Practical Vim Examples

Jordan Mandel

September 14, 2022

Intro

I like the book 'Practical Vim' by Drew Neil, but the examples are scattered through many text files. This brings them together under a single markdown document that somewone could work through similarly to the vimtutor document that comes with vim. Also included are summaries of tips that don't have accompanying examples. To learn vim I first recommend going through the vimtutor document a few times, then going through this document while reading the full text of Practical Vim. I recommend going through vimtutor before this to learn the very basics however. I may consider splitting this file into several by sections if having it all in one document makes limiting the scope of commands to the example we are working on cumbersome. Also worth noting that not all tips have example files. Ones that do have examples are marked with *Example*.

Another random note: I treat `<Esc>` as synoymous with `<C-[>`.

1. The Vim Way

Tip 1 *Example*: The dot command

- Indent the lines with `>G`
- Delete with `x`.
- Delete lines with `dd`
- Repeat any of these with `.`

---

practical-vim-2nd/the_vim_way/0_mechanics.txt

```
Line one
Line two
Line three
Line four
```

Tip 2 *Example*: Don't Repeat Yourself

Append semicolons with `A;`, use `j.` to repeat.

---

practical-vim-2nd/the_vim_way/2_foo_bar.js

```
var foo = 1
var bar = 'a'
var foobar = foo + bar
```

We also have:

- `C` same as `c$`
- `cc` same as `^C`. `S` does this too but the keybinding might be taken up by the plugin `lightspeed` for neovim.
- `I` inserts at betinning of line.
- `A` inserts at end of line.
- `o` opens line below
- `O` pushes current line down and opens new line on curren line

Tip 3 *Example*: Take One Step Back, Then Three Forward

- start at betinning and use `f+` to get to the first `+`
- then `s + <esc>` followed by `;` and `..` Note that `lightspeed` might break this.

---

practical-vim-2nd/the_vim_way/3_concat.js

```
var foo = "method("+argument1+","+argument2+")";
```

Tip 4: Act, Repeat, Reverse

Goes over `./u`, `;/,` and `n/N`. Ways of reversing.

More advanced content for `:s/target/replacement`: `&/u` and `qx{changes}q`: `@x/u` is mentioned later in the book.

Note mentioned is `<C-O>` and `<C-I>`. `<Tab>` is synonymous with `<C-I>`. as discussed in the tutor. Neither is `U` which undoes all changes on a particular line, which itself is a change that is added to the undo history.

Tip 5 *Example*: Find and Replace by Hand

Can't use `:%s/content/copy/g` because the word `content` has two meanings. Instead `*` to search for word under cursor, makd change with `cw`, get to nexc occurrence with `n` and decide whether to make changes with `. .`

---

practical-vim-2nd/the_vim_way/1_copy_content.txt

```
...We're waiting for content before the site can go live...
...If you are content with this, let's go ahead with it...
...We'll launch as soon as we have the content...
```

Tip 6: Meet the Dot Formula

One keystroke (some motion repeater like `n` or `;`) to move, and one keystroke (`.`) to execute.

Part I - Modes

2. Normal Mode

Tip 7: Pause With your Brush Off The Page

A painter doesn't have his brush on the canvas the whole time. Vim is like that too with normal mode.

Tip 8: Chunk Your Undos

`u` undoes a change. A change is an edit done while in insert mode, or an edit using commands in normal or command line mode. Might want to leave insert mode from time to time to break an edit into multiple changes to give undo more granularity.

Tip 9 *Example*: Compose Repeatable Changes

The idea here is to choose between (starting on `h`) `dbx`, `bdw`, or `daw`. `daw` wins because it is repeatable with `..`

---

practical-vim-2nd/normal_mode/the_end.txt

```
The end is nigh
```

Tip 10 *Example*: Use Counts to Do Simple Arithmetic

`<C-a>` and `<C-x>` respectively increment or decrement numbers, and we can use a count. It looks for the next occurrence of the number in th buffer. By default numbers starting with a zero are seen as octal. To avoid this you might have to include `set nrformats-=octal` in your vimrc. Note that `<C-a>` is a common tmux prefix. If it is a tmux prefix you will have to press `<C-a>` twice.

The instructions here are (starting on the first `.` of the second line) `yyp`, `cW.news<Esc>` to add a new line and change `.blog` to `.news` and then `180<C-x>` to decrement.

