directory in *$files*. In PowerShell, check out the -exclude flag as well as the Get-ChildItem cmdlet.
`|  >  >>  2>  2>>`	`|  >  >>  2>  2>>`	Piping, output and error redirection. In BASH, output redirected to `/dev/null` is gone. In PowerShell, output redirected to `$null` is gone.

printArg`()`
`{`
`  `*echo* `$1`
`}`
	`function` *printArg*
	`{`
	`  param (`$*p1*`)`
	`  `*echo $p1*
		function to print the first argument to the screen.

	} 	

return_five()
{
  return *5*
}
 	
return_five
echo $?
	function *return_five*
	{
	  echo *5* (or) return *5*
	}
	
	$*value* = *return_five*
	echo $*value*
		Function returns 5, which is printed after the function call. In PowerShell, any output in a function that is not caught is returned. The return statement merely ends the function. The return value of a BASH function is stored in the variable $?.

File Information/Operations		
ls		Listing of files. For bash, learn the options of `-lisa`, `-r`, `-R`.
	ls	Listing of files. For PowerShell, learn `-f`, `-r`, `-filter`, and `-exclude`
tree	tree	Graphically displays the directory structure of a drive or path.
cat	cat	List the contents of a file on the `stdout`
more	more	List the contents of a file on the `stdout`, pausing after each page
mkdir	mkdir	Creates a directory.
rmdir	rmdir	Deletes a folder if it is empty
pwd	pwd	print working directory
cd	cd	Change the current directory to the one given as argument.
pushd	pushd	Saves the current directory name on the

		stack, and then cd's the one given as argument.
`popd`	`popd`	Pop off the top-most name on the stack, and then cd to it
`mv`	`mv`	Moves or renames files. In PowerShell, check out the -force and -WhatIf flags. In BASH, check out the -f flag.
`cp -r`	`cp -r`	Copies files and directory trees recursively.
`cp`	`cp`	Copies files. In PowerShell, check out the -force and -WhatIf flags. In BASH, check out the -f flag.
`rm`	`rm`	Deletes a file. Check out the -r flag. In PowerShell, check out the -force and -WhatIf flags. In BASH, check out the -f flag.
`cat`	`cat`	show the contents of each file in sequence
`more`	`more`	pagination
`rm`	`rm`	Remove files
`ln`		Link (hard or soft) to an existing file.
	`mklink`	Link (hard or soft) to an existing file. Type `cmd /c mklink` to use it in PowerShell
`chmod`	`attrib`	Change file permissions/attributes
	`icacls`	Displays or modifies access control lists (ACLs) of files

`chown`	`icacls`	Change ownership of a file. In PowerShell, multiple steps are necessary
`umask`		get/set the file mode creation mask; packed vector of bits controlling the initial permissions on a newly created file
`du`	`measure`	Disk space Used. In PowerShell, try `gci . -r | measure -property length -sum`
`wc`	`Measure-Object`	word count, etc.
`od`		Octal dump of file content. Almost always used with -x for hexadecimal dump
`tr`		Translate/substitute characters; useful in improving interoperability
	`assoc`	List associations of commands with extensions. Type `cmd /c assoc` to use it in PowerShell
`file`		Heuristically determine the type of file content
`grep`	`select-string`	Search for a string in a file's content. For now, learn it without regexp.
`find`	`gci`	Locate a file. By name, etc. For now, learn it without regexp.
`which`		Gives the full path name of a command
	`where`	Gives the full path name of a command. Type `cmd /c where` to

		use it in PowerShell
diff	diff	List the differences between two text files
cmp, diff	compare, diff	show the differences between two files
	gci . -r | sort length -descending | select -first 10	get a list of the 10 largest files in the current directory (recursive)
vi	vim	A powerful text editor. For now, learn to edit simple text files with it.
kate, leafpad	notepad	Simple text editors.
emacs	emacs	A very powerful multi-purpose text editor. For now, learn to edit simple text files with it.
Processes		
time	Measure-Command	times commands, etc.
ps	ps	shows current processes
	gps | sort ws | select -last 5	Get a list of the 5 processes using the most memory
	gsv | where {$_.Status -eq "Stopped"}	Get a list of stopped services
top		like ps, but with continuous updates
bg		place a STOPped process in the background
fg		bring a backgrounded process to foreground
kill	kill	kills a running program
ltrace		show lib calls made
strace		show sys calls made

System		
man	man	show reference pages
set	set	set the values of shell variables
set	gv	get and show the values of shell variables
env	ls env:\	lists the current environment variables
$PATH	$env:path	the search path
links		WWW/News/Mail browser
sftp, filezilla	filezilla	transfer files securely to/from a remote machine
ssh, putty	sshclient, putty	remote login securely
w		who is on the system, and what they are doing
df	gdr	show mounted volumes, etc.