CS2042 - Unix Tools

Fall 2010

Lecture 10

Hussam Abu-Libdeh

based on slides by David Slater

September 29th, 2010

Vim = Awesome!

- Vim is a powerful lightweight text editor.
- The name "Vim" is an acronym for "Vi IMproved"
  - `vi` is an older text editor
- Ports of Vim are available for all systems
  - including Microsoft Windows

Vim allows you to perform text editing tasks much faster than most other text editors!

- Though it does have a learning curve

A modal text editor

- One of the reasons that `Vim` allows you to performs tasks quickly is because it works in modes.
- Without modes, users would have to either use command menus (with a mouse or keyboard), or use complex/long command shortcut keys involving the control key (`ctrl`) or the alt key (`alt`).
- `Vim` uses modes to speed up editing by not relying on command keys or menus.

You can do all of your editing by just using the keyboard which is super fast!!

The 3 main modes of Vim

- **Normal mode:**
  - Launching pad to issue commands or go into other modes
  - Allows you to view the text but not edit it
  - Vim starts in normal mode
  - You can jump to normal mode by pressing the `Escape` (`Esc`) key on your keyboard
- **Visual mode:**
  - Used to highlight text and perform operations on selected text
  - You get to visual mode from normal mode by pressing the `v` key on your keyboard
- **Insert mode:**
  - Used to type text into the buffer (file)
  - This probably what you're used to from your text editor
  - You get to the insert mode by pressing the `i` key on your keyboard

Command line commands

- You can issue "command-line commands" from inside Vim to perform some functionalities
  - write to disk, quite, get help, split screen, ...etc
- To issue a command, go to normal mode and the type : followed by your command

Launching Vim help

```
:help
```

Moving around

Fast

You can use your mouse to move around in Vim (assuming a graphical interface as in `gVim`

Faster

However it is much faster to just use your keyboard, and for that you can just use the arrow keys to move up/down/left/right.

Fastest

You can even be more efficient by not leaving the main area of the keyboard and using "h" to go left, "j" to go down, "k" to go up, and "l" to go right.

To start off, I recommend you just use the arrow keys.

Getting started

To get started, launch Vim and go through the built in help.

```
:help
```

You can search for help on a specific topic as well.

```
:help [topic to search for]
```

Helpful commands

I can't possibly teach you about all the power of Vim in a few minutes, however here are a few commands to help you get started.

Getting help

```
:help
```

Entering normal mode

```
<Esc>
```

Entering insert mode (from normal)

```
<i>
```

Entering visual mode (From normal)

```
<v>
```

Helpful commands

Save text to filename.txt

```
:w filename.txt
```

Exit

```
:q
```

Quit without saving

```
:q!
```

Open another file

```
:e [filename]
```

Helpful commands

Turn on syntax highlighting

```
:syntax on
```

Turn on line numbering

```
:set number
```

Turn on spell check

```
:set spell
```

Helpful commands

Split screen horizontally

```
:sp
```

Split screen vertically

```
:vsp
```

Move between split regions

```
<ctrl-w w>
```

The most helpful command

The Most Helpful Command By Far

```
:help
```

GUI + Vim

Vim can run right in your shell, but there are also implementations of it that run in a nice GUI window (with menus, toolbars, and mouse)

- Use `gVim` for that
  - I also heard about (but not used) `MacVim` for Mac OS X

Useful links

Vim project website

`http://www.vim.org/`

Vim tips and tricks

`http://www.cs.swarthmore.edu/help/vim/home.html`

Vim recipes

`http://vim.runpaint.org/toc/`

.. switching gears ..

Process? What Process?

Definition

A `process` is an instance of a running program

- More specific than "a program" because it's being executed.
- More specific than "a running program" because the same program can be run multiple times simultaneously

Example:

Many users could be simultaneously running `ssh` to connect to other servers. In this case, each *instance* of `ssh` is a separate process.

Process Identification

How do we tell one process from another?

- Each process is assigned a unique "Process ID" (or PID) when it is created
- These PIDs are used to differentiate between separate instances of the same program

How do we find out which processes are running, and with which PIDs?

The **P**rocess **S**napshot Command

```
ps [options]
```

- Reports a snapshot of the current running processes, including PIDs

By default, `ps` is not all that useful because it only lists processes started by the user in the current terminal. Instead...

`ps` Options

- `ps -e` – Lists every process currently running on the system
- `ps -ely` – Gives more info about your processes than you'll ever need
- `ps -u username` – Lists all processes for user username.
- NOTE: Options for BSD venison are different! (See manpage)

- To see information about a specific process, pipe through `grep`.
- For example,

```
ps -e | grep firefox
```

shows us information about all firefox processes

Priority

Suppose you want to run some long running scientific calculation that might take days and consume 100% of the CPU on some server. Wouldn't it be nice if there was some way to tell the server to give your process less priority with CPU time?