---

practical-vim-2nd/normal_mode/sprite.css

```
.blog, .news { background-image: url(/sprite.png); }
.blog { background-position: 0px 0px }
.news { background-position: 0px 0px }
```

Tip 11: Don't Count When You Can Repeat

To delete two words, `dw.` is better than `2dw` or `d2w` because it is more granular. Pressing `.` over and over again can be good too; in case you go too far you can just press `u` once or twice.

But consider `I have a couple of questions` to `I have some more questions`, starting on `a`. Use `c3w` for a clean change, rather than the dot command twice. The granular change is the right size with this method.

Tip 12: Combine and Conquer

We have motions followed by operators. Or operators followed by text objects. Duplicating an operator makes it work linewise.

- `c` is change
- `d` is delete
- `y` is yank
- `g~` is swap case
- `gu` is make lowercase; repeat with guu.
- `gU` is make uppercase
- `>` is indent right
- `<` is indent left
- `=` is autoindent

- `!` filter lines through external program

Be aware of what operator pending mode is.

3. Insert Mode

Tip 13: Make Corrections Instantly from Insert Mode

- `<C-h>` works as backspae
- `<C-w>` delete back one word
- `<C-u>` delete back to start of line These can be used other places

Tip 14: Get Back To Normal Mode

Can use `<Esc>` or `<C-[>` to get to normal mode from insert mode. Can use `<C-o>` to get into *insert-normal mode*, which allows us a single command. For example we could use `<C-o>zz` to quickly center the screen.

Tip 15 *Example*: Paste from a Register Without Leaving Insert Mode

First it gives advice on remapping the caps-lock key. I've followed the advice of mapping it to *control* and use `<C-[>` in place of escape.

Then the basic idea of this hint is that you can paste from a register with `<C-r>{register}`, but it might not respect autoindentation or textwidth, and inserts as if it's being typed. `<C-r><C-p>{register}` is smarter; it inserts all at once and without some of the problems mentioned but it is difficult to type.

- Start on `P`.
- Editing commands: `yt,. jA`
- Pasting from the register: `<C-r>0`.

---

practical-vim-2nd/insert_mode/practical_vim.txt

```
Practial Vim, by Drew Neil
Read Drew Neil's
```

Tip 16 *Example*: Do Back-of-the-Envelope Calculations in Place

The expression register, `=`, evaluates any vimscript code and returns the result, including for arithmetic expressions.

- `A` to get to the end of the line.
- `<C-r>=6*35<CR>` to get into expression register, evaluate and paste.

./practical-vim-2nd/insert_mode/back-of-envelope.txt

```
6 chairs, each costing $35, totals $
```

I'll add that you can access this register from normal mode as well. For example `"=3*3<CR>p` will insert nine.

Tip 17: Insert Unusual Character by Character Code

Insert unusual character with `<C-v>{code}`. Code for uppercase `A` for example is `065`.

Insert longer character code with `<C-v>u{code}`. Uses hexadecimal code.

`ga` will give codes for character under cursor. Lookup unicode tables for code otherwise.

Tip 18: Insert Unusual Characters by Digraph

- `<C-k>{char1}{char2}` inserts character by digraph.
- The digraphs can be induitive, for example `12` inserts ½. `<<` and `>>` insert « and ». `?I` inserts ¿ (*I*nverted).
- `:h digraphs-default` describes the conventions.
- `:digraphs` puts out a complete list.
- `:h digraphs-table` is more user-friendly than `:digraphs`

Tip 19 *Example*: Overwrite Existing Text with Replace Mode

- Start on `T`
- `f.` to get to the period.
- `R, b<Esc>` to ovwerwrite.

./practical-vim-2nd/insert_mode/replace.txt

```
Typing in Insert mode extends the line. But in Replace mode
the line length doesn't change.
```

- `gR` will treat tab characters as spaces. There is also `r` and `gr` for single character replacement.

4. Visual Mode

Tip 20: Grok Visual Mode

- Visual mode has a lot of motions like normal mode
- We define area to operate on first, then call the operator. This is the opposite of normal mode.
- For example `viwc` to change a word.

Tip 21: Define a Visual Selection

- character-wise (`v`), line-wise (`V`), and block-wise visual modes (`<C-v>`).
- `gv` re-selects the last selection regardless of what mode was used.
- You can toggle between visual modes while in a particular visual mode; just press one of the commands mentioned. Press the command for the mode you are in, or `<Esc>` to get out.
- `o` switches the end of the visual selection.

