Introduction to Linux and Bash

Danilo Croce

Original Material from

Robert Putnam
Research Computing, IS&T
putnam@bu.edu

Overview

- The Bash shell
- I/O redirection (pipes, etc.)
- Navigating the file system
- Processes and job control
- Editors
- Hello,world in C

What is Linux?

- Linux is a Unix* clone begun in 1991 and written from scratch by Linus Torvalds with assistance from a loosely-knit team of hackers across the Net.
- 64% of the world's servers run some variant of Unix or Linux. The Android phone and the Amazon Kindle run Linux.

*kernel

What is Linux?
"Small programs that do one thing well"

- From The Unix Programming Environment, Kernighan and Pike:

  … at its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves. Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools.

What is Linux: Selected text processing utilities

- awk — Pattern scanning and processing language
- cat — Display file(s)
- cut — Extract selected fields of each line of a file
- diff — Compare two files
- grep — Search text for a pattern
- head — Display the first part of files
- less — Display files on a page-by-page basis
- od — Dump files in various formats
- sed — Stream editor (esp. search and replace)
- sort — Sort text files
- split — Split files
- tail — Display the last part of a file
- tr — Translate/delete characters
- uniq — Filter out repeated lines in a file
- wc — Line, word and character count
- tar — File archive (similar to zip)

The Shell

- A shell is a **computer program that interprets the commands you type and sends them to the operating system.**
  - On Linux systems (and others, like DOS/Windows), it also provides a set of built-in commands and programming control structures, environment variables, etc.
- Most Linux systems, support at least two shells: TCSH and BASH.
  - The default shell for your account is BASH.

- "BASH" = "Bourne-again Shell"
  - (GNU version of ~1977 shell written by Stephen Bourne)

Bash environment variables

- Variables are named storage locations. So-called "environment variables" are conventionally used by the shell to store information such as where it should look for commands (i.e., the PATH). Environment variables are shared with programs that the shell runs.
- To see the current value of PATH, do:
  - `echo $PATH`
- To see all currently defined environment variables do:
  - `printenv`

Bash variables

- To create a new variable, use the assignment operator '=':
  - `foo="this is foo's value"`
  - E.g., `foo=5`
- The foo variable will now be shown if you run the 'set' command.
  - To make foo visible to programs run by the shell (i.e., make it an "environment variable"), use export:
  - export foo
- Variables are used extensively in shell scripts (about which, more later)

Using the Shell

- After you connect, type
  - `shazam` `# bad command`
  - `whoami` `# my login`
  - `hostname` `# name of this computer`
  - `echo "Hello, world"` `# print characters to screen`
  - `echo $HOME` `# print environment variable`
  - `echo my login is $(whoami )` `# replace $(xx) with program output`
  - `date` `# print current time/date`
  - `cal` `# print this month's calendar`
- Commands have three parts;
  - *command*, *options* and *parameters*.
  - Example: `cal -j 3 1999`. "`cal`" is the command, "`-j`" is an option (or switch), "`3`" and "`1999`" are parameters.
- Options have long and short forms. Example:
  - `date -u`
  - `date --universal`

What is the nature of the prompt?
What was the system's response to the command?

Command History and Simple Command Line Editing

- Try the **history** command
- Choose from the command history by using the up ↑ and down ↓ arrows
- To redo your last command, try !!
- To go further back in the command `history` try !, then the number as shown by history (e.g., !132).
  - Or, !ls, for example, to match the most recent 'ls' command.

Help with Commands

- Type
  - **`date --help`**
  - **`man date`**
- [And yes, you can always Google it]
- For a list of BASH built-in commands, just type the command ‘help’
  (and see also ‘man bash’)

On using ‘man’ with ‘less’

- The ‘`man`’ command generally pipes its output through a pager called ‘`less`’, which supports many ways of scrolling through text:

```
◦ Space, f          # page forward
◦ b                 # page backward
◦ <                 # go to first line of file
◦ >                 # go to last line of file
◦ /                 # search forward (n to repeat)
◦ ?                 # search backward (N to repeat)
◦ h                 # display help
◦ q                 # quit help
```

Plug: emacs has a man page mode that is convenient.

