LIS-3353

Linux/Unix Command Line Goodness

Why isn't “scripting” a bigger thing?

Why is “scripting” NOT considered “real programming?”

- Quick and dirty
- Performance can be slow (especially as compared to compiled)
- Lacks “libraries” or “frameworks”
- Few tools/structures designed for reuse or collaboration

Why is “scripting” NOT considered “real programming?”

Or, perhaps:

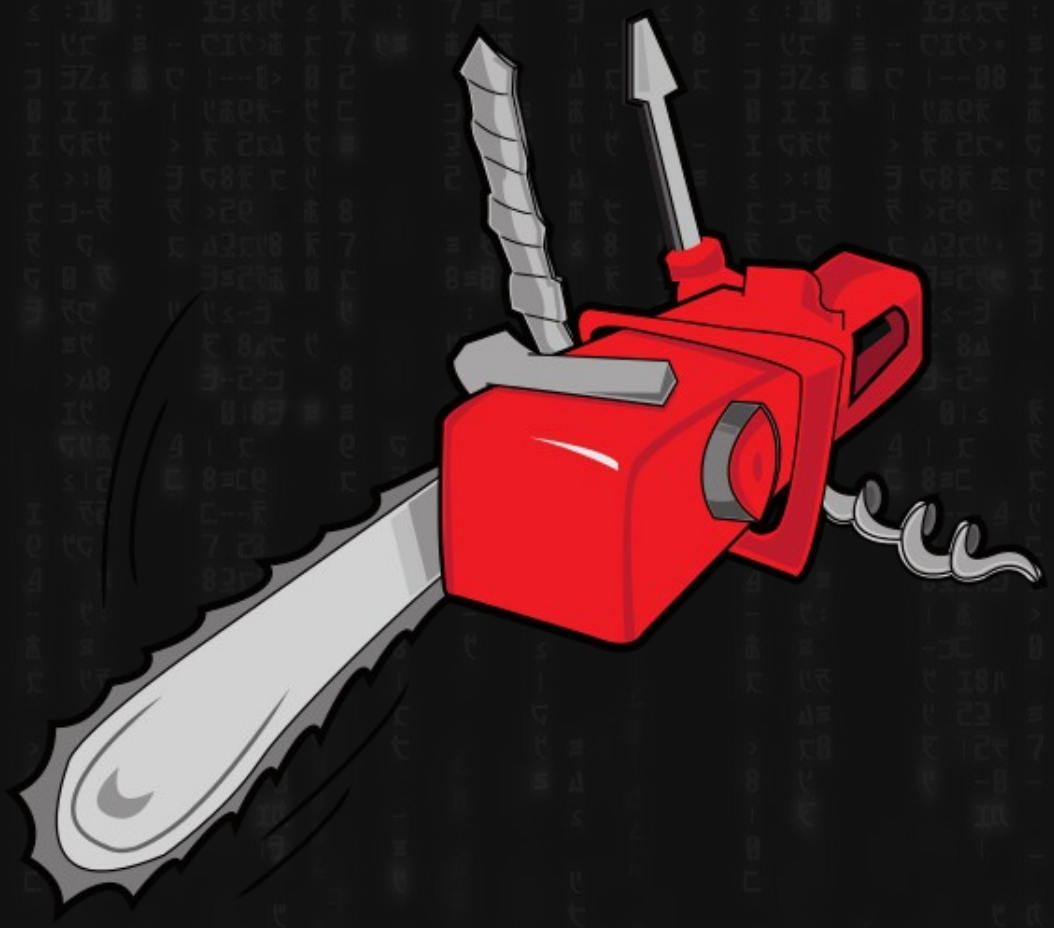
- It lets you be individually powerful, across a variety of topics, ideas, and skills…

  and not a replaceable cog in a factory? :)

“Real” Programming Languages

(C, Python, Ruby, Java etc.)

Bash/Shell scripting



For example...

```
sudo rm -rf /
```

- seriously, don't do this.

Users and Permissions
(they actually mean something here)

ROOT – Like “Administrator” or maybe “God”