Tip 22 *Example*: Repeat Line-Wise Visual Commands

The idea here is to select the line with `print` and the line below, linewise with `V` and then you can indent them once with `>` and then again with `..` Could use `gv` to reselect but better to use the dot formula.

./practical-vim-2nd/visual_mode/fibonacci-malformed.py

```
def fib(n):
    a, b = 0, 1
        while a < n:
print a,
a, b = b, a+b
fib(42)
```

Tip 23 *Example*: Prefer Operators to Visual Commands Where Possible

The idea here is to make the contents of the tags uppercase.

- `vitU` uppsrcases `one` and `two` but then only the first three letters of three.
- `gUit` would be better because we can then just go `j.j.` to uppercase the next two tabs.

./practical-vim-2nd/visual_mode/list-of-links.html

```
<a href="#">one</a>
<a href="#">two</a>
<a href="#">three</a>
```

Tip 24 *Example*: Edit Tabular Data with Visual-Block Mode

- Start in middle of firsline
- `<C-v>3j` to select a column. `x...` do delete some columns.
- `gv` to select the column again. `r|` to insert vertical bars.
- `yyp` to duplicate first line. `Vr-` to replace with dashes.

./practical-vim-2nd/visual_mode/chapter-table.txt

```
Chapter             Page
Normal mode           15
Insert mode           31
Visual mode           44
```

Note that `r` replaces every visually selecte character. Works in Block mode too.

Tip 25 *Example*: Change Columns of Text

With cursor in first `i` of `images`, select text with `<C-v>jje`, `c`, then type a new directory name. All the text in the selected column will change on each line.

./practical-vim-2nd/visual_mode/sprite.css

```
li.one   a{ background-image: url('/images/sprite.png'); }
li.two   a{ background-image: url('/images/sprite.png'); }
li.three a{ background-image: url('/images/sprite.png'); }
```

Tip 26 *Example*: Append after a Ragged Visual Block

Start on `1`. `<C-v>jj$` to select to end of each line. `A;` to append `;`, but could append whatever text we want.

./practical-vim-2nd/the_vim_way/2_foo_bar.js

```
var foo = 1
var bar = 'a'
var foobar = foo + bar
```

5. Command Line Mode

Tip 27: Meet Vim's Command Line

- `:[range]d[elete] [x]`
- `:[range]y[ank] [x]`
- `:[line]pu[t] [x]`
- `:[range]co[py] {address}`
  - `t` is also a synonym
- `:[range]mo[ve] {address}`
- `:[range]jo[in]`
- `:[range]norm[al] {commands}`
- `:[range]substitute/{pattern}/{string}/[flags]`
- `:[range]global/{pattern}/[cmd]`

`[x]` is a register.

Can use `<C-w>`, `<C-u>`, `<C-h>` to delete like in insert mode.

Can use `<C-v>` or `<C-k>` to insert unusual characters.

Can use `<C-r>{register}`

Some mappings that are not found in insert mode.

These limited commands are supplemented by the commands available in the command history window.

Tip 28 *Example*: Execute a Command on One or More Consecutive Lines

---

./practical-vim-2nd/cmdline_mode/practical-vim.html

```
<html>
  <head><title>Practical Vim</title></head>
  <body><h1>Practical Vim</h1></body>
</html>
```

- `:1` jumps to beginning of file. `:$` jumps to end of file. `:3p` will jump to the third line and then print it.
- `:2,5p` prints lines 2 to 5 inclusive, leaving the cursor there.
- `.` represents current line. So could go `.,$p`.
- `%` represents the whole file; will take you to end of file.
- Pressing `:` with a visual selection gets us `:'<,'>`, which just means the visual selection.
  - `'<` and `'>` by themselves pick up the latest visual selection.
  - can use `:write` with a visual selection to write the visually selected part of a file to a new file.
- Searches for the patterns: `/<html>/,/<\/html>/p`
- Searches for the patterns: `/<html>/+1,/<\/html>/-1p` adds or subtracts.
- line `0` is the virtual first line of the document. Can be useful in some cases.

Tip 29 *Example*: Duplicate or Move Lines Using `:t` and `:m` Commands

Starting on the `H` of hardware store: Examples of things you could do include:

- `:6t.`: copy line six
- `:t6`: copyy current line to just below line six.
- `:t.`: duplicate current line
- `:t$`: copy current line to end of file

- `:'<,'>t0`: copy visually selected lines to start of file

`6` doesn't apply in this document because everything is on another line of the document.

