GIT INSTRUCTION MANUAL

EVAN BERGERON
FREDERICK LEE
ZIYANG WANG
SUNNY GAKHAR
NISHAD GOTHOSKAR

Contents

Introduction

This is a manual on how to set up Git on your computer and set up a basic Git workflow. The document covers installing and setting up Git and how to work with Git. The majority of this document is primarily intended for users who want to learn Git to use it in a fast-paced setting such as a hackathon, this document is also useful as a reference for experienced Git users who want to refer to some specifc concept or command which they need.

Easily Reordering Commits

Weve all been there. You pull from master, take several minutes to clean up the various merge conflicts, and then are ready to push. You pull one last time before pushing, and what do you know - someone has pushed again in the last couple minutes.

Previously, my workflow for this situation would look something like

```
$ git checkout -b tmp
$ git checkout master
$ git reset --hard HEAD~
$ git cherry-pick tmp
```

This works fine, but theres a much easier way – one that involves very little typing. We can simply say

```
git rebase -i HEAD 2
```

This will open an interactive git rebasing session (the `-i` stands for interactive). The window will display something along the lines of

```
pick 370e221 Commit one
pick c342396 Commit two
```

In whichever text editor were in, we may simply reorder these lines to reorder the commits. Much shorter!

Adding partial files

I just used this, actually. Suppose youve changed a single file `foo.c` in different sections, and each of these changes are logically different. For instance, maybe you refactor some function `foo`, while at the same time fixing a bug in function `bar`. Rather than create a separate branch and manaully edit the files, we can simply say

```
git add -p foo.c
```

This will bring up an interactive prompt. It will automatically cycle through all the different areas of the diff, asking you if you want to stage each section. You may hit `y` or `n` for yes or no.

Once youre done adding the subset of changes you want to commit, you can double-check you have the right changes staged by saying

```
git diff --cached
```

Once youre sure that youre good to go, just commit your changes as normal. You can repeat this process for the remaining changes. (Or just do a normal `git add` at this point).

Git aliases

Git aliases are a good way to save yourself a lot of typing. I frequently want to see the git commit history, but dont especially care about the body of each commit. Heres the command Ive added to my configuration:

```
git config --global alias.l "log --oneline"
```

I can then just type `git l` to see a one line log of this commit history. If there are several long commands you use frequently, this can be a great way to save yourself some time.

Vim workflows

At the time of writing, perhaps the most feature complete vim-git plugin is Tim Popes vim-fugitive. Consequently, we will assume usage of this plugin throughout the entire vim workflows tutorial.

Installing vim-fugitive

There are a number of ways to install vim-fugitive. The one suggested by Tim Pope is as follows:

```
$ cd ~/.vim/bundle
$ git clone git://github.com/tpope/vim-fugitive.git
$ vim -u NONE -c "helptags vim-fugitive/doc" -c q
```

Vundle is a great plugin manager for vim – if you use this, you may simply add the line

```
Plugin 'tpope/vim-fugitive
```

to your vimrc and run the `PluginInstall` command.

Easy git blame

Youre browsing some file and discover a horrible bug written by one of your coworkers. Youre about to storm over to someones desk and verbally abuse them for producing incorrect code. Before you deliver your diatribe, you need to know who to blame.

Before using `vim-fugitive`, you would have to exit vim, manually type `git blame <filename>`, and then search for the relevant line in the output. Now, you can simply type `:Gblame` in your vim prompt, and a vertical split will open up right next to the line in question. You could even establish a keybinding to do this for you! What was once several lines of typing is now a single keystroke away! Your coworkers have never been so scared...

Emacs workflows

Similarly to vim-fugitive for vim, Magit is (at the time of writing), the most feature-complete git wrapper for emacs. We will thus assume usage of this package.

Installing Magit