Chapter 2

Tools

Know thy tools!

Pick an editor, and learn it well.

Be willing to keep your knowledge of tools fresh. This process is one of balancing between expertise with your current tool set, avoiding fads, and learning useful new tools that can significantly improve your productivity.

Pick your tools carefully; most take some time to learn, and longer to become expert, and expert is what you should aim for with any tool worth using.

2.0.1 Editors

Knowing your editor and knowing it well can save you countless keystrokes and small bits of time. In aggregate, these savings add up rapidly.

The `vi` vs `emacs` debates have raged and will continue to rage about people's favorite editor. In many cases, it doesn't matter. There are outstanding, efficient systems hackers who use either. This doesn't mean that all choices are equivalent—writing code with "notepad" or "cat" isn't going to cut it. An advantage to either of these tools is that they are cross-platform. We suggest picking one, and becoming an expert at it. The skills will serve you well for years.

2.0.2 Emacs Tips

This section is derived and informed by a writeup by Adrian Perrig.

How shall we use emacs? Return to our principle of laziness. In this section, we seek to understand our editor well enough to find:

- Minimum keystrokes for maximum return
- Minimum memorization for maximum return

Conventions

Commands and typed text is in `Mono-spaced font.` The keystroke `C-e` means control-e. The keystroke `M-C-e` means Meta-Control-e. In Emacs, Meta is accomplished either by holding down the "Alt" key or by first typing "Escape."

Basic Movement

In keeping with our principles, we seek movement keys that require minimum hand motion and provide us with the most bang for the buck. The following commands seem to accomplish that as a good starting point:

Command	Effect
C-f	Next character
C-b	Previous character
C-p	Previous line
C-n	Next line
M-f	Next word
M-b	Previous word
C-e	End of line
C-a	Beginning of line
C-v	Next page (down)
M-v	Previous page (up)

Command	Effect
C-s and C-r	Incremental search forward or backward.

Command	Effect
M-C-n	Jump forward to the matching parenthesis
M-C-p	Jump backward to the previous parenthesis

Multiple Windows

Command	Effect
C-x o	Switch to other window
C-x b	Exchange the current buffer with a different one
C-x 4 b	Exchange the other half screen buffer with a different one, creating that buffer if it didn't exist.

Auto-completion

Command	Effect
M-/	Find closest word, dynamic abbreviation. Looks backwards and expands to the closest word that starts with the same letters already typed. Searches forward if it couldn't find anything back.
M-TAB	Expand tab: Tries to complete the word to a function or constant name. May open a separate buffer for disambiguation.

Cutting and Pasting

Emacs stores all deleted text (other than backspaced characters) in a "kill ring." If you kill multiple things in a row without intervening commands, it appends these to the same buffer. Otherwise, it starts a new buffer in the kill ring.

C-y will "yank" text from the kill ring. If you hit M-y *after* yanking some text, emacs will change the yanked text to the next older entry in the kill ring, and so on. Give it a try.

Macros

An amazingly useful feature in emacs is the quick construction of small macros.

Command	Effect
C-x (	Start recording a keyboard macro
C-x)	End recording
C-x e	Invoke the last created macro
M-x name-last-kbd-macro	Assigns a name to the last created macro.
M-x insert-kbd-macro	Inserts a definition of the keyboard macro into the current buffer. You can use this to "capture" your recorded macro and save it in your .emacs file for later use. Consider binding

2.0.3 Tags

Tags are a handy, powerful feature in good editors that give the editor a rudimentry knowledge of the symbols, function names, etc., in your programs. Using tags, you can quickly jump to the definition or prototype for a function or use auto-completion for longer symbol and function names.

Emacs uses a TAGS file created by the program `etags`. Creaing the file is simple: run `etags *.[ch]`. Actually, I suggest either creating a makefile target or an alias for it that's a little more complete:

```
etags *.{c,C,cc,cpp,h,hh,hpp,cpp,java} Makefile
```

to cover most of the bases. Once the TAGS file exists, using it is a piece of cake:

Command	Effect
M-.	Visit tags table.
C-u M-.	Finds the next tag
M-0 M-.	Synonym for find-next tag. Easier to type.
C-u -M-.	Find previous tag (negative modifier)
M-tab	Tag complete symbol. Type the first few characters of a function or variable and hit M-tab to autocomplete it.)
M-,	Search for next tab. Find tag is a strict match; this finds sub expressions that contain what you're looking for. Very handy.
Tab	In some prompts from emacs, the tab key will also complete on tag values.

2.0.4 Integrated Development Environments

Unless you have a compelling reason to learn a particular IDE (existing job, project, etc.) we don't suggest a particular IDE. If you do choose one, consider one that works with a variety of underlying languages and operating systems, such as eclipse. Note, however, that the integration and "do it for you" features in some IDEs can work against you in a systems programming context, when debugging involves multiple processes or machines or operating system kernel code.

2.1 Tools for Code Checking

Always, always, always use gcc -Wall