./practical-vim-2nd/cmdline_mode/shoppng-list.todo

```
Shopping list
    Hardware Store
        Buy new hammer
    Beauty Parlor
        Buy nail polish remover
        Buy nails
```

Also for above: start on `H` of hardware. `Vjj`. `:'<,'>m$`. Moves selected text to end of file. `@:` repeats laxt ex command.

Tip 30 *Example*: Run Normal Mode Commands Across a Range

Start on first `v`. `A;`. `jVG`. `:normal.` Another option is `:%norm A;`

./practical-vim-2nd/cmdline_mode/shoppng-list.todo

```
var foo = 1
var bar = 'a'
var baz = 'z'
var foobar = foo + bar
var foobarbaz = foo + bar + baz
```

Tip 32: Tab Complete Your Ex Commands

Can complete using `<Tab>` or `<C-d>`. Other advice regarding command completion. When making custom `Ex` commands we might want to look at `:h command-complete`.

Tip 33 *Example*: Tab Complete Your Ex Commands

Start on first `t` of tally. `*` and then `cwcounter<Esc>`. Then `:%s//<C-r><C-w>/g` to replace everything. We don't have to enter a search pattern because it is saved from when we used `*`.

./practical-vim-2nd/cmdline_mode/loop.js

```
var tally;
for (tally=1; tally <= 10; tally++) {
  // do something with tally
};
```

Tip 34 *example*: Recall Commands From History

Can use `<Up>` or `<Down>` to go through history of both `:` commands and `/` searches. or also work. Will narrow down history based on what you've already typed.

When writing a Ruby script we might start with `:write` followed by `:ruby %`. Can combine them into `:write | !ruby %`

Might do this using command-line window: `q:`. Can also do `q/` window. Can press `<C-f>` in rregular search or command line mode to get to the command-line window or search window.

Quick editing example (not from an example file). Start on `w`. `A |<Esc>`. Then `J` to join lines. `0ceupdate` to change word. The book uses `:s/write/update` command but that makes no sense.

```
write
!ruby %
```

Tip 35 *Example*: Run Commands In The Shell

Can use `!` to run commands in the shell. Can use address symbols like `%` to pass the contents of the file. Can open a shell with `:term` or `:shell` the latter of which was removed in Neovim. Preferred to use `<C-z>` and then `fg` commands.

- `:read !ls` will read the output of the `ls` command into the active buffer.
- `write` is generally used for saving buffers. But if we append a command with `!` the contents of the buffer will be passed to the command.
- Do not put an exclamation point in the wrong place and do `:write` which can acidentally overwrite a file.

Filtering through range. The example they give here is `:2,4!sort -t',' -k2` which sorts on last name. Can use `!{motion}` to automaticaly populate the range with something to pipe to the filtering command.

./practical-vim-2nd/cmdline_mode/emails.csv

```
first name,last name,email
john,smith,john@example.com
drew,neil,drew@vimcasts.org
jane,doe,jane@example.com
```

In summary there is:

- `:shell` and `:term`
- `:!{cmd}`
- `:read !{cmd}`
- `:[range]write !{cmd}`
- `:[range]!{filter}`

Some commands are special like `make` and `grep` and have wrappers in Vim.

Tip 36 *Example*: Run Multiple Ex Commands as a Batch

./practical-vim-2nd/cmdline_mode/vimcasts/episodes-1.html

```
<ol>
  <li>
    <a href="/episodes/show-invisibles/">
      show invisibles
```

```
    </a>
  </li>
  <li>
    <a href="/episodes/tabs-and-spaces/">
      tabs and spaces
    </a>
  </li>
</ol>
```

Want to transform it to:

./practical-vim-2nd/cmdline_mode/episodes-1.html

```
Show invisibles: http://vimcasts.org/episodes/show-invisibles/
Tabs and Spaces: http://vimcasts.org/episodes/tabs-and-spaces/
```

Drew Neil goes about this by

```
:global/href/j
:vglobal/href/d
:%norm A: http://vimcasts.org
:%norm yi"$p
:%s/\v^[^\>]+\>\s//g
```

where `global` and `vglobal` can respectively be abbreviated `g` and `v`. `:[range]g[lobal]/{pattern}/[cmd]` executes command onlines where pattern matches. `vglobal` (or `g[lobal]!` does it where it does not match.

