Shortcut key	Arguments / Options	Purpose
ls	-l, -i, -d, -r, -R , -t, -a, -A, -p, -F, dir-pathname, -o, -g, -G	• List files of directory • ls -l : displays a detailed listing of filenames in the current directory , -l can be used to help determine file type. (d: directory file, -: regular file, b or c: device file) • -i, --inode : display the i-node number for each file • ls /bin : displays a listing of filenames in the /bin directory (as opposed to your current directory) • ls -d : lists the directory itself (not contents) • -r, --reverse : reverse order while sorting • ls -R : displays directories and subdirectory contents. (similar to tree) • ls -t : sort by modification time, newest first • ls -a : show all files including hidden and nonhidden. Current and Parent directories (. and ..) are displayed. • ls -A : show all files including hidden and non-hidden. Current and Parent directories (. and ..) are NOT displayed. • ls -p / ls -F : adds a / at the end of directories, it helps you easily detect which one of the outputs is a directory and which one is a file. (/ --> directories, @ --> symbolic links, | --> fifo files) • ls -o : like -l, but do not list group information • ls -g : like -l, but do not list owner • ls -G, --no-group : in a long listing, don't print group names
tree	-a	• tree -a : All files are printed. By default tree does not print hidden files (those beginning with a dot `.'). In no event does tree print the file system constructs `.' (current directory) and `..' (previous directory).
	Filename expansion ls *.txt	• Use special characters to allow the shell to match files that share the same characteristics to help the user save time (Only shell knows it is file expansion, the command itself don't) • * : represent 0 or more characters • ? : represent exactly one character (any character) • [ ] : represent and match for the character enclosed within the square brackets. It represents ONLY ONE character: [0-9][a-z][A-Z] • [!] : represent and match and OPPOSITE character for the character enclosed within the square bracke • commands that support filename expansion: ls, cp, mv, rm, find, grep

	regular expression	• Regular expressions are used to search, edit and manipulate text. This can represent text contained in a file or within a pipeline command command. • The ^ symbol anchors the pattern at the beginning of the string. • The $ symbol anchors the pattern at the end of the string. • The period symbol "." represents a single character which could represent any character. • [ ] single character class • [^] means opposite of the contents within the character class • * symbol means zero or more occurrences of the previous character • zero or any character: .* • commands that support regular expression: grep, sed, awk, find, egrep, grep -E, perl, python • use a forward slash / to specify a regular expression with man , more , less, vi
grep	-i, -v, -n, -w, -x, -c "keyword" dir-pathname	Display and filter lines has that keyword in the file • grep -i, --ignore-case : Perform case insensitive matching. By default, grep is case sensitive. • grep -v, --invert-match : Selected lines are those not matching any of the specified patterns. (exclude) • grep -n, --line-number : Each output line is preceded by its relative line number in the file, starting at line 1 • grep -w, --word-regexp : The expression is searched for as a word • grep -x, --line-regexp: Select only those matches that exactly match the whole line. • grep -c, Count number of lines that match the pattern • grep "Linux i*" data.txt
egrep / grep -E	Extended regular expression {min, max} {1,} : + {0,1} : ? group:(space important) | : alternative in group "keyword" dir-pathname	• grep will not work • {min,max} minimum and maximum number of occurrences: • a{2,5} 2 to 5 occurrences of the character a • [0-9]{1, } 1 or more occurrences of a number = [0-9]+ • [a-z]{0,1} zero or 1 occurrence of a lowercase letter = [a-z]? • ( ){ } repetition of a group of characters (space important): • egrep "(the)){2, }" data.txt -> Time to go to the the store • (|) "or" symbol to provide alternative within a group • use %s/uli101/ULI101/g to search and replace text globally (all lines) with vi
sed	-n 'address instruction' filename can also be double quotes	• -n to suppress the default print action (silence) • range of line numbers : sed -n '3, 6 p' readme -> only display lines 3 through 6 • without -n : print all lines with line 3-6 two times • Instructions: • p : print lines that match