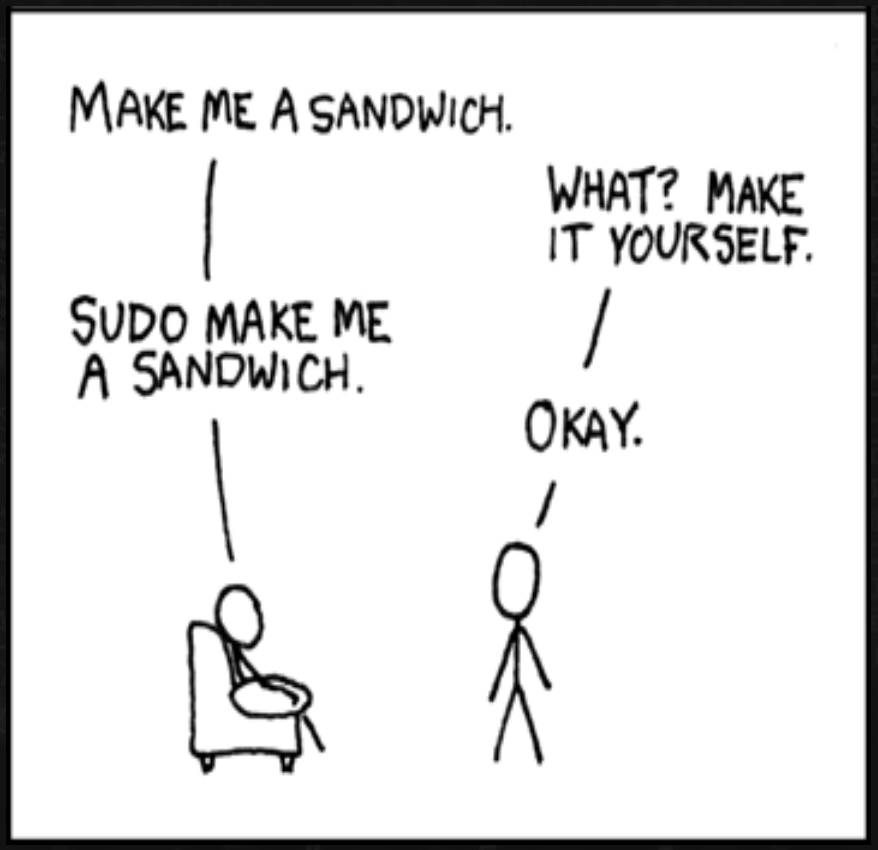
users – humans

(..and others – fake “users” to get tasks done)

Some systems (eg Ubuntu) allow for Super Users

S.U.- do “this” = `sudo`

And now...this makes sense



Why Linux had (has?) MUCH fewer virus problems

Windows historically does not distinguish between:

files you're meant to read/watch/hear/edit, and
files you're meant to run.

A piece of paper that says “Go jump off a bridge” is pretty harmless...unless....

P.S.

In this set of slides, I will **not** test you on anything from here, forward...

Permissions

aka why original windows was amazingly stupid because multiple people might want to sometimes use the same computer

Three major things you can do with files

READ (look at, view, listen to)

WRITE (and delete and edit)

EXECUTE (run as a program)

Three important "groups"

owner of the file

owner's group

everybody else

Permissions

Quick note on permissions for directories (kind of non-intuitive)

READ: Is able to read the directory listing

WRITE: Is able to change contents of the directory

(create new/delete existing files, or rename them)

EXECUTE: Is able to access/ go to the directory

Permissions

(that funky line when you do a ls -l)

0123456789

-rwxr--r--

dr-x------

Permissions
(also, how computers work)

Octal	Text	Binary	Meaning
0	---	000	All types of access are denied
1	--x	001	Execute access is allowed only
2	-w-	010	Write access is allowed only
3	-wx	011	Write and execute access are allowed
4	r--	100	Read access is allowed only
5	r-x	101	Read and execute access are allowed
6	rw-	110	Read and write access are allowed
7	rwx	111	Everything is allowed

Permissions

Thus – permission types like

644

oge

owner can read and write (4+2)

group can only read (4)

others can only read (4)

Practical Permission problems you are likely to encounter:

- If you're unable to view, execute, or delete/change a file, try this.
- If you write a little shell script (.sh), remember to set it executable. (The only permission command I use on a regular basis is chmod +x “file.sh”)
- FAT and NTFS filesystems (the ones Windows use) don't have permissions, but Linux has to occasionally pretend they do, this causes problems.
- When you're taking a website online, this is often a difficult issue. (For a good reason; you don't want website visitors overwriting your critical files!)

File Paths

File paths are HIERARCHICAL and DELIMITED by backslashes, starting with root, at “/”, e.g.

`/media/cdrom/mypaper.txt`

signifies a file “mypaper.txt” in a folder called “cdrom”, and THAT folder is in a folder called media – and “media” is in the root directory.

SPECIAL FOLDERS:

`~` or `~/` signifies the user's home folder. i.e. if your username is fsmith, and you are logged in: `~/` = `/home/fsmith/`

. (one period) refers to your current folder

..(two periods) refers to one folder up. Thus, if you're currently in `/home/fsmith` then `../` would refer to `/home`.

The LINUX Filesystem (EVERYTHING is a file!)