Can put all of these commands into a `batch.vim` file and execute it on the current buffer by `:source batch.vim` Can then use `:next`/`:prev` to switch buffers (going through the arglist) and source the script. Or can open them all at once and run on all open buffers with `:argdo source batch.vim`.

Part II - Files

6. Manage Multiple files

Tip 37: Track Open Files withthe Buffer List

Buffers are what we are editing in memory, files are on the disk. `:ls` lists the open vuffers. `%` marks the current buffer. `#` represents the alternate file that we can toggle to/from with `<C-6>` (listed as `<C-^>` for some reason).

There is `:bn`, `:bp`, `:bf`, `:bl`. Look at Tim Pope's Vim Unimpaied for other mappings.

Also `:b{buffer number}` Or `:b {enough of a match}`, can use tab completion.

`:bufdo` is similar to `:argdo` which will be covered later.

`N,Mbd` and `bd M N` delete buffers. First one is for range `N` to `M`.

Might be easier in general ot use the argslist, and tabs.

Tip 38 *Example*: Group Bufferes into a Collection with the Argument List:

The arglist is changeable while the buffer list is static. `:args` lists all of the files that opens when we opened vim, but we can counterintuitively change it.

Can also do `:args filename1 filename2 ...`; this method accepts wildcards includint `**`.

./practical-vim-2nd/files/.chapters

```
# START:head
the_vim_way.pml
normal_mode.pml
insert_mode.pml
visual_mode.pml
# END:head
ex_mode.pml
managing_files.pml
files.pml
motions.pml
jumps.pml
copy_and_paste.pml
macros.pml
patterns.pml
search.pml
substitution.pml
global_commands.pml
ctags.pml
quickfix.pml
grep.pml
auto_complete.pml
spell_check.pml
```

Then can execute (from within the `files` dierectory) which will add the chapters to the arglist

`:args `cat .chapters``

can use `:argdo [commands]` to run `ex` commands on each file in the arglist.

Tip 39: Manage Hidden Files

- `+` means unsaved changes
- `h` means hidden
- `%` means active file
- `#` means alternate file
- `:qa[ll]` is quit all
- `:wa[ll]` is write all
- `:w[rite]` is write one file
- `:e[dit]!` is write all

Hidden setting lets us do `:bn`, etc. without `!`. Also lets us do `:argdo` etc without interference.

- `:wn` write and edit next arg; good to use after `:*do` commands to check each file. Or use `:argdo write` or `:wall`.

Tip 40: Divide your Workspace into Split Windows

Can continue holding down `<C>` whie pressing second key.

Splitting:

- `<C-w>s`: horizontal split
- `<C-w>v`: vert split
- `:sp[lit] {file}`: horizontal split to file
- `:vsp[lit] {file}`: vert split to file

Switching windows:

- `<C-w>hjkl`: go to window in direction
- `<C-w>w`: rotate windows

Closing Windows

- `<C-w>c`: close active window and `h: window-moving`
- `<C-w>o`: close all but active window

Resizing/arranging Window

- see `h: window-resize` for more
- `<C-w>=` equalize width and height of all windows.
- `<C-w>_` maximize height of active window
- `<C-w>|` maximize width of active window
- `[N] <C-w>_` set height to [N] rows
- `[N] <C-w>|` set width to [N] columns

see for more.

Not mentioned: `<C-w>R` and `<C-w>r` (for rotating windows in column or row), and `<C-w>x` (for exchanging windosw)

`<C-w>{HJKL}` for sending window to extremities of screen.

Tip 41: Organize Your Window Layouts with Tab Pages

- For more details see `:h tabpage`; a bit to unpack there.
- `:lcd {path}` lets us change directory of current window.
- New tab with `:tabe[dit] {filename}`
- `<C-w>T` send window to new tab.
- `:tabc[lose]` close current tab
- `:tabo[nly]` close all tabs but current
- `{N}gt` or `:tabn[ext] {N}` switch to tab page number `N`, or just next tab when `N` is omitted.
- `gT` or `:tabp[revious]` previous tab page.
- `{N}tabmo[ove]{N}`; n can be in either position, moves to after `N`, use `0` for first, no number for last.

7. Open Files and Save Them to Disk

Tip 42: Open a File by Its Filepath Using `:edit`

- `:edit %<Tab>`: expands to path of current buffer from working directory of vim.
- `:edit %:h<Tab>` expands to full path of current buffer of vim.

