

Slide 1

Administrivia

- Reminder: Homework 3 due today. “Programming” problems, so submit by e-mail.
- Homework 4 on the Web; due next Monday.

Slide 2

Minute Essay From Last Lecture

- Can you multithread in shell scripts? Kinda sorta: In my usage “multithreading” implies shared memory, and that’s not (as far as I know) possible in shell scripts. But you can start multiple processes (e.g., running some in the background with `&`) and wait for them to finish with `wait`.
- Do I have a list of “must have” shell scripts? I have some I use a lot, but most of them are such crude hacks I hesitate to share them.

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” 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?

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 may not be as close to universally available, and (I think!) often are more novice-friendly than expert-friendly.

Which Text Editor?, Continued

- `vi` (or one of its clones) is slightly more universally available.
- Plain `vi` is lightweight but pretty primitive.
- `vi` under Linux is really `vim`, and has lots of extra features. Can be 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`. (Try it sometime?)

Slide 5

Which Text Editor?, Continued

- `emacs` is almost as close to universally available and highly customizable — can do almost anything (compile and test programs, send e-mail, etc.) from within it. (An old joke claims that `emacs` is a wonderful operating system, lacking only a good text editor. I say more “command shell” than O/S, but — yeah.)
- Over the years people have written some truly, um, remarkable(?) customizations/add-ons (in `emacs`’s version of the functional language Lisp).
- (If I had it to do over again, I might well choose `emacs`!)

Slide 6

vi Basics

Slide 7

- `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` and do simple things. But if you normally use almost nothing but insert mode, you aren’t using this tool to anywhere near its potential. A little (more) learning may pay off!

vi Basics, Continued

Slide 8

- 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. Pressing `*` searches for the “word” under the cursor. (I only discovered this relatively recently. I like it!)

vi Not-So-Basics

Slide 9

- 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

Slide 10

- 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`.

Slide 11

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.

Slide 12

vi Not-So-Basics, Continued

- To work with blocks of text, can use `ex` commands that reference lines; this works even with base `vi` but can be cumbersome.
- Most involve a “range of lines”, which 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” (next slide).
- `: 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 mtarget-lineto` move lines. Replace `m` with `copy` to copy.

vi Not-So-Basics, Continued

- Can “mark” lines (invisibly, bleah) with `mc` for any single letter `c`.
- Can then reference with `'c` in commands on previous slide.
- Not very easy to use, but works even in base `vi`.

Slide 13

vi Not-So-Basics, Continued

- To search and replace, can use search (`/`), replace (`cw`), and repeat (`.`).
- Or use `ex` command `s`
 - `: 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. Basic idea is our next topic.
 - 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`.

Slide 14

vi Not-So-Basics, Continued

Slide 15

- 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.

Another way: Put a command to execute on a line and then use `.!sh` to execute it and get its output.

vi Not-So-Basics, Continued

Slide 16

- 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.
- `:bufdo` applies a command to all files being edited. Could be useful for search and replace across multiple files.

Customizing `vi`

Slide 17

- 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`). (Could make this a shell alias.)
- Customizations can include settings of `vi` options, key mappings, abbreviations, macros, etc., etc.
- Examples on the “sample programs” page.

How is `vim` “Vi iMproved”?

Slide 18

- 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.
- Online help with `:help`. `:q` to exit help. Not optimally organized, but not bad for free software.
- If you must have something with little pictures across the top — `gvim`. (Actually might be useful while learning.)

How is vim “Vi iMproved”?, Continued

Slide 19

- “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).
`: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. Helpful command is `=` to reindent according to current scheme. `==` to reindent current line, `gg=G` to reindent all.

How is vim “Vi iMproved”?, Continued

Slide 20

- Multiple “windows”. `:help split` for more info.
- “Macros”: Can 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).

Slide 21

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 code in a dialect of Lisp.
- 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`.

Slide 22

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 and/or faster.

Minute Essay

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

Slide 23