The Command Line

Matthew Bender

CMSC Command Line Workshop

October 23

Section 1

Text Editors

Text Editors

Text editors do exactly what they say - they edit text.
They don't give you formatting like bold, italics, or allow different fonts.
Good text editors will provide features like syntax highlighting, code indentation, ability to edit remote files, etc.
It is important to learn a command line text editor for when you have to SSH into other machines.

Command Line Editors

- `ed` (1971) : a line editor - reads in a command and takes an action (not fun to use)

Command Line Editors

- `ed` (1971) : a line editor - reads in a command and takes an action (not fun to use)
- `nano` (1991) : basic, easy to use, but limited

Command Line Editors

- `ed` (1971) : a line editor - reads in a command and takes an action (not fun to use)
- `nano` (1991) : basic, easy to use, but limited
- `emacs` (1976) : powerful editor, very extensible through macros in Lisp

Command Line Editors

- `ed` (1971) : a line editor - reads in a command and takes an action (not fun to use)
- `nano` (1991) : basic, easy to use, but limited
- `emacs` (1976) : powerful editor, very extensible through macros in Lisp
- `vi` (1976, `vim` in 1991) : powerful modal editor, edit through combination of actions and motions

Section 2

Vim

Vi/Vim

`vi` was originally written by Bill Joy in 1976 as a visual mode for the line editor `ex`
Later `vim` (**vi** i**m**proved) written by Bram Moolenaar as a superset of `vi`'s features.
On most systems today, `vi` is symlinked to `vim` so that typing either will run `vim`
Running the program `gvim` will also give you a window with a graphical version of `vim` running.
`vim` also comes installed with the `vimtutor` command to help teach you how to use it.

Vim modes

`vim` is a modal text editor - which means it has several editing modes that do different things.

- Insert mode - for editing text. Anything you type here will be entered literally.
- Normal mode - for modifying text efficiently. Characters typed are interpreted as actions to take on text (e.g. `dw` will delete the current word)
- Command mode - for longer commands to change the editor (e.g. `set number` to turn on line numbers) or edit the file (e.g. `s/old/new/g` to replace all instances of `old` with `new` on the current line)
- Visual mode - for selecting text and performing an action on it. We will ignore this for now

Switching between modes

When Vim starts, it will be in normal mode. To get to normal mode from any other mode, press `<Esc>`.
To get to insert mode from normal mode, press `i` (there are many other ways, but this is the simplest)
To get to command mode from insert mode, press `:`. This will start a command prompt at the bottom of the screen starting with `:` waiting for you to enter commands.

Saving and Exiting

I will use the following notation: `:text` means go to command mode (type `:` in normal mode), type `text`, and hit `Enter`.

To save, use the command `:w` (think **w**rite)
To quit, use the command `:q` (use `:q!` to quit without saving)

You now know how to edit (`i`), save, and quit Vim!

Motions

A good Vim user knows how to use normal mode. Some keys in normal mode are used for *motions*:
`h`, `j`, `k`, and `l` mean to move left, down, up, and right, respectively.
`w`, `e`, `b` move to the next word, end of word, and back a word, respectively. Words are broken by punctuation or whitespace (use capital versions to only be broken by whitespace)
`^` and `$` move to the beginning/end of line (remember these meant beginning/end of line in regexes!)
`gg` and `G` move to the first/last line of the file. Use `92G` to go to line `92`.

Searching

Searching is considered a motion as well
Type `/stuff` to go to the next instance of `stuff` in the file.
`stuff` is a regular expression, so some characters are special (and more will be made special by escaping them)
After searching for something, type `n` and `N` to move to the next/previous instance of `stuff`

Actions

Actions take a motion, and (usually) modify the text in that motion by some way. For example, the `d` action deletes the text in the given motion. Type `dw` to delete the current word, `d^` to delete to the beginning of the line, and `dG` to delete to the end of the file.
The `y` action copies the text in the given motion (`y` for **y**ank)
The = action indents code properly in the given action. For example, to format the entire file, type `gg=G` (`gg` to go to the top, `=G` to format to the last line)
The > and < commands indent/dedent the given motion 1 tab width.

If the same command is given instead of a motion, the current line is used (e.g. `dd` deletes the current line, `yy` copies the current line, etc.)

Actions

Not all actions take a motion:
`p` and `P` pastes the last copied/deleted text after/before the cursor
`x` and `X` deletes the character at/before the cursor

Motions and actions can also take a count for how many times to do it:
`6b` will move 6 words backward
`3dd` will delete 3 lines of text
`2p` will paste whatever was copied/deleted last twice
`5yw` and `y5w` will both copy 5 words.

Commands

Command mode allows to you change the editor or perform complicated actions on text.

`set number` - turn on line numbers
`syntax on` - turn on syntax highlighting
`set hlsearch` - highlight search matches
`set ignorecase` - ignore case when searching
`set cindent` - set auto-indentation to C syntax
`set nocompatible` - don't worry about compatability with `vi` (fixes some ugly things)
`set backspace=eol,indent,start` - allows backspacing through certain things

These commands can also be put in a `~/.vimrc` file so they will be run when Vim starts up

Vim's help system

Vim provides help with all of its commands.

For general help, run `:help`
For help with a:

Normal mode command (like `d`) - `:help d`
Insert mode command (like `<Esc>`) - `:help i_<Esc>`
Command mode command (like `set`) - `:help :set`
Option (like `hlsearch`) - `:help 'hlsearch'`

More Motions

- `*`,`#` - go forward/back to next/prev instance of current word
- `f<char>`,`F<char>` - go forward/back to next instance of `<char>` on current line
- `t<char>`,`T<char>` - go forward/back right before next instance of `<char>` on current line
- {, } - move a paragraph forward/backward
- `%` - move to matching {}, `()`, or `[]`

More Ways to get to Insert Mode

- `i` : insert in front of current character
- `I` : insert in front of current line
- `a` : insert after current character
- `A` : insert after current line
- `o` : insert on new line below current line
- `O` : insert on new line above current line
- `s` : delete current character and insert
- `S` : delete current line and insert
- `c<motion>` : delete characters in `<motion>` and insert
- `C` : delete to end of line and insert (same as `c$`)