	address: /regex/ instructions: p, d, q, s, a, i, c	• q : quit after first line that matches • d :delete lines that match • s : substitute • a : append (3a: 第四行插字) • i : insert (3i: 第三行插字) • c : change (3c: 第三行換字) • -> g flag for global change • -> you can also specify the occurrence • -> number : specify 第幾個 match in each line • -> i : ignore case • If the line matches the address , then it will perform the instruction // If NO address is present, the instruction will apply to ALL lines • sed '/line/ p' readme • sed 's/^./\t&/' readme : \t (replacement string - tab) • sed 's/^./1&/' test : 1 (replacement string) • & takes on the value of what the regular expression matched. • \s : whitespace • Using sed utility, display only the cars that are less than $70000 from cars.txt ○ sed -n '/[0-6][0-9][0-9][0-9][0-9]$/ p' cars.txt
awk	-F 'selection criteria {action}' filename $1 $2 ... ${10}...${n} ~ !~ ; 好重要 >, >=, <, <=, ==, != || && selection: /regex/ action: {print} {print $1,$2}	• -F";": specify delimiter ; (default is whitespace, spaces or tabs) • "\n" : newline • NR -> Number of record, line number • NF -> Number of field • $NF -> last field (column) ***If not all lines has the same number of field, but you still want to access the last field of each line. • $0 -> all fields, entire input record (awk '{print NR,$0}' customer.dat) 1 A100 Acme Acme-Inc. 5400 2 R100 Rain Rain-Ltd. 11224 3 T100 Toy Toy-Inc. 3413 • $1 -> first field • ${10},${11}....${n} • $USER is a shell variable that stores the username of the current user who is logged in • ~ test whether a specific field matches the regex • !~ test no match (awk '$1 ~ /^[F-Z]/ {print}' data.txt) • numeric/string comparisons: >, >=, <, <=, ==, != • || (OR) && (AND) operators (awk '$3 >= 5000 && $3 <= 10000 {print}' customer.dat) • BEGIN, END • \<: Matches the beginning of a word. • \>: Matches the end of a word. • '{n+=5} END {print n}' cars (only print the total)

		• ‘{if (m < $5) {m=$5; line=$0}} END {print line}’ cars (print the line that has the largest value in the fifth column) • If no pattern is specified, awk selects all lines in the input • If no action is specified, awk copies the selected lines to standard output • FS -> Input field separator (default: SPACE or TAB) • OFS -> Output field separator (default: SPACE) • ORS -> Output record separator (default: NEWLINE) • RS -> Input record separator (default: NEWLINE)
	\ , ‘ ’ , “ ”	• ignore the special character meaning and act as regular text • \ and ‘ ’ works for all special characters • “ ” works for most special characters (not work for $HOME, $PATH)
man	“command”, -k	• Provide information on how to use a command • man -k “text pattern” (used with -k option to match a text pattern for you don’t know the name of command)
--help	command -- help	
pwd		Display current working directory
cd	dir-pathname	Change directory
cal	Month, year	Display calendar
date		Display date and time
who		List users logged into server
whoami		Display username of user logged in
clear		Clear Screen
passwd	username	Change user’s password
mkdir	dir-pathname , -p	Creates a directory mkdir -p : if the specified directory doesn’t exist, create for me
rmdir	-p	Only Remove empty directories • when your current directory contains the directory, rmdir directoryName = rmdir ./directoryName • No rmdir -r • rmdir -p, --parents : remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is similar to 'rmdir a/b/c a/b a'
rm	-r, -R, -i, -I, -d, -v, -f	• rm : remove files only (No directory) • By default, rm does not remove directories. • rm -r / -R : remove non-empty or empty directories / files and their contents • rm -i : prompt user to confirm deletion of directory contents • rm -I : prompt once before removing more than three files, or when removing recursively; less intrusive than -i, while still giving protection against most mistakes • rm -d, --dir : remove empty directories (same as rmdir) • rm -v, --verbose : explain what is being done • rm -f, --force: ignore nonexistent files and arguments, never prompt • Remove hidden files:

		rm -i /path/to/.fileName rm -i /path/to/.dirName rm -rf /path/to/dir/.* 1. touch ./-f (create a file called -f) 2. rm ./-f (Use rm -f to remove - Doesn't work)
cp	-r, -R	• cp -r/ -R : Copy directory and its contents (recursive) to a different directory • Cp movies/harryPotter[57] books/novels
mv	dir-pathname	• Moves directory and its contents to a different directory • If directory doesn't exist, rename the directory • mv ../../movies/harryPotter[124] .
touch		• Create empty file(s) • Update existing file's date / time stamp • Touch movies/harryPotter{1..7}
cat	dir-pathname	Display text file's contents without editing (small files)
more, less	dir-pathname	Display/navigate within large text files without editing
head, tail	dir-pathname	View lines at top/bottom of file: head -3: first three lines tail -3: last three lines no parameter: display the last 10 lines of the file by default. tail -n 5 myfile.txt will display the last 5 lines of the file, while tail -n 20 myfile.txt will display the last 20 lines of the file.
cut	-d":" -f1 -s dir-pathname	• -d, --delimiter=DELIM : use DELIM instead of TAB for field delimiter • -f, --fields=LIST : select only these fields; also print any line that contains no delimiter character, unless the -s option is specified (-f1,2 ,-f1-9) • -s, --only-delimited : do not print lines not containing delimiters • -c, --characters=LIST : select only these characters (cut -c1-8 : specifies 1-8 character positions)
tr		translate or delete characters: tr "[a-z]" "[A-Z]" < filename tr 'a-z' 'A-Z' < filename tr [A-Z] [a-z] < filename Doesn't support filename expansion, regular expressions
sort		Display contents of file in sorted order • -r, --reverse : reverse the result of comparisons • -f, --ignore-case : fold lower case to upper case characters • sort unsorted.txt | uniq : repeated lines display once • sort unsorted.txt | uniq -u : repeated lines will not display at all • -n, --numeric-sort :compare according to string numerical value
wc		• Find out number of lines, word count, byte and characters count in the files specified in the file arguments. • By default it displays four-columnar output.

		• First column shows number of lines present in a file specified • Second column shows number of words present in the file • Third column shows number of characters present in file • Fourth column itself is the file name which are given as argument. • -w, --words print the word counts • -m, --chars print the character counts • -c, --bytes print the byte counts (for international language) • -l, --lines print the newline counts/ line counts
Standard input (stdin)	< 左耳入右耳出	• describes from where a command receives input. • provide input to a command from a file instead of the console • < only apply to Unix/Linux commands that can accept stdin like cat, more, less, sort, grep, uniq, head, tail, tr, cut, and wc.
Standard output (stdout)	> , >>	• describes where a command sends its output. • save the output of a command to a file • allow you to manage input and output from a single source or destination • > Either creating a new file if it doesn't exist or overwriting the content of an existing file. • >> Either creating a new file if it doesn't exist or adding stdout to the bottom to the existing file's contents.
Standard Error (stderr)	2>, 2>>	• describes where a command sends its error messages. • allow you to manage input and output from a single source or destination • 2> Either creating a new file if it doesn't exist or overwriting the content of an existing file. • 2>> Either creating a new file if it doesn't exist or adding stdout to the bottom to the existing file's contents. • /dev/null (bit bucket or black hole)
Here document	command <<delimiter text delimiter	• allows stdin to be redirected into a command within the command-line. • Here Documents are useful for passing a block of text to a command as input, like passing configuration data and templates • allow you to manage input and output from a single source or destination • delimiter can be any symbol (END, +) • You want to create a file named "example.txt" containing the following three lines of text: cat << END > example.txt This is a mult-line here document.

