**Administrivia**

- Reminder: Homework 3 due today (5pm).
  (You will turn this in by e-mail. Please put something in the subject line that names the course and the assignment!)
- Homework 4 on Web; due next Monday.

**Why Text Editors?**

- In traditional UNIX world, everything is a text file (source code, configuration files, e-mail, input to text formatting programs, etc., etc.), so mastering a cryptic but powerful "text editor" can pay off.
- Does this approach still make sense? Maybe, though you have to choose your other tools carefully to get maximum payoff. But a determined person can use the same text editor to write programs, compose e-mail messages, "word process", etc.)
Which Text Editor?

- Traditionally a “religious war” topic, with vi and emacs having the most supporters. Both very powerful and very widely available. There are others, but they’re not as widely available, and often are more novice-friendly than expert-friendly.

- vi (or one of its clones) slightly more universally available. Plain vi is lightweight but a little primitive. vi under Linux is really vim, and has lots of extra features. Useful to know which are not “real” vi in case you ever have to use real vi. :set cp makes vim behave almost like “real” vi.

- emacs is almost as available and highly customizable — can do almost anything (compile and test programs, send e-mail, etc.) from within it. (If I had it to do over again, I might well choose emacs!)

vi Basics

- vi is “modal” — input mode and command mode. (A subset of command mode is “ex mode”, where you enter commands understood by the line editor ex. These are the ones that start with :.)

- You know how to start vi. To quit (saving changes), :wq. To quit (not saving changes), :q!. To save changes but not quit, :w.
vi Basics, Continued

- To move around, arrow keys usually work (and in vim you can use them in insert mode). Old way — which always works, but requires command mode — h, j, k, l. Does anyone still use those keys? Fanatical touch typists, maybe!
- Scrolling up and down — ctrl-F and ctrl-B. Moving to start or end of line — ^ and $.
- Many other “cursor-movement” commands, e.g., w (next word) which can be usefully combined with commands to do something (next slide).
- To find foo, /foo<CR>. (<CR> means “enter” here.) Repeat with /<CR> (forward) or ?<CR> (backward), or n to repeat search in same direction.

vi Not-So-Basics

- A lot of vi functionality is built around the idea of combining commands to do something (e.g., d to delete, y to “yank” (copy to buffer)) with commands that move the cursor (e.g., w to move forward a word, $ to move to end of line).
- So, dw deletes a word, y$ copies text from cursor to end of line, etc. For many of the commands, the letter twice applies it to a whole line (e.g., dd).
- Other useful ways to move the cursor: fc to move to next c, tc to move to just before next c. Several more; in vim,:help cursor-motions to learn more.
**vi Basics, Continued**

- Inserting text — a (after cursor) or i (after cursor), <ESC> to exit insert mode.
- Deleting text — x to delete a character, dw to delete a "word", dd to delete a line.
- To undo most recent change, u. (vim supports multiple undo. Real vi does not!)
- To read in file foo, :r foo.

**vi Not-So-Basics**

- . to repeat previous command. Precede any command with n to repeat it n times (e.g., 10dd to delete 10 lines).
- Deleted text (with x, dw, dd) goes into a “cut/copy” buffer. p pastes it back after the cursor, P before. To copy rather than delete, “yank” — yw, yy. There are also 26 more buffers, referred to by lowercase letters. E.g., "ayy to copy current line into buffer a. "ap to paste it back. (Yes, those are unmatched double quotes.)
- cw to change a word, r to replace a single character, R to go into overwrite/replace mode.
vi Not-So-Basics, Continued

- To work with blocks of text, can use `ex` commands that reference lines:
  - `: range-of-lines` `d` to delete lines. (They go into the “cut/copy” buffer and can be retrieved with `p` or `P`.) Replace `d` with `y` to yank rather than delete.
  - `: range-of-lines` `m` `target-line` to move lines. Replace `m` with `copy` to copy.