I/O redirection with pipes

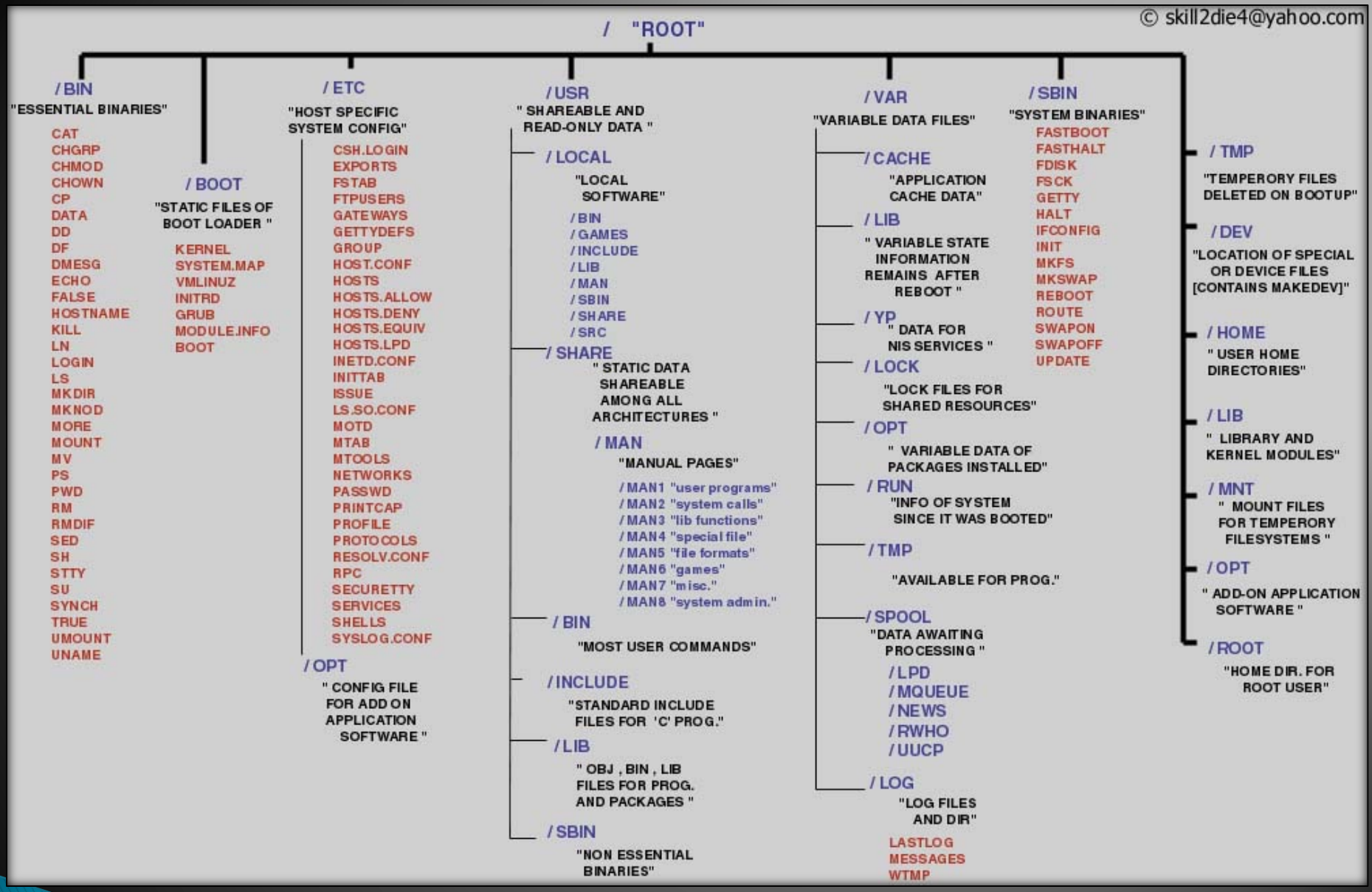
- Many Linux commands print to “standard output”, which defaults to the terminal screen. The ‘|’ (pipe) character can be used to divert or “redirect” output to another program or filter.
  - **w** # show who’s logged on
  - **w | less** # pipe into the ‘less’ pager
  - **w | grep ‘danilo’** # pipe into grep, which will print only lines containing ‘danilo’
  - **w | grep –v ‘danilo’** # print only lines *not* containing ‘danilo’
  - **w | grep ‘danilo’ | sed s/danilo/scholar/g** # replace all ‘danilo’ with ‘scholar’