– Remapped that with `cnoremap <expr> %% getcmdtype() == ':' ? expand('%:h').'/' : '%%'`

Tip 43: Open a File by Its Filename using `:find`

Form `:h file-searching`: -Downward Search: - `**` in `:find` only works at begining of a name, only works for directory. - `*` works pretty much the same - Upward Search: - Searches all files upwards from the given directory, but will go one down into directory of that name in directories above. Append upward stop directory with `;`: ie `:set path=include;/u/user_x`

Can do `:set path+=app/**` to add it. Then can find files with `:find app/**` using the command line. Or `:set path={path}` to completely reset it.

Press `<Tab>` to go to deeper match if a shallower one is matched first.

Tip 44: Explore the File System with netrw

Open for active buffer (short for `:Exlplore`:

`:E`

Other commands like `d` to make directory, `%` make new file, `rename`.

Can also open files across a network.

Tip 45: Save Files to Nonexistent Directories

Can make a directory with `:!mkdir -p %:h` with `-p` saying make intermediate directories. Recall `%:h` makes path to current directory without the current file name. I've remapped it to `%%` with

```
cnoremap <expr> %% getcmdtype() == ':' ? expand('%:h').'/' : '%%'
```

Tip 46: Save a File as the Super User

To write with sudo permissions we can go

```
:w !sudo tee % > /dev/null
```

Writes to current file, which is what `%` expands to. `/dev/null/` is just so that we don't see the output.

Part III - Getting Around Faster

8. Navigating Inside Files With Motiions

Tip 47: Keep Your Fingers on the Home Row

Just says to use `h`, `j`, `k`, and `l`.

Tip 48: Distinguish Between Real Lines and Display Lines

Talks about using `g` + `[motion]` to treat wrapped lines as regular lines.

Tip 49: Move Word Wise

Talks about basic motions, differences between `WORDS` and `words`.

Tip 50: Find By Character

Talks about `t` and `f`.

Tip 51: Search to Navigate

Talks about basic searching with `/` and using it as a motion with operators.

Tip 52 *Example*: Trace Your Selection with Precision Text Objects

---

motions/template.js

```
var tpl = [
  '<a href="{url}">{title}</a>'
]
```

- Can type text object (`it`, for example) to change selection while already in visual mode.

Tip 53 *Example*: Delete Around, Change Inside

Recommends using `i` text objects for changes, `a` text objects for deletions. Note that for text objects where the delimeter is also a delimeter for another text object(`s`,`p`,`w`,`W`).

Tip 54: Mark your Place and Snap Back to It

- `m{a-zA-Z}`
- lowercase for particular buffer, uppercse across buffers
- `'{mark}` for the beginning fo the line (first non-whitespace character)
- `` `{mark} `` for exact positon ‘;

Automatic Marks

- `` `. `` for last change (related to change list)
-

``` ``

for last jump (related to jump list)

- `` `^ `` last inserction
- `` `{[\]} `` for start or end of last change or yank
- `` `{<\>} `` for start or end of last visual selection

Tip 55 *Example* : Jumping Between Parentheses

Basically says to jump delimeters with `%` so that we can jump back with `` ` ` `` or `<C-o>` and change the other delimeter.

./practical-vim-2nd/motions/parentheses.rb

```
cities =  %w{London Berlin New\ York}
```

---

9. Navigate Between Files With Jumps

Tip 56: Traverse The Jump List

- `:jumps` describes jumps
- `<C-o>` anc `<C-i>` for jumps.
```

- `<C-i>` and `<Tab>` are identical in vim.
- Some motions are jumps and some aren't.

Tip 57: Traverse The Change List

- `:changes` shows the changelist
- `g;` and `g,` traverse the changelist.
- The `.` mark is for the last change but don't need it with the previous bullet point.
- The `^` mark is for the last insertion and you can get to it using `gi`

Tip 58: Jump to the Filename Under the Cursor

- `gf` above file name jumps to file.
- `suffixesadd` option
  - `:set suffixesadd+=.rb`
- `:set path?` shows the path. we can modify it.
- `<C-]>` is similar but requires more setup.

Tip 59: Snap Between Files Using Global Marks

- `m{capital letter}` makes a mark persistent across files and editing sessions Good to use any time you think you might jump around.

