**Everyday Vim – A Basic Vim Commands Cheat Sheet**

[Vim](http://www.vim.org/) is a pretty great text editor, but learning to use it effectively can be a challenge. Even if you keep a quick-reference card or cheat sheet around, it can be difficult to figure out which commands are the most useful. But the truth is, Vim can still be super helpful if all you know is a few commands. So I’ve compiled a few of the Vim commands that I use every day.

**Movement**

**h j k l**

Basic movement keys. A step up from the cursor keys simply because they are already under your fingers. Most useful when prefixed with a number (e.g. if you need to move down by about 10 lines, hit “**10j**” instead of just holding **j** until you get there).

**b w B W**

Move back by token/forward by token/back by word/forward by word.  (A token is a sequence of letters, digits, and underscores. For the capital letter variations, a word consists of anything that’s not whitespace.) Faster than holding down a simple directional key.

**0 ^ $**

Jump to first column/first non-whitespace character/end of line, like Home and End. Faster than moving by words if you’re trying to get to the opposite end of the line.

**ctrl+u ctrl+d**

Basically Page Up and Page Down, but moves by half a screenful and doesn’t lose your cursor position.

*<line-number>***G**

Jump directly to a specific line number. Most helpful if you also have line numbering enabled (:set number).

**H M L**

Move to the top/middle/bottom of the screen (i.e. High/Middle/Low). A good first step in getting approximately to where you want to go.

**# \***

Find the previous/next occurrence of the token under the cursor.

**n N**

Repeat the last *find* command forward/backward.

**“**

(That’s two back-ticks). Jump back to where you just were. This will jump back and forth between the same two locations if you keep pressing it.

**ctrl+o ctrl+i**

Move backward/forward through the jump history. Useful if you have followed a chain of method calls and need to get back to where you were.

**Editing**

In Vim, you spend most of your time in “normal” mode, switching to “insert” mode only when you need to add or change some text. This way, all edits become their own self-contained operations that can be repeated or chained with other operations. Most editing commands may optionally be preceded by a number in order to apply it more than once (e.g. to delete three lines, press **3dd**).

**i a I A**

Enter insert mode (insert at cursor/append after cursor/insert at beginning of line/append to end of line). Press Esc to exit insert mode and return to normal mode.  It’s rarely useful to precede one of these commands with a number, but it can come in handy. Need a comma-separated list of eight 1s? Just hit “**8i1,** <esc>” then delete the trailing comma.

**o O**

Open new line (below the current line/above the current line). A quick “**o**<esc>” will add a blank line below the current line, no matter where your cursor is.

**cw cW**

Correct the token(s)/word(s) following the cursor. Basically combines delete and insert into one step.

**cc**

Correct line(s) by clearing and then entering insert mode. Starts inserting at the current indent level.

**dd**

Delete line(s). Quickly rearrange lines by deleting them, moving to the new location, and pasting with “**p**”.

**ct cf ci ca**

**dt df di da**

Correct/delete up to or including specific characters. Since there are many variations, I break it down in the section below about correcting text.

**s**

Delete character(s) at the cursor and then enter insert mode.  **cw** is usually faster if you want to correct an entire word, but this is useful for correcting a fixed number of characters (e.g. “**5s**” will correct the next five characters).

**yy**

Copy line(s).  The “y” is for “yank.”

**yw yW**

Copy token(s)/word(s).

**p P**

Paste the last thing that was deleted or copied before/after cursor (for more advanced usage, you can precede it with a *register* specification, but that’s a topic for another day).

**u ctrl+r**

Undo and redo.

**.**

(That’s a period). Repeat the previous edit command. I use this *all the time*. Did you just add a line (e.g. using “**o**” or “**O**”) that you need to duplicate five more times with only slight modifications?  Hit “**5.**” to repeat that operation, then make your modifications; no copy/paste needed.

**Correcting Text**

In a boring text editor, you are limited to very basic operations: highlight some text, delete it, type more text. Vim has the ability to highlight (it’s called “visual” mode), but I rarely use it.  It is often much faster to achieve the same thing using a few editing commands.

I frequently need to correct some text that doesn’t fall neatly onto a token or word boundary. Fortunately, operations like “**c**” (correct) and “**d**” (delete) have a number of operators that may be applied to them:

* **t***<char>* – exclusive match: continue up to (but not including) the next *<char>* on this line
* **f***<char>* – inclusive match: continue up to (and including) the next *<char>* on this line
* **i***<char>* – exclusive inner match: apply to text bounded by *<char>*, where *<char>* is from a limited set of characters that come in pairs, like quotes, parentheses, brackets, etc.
* **a***<char>* – inclusive inner match: same as above, except it includes *<char>* on both ends

Say that I have the code below, and I want to completely replace the code inside the map() with something else:

signal.map(count -> String.format(“%d cookies, ah ah ah”, count));

The first thing I need to do is get my cursor *anywhere* inside the parentheses belonging to map(). The exact command I use depends on where I end up:

* If the cursor is just inside the open paren for map(), I could use “**cf)**”. This corrects all text up to and including the next “)” on this line.
* Say the cursor is on the word “format”. I would do “**ci(**”. Vim will search backward to find the first open paren, then search forward to find its match, and correct the text between (but not including) those characters.
* Maybe my cursor was already closest to the word “cookies.” To break out of the inner parentheses, I would need to add a count and do “**2ci(**”. This is almost identical to the last example, except that the 2 is needed due to the nested parentheses.