More examples of I/O redirection

- Try the following (use up arrow to avoid retyping each line):
  - **`w | wc`** `# count lines, words, and characters`
  - **`w | awk -F" " '{print $1}' | less`** `# extract first column, page with 'less'`
  - **`w | awk -F" " '{print $1}' | sort`** `# sort users (with duplicates)`
  - **`w | awk -F" " '{print $1}' | sort | uniq`** `# eliminate duplicates`
- We can also redirect output into a file:
  - **`w | awk -F" " '{print $1}' | sort | uniq > users`**

The Linux File System

- The structure resembles an upside-down tree
- Directories (a.k.a. “folders” in Windows) are collections of files and other directories.
- Every directory has a parent except for the root directory.
- Many directories have subdirectories.
- Unlike Windows, with multiple drives and multiple file systems, a Unix/Linux system only has ONE file system.



The Linux File System

Navigating the File System

- **Essential navigation commands:**
  - **`pwd`** `print current directory`
  - **`ls`** `list files`
  - **`cd`** `change directory`

Navigating the File System

- We use “pathnames” to refer to files and directories in the Linux file system. There are two types of pathnames:
  - Absolute – the full path to a directory or file; begins with /
  - Relative – a partial path that is relative to the current working directory; does not begin with /
- Special characters interpreted by the shell for filename expansion:
  - ~ your home directory (e.g., /usr1/tutorial/tuta1)
  - . current directory
  - .. parent directory
  - * wildcard matching any filename
  - ? wildcard matching any character
  - TAB try to complete (partially typed) filename

Navigating the File System

- Examples:
  - **`cd /usr/local/lib`** # change directory to /usr/local/lib
  - **`cd ~`** # change to home directory (could also just type 'cd')
  - **pwd** # print working (current) directory
  - **cd ..**
  - **cd** / (root directory)
  - **ls -d pro*** # (a listing of only the directories starting with "pro")

The ls Command

- Useful options for the "**ls**" command:
  - **ls -a** List all files, including hidden files beginning with a period "**.**"
  - **ls -ld *** List details about a directory and not its contents
  - **ls -F** Put an indicator character at the end of each name
  - **ls -l** Simple long listing
  - **ls -lR** Recursive long listing
  - **ls -lh** Give human readable file sizes
  - **ls -lS** Sort files by file size
  - **ls -lt** Sort files by modification time (very useful!)

Some Useful File Commands

- **cp** [file1] [file2] copy file
- **mkdir** [name] make directory
- **rmdir** [name] remove (empty) directory
- **mv** [file] [destination] move/rename file
- **rm** [file] remove (-r for recursive)
- **file** [file] identify file type
- **less** [file] page through file
- **head -n** [file] display first n lines
- **tail -n** [file] display last n lines
- **ln** -s [file] [new] create symbolic link
- **cat** [file] [file2…] display file(s)
- **tac** [file] [file2…] display file in reverse order
- **touch** [file] update modification time
- **od** [file] display file contents, esp. binary