		END
Pipeline	|	• Pipelines allow you to connect multiple commands together, so that the output of one command is used as the input of the next without having to use temporary files • allow you to manage input and output between multiple sources and destinations.
Command substition	command1 $(command2) or command1 `command2`	Command substitution is a facility that allows a command to be run and its output to be pasted back on the command line as arguments to another command. file $(ls) echo "The current directory is $(pwd)"
tee	dir-pathname	tee - read from standard input and write to standard output and files tee -a : can be used to add content to the bottom of an existing file as opposed to overwriting the file's previous contents.
bc	obase : output base ibase : input base	• Binary to Octal: bc, obase = 8, ibase = 2 • Octal to Binary: bc, obase = 2, ibase = 8 • Binary to Hex: bc, obase = 16, ibase = 2 • Hex to Binary: • bc, obase = 2, ibase = 16
chmod (symbolic)	-R ugo +/-/= r/w/x dir-pathname	• chmod -R : set permissions for directory, subdirectory and directory contents recursively • chmod ugo+x dir-pathname • chmod go-w dir-pathname • chmod u = rwx, go = x ~ (home directory) • chmod can use symbols to add, remove, and set rwx permissions
chmod (octal)	-R octal number dir-pathname	• chmod -R : set permissions for directory, subdirectory and directory contents recursively • chmod 500 dir-pathname • chmod 711 ~ (Set "pass-thru" permissions of home directory, which is at least grant x permission) • chmod can use octal numbers to only set permissions. *** Octal to Binary, Hex to Binary add leading zero if needed. ***Dec to Hex: 100 /16 6(6.25) 4(0.25*16) 6/16 6 64
umask	three digits	• without arguments: display current umask value
;		Adding ; to make multiple commands can be run within a single command line. (command1;command2;command3)
()		Multiple commands can also be grouped by using parentheses to force commands to be run together (echo "Who is logged in:"; who) > whoson All command output is sent to a file
\	Multi-Line Commands	Adding a "\" at the end of a line to spread-out commands over multiple lines echo "This will be split over multiple \

		lines. Note that the shell will realize \ that a pipe requires another command, so \ it will automatically go to the next line" | tr '[a-z]' '[A-Z]'
history | more		Display all stored commands
history | tail	number of commands	• Display last 10 commands with date and time by default • history | tail -12 : Display last 12 commands with date and time by default
fc -l	number of starting point	• Display last 16 commands • fc -l 100: Display all commands starting from command number 100
!	command number #	re-executes command by command number (obtained from fc -l) !480 : re-executes command by command number 480
!	string	re-executes last command beginning with string !abc : re-executes last command beginning with string ”abc”
!!		re-execute the last command
diff		• Display the content differences between 2 files
uniq		• Display identical adjacent lines only once
file	dir-pathname	• Give info about the contents of the files with no extension, determine (directory, empty, ASCII text)
find	-name, -size, -mmin	• file . -name “file*” : lists pathname of any filenames beginning with "file", from the current directory and any subdirectories • find . -size +50k : lists pathname of any files larger than 50 kb, from the current directory and any subdirectorieslists • find . -mmin -5 : lists files modified less than 5 minutes ago
scp	copies files between hosts on a network.	Upload: 1. scp local.file username@host: scp other.txt yoursenecaid@matrix.senecacollege.ca: 2. scp local.file username@host:destination-pathname To the ~/remote directory in Matrix renaming it as different.txt: scp other.txt yoursenecaid@matrix.senecacollege.ca:remote/different.txt Download: 1. scp user@host:file-pathname local-pathname scp yoursenecaid@matrix.senecacollege.ca:remote/myfile.txt .
sftp	Secure File Transmission Control Protocol	sftp yoursenecaid@matrix.senecacollege.ca local vs remote: • lls vs ls • lpwd vs pwd • lmkdir vs mkdir • lcd vs cd upload: put file-pathname download: get file-pathname