- Remember that although UNIX seems to run tens or hundreds of processes at once, one CPU can only run one process at a time.
- Quick switching back and forth between processes makes it seem as though they are all running simultaneously

UNIX Developers saw this type of situation coming - each process is given a `priority` value when it starts.

Initial Priority

Start a process with a non-default priority:

The `nice` command

```
nice [options] command
```

- Runs command with a specified "niceness value" (default: 10)
- Niceness values range from -20 (highest priority) and 19 (lowest priority)
- Only root can give a process a negative niceness value!
- Commands run without nice have priority 0.

Example:

```
nice -n 10 azureus
```

- Keeps torrent downloads from hogging all our CPU time!

Adjusting Priority

Adjust the niceness of a running process:

The `renice command`

`renice <priority> -p <PID>`

- Changes the niceness of the indicated process to `<priority>`
- Again, only root can go below 0!
- Can only renice processes YOU started!

Example:

`renice 5 -p 10275`

- Sets the niceness of the process with PID 10275 to 5 (slightly lower priority than default)

`renice 19 -u hussam`

- renice all my processes to 19

Going Rogue...

To end a process running in the foreground simply hit `Ctrl + C`

- What about a background process that stops working?
- What is the UNIX version of CTRL + ALT + DELETE?

The `kill` command

`kill`

`kill [-signal] <PID>`

- Sends the specified signal to the process
- By default, terminates execution

So to terminate a process:

- Look up the process's PID with `ps`
- Use that PID to `kill` the process

Useful Kill Signals

Signal used with `kill` can either be specified by their names or numerical values.

- `TERM` or 15 : Terminates execution (default)
- `HUP` or 1 : Hang-up (restarts the program)
- `KILL` or 9 : Like bleach, can kill anything

Generally speaking, the default `TERM` will get the job done.

Example:

- `kill 9000` : terminates process 9000
- `kill -9 3200` : REALLY kills PID 3200
- `kill -HUP 12118` : Restarts 12118 (handy for servers & daemons)

all in one: top

`top` is a useful little program that lists and dynamically updates information about running programs. It also allows the user to kill and renice other processes.

top

`top [-options]`

- Lists processes by default by percentage of CPU usage
- Customizable display, hit `h` for a list options
- u - show specified user only
- Can manipulate tasks: 'k' kill; 'r' renice

Introduction to Jobs

Jobs

A Job is a process running under the influence of a job control facility.

- What does that mean?!?!?!?!?

- Job control is a built-in feature of most shells, allowing the user to pause and resume tasks, as well as run them in the background (so that the shell is usable while it executes!)

How does this help?

Lets use the `ping` command as an example.

Ping

`ping <server>`

- Measures the network response time (or network latency) to a remote server.
- Sends short bursts to the server, then measures the time until they are returned.
- Can be a good way to check your network connection

Try pinging a reliable location, like `google.com`

Example:

```
ping google.com
```

Why We Need Job Control

As long as `ping` runs, we lose control of our shell. This happens with many applications which run either indefinitely or for long periods:

- Moving large quantities of files
- Compiling source code
- Playing multimedia
- Doing Scientific Computing

Example:

```
mpg123 song.mp3
```

Starting a Job in the background

To run a job in the background, we will use a new command-line operator:

&

`<command> [arguments] &`

- Runs the specified command as a background job
- Unless told otherwise, will send output to the terminal!

Since `cat` runs indefinitely with no arguments, this will illustrate our point:

Example:

```
cat &
```

- Try it without the &!

Backgrounding a Running Job

What if we start a process normally and it's taking too long?

Pausing a Job

Press CTRL + Z to pause a running process!

- When we do this, the shell tells us the paused job's JOB ID
- This Job ID is used like a process's PID
- Once we have a process paused, we can tell it to continue in the background...

The Background Command

bg's Usage

`bg <Job ID>`

- Resumes a paused job in the background
- Without a Job ID resumes the last job placed in the background

how do we find these Job IDs?

the Job Table

`jobs`

- Prints currently running, paused, or recently stopped jobs
- Prints jobs with their Job IDs

Foregrounding an Existing Job

What if we want to resume a job in the foreground?

`fg`'s usage

`fg <Job ID>`

- Resumes a paused job in the foreground
- Again without a Job ID resumes the last command placed in the background

To kill a job, either foreground it and the hit CTRL+C, or you can use the `kill` command with the PID

Dealing with Excess Output

Many programs output continuously as they run. For example `ping` and `play` both clutter up the terminal with output even when they are backgrounded.

- The solution is to use output redirection

Example:

```
ping google.com > testping.log &
```

- When you care about a program's output, redirect it to a log file.

Example:

```
play somesong.mp3 > /dev/null &
```

- If the text output doesn't matter, redirect it to `/dev/null`.

/dev/null - the black hole

`/dev/null` is a special file which has the following properties:

- Any user can write to it.
- Anything written to it goes nowhere
- it always reports a successful write.

It works like a black hole for data - you can output to it all day and it will never fill up. Anything you redirect to /dev/null just disappears.