Manipulating files and directories

- Examples:
  - **cd** (also takes you to your home directory like cd ~)
  - **mkdir test**
  - **cd test**
  - **echo 'Hello everyone' > myfile.txt (create the file)**
  - **echo 'Goodbye all' >> myfile.txt (append to the file)**
  - **less myfile.txt**
  - **mkdir subdir1/subdir2** (FAILS)
  - **mkdir -p subdir1/subdir2** (Succeeds)
  - **mv myfile.txt subdir1/subdir2**
  - **cd ..**
  - **rmdir test** (FAILS)
  - **rm -rf test** (Succeeds)

Symbolic links

- Sometimes it is helpful to be able to access a file from multiple locations within the hierarchy. On a Windows system, we might create a "shortcut." On a Linux system, we can create a symbolic link:
  - mkdir foo # make foo directory
  - touch foo/bar # create empty file
  - ln -s foo/bar . # create link in current dir.

Finding a needle in a haystack

- The '`find`' command has a rather unfriendly syntax, but can be exceedingly helpful for locating files in heavily nested directories.
- Examples:
  - **find . -name my-file.txt** # search for my-file.txt in .
  - **find ~ -name bu -type d** # search for "bu" directories in ~
  - **find ~ -name '*.txt'** # search for "*.txt in ~

File access permissions

- Linux files have a set of associated permissions governing read, write, and execute status for the owner, members of the owner's group, and everyone else. To see a file's permissions, use the -l flag to ls:

```
[tuta0@scc1 ~]$ touch foo
[tuta0@scc1 ~]$ ls -l foo
-rw-r--r-- 1 tuta0 tutorial 0 Sep  4 10:25 foo
```

Changing file access permissions with chmod

- We can change a file's access permissions with the chmod command. There are a couple of distinct ways to use chmod. With letters, u=owner, g=group, o=other, a = all

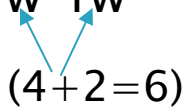
  r=read, w=write, x=execute:

```
[tuta0@scc1 ~]$ chmod ug+x foo
[tuta0@scc1 ~]$ ls -l foo
-rwxr-xr-- 1 tuta0 tutorial 0 Sep  4 10:03 foo
[tuta0@scc1 ~]$ chmod a-x foo
[tuta0@scc1 ~]$ ls -l foo
-rw-r--r-- 1 tuta0 tutorial 0 Sep  4 10:03 foo
```

Changing file access permissions with chmod (cont.)

- The chmod command also works with the following mappings, read=4, write=2, execute=1, which are combined like so:

```
[tuta0@scc1 ~]$ ls -l foo
-rw-r--r-- 1 tuta0 tutorial 0 Sep  4 10:20 foo
[tuta0@scc1 ~]$ chmod 660 foo
[tuta0@scc1 ~]$ ls -l foo
-rw-rw---- 1 tuta0 tutorial 0 Sep  4 10:20 foo
```



Changing file access permissions with chmod (cont.)

- The table

#	Permission	rwx
0	none	000
1	execute only	001
2	write only	010
3	write and execute	011
4	read only	100
5	read and execute	101
6	read and write	110
7	read, write, and execute	111

Bash startup files – dot files

- **Bash Initialization**:
  - Bash reads multiple startup files upon login.
  - Some files are system-protected.
  - Others reside in the home directory for customization.
- **Hidden Files**:
  - Start with a ..
  - Hidden by default.
- **Reveal Hidden Files**:
  - Use ls -al to display them.
  - Action: Type ls -al in your terminal.

.bash_profile, .bashrc, alias

- View .bash_profile (with less, or cat).
  - This file is executed when you log in.
  - Note that PATH is set here.
- View .bashrc
  - This file is executed when a new shell is created.
  - Note this line: alias rm='rm –i' (ask for confirmation when deleting files)

.bash_profile, .bashrc, alias

- **Removing an Alias**:
  - Edit .bashrc using editors: gedit, emacs, or vim.
  - Comment out the alias using # at the start of the line.
- **Effects**:
  - Takes effect upon the next bash shell creation.
  - For immediate removal: Type unalias rm.
- **Checking Alias Status**:
  - Use which rm or type rm to see if alias is active.
- **View All Aliases**:
  - Simply type alias.