ssh		To run a command on your remote matrix server, without having to establish a continuous connection to it: ssh yoursenecaid@matrix.senecacollege.ca ls -l other.txt
mail	-a: attach a file -s: specify a subject line EOT: End of transmission	1. mail yoursenecaid@myseneca.ca 2. enter the subject line 3. enter body message 4. ctrl-d to send your email message Attach myfile.txt to an email: mail -a ~/remote/myfile.txt yoursenecaid@myseneca.ca The email takes the content of myfile.txt as the body message with the subject "email with attachment": mail -s "email with attachment" yoursenecaid@myseneca.ca < ~/remote/myfile.txt
ln	links files -s	1. ln: hard links (backups, no original files, all sync together) If accidently remove one file, the rest files will be linked and synced together (take-up extra space, cannot hard link directories, only within one server) 2. ln -s: symbolic links (shortcuts, can be broken link) (Not good for backup purposes)
	environment variable They can be used to configure the behavior of the shell, set system-wide settings, and more	• USER is an environment variable that stores the name of the currently logged in user. • PATH is an environment variable that stores a list of directories where the system looks for executables when a command is entered in the terminal. • PWD is an environment variable that stores the current working directory of the shell.
ps	-l, -ef, aux, -U username	Without argument : Basic listing of processes in current user's terminal (PID, process names names) -l : Detailed listing Detailed listing in current user's terminal (PID, PPID, status) -ef : Detailed listing ALL processes running on entire system. aux : Detailed listing of processes for ALL users and background running services -U username : Basic listing of processes running for a particular user. *Each process has a unique ID (PID) and processes keep their PID for their entire life. The top command provides real-time status of all running processes (press ctrl-c to exit top command)
ctrl-c		Terminates a process running in the foreground
ctrl-z		Sends a process running in the foreground into the background. Process is stopped (suspended) in background and requires bg command to run in background.
bg		The bg utility resumes suspended jobs from the current environment. Without arguments will run the most recent process was placed into the background.
fg	contrast to ctrl-z	The fg command moves a background job into the foreground. Without arguments will place the most recent process in the background to the foreground.

jobs		The jobs utility displays the status of jobs that were started in the current shell environment • sign "+" indicates the most recent process placed into the background. • sign "-" indicates the second recent process placed into the background. • Can use ( ) to enclose multiple commands to make them run in a group as just one process • ampersand character "&" indicates that this process in the background is running in the background.
command &		You can place an ampersand "&" after Linux commands to run processes automatically in the background without having to issue ctrl-z and bg short-cut keys.
kill		kill %jobnumber kill PID kill -9 %jobnumber kill -9 PID
alias		Set a nickname to an existing command or group of commands. alias dir=ls alias lal='ls -al' 花名＝real command Without argument : wil display all the aliases currently set It will be lost when your current Linux session ends, unless the alias is set in a start-up file (e.g. ~/.bashrc.)
unalias		Remove existent aliases alias clearfile='cat /dev/null >' unalias clearfile
which		• To see if the filename is recognized as a Unix/Linux command • Display the absolute path of the a command
Extension	.bash .csh	Adding an extension to your shell script filename will help to identify the type of shell that the shell script was designed to run.
comment		# This is a comment
Shebang Line		#!/bin/bash (force the shell script to run in a specific shell) It must appear on the first line and at the beginning of the line, otherwise, it will be treated as a regular comment and ignored.
echo	" " (recommend " " instead of ' ') -n	display text echo "My username is: $USER" My username is: twwong9 –n: will display text without the newline character.
read	-p	-p: prompts the user for data without requiring echo command.
Need execute permission		chmod u+x myscript.bash You can run a shell script without execute permissions by issuing: bash myscript.bash
User-created Variables		• Must start with a letter (a-z or A-Z) or an underscore (_). • They cannot start with a number. • Can contain letters, numbers, and underscores. • Case-sensitive.

