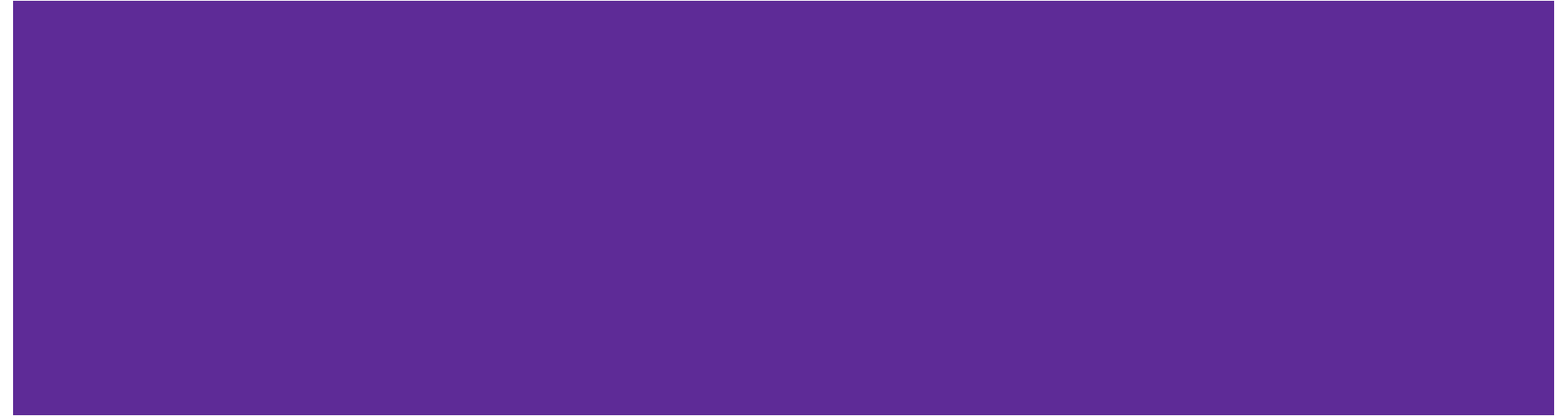# What do you think?

Work with a partner(s):

Write down as many linux commands as you can name.

CSE 374 Lecture 5
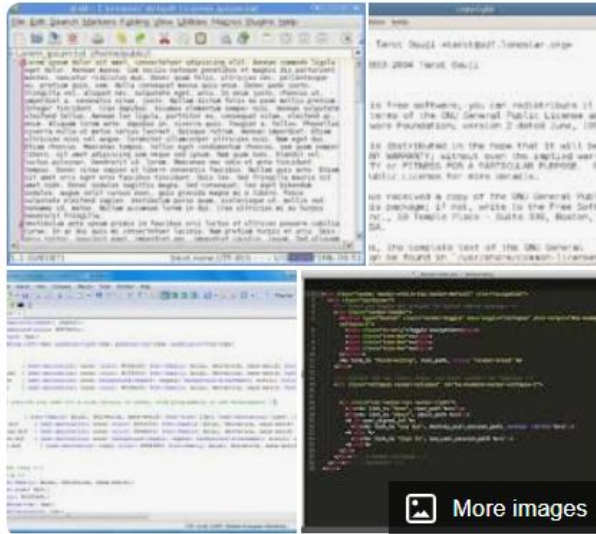
Scripting Continued

TODAY

Brief review

Scripting notes

Practice

___

Text Editors

A text editor is a type of computer program that edits plain text. Such programs are sometimes known as "notepad" software, following the naming of Microsoft Notepad. Wikipedia

Vi (Vim)

- Move around, mark edit using letter keys <hjkl>
- Get to insert mode by typing 'i', 'esc' out
- Get to menu by typing ":"
- Save and quit ":wq", no-save and quit: ":q!"
- Vim is 'improved' vi - more powerful
- Vimtutor at prompt

Emacs

- Endlessly extendable; can use arrows to move
- Menu commands use C (control) or M (meta/alt)
- Quit: C-x C-c
- Tutorial, C-h for help

Stuff in .emacs

```
14 ;; use spaces to indent
15 (setq-default indent-tabs-mode nil)
16
17 ;; change the default tab width
18 ;; and then use that width
19 (setq-default tab-width 2)
20 (setq indent-line-function 'insert-tab)
21
22 ;; always display the line numbers
23 (global-display-line-numbers-mode 1)
```