Executing a Command or Executable in Bash

- **Command**: Simply type the command name and press Enter.
  - `command_name`
    - no need to specify anything else if the file is in one of the folder in the PATH varibale
- **Running an Executable File**
  - To run an executable script or program from the current directory:
  - `./executable_name`
    - **Note**: The `./` prefix specifies that the executable is in the current directory.
    - Ensure the file has execute permissions
      - `chmod +x executable_name`

Specifying the Interpreter in a Script

- We can execute a bash script with
  - `bash scriptname.sh`
- Scripts in Bash, Python, or other languages can specify their interpreter on the **first line**.
  - This is called a **shebang** (#!) and tells the system which interpreter to use:
    - For Bash scripts: `#!/bin/bash`
    - For Python scripts: `#!/usr/bin/python3`
- This eliminates the need to call bash or python explicitly:
  - `./my_script.sh  # runs with Bash if #!/bin/bash is at the top`
  - `./my_script.py  # runs with Python if #!/usr/bin/python3 is at the top`

Running Commands in the Background

- Add & after a command to run it in the background:
  - command_name &
- **Examples**
  1. **Running a Simple Command:** `ls -l`
  2. **Running a Script in the Current Directory:** `./my_script.sh`
  3. **Running in the Background:** `./long_task.sh &`

Processes and job control

- As we interact with Linux, we create numbered instances of running programs called “processes.”
- You can use the ‘ps’ command to see a listing of your processes (and others!). To see a long listing, for example, of all processes on the system try:
  - ps –ef
- To see all the processes owned by you and other members of the class, try:
  - ps –ef | grep danilo
- To see the biggest consumers of CPU, use the top command (which refreshes every few seconds):
  - top

Foreground/background

- Thus far, we have run commands at the prompt and waited for them to complete. We call this running in the "foreground." It is also possible, using the "&" operator, to run programs in the "background", with the result that the shell prompts immediately without waiting for the command to complete:
  - $ mycommand &
  - [1] 54356 ←-------- process id
  - $

Process control

- To get experience with process control, let's look at the "countdown.sh" script
  - cat countdown.sh
- Make the script executable with chmod:
  - chmod +x countdown
- First, run it for a few seconds, then kill with Control-C.

Process control

- Now, let's try running it in the background with &:
  - countdown 20 &
- The program's output is distracting, so redirect it to a file:
  - countdown 20 > c.txt &
- Type 'ps' to see your countdown process.
- Also, try running 'jobs' to see any jobs running in the background from this bash shell.

Process control

- To kill the job, use the ‘kill’ command, either with the five-digit process id:
  - kill 56894 #for example
- Or you can use the job number, with ‘%’:
  - kill %1 #for example

Backgrounding a running job with C-z and 'bg'

- Sometimes you start a program, then decide you want to run it in the background. Here's how:
  - `countdown 200 > c.out`
  - Press `Ctrl+z` to suspend the job.
  - Type '`bg`' at the command prompt.
  - The job is now running in the background. To bring it back to the foreground, type 'fg' at the command prompt.
  - If more than one process is suspended, you can use `jobs` and then `bg` and/or `fg` with the specific job number
    - e.g., `bg 2` to put in background the job number 2

The `screen` Command

- **What is screen?**
  - screen is a terminal session manager that allows you to start and maintain processes even if you disconnect from the terminal.
  - Ideal for commands that require significant time to complete.
- **Key Features of screen**
  - **Session Persistence**: Keeps your processes running even if the connection is lost.
  - **Multiple Windows**: Allows multiple terminal windows within a single session.
  - **Detaching and Reattaching**: You can detach from a screen session and reattach later, from the same or another terminal.
- **Basic Usage**
  - Start a new session: `screen`
  - Optional: you can run you command
  - Detach from a session: `Ctrl + A` followed by `D`
  - List active sessions: `screen -ls`
  - Reattach to a session: `screen -r <session_id>`
  - Optional a simple: `Ctrl + D` to terminate the screen