Environment Variables		PS1 : Primary shell prompt PWD : Absolute path of present working directory HOME: Absolute path to user's home PATH : List of directories where commands / programs are located HOST : Host name of the computer USER : Name of the user logged in SHELL : Name (type) of current shell used
$	Placing a dollar sign $ before a variable name	Cause the variable to expand to the value contained in the variable.
\	Ignore the special meaning	set 10 9 8 7 6 5 4 3 2 1 echo "\$0 is: $0" $0 is: ./positional.bash echo "\$10 is: $10" $10 is: 100
=	Assign value by using the equal sign	Without space: name=value With spaces or tabs: fullName="David G Ward"
Remove variable value		1. variableName= 2. unset variableName
readonly	variable name	Prevents the user from changing the value of the variable while the shell script is running or during the duration of your shell session.
positional parameters	set $1 $2	1. Can be inside or outside shell script set apples oranges echo $1 (apples) echo $2 (oranges) 2. Inside a shell script - myScript.bash: echo "First argument is $1" echo "Second argument is $2" ./myScript.bash apples oranges First argument is apples Second argument is oranges
shift	number	shift (no argument) 左移 1 個 position shift 2 左移 2 個 positions
		• $* : Display all positional parameters. (If the script is called with "script.sh arg1 arg2 arg3", then $* will be equal to "arg1 arg2 arg3".) • "$*" : Containing values of all arguments separated by a single space • "$@" : Multiple double-quoted strings, each containing the value of one argument • $# : Represents the number of parameters (not including the script name) • $? : Exit Status of previous command (0:True, Non-zero:F) echo hi there | grep hello | more echo $? 0 (grep hello : 1, more : 0) • $0 : the name of the currently running script or shell program.

test	$name =, != “string” $num1 -eq, -ne, -lt, -le, -gt, -ge $num2 -f, -d, -s, -w, -z file_pathname || &&	test $name = “Murray” (means == sign) -eq : Equal to (for comparing two integer values, not strings) -ne : Not equal to -lt : less than -gt : greater than -le : less than or equal to -ge : greater than or equal to X test $num1 > $num2 (唔 work) -f : Regular filename exists -d : Directory filename exists -s : Regular filename is non-empty -w : File exists / write permission is granted -z : check whether a variable or a string is empty or not if test "$num" -eq 5; ----> if ["$num" -eq 5];
[ ]	can replace the test command	There must be spaces between square brackets and the test condition if ["$num" -eq "$num"]
[[ ]]	pattern matching and regular expressions	
Math operation	(( )) + - * / % remainder ** exponentiation ++ --	num1=5;num2=10 1. echo "$(($num1+$num2))" (15) 2. echo "$((num1+num2))" (15) ((product=num1*num2)) echo "$product" (50) let used to perform the integer division, the resulting value will be an integer
Control Flow		1. if test condition then command(s) fi 2. if test condtion then command(s) else command(s) fi 3. if test condition then command(s) elif test condition then command(s) else commands(s) fi for item in list do command(s) done

1. A series of arguments separated by spaces:

```
for x in apples oranges bananas
do
echo “The item is: $x”
done
```

2. for 1 in {1..5}

```
do
read -p "Enter grade for subject #$i: " num
done
```

3. for ((i=2;i<=$num;i++))

```
do
      for ((j=1; j<=i; j++))
      do
            echo -n "$i"
      done
      echo
done
```

4. Supplied by command substitution:

```
for var in $(ls)
do
echo “Filename is: $var”
done
```

* run ls, take the output of ls

```
seq 5 : 1 2 3 4 5
seq 5 10 : 5 6 7 8 9 10
for i in $(seq 1 20)
do
      echo $i
done
1 2 3 4 5 ..... 20

while test condition
do command(s)
done

while [[ "$valid" = false ]]
do
      read -p "Type in a number less than 10: " num

if [ $(echo $num | grep "[^0-9]") ]
then
      echo "Incorrect data input!"
elif [[ num -lt 1 || num -gt 9 ]]
then
      echo "Number must be between 1 and 9!"
else
      echo "You entered $num"
      valid=true
```