UU-:----F1   .emacs   Bot L24   (Emacs-Li

Some useful utilities

Use `man -k`: find commands with subject search

Use `find`: location a file on a computer (`locate`: locate a file in the directory database)
`whereis`: finds files with a program's name, `which`: where the executable in your path is found

Use `!?phrase`: execute the last command containing phrase

Use `^typo^correct`: correct a typo in the last command

Use `diff f1 f2`: find lines that are different in f2 than in f1 (or `sdiff`)

Process Management

Figure out what's running:

- Top
- ps (many options)

Stop processes:

- Ctrl-c (Send interrupt command)
- Kill (with options) PID

Manage processes:

- Ctrl-z (suspend process) / fg
- nice

Variables useful in a script

$# stores number of parameters (strings) entered

$0 first string entered - the command name

$N returns the Nth argument

$? Returns state of last exit

$* returns all the arguments

$@ returns a space separated string with each argument

　　(* returns one string with spaces, @ returns an array of words)

Exit Codes

Command 'exit' exits a shell, and ends a shell-script program.

Exit with no error:

    Use `exit` or `exit 0`

Exit with error:

    User `exit 1` or.. {1-255}

___

Arithmetic

Variables hold strings, so we need a way to tell the shell to evaluate them numerically:

K=$i+$j does not add the numbers

Use the shell function ((

k=$(($i+$j))

Or let k="$i+$j"

The shell will automatically convert the strings to the numbers

fibo

Conditionals

Binary operators: -eq -ne -lt -le -gt -ge

Can use the [[shell command to use < , > , ==

Syntax is a little different, but commands works as expected

if *test*; then
    commands
fi

while *test*; do
    commands
done

for *variable* in *words*; do
    commands
done

Flow control

test *expression* or [*expression*]

```
if [ -f .bash_profile ]; then
    echo "You have a .bash_profile.
Things are fine."
else
    echo "Yikes! You have no
.bash_profile!"
fi
```

http://linuxcommand.org/lc3_man_pages/testh.html

Details

Command substitution:

`$(command)` or `` `command` ``

Test:

`test condition` or `[ condition ]`

Upgrade: `[[ condition ]]`

Subshell: `( command )`

Math: `(( expression ))`

Convert to string: `$( )` or `$(( ))`

sdel

- And more:

Tests:

`-eq` : equals.

`-lt` : Less than.

`-e <file_a>`: File_a exists.

`-f <file_a>`: File_a exists and a regular file.

`-d <file_a>`: File_a exists and is a directory.

`-w <file_a>`: File_a exists with write permissions.

`-x <file_a>`: File_a exists with execute permissions.

Functions and local variables

Yes, possible

Generally, a script's variables are global

```
name () compound-command [ redirections ]
or
function name [()] compound-command [ redirections ]

Ex:

func1()
{
    local var='func1 local'
    func2
}
```

Stuff to watch out for

White space: spacing of words and symbols matters

Assign WITHOUT spaces around the equal, brackets are WITH SPACES

Typo on left creates new variable, typo on right returns empty string.

Reusing variable name replaces the old value

Must put quotes around values with spaces in them

Non number converted to number produces '0'

Shell-scripting Notes

Bash Scripting

Interpreted

Esoteric variable access

Everything is a string

Easy access to files and program

Good for quick & interactive programs

Java Programming

Compiled

Highly structured, Strongly typed

Strings have library processing

Data structures and libraries

Good for large complex programs

Scripting Style Guide

Scripts should generally be <200 lines

Do one thing and do it well.

Always use spaces, not tabs (indent line with two spaces)

Comment code with '#'

https://google.github.io/styleguide/shell.xml

Up next: Regular expressions

Regular expressions: string of symbols and characters used for pattern matching

grep

(run man grep now!)

% grep 'echo' script
% ls | grep shift

Grep - a program to do matching using regular expressions

Grep -e / egrep - uses extended regex

___

Grep Regex

By default, grep matches each line against .*p.*

You can anchor the pattern with ^ (beginning) and/or $ (end) or both (match whole line exactly)

These are still "real" regular expressions