The `nohup` Command

- **What is `nohup`?**
  - nohup stands for “no hang up,” and it allows a process to keep running even after the terminal has been closed.
  - Useful when you want to run a command that continues independently of your terminal session.
- **Basic Usage**
  - Run a command with nohup: `nohup command &`
    - The output is saved to nohup.out by default unless redirected.
  - Running a script that doesn’t stop when you log out:
    - `nohup ./data_processing_script.sh &`
  - Combine `nohup` with `&` to start a command in the background:
    - `nohup ./heavy_task.sh > output.log 2> errors.log &`

When to Use `screen` vs. `nohup`

Feature	screen	nohup
Session Persistence	Yes, can reattach	No, runs detached permanently
Multiple Windows	Yes	No
Use Case	Interactive long-running tasks	Non-interactive, background tasks

Creating a user account using useradd command on Ubuntu

- You can use the useradd command is a low level utility for adding users on Ubuntu. The syntax is:
  - `sudo useradd -s /path/to/shell -d /home/{dirname} -m -G {secondary-group} {username}`
- To assign a password
  - `sudo passwd {username}`
- **sudo means that you must be an user with the root privilages**

Creating a user account using useradd command on Ubuntu

- Example: let's create a new user named vivek using the `useradd` command on Ubuntu:
  - `sudo useradd -s /bin/bash -d /home/vivek/ -m -G sudo vivek`
  - `sudo passwd vivek`
- Where,
  - **`-s /bin/bash`** – Set `/bin/bash` as login shell of the new account
  - **`-d /home/vivek/`** – Set `/home/vivek/` as home directory of the new Ubuntu account
  - **`-m`** – Create the user's home directory
  - **`-G sudo`** – **`[Optional]`** Make sure `vivek` user can sudo i.e. give admin access to the new account

More info:
`https://www.cyberciti.biz/faq/create-a-user-account-on-ubuntu-linux/`

Regular expressions

- Many Linux tools, such as grep and sed, use strings that describe sequences of characters. These strings are called regular expressions. (In fact, grep is an acronym for “general regular expression parser”.) Here are some examples:
  - ^foo # line begins with “foo”
  - bar$ # line ends with “bar”
  - [0-9]\{3\} # 3-digit number
  - .*a.*e.*i.*o.*u.* # words with vowels in order*

*to apply this against a dictionary, run ~/linux-materials/scripts/vowels.sh

File Editors

- emacs
  - Swiss-army knife, has modes for all major languages, and can be customized ad infinitum (with Emacs lisp). Formerly steep learning curve has been reduced with introduction of menu and tool bars. Can be used under Xwindows or not.
- vim
  - A better version of 'vi' (an early full-screen editor). In the right hands, is efficient, fast. Still popular among systems programmers. Non-Xwindows.
- gedit
  - Notepad-like editor with some programming features (e.g., keyword highlighting). Requires Xwindows.
- Nano
  - Lightweight editor. Non-Xwindows.

Vim modes

- Normal – navigation, text manipulation
  - Arrow keys, j,k,l,m…
  - p to put yanked text
  - x to delete character under cursor
  - dd to delete current line
  - : to enter command mode
- Insert – for adding new text
  - Enter by typing i when in normal mode
  - Exit by hitting ESC
- Visual – for selecting text
  - Enter by typing v when in normal mode
  - Copy (yank) text by typing y

Vim modes (cont.)

- Command (entered via ":" from normal mode)
  - q Quit
  - q! Quit without saving
  - w filename Write filename
  - help Get help (extensive)

Questions?