		fi done
export		When a variable is exported using the export command, its value can be used by programs or scripts that are executed in the shell environment. var=10 export var
break		terminate a loop
exit		terminate a shell script exit 0: Success of execution exit any other value: Sets the exit status to 1, indicating that the script has encountered an error.
>&2		Redirect the error message to standard error instead of standard output. By doing so, any error message produced by the script will be printed to the console as an error message, rather than as normal output, making it easier to xt editor (eg. with vi: press identify and diagnose errors.
/etc/profile		first start-up file that executes when you log in, regardless of shell.
/etc/bashrc		setting the default Bash shell environments for users
~/.bash_profile ~/.bash_logout		~/.bash_profile : User-specific config start-up files ~/.bash_logout : Reset or restore the environment or properly shut-down running programs when the user logs out of their shell.

Shortcut Key(s)	Purpose
<ctrl><l>	Clear Screen
<ctrl><u>	Clear Command Line
<Up Arrow> ,<Down Arrow>	Scroll Up / Down Command History
<backspace> , <ctrl><backspace> ,<ctrl><h>	Delete character before the cursor
<ctrl><w>	Delete word before the cursor
<ctrl><a>	Move cursor to beginning of command line
<ctrl><e>	Move cursor to end of command line
<alt>f/<alt>b **(Mac: OPTION+Right/Left-Arrow)**	Move Forward/Backward one word

Common Unix / Linux Directories and their purpose

Directory Pathname	Purpose
/	Root directory (ancestor to all directories)
/home	Used to store users' home directories
/home/username	A **specific** User's Home Directory
/bin , /usr/bin	Common system binaries (commands)
/usr/sbin	Common utilities for system administration
/etc	System administration files (eg. passwd)
/var variable log	Dynamic files (log and mail files)
/tmp , /var/tmp	Temporary files for programs
/dev	Device driver files (terminals, printers, etc.)

1. Absolute pathnames : always begins from the root directory regardless of your current directory location.

/home/userid/uli101/cars.txt (absolute pathname)

- Useful if you do NOT know your current directory location
- Helps you to understand the FULL layout of pathname

2. Relative pathname : always begins from your current directory.

../../../bin
examples OR ./examples

- Possibly a shorter pathname (less typing)

3. Relative-to-home Pathnames : always begins with the tilde character (i.e. ~) to represent the current user's home directory.

~ = /home/current-user-id
~jane = /home/jane
~/uli101/examples= /home/current-user-id/uli101/examples

- Possibly a shorter pathname (less typing)

Extra demo of grep:

```
[ysseo@mtrx-node01pd test]$ grep "ford" *
grep: abc: Is a directory
cars:ford     mustang 65      45      17000
cars:ford     ltd     83      15      10500
cars:ford     thundbd 84      10      17000
cars:ford     bronco  83      25      9525
cars.bad:ford     mus tang 65      45      17000
cars.bad:ford     ltd     83      15      10500
cars.bad:ford     thundbd 84      10      17000
cars.bad:ford     bronco  83      25      9525
cars.blanklines:ford     mustang 65      45      17000
cars.blanklines:ford     ltd     83      15      10500
```

echo

```
[ysseo@mtrx-node06pd test1]$ echo "whoami"
whoami
[ysseo@mtrx-node06pd test1]$ echo "whoami" > file3
[ysseo@mtrx-node06pd test1]$ cat file3
whoami
[ysseo@mtrx-node06pd test1]$ date
Mon Feb  6 16:07:10 EST 2023
[ysseo@mtrx-node06pd test1]$ echo date >> file3
[ysseo@mtrx-node06pd test1]$ cat file3
whoami
date
[ysseo@mtrx-node06pd test1]$ echo "echo This is echo command" >> file3
[ysseo@mtrx-node06pd test1]$ cat file3
whoami
date
echo This is echo command
```