/bin, /sbin – Systemwide binaries

/boot – Boot Stuff

/dev - devices

/etc – (Some) helper files

/home/user – YOUR files & config (you can just back this up)

.files (dotfiles)

/lib – Libraries (kind of like dlls)

/lost+found – improper shutdown?

/opt – non-default/weird programs

/mnt, /media – generic “mount points”

/proc – the actual running processes whooa

/usr – User stuff (mostly binaries)

/tmp – temp files

/var – other spooling data, logs

Getting help

`man` (command)

`info` (might give you more info)

`apropos` (keyword to search)

`help` (pretty basic stuff)

but seriously, Google/Duckduckgo etc

Linux/Unix Commands (verbs)

Any action or program the computer can do

Commands often have ARGUMENTS, either:

OPTIONS (adverbs)

- One dash + a letter (ls -a)
- Two dashes + words (sort --reverse)

EXPRESSIONS (nouns)

- Text, numbers, files, streams, anything you want to manipulate

File Manipulation

`ls` - list

`cd` – change directory

`rm` –remove (delete for good)

`mv` – move OR rename (they are literally the same thing, weird)

`cp` - copy

Viewing text and files

`cat` - “concatenate”

`less` - this is such a terribly bad joke I hate even explaning it

...but what about editing?

Editing Files

`nano/pico` (text-based, “normal” keys)

`vi/vim` (hardcore choice 1 universal,modal)

`emacs` (hardcore choice 2)

When you turn on your computer

- 1) Electricity and Magic
- 2) BIOS/EFI/UEFI
- 3) Bootloader (Grub or windows)
- 4) Operating System

Multiple commands, one line

& - Run both simultaneously

&& - Run the first one, and then the second ONLY IF the first “succeeds,” otherwise stop.

; - Run the first one, then the second regardless of what happens.

Even MORE command line.

One quick command I totally forgot:

echo

(puts argument through stdout)

Pipes and redirects

Default behavior:

read from “stdin”, write to “stdout”

>	(over)write/replace a file
>>	write to/append to file
<	read from file
|	pipe output from first command into 2nd
tee	pipe AND write to stdout

BASH

BASH (Bourne Again) Shell - others are fish and zsh, etc

Lots of “tricks” are available here, eg

- Tab completion
- Up arrow key for history
- Ctrl-R to search history

and many MANY more

More BASH

Furthermore, you can modify this environment to fit your needs, via:

`.bashrc`

(stuff here will be run everytime you open a terminal)

A great example is the “alias” command. If a command doesn't exist for what you want to do, just ,ake up your own!

```
alias modbash='gedit ~/.bashrc'
```

Linux/Unix Commands

An action or program that a computer can do

Find them with “apropos,” learn about them with “man”

(check these out http://www.oreillynet.com/linux/cmd/)

Commands can optionally have ARGUMENTS, in the form of:

OPTIONS

- one dash + letter (`ls -a`)
- two dashes + words (`sort --reverse`)

EXPRESSIONS

- text; numbers; files; streams – things to be manipulated

Opening Files

IN TERMINAL

```
less
cat  (stdout)
```

COMMAND/ARGUMENT STYLE

```
gnome-open file
vim textfile
firefox localfile.html
firefox http://slashdot.org
```

SORT

- - i = case INSENSITIVE
- - r = REVERSE
- - g = numbers
- - R = random

GREP (line matching)

`grep OPTIONS PATTERN (FILE)`

Can search over FILES or STDIN

Also, can search ONE FILE or MANY (check -d or -R)

useful flags:

-i (case insensitive)

-v (invert search/show NON-matches)

-l (just show matching FILES, not lines)

FIND (files)

Searches directory tree rooted at given filename (default current)

Good if you also want to use parameters like “date”, “last accessed”, “size” and so forth.

Often used with -name or -iname

Also, consider “locate” (database must be setup beforehand

SED (stream editor)

Considered an entire language

Usually used with “s” for substitution

Delimiters are usually slashes but can be anything

REGULAR EXPRESSIONS

echo “Good day” | sed 's/day/night/'

http://www.grymoire.com/Unix/Sed.html

http://sed.sourceforge.net/sed1line.txt

AWK

`awk <search pattern> {<program actions>}`

Also a text-processor, good for flat-file databases

Also, an entire language

awk ' /apples/ { print $2 “ “ $1 } '

http://www.vectorsite.net/tsawk.html

http://www.pement.org/awk/awk1line.txt

CLI v GUI?

- Command Line Interface
-
- Vs
-
- Graphical User Interface

…..why not both?

CLI, but GUI-ish

- Nano
- Mc (midnight commander)

From CLI to GUI

- Opening file on command line

  firefox home.html

  (Remember, closing the terminal will also close the program)