Part IV - Getting Around Faster

10. Navigating Inside Files With Motiions

Tip 60: Delete, Yank, and Put with Vim's Unnamed Register

- `ddp` to transpose lines
- `xp` to transpose characters

Tip 61: Grok Vim's Registers

- `"` is tne unnamed register, easy to overwrite
- `0` is the yank register
- With named registers `a-z` we overwrite when it is lowercase and append when it is uppercase.
- `_` black hole register.
- `+` is the system clipboard
- `*` is the most recently selected text, can paste with middle mouse button (for linux). Same as `+` on other systems.
- `=` is expressionr register.

Other Registers include: - `%` name of current file - `#` name of alternate file - `.` last inserted text - `:` last ex command - `/` last last search pattern

Tip 62: Replace a Visual Selection with a Register

In visual mode `put` replaces the contents of the visual selection.

They shwo a technique for swaping words. Basically delete, mark, go, replace, hop back to mark, put again.

Tip 63: Paste From a Register

Only new things are the `gp` and `gP` commands. They just change where your cursor is after pasting.

Tip 64: Inteact with System Clipboard

Talks about paste toggle and paste mode to avoid problems with audotinent. I get around this by remapping `<C-r>` to `<C-r><C-p>`. Also by pasting from `+` and `*` registers.

11. Macros

Tip 65 : Record and Execute a Macro

- `q[register]` to record a macro.
- `@[register]` to play back.
- `@@` to replay most recent macro.

Tip 66 : Record and Execute a Macro

Make sure we are using context aware commands. Macros will abort when they fail.

Tip 67 *Example*: Play Back with a Count

Start on the begining. `f+` followed by `s + <Esc>`. then `qq;.q` to record a macro. Then `22@q` to execute it. Lightspeed seems to interfere.

```
x="(" + a + "," + b + ","+c + "," + d + "," + e")";
```

Tip 68 *Example*: Repeat A Change on Contiguous Lines

record macro (cursor on `1.` of one): `qa0f.r)W~jq`

---

practical-vim-2nd/macros/consecutive-lines.txt

```
2. one
2. two
3. three
4. four
```

Can play it back several times: `@a`

Can also linewise-visually select lines and go `:<,>norm @a`

Tip 69: Append Commands to a Macro

Use a capital letter to append to a macro. Works for yanking as well. Stored in the same registers as yanks.

Tip 70: Act Upon a Collection of Files

practical_vim/macros/ruby_module/animal.rb

```
class Animal
  # implementation
end
```

---

source `practical_vim/macros/ruby_module/rc.vim`

`cd code/macros/ruby_module`
`:args *.rb`

hit `:first` to go to beginning of argument list.

can record macro; then undo changes with `:edit!` and then go `:argdo normal @a` (note that `:edit!` won't work if we've saved the file.

can make the buffer advance files in arglist at end: `qA:nextq` then `22@a`

can save everything with `:wall` or `:argdo write`

Tip 71: Append Commands to a Macro

---

practical-vim-2nd/macros/incremental.txt

```
1) partridge in a pear tree
2) turtle doves
3) French hens
4) Calling birds
5) golden rings
```

---

macro to record is:

```
let i=1
qa
I<C-r>=i<CR>
<esc>
let i += 1
q
```

Tip 72: Append Commands to a Macro

Can put a macro with `:put a`, edit the macro, and re-yank it; better not to do a line-wise yank because then there might be a new-line character.

Can use the substitute command `:let @a=substitute(@a, '\~', 'vU', 'g')`

Part V - Patterns

Chapter 12: Matching Patterns and Literals

Tip 73: Tune Case Sensitivity of Search Patterns

can type `\c` or `\C` at end for case insensitivity or sensitivity respectively

Tip 74: Use the P̌attern Switch for Regex Searches

it makes searching with a regex much eaiser so that everything but `-`, and alphanumeric characters have a special meaning; closer to perl regex than posix, which is what is used without `\v`.

`\{x}` matches an alphanumeric class

Tip 75: Use the `\V` Literal Swtitch for Verbatim searches

after `\V` only the backslash has special meaning.

Tip 76: Use Parenthese to Capture Submatches

can reference previous matches in parenthases using `\1` ... and higher numbers up to `9` .

`\_s` matches white space or line break. `\_a` matches alphabetic character, or line break `\a` just alphabetic character. `\n` is just for line break.

Can use `%()` to avoid capturing the pattern in a register