GitHub Cheat Sheet

A collection of cool hidden and not so hidden features of Git and GitHub. This cheat sheet was inspired by Zach Holman's Git and GitHub Secrets talk at Aloha Ruby Conference 2012 (slides) and his More Git and GitHub Secrets talk at WDCNZ 2013 (slides).

Shortlink: `http://git.io/sheet`

Read this in other languages: English, 한국어, 日本語, 简体中文, 正體中文.

Table of Contents

GitHub

Ignore Whitespace

Adding `?w=1` to any diff URL will remove any changes only in whitespace, enabling you to see only that code that has changed.

Read more about GitHub secrets.

Adjust Tab Space

Adding `?ts=4` to a diff or file URL will display tab characters as 4 spaces wide instead of the default 8. The number after `ts` can be adjusted to suit your preference. This does not work on Gists, or raw file views, but a Chrome or Opera extension can automate this.

Here is a Go source file before adding `?ts=4` :

…and this is after adding `?ts=4` :

Commit History by Author

To view all commits on a repo by author add `?author={user}` to the URL.

```
https://github.com/rails/rails/commits/master?author=dhh
```

Read more about the differences between commits views.

Cloning a Repository

When cloning a repository the `.git` can be left off the end.

```
$ git clone https://github.com/tiimgreen/github-cheat-sheet
```

Read more about the Git `clone` command.

Branch

Compare all Branches to Another Branch

If you go to the repo's Branches page, next to the Commits button:

```
https://github.com/{user}/{repo}/branches
```

… you would see a list of all branches which are not merged into the main branch.

From here you can access the compare page or delete a branch with a click of a button.

Comparing Branches

To use GitHub to compare branches, change the URL to look like this:

```
https://github.com/{user}/{repo}/compare/{range}
```

Where `{range} = master...4-1-stable`

For example:

```
https://github.com/rails/rails/compare/master...4-1-stable
```

`{range}` can be changed to things like:

```
https://github.com/rails/rails/compare/master@{1.day.ago}...master
https://github.com/rails/rails/compare/master@{2014-10-04}...master
```

Dates are in the format `YYYY-MM-DD`

Branches can also be compared in `diff` and `patch` views:

```
https://github.com/rails/rails/compare/master...4-1-stable.diff
https://github.com/rails/rails/compare/master...4-1-stable.patch
```

Read more about comparing commits across time.

Compare Branches across Forked Repositories

To use GitHub to compare branches across forked repositories, change the URL to look like this:

```
https://github.com/{user}/{repo}/compare/{foreign-user}:{branch}...{own-branch}
```

For example:

```
https://github.com/rails/rails/compare/byroot:master...master
```

Gists

Gists are an easy way to work with small bits of code without creating a fully fledged repository.

Add `.pibb` to the end of any Gist URL (like this) in order to get the *HTML only* version suitable for embedding in any other site.

Gists can be treated as a repository so they can be cloned like any other:

```
$ git clone https://gist.github.com/tiimgreen/10545817
```

This means you also can modify and push updates to Gists:

```
$ git commit
$ git push
Username for 'https://gist.github.com':
Password for 'https://tiimgreen@gist.github.com':
```

However, Gists do not support directories. All files need to be added to the repository root.

Read more about creating Gists.

Git.io

Git.io is a simple URL shortener for GitHub.

You can also use it via pure HTTP using Curl:

```
$ curl -i http://git.io -F "url=https://github.com/..."
HTTP/1.1 201 Created
Location: http://git.io/abc123

$ curl -i http://git.io/abc123
HTTP/1.1 302 Found
Location: https://github.com/...
```

Read more about Git.io.

Keyboard Shortcuts

When on a repository page, keyboard shortcuts allow you to navigate easily.

- Pressing `t` will bring up a file explorer.
- Pressing `w` will bring up the branch selector.
- Pressing `s` will focus the search field for the current repository. Pressing Backspace to delete the "This repository" pill changes the field to search all of GitHub.
- Pressing `l` will edit labels on existing Issues.
- Pressing `y` **when looking at a file** (e.g. `https://github.com/tiimgreen/github-cheat-sheet/blob/master/README.md`) will change your URL to one which, in effect, freezes the page you are looking at. If this code changes, you will still be able to see what you saw at that current time.

To see all of the shortcuts for the current page press `?`:

Read more about search syntax you can use.

Line Highlighting in Repositories

Either adding `#L52` to the end of a code file URL or simply clicking the line number will highlight that line number.

It also works with ranges, e.g. `#L53-L60`, to select ranges, hold `shift` and click two lines:

```
https://github.com/rails/rails/blob/master/activemodel/lib/active_model.rb#L53-L60
```

Closing Issues via Commit Messages

If a particular commit fixes an issue, any of the keywords `fix/fixes/fixed`, `close/closes/closed` or `resolve/resolves/resolved`, followed by the issue number, will close the issue once it is committed to the master branch.

```
$ git commit -m "Fix screwup, fixes #12"
```

This closes the issue and references the closing commit.

Read more about closing Issues via commit messages.

Cross-Link Issues

If you want to link to another issue in the same repository, simply type hash `#` then the issue number, and it will be auto-linked.

To link to an issue in another repository, `{user}/{repo}#ISSUE_NUMBER` e.g. `tiimgreen/toc#12`.

Locking Conversations

Pull Requests and Issues can now be locked by owners or collaborators of the repo.

This means that users who are not collaborators on the project will no longer be able to comment.

Read more about locking conversations.

