Administrivia

- Reminder: Abelson lecture 4pm today, Chapman Auditorium. Other talks:
    - "Amorphous computing" in Dr. Myers's AI class tomorrow at 11:20am in HAS 340.
    - Cryptography at 3:30pm in HAS 228.
- Sample solutions to homeworks 1, 2, and 3 on Web.

Slide 1

Minute Essay From Last Lecture

- Question: The command `ping -c 1 Janus00` will test to see if `Janus00` is network-reachable. Write a few lines of `bash` input that would let you "ping" all the `Janus` machines.
- Answer?

Slide 2

Why Text Editors?

Slide 3

- In traditional Unixworld, 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" pays off. Currently, you have to choose your other tools carefully to get maximum payoff.

Which Text Editor?

Slide 4

- 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`.

- `emacs` is almost as available and highly customizable — can do almost anything (compile and test programs, send e-mail, etc.) from within it.

`vi` Basics

Slide 5

- `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

Slide 6

- 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`.

- Scrolling up and down — `ctrl-F` and `ctrl-B`. Moving to start or end of line — `^` and `$`.

- 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` Basics, Continued

Slide 7

- 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

Slide 8

- `.` 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.

$\texttt{vi}$ Not-So-Basics, Continued

Slide 9

- To work with blocks of text, can use $\texttt{ex}$ commands that reference lines:
  - $\texttt{:}$ *range-of-lines* $\texttt{d}$ to delete lines. (They go into the "cut/copy" buffer and can be retrieved with $\texttt{p}$ or $\texttt{P}$.) Replace $\texttt{d}$ with $\texttt{y}$ to yank rather than delete.
  - $\texttt{:}$ *range-of-lines* $\texttt{m}$ *target-line* to move lines. Replace $\texttt{m}$ with $\texttt{copy}$ to copy.
- *range-of-lines* can be one line, two lines with comma between, or $\texttt{%}$ for all lines. Can reference lines with:
  - Absolute line numbers ($\texttt{:set nu}$ to see line numbers). $\texttt{\$}$ is last line.
  - Relative line numbers — $\texttt{.}$ is the current line, $\texttt{.+1}$ is the next line, etc.
  - "Marks" (lowercase letters). Mark current line with, e.g., $\texttt{a}$. Reference as $\texttt{'a}$. E.g., $\texttt{:'a,'bm.}$. No visual confirmation of marks.

$\texttt{vi}$ Not-So-Basics, Continued

Slide 10

- To search and replace, can use search ($\texttt{/}$), replace ($\texttt{cw}$), and repeat ($\texttt{.}$).
- Or use
  $\texttt{:}$ *range-of-lines* $\texttt{s}$ / *old* / *new* / $\texttt{g}$
  - *range-of-lines* is as before ($\texttt{%}$ for all lines).
  - *old* is a "regular expression" (can include wild-card-type expressions). Can be very powerful, though syntax is cryptic! In $\texttt{vim}$, $\texttt{:help regexp}$ to read more.
  - Omit $\texttt{g}$ to change only the first occurrence on each line. Add $\texttt{c}$ to be prompted before each change.
  - Can use any character (not just $\texttt{/}$) to delimit *old* and *new*.

`vi` Not-So-Basics, Continued

- `:` *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.

- Can edit multiple files by giving list of file names (e.g., `vi file1 file`).
  `:n` cycles through files; `:rew` ("rewind") to go back to first.

Slide 11

Customizing `vi`

- Customizations go in `.exrc` (or, for `vim`, `.vimrc` and/or `.gvimrc`) in home directory.

- Customizations can include settings of `vi` options, key mappings, and abbreviations.

Slide 12

How is `vim` "Vi iMproved"?

- If you must have something with little pictures across the top — `gvim`. (Actually might be useful while learning.)

- `vimtutor` (from command line, not from within `vim`) starts a tutorial.

- Online help with `:help`. `:q` to exit help. Not optimally organized, but not bad for free software.

- "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).

Slide 13

How is `vim` "Vi iMproved"?, Continued

- 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.

Slide 14

emacs

- 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).

- If you must have something with little pictures across the top — `xemacs`. Actually a different program / code base from `emacs`, though very similar.

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

Slide 15

More Unsolicited Advice

- Both `vim` and `emacs` are powerful editors and worth the trouble to learn unless you plan to do all or most of your editing with programs that have their own editor. My advice is to try both and see which one 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.

Slide 16

Slide 17

Minute Essay

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