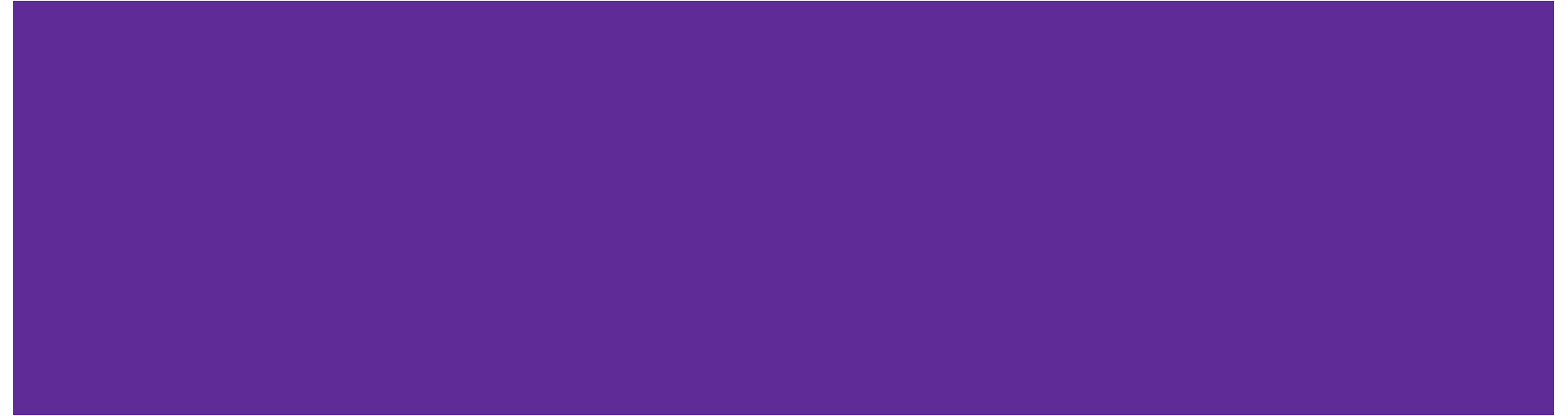


CSE 374 Lecture 5

Scripting Continued



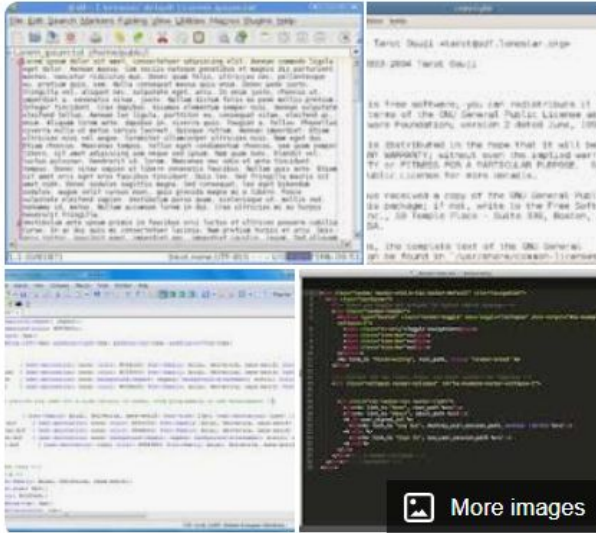
TODAY

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

```
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)
```

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

```
OpenSSH SSH client
top - 21:37:55 up 19 days, 14:42, 10 users, load average: 6.04, 6.09, 6.47
Tasks: 937 total, 5 running, 932 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.4 us, 0.4 sy, 0.0 ni, 87.6 id, 0.0 wa, 0.4 hi, 0.1 si, 0.0 st
MiB Mem : 128689.4 total, 62881.4 free, 11123.5 used, 54684.5 buff/cache
MiB Swap: 131072.0 total, 131071.7 free, 0.3 used, 116189.7 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 3613768 ballj10  20   0 1667572 61636 23064 S 100.0  0.0   37:55.83 qemu-system-x86
2246599 saad99   20   0 323720 56508 26620 R 99.7  0.0   5657:53 gdb
2248611 saad99   20   0 323720 46288 26628 R 99.7  0.0   5631:13 gdb
2320519 saad99   20   0 324928 47372 26548 R 99.7  0.0   4576:23 gdb
3331417 laviniad 20   0 7049448 395728 31008 R 99.7  0.3  459:44.62 python
1201798 mac98    20   0 20.9g   2.0g  644096 S 27.2  1.6  2721:50 ld-linux-x86-64
3383119 ssoetomo 20   0 20.8g   2.2g  749508 S 25.8  1.8  107:39.97 ld-linux-x86-64
3618114 hcybay   20   0 1006508 153140 34568 S 4.3  0.1  0:19.06 node
1312755 kmgraham 20   0 920476 59772 31796 S 0.3  0.0  0:45.82 node
3308298 kaylah18 20   0 962244 57640 30140 S 0.3  0.0  0:45.71 node
3343909 hy2919  20   0 879476 52956 35240 S 0.3  0.0  0:08.51 node
3458128 hcybay   20   0 983528 115872 32208 S 0.3  0.1  0:41.54 node
3513610 ashwin23 20   0 968196 85604 33992 S 0.3  0.1  0:08.66 node
3550764 byr      20   0 946132 48676 29976 S 0.3  0.0  0:03.99 node
3594157 cheale   20   0 948896 81028 31948 S 0.3  0.1  0:06.79 node
3599891 ballj10  20   0 945760 68848 32072 S 0.3  0.1  0:03.36 node
3606165 jasonm36 20   0 898620 80264 34200 S 0.3  0.1  0:14.36 node
3606541 saad99   20   0 1018740 135632 34512 S 0.3  0.1  0:09.65 node
3611500 mdj17    20   0 1007460 128032 38744 S 0.3  0.1  0:06.09 node
3617432 root     20   0 0 0 0 I 0.3  0.0  0:00.66 kworker/u97:4-nfsiod
3620682 mh75     20   0 55244 5408 3636 R 0.3  0.0  0:00.06 top
  1 root    20   0 240492 13056 8112 S 0.0  0.0  2:46.35 systemd
  2 root    20   0 0 0 0 S 0.0  0.0  0:03.81 kthreadd
```

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

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.

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>

Practice ...

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

W What steps can you take to get started on this?

Top



Up next: Regular expressions

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