CI Status on Pull Requests

If set up correctly, every time you receive a Pull Request, Travis CI will build that Pull Request just like it would every time you make a new commit. Read more about how to get started with Travis CI.

Read more about the commit status API.

Filters

Both issues and pull requests allow filtering in the user interface.

For the Rails repo: https://github.com/rails/rails/issues, the following filter is built by selecting the label "activerecord":

`is:issue label:activerecord`

But, you can also find all issues that are NOT labeled activerecord:

`is:issue -label:activerecord`

Additionally, this also works for pull requests:

`is:pr -label:activerecord`

Github has tabs for displaying open or closed issues and pull requests but you
can also see merged pull requests. Just put the following in the filter:

`is:merged`

Read more about searching issues.

Finally, github now allows you to filter by the Status API's status.

Pull requests with only successful statuses:

`status:success`

Read more about searching on the Status API.

Syntax Highlighting in Markdown Files

For example, to syntax highlight Ruby code in your Markdown files write:

~~~
```ruby
require 'tabbit'
table = Tabbit.new('Name', 'Email')
table.add_row('Tim Green', 'tiimgreen@gmail.com')
puts table.to_s
```
~~~

This will produce:

```ruby
require 'tabbit'
table = Tabbit.new('Name', 'Email')
table.add_row('Tim Green', 'tiimgreen@gmail.com')
puts table.to_s
```

GitHub uses Linguist to perform language detection and syntax highlighting. You can find out which keywords are valid by perusing the languages YAML file.

Read more about GitHub Flavored Markdown.

Emojis

Emojis can be added to Pull Requests, Issues, commit messages, repository descriptions, etc. using `:name_of_emoji:` .

The full list of supported Emojis on GitHub can be found at emoji-cheat-sheet.com or scotch-io/All-Github-Emoji-Icons.

A handy emoji search engine can be found at emoji.muan.co.

The top 5 used Emojis on GitHub are:

1. `:shipit:`
2. `:sparkles:`
3. `:-1:`
4. `:+1:`
5. `:clap:`

Images/GIFs

Images and GIFs can be added to comments, READMEs etc.:

```
![Alt Text](http://www.sheawong.com/wp-content/uploads/2013/08/keephatin.gif)
```

Raw images from the repo can be used by calling them directly.:

```
![Alt Text](https://github.com/{user}/{repo}/raw/master/path/to/image.gif)
```

All images are cached on GitHub, so if your host goes down, the image will remain available.

Embedding Images in GitHub Wiki

There are multiple ways of embedding images in Wiki pages. There's the standard Markdown syntax (shown above). But there's also a syntax that allows things like specifying the height or width of the image:

```
[[ http://www.sheawong.com/wp-content/uploads/2013/08/keephatin.gif | height = 100px ]]
```

Which produces:

Quick Quoting

When on a comment thread and you want to quote something someone previously said, highlight the text and press `r`, this will copy it into your text box in the block-quote format.

Read more about quick quoting.

Pasting Clipboard Image to Comments

(Works on Chrome browsers only)

After taking a screenshot and adding it to the clipboard (mac: `cmd-ctrl-shift-4`), you can simply paste (`cmd-v / ctrl-v`) the image into the comment section and it will be auto-uploaded to github.

Read more about issue attachments.

Quick Licensing

When creating a repository, GitHub gives you the option of adding in a pre-made license:

You can also add them to existing repositories by creating a new file through the web interface. When the name `LICENSE` is typed in you will get an option to use a template:

Also works for `.gitignore`.

Read more about open source licensing.

Task Lists

In Issues and Pull requests check boxes can be added with the following syntax (notice the space):

```
- [ ] Be awesome
- [ ] Prepare dinner
  - [ ] Research recipe
  - [ ] Buy ingredients
  - [ ] Cook recipe
- [ ] Sleep
```

When they are clicked, they will be updated in the pure Markdown:

```
- [x] Be awesome
- [ ] Prepare dinner
  - [x] Research recipe
  - [x] Buy ingredients
  - [ ] Cook recipe
- [ ] Sleep
```

Read more about task lists.

Task Lists in Markdown Documents

In full Markdown documents **read-only** checklists can now be added using the following syntax:

```
- [ ] Mercury
- [x] Venus
- [x] Earth
  - [x] Moon
- [x] Mars
  - [ ] Deimos
  - [ ] Phobos
```

- [ ] Mercury
- [x] Venus
- [x] Earth
  - [x] Moon
- [x] Mars
  - [ ] Deimos
  - [ ] Phobos

Read more about task lists in markdown documents.

Relative Links

Relative links are recommended in your Markdown files when linking to internal content.

```
[Link to a header](#awesome-section)
[Link to a file](docs/readme)
```

Absolute links have to be updated whenever the URL changes (e.g. repository renamed, username changed, project forked). Using relative links makes your documentation easily stand on its own.

Read more about relative links.

Metadata and Plugin Support for GitHub Pages

Within Jekyll pages and posts, repository information is available within the `site.github` namespace, and can be displayed, for example, using `{{ site.github.project_title }}`.

The Jemoji and jekyll-mentions plugins enable emoji and @mentions in your Jekyll posts and pages to work just like you'd expect when interacting with a repository on GitHub.com.

Read more about repository metadata and plugin support for GitHub Pages.

Viewing YAML Metadata in your Documents

Many blogging websites, like Jekyll with GitHub Pages, depend on some YAML-formatted metadata at the beginning of your post. GitHub will render this metadata as a horizontal table, for easier reading

Read more