CSE 374 Lecture 5

Scripting Continued

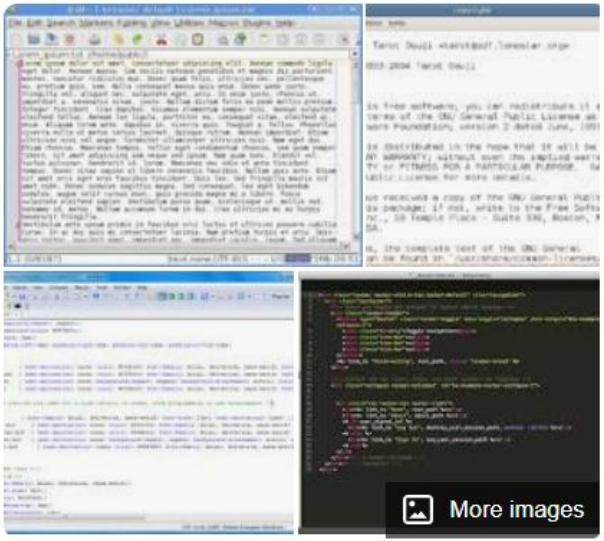
TODAY

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

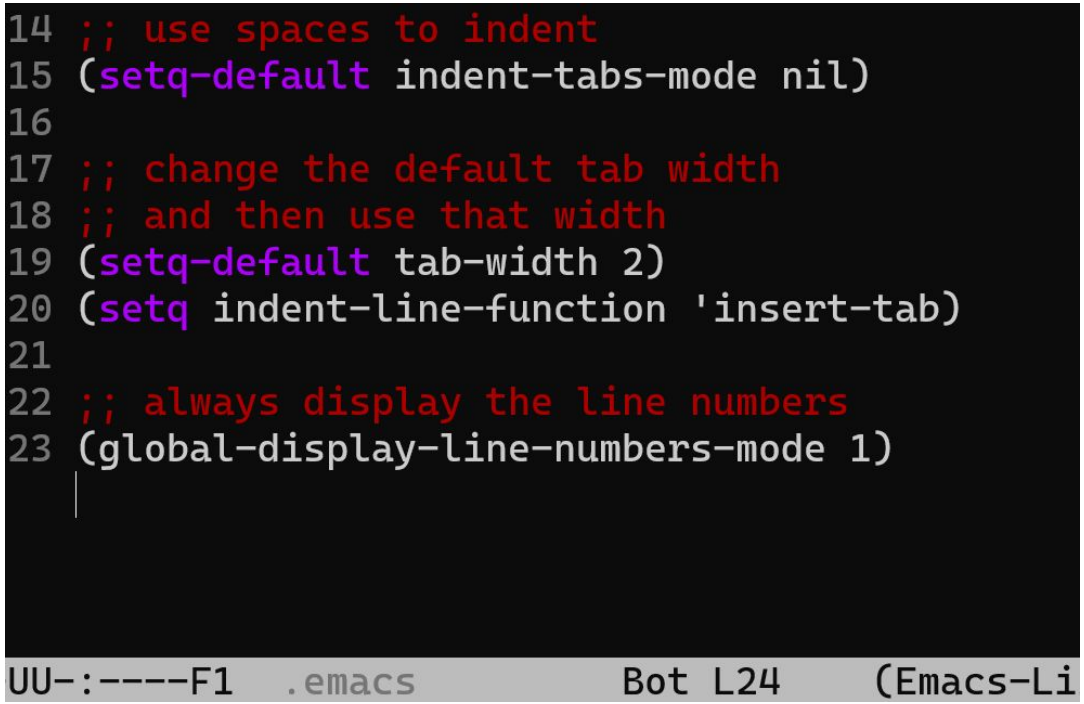
Vi (Vim)

- Move around, mark edit using letter keys
- Get to menu by typing ":"
- Save and quit ":wq", no-save and quit: ":q!"
- Vim is 'improved' vi - more powerful

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



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:

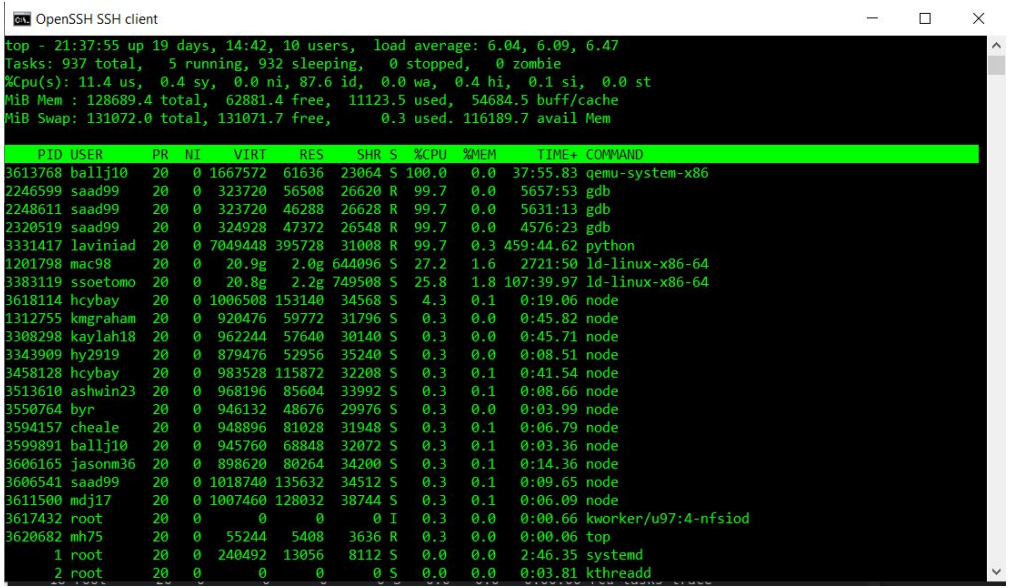
- Top
- ps (many options)

Stop processes:

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

Manage processes:

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



Variables

Shell has a state, which includes shell variables

All variables are strings (but can do math, later)

White space matters - not spaces around the ‘=’

Create: myVar= or myVar=value

Set: myVar=value

Use: $myVar

Remove: unset $myVar

List variables (use ‘set)

Export Variables

Use: export myVar

To make variable available in the current shell environment (so it can be used in sub-calls).

If a program changes the value of an exported variable it does not change the value outside of the program

: export -n remove export property

Variables act as though passed by value

Quoting Variables

In order to retain the literal value of something use ‘single quotes’

In order to retain all but $, ` , \ use “double quotes”

Put $* and $@ in quotes to correctly interpret strings with spaces in them.

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)

Okay, lets make a script!

1. First line of file is #!/bin/bash (specifies which interpreter to execute)
2. Make file executable (chmod u+x)
3. Run a file ./myNewScript
4. Shell sees the shell program (/bin/bash) and launches it to run the script
5. Can include
    a. String tests (string returns true if non-zero length, string < string, etc.)
    b. Logic (&&,||,!) - use double brackets
    c. File tests (-d : is directory, -f: is file, -w: file has write permission etc.)
    d. Math - use double parens

Script Arguments & Errors

Script refers to i^{th} argument at $i ; $0 is the program

Use ‘shift’ to move arguments towards left ($i become $i-n)

Exit your shell with 0 (normal) or 1 (error)

shiftdemo

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: `$( )` or `$(( ))`

- And more:

https://dev.to/rpalo/bash-brackets-quick-reference-4eh6

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.
```

sdel

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	Java Programming
Interpreted	Compiled
Esoteric variable access	Highly structured, Strongly typed
Everything is a string	Strings have library processing
Easy access to files and program	Data structures and libraries
Good for quick & interactive programs	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

Practice …

https://courses.cs.washington.edu/courses/cse374/23sp/assignments/exercises5.html

What steps can you take to get started on this?

Top

To

0

Up next: Regular expressions

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