- `range-of-lines` can be one line, two lines with comma between, or `%` for all lines. Can reference lines with:
  - Absolute line numbers (`:set nu` to see line numbers). `$` is last line.
  - Relative line numbers — . is the current line, .1+ is the next line, etc.
  - “Marks” (lowercase letters). Mark current line with, e.g., a. Reference as `’a`. E.g., `: ’a’, ’b’ m`. No visual confirmation of marks.

vi Not-So-Basics, Continued

- To search and replace, can use search (`/`), replace (`cw`), and repeat (`:`).
- Or use
  : `range-of-lines` `s` `old` `/new` `/g`
  - `range-of-lines` is as before ( `%` for all lines).
  - `old` is a “regular expression” (can include wild-card-type expressions). Can be very powerful, though syntax is cryptic! In `vim`, `:help regexp` to read more.
  - Omit `g` to change only the first occurrence on each line. Add `c` to be prompted before each change.
  - Can use any character (not just `/`) to delimit `old` and `new`. 
Another plus of vi (to its fans) is interoperability with other old-style UNIX tools.

- :range-of-lines ! pgm to "filter" range-of-lines using program pgm. E.g., :%!sort to sort the whole file.
- :r! pgm to insert output of pgm after current line. E.g., :r!ls to get a list of files in the current directory.

Can edit multiple files by giving list of file names (e.g., vi file1 file2). :n cycles through files; :rew ("rewind") to go back to first. This allows making similar changes in several files, or cutting and pasting text from one file to another.
Customizing vi

- Customizations go in .exrc (or, for vim, .vimrc and/or .gvimrc) in home directory. Several ways to use different options for different needs; one involves starting vim with different configuration file (vim -u someotherfilename).
- Customizations can include settings of vi options, key mappings, abbreviations, macros, etc., etc.
- The "sample programs" page (here) has a .vimrc file with the settings I use for code (automatic indentation, etc.).

How is vim "Vi iMproved"?

- If you try plain vi (or vim in "compatibility mode") — well, vim has a lot more features. Partial list on next slide.
- vimtutor (from command line, not from within vim) starts a tutorial.
- If you must have something with little pictures across the top — gvim. (Actually might be useful while learning.)
How is `vim` “Vi iMproved”?, Continued

- “Visual mode” (to select text to delete/yank/etc.). `v` to start, move cursor to continue selecting. When the text you want is selected, `d` to delete, `y` to yank, `:` to start a : command (e.g., `:s` to search and replace).
  
  ```lang-none
  :help visual-mode for more info.
  ```

- Syntax highlighting. Can be based on filename’s extension, different for different types of files. :help syntax for more info.

- Automatic indenting of code. :help C-indenting for more info.

- Multiple “windows”. :help split for more info.

- Record sequences of commands and play back. :help record for more info.

- “diffs” mode. Start it with `vimdiff file1 file2` (`-o` to split vertically rather than horizontally).

---

**emacs**

- `emacs` is (IMO) the other major player in the text-editor wars. May be more powerful and customizable overall. Some other programs (e.g., `bash`) use some of the same key bindings.

- Add-ons available to do — “everything”? Maybe! (Try `<ESC>-x` `doctor. ctrl-x ctrl-c` to quit.)

  Add-ons/customization are done with Lisp code (similar to Scheme).

- Online help available — `ctrl-H`. `ctrl-H` `T` starts a tutorial.

- If you must have something with little pictures across the top — actually these days `emacs` started in a graphical environment has that. If you want the old-style text-only interface, use the command-line switch `-nw`. (There is also `xemacs`, but it’s a different code base.)
More Unsolicited Advice

- Both `vim` and `emacs` are powerful editors and may be worth the trouble to learn — unless you plan to do all or most of your editing with programs that have their own editor. If nothing else, they will show you a different way of doing things! My advice is to try both and see if one of them appeals to you.

- As with other UNIX things, a good way to learn them is incrementally — learn a few things, practice them, then learn a few more. The online help/tutorials are good sources of new things to try. So is your local expert. A good approach is to think of something you do often and find tedious, and try to find a way to make it easier / faster.

Minute Essay

- What text editor do you currently use under Linux? What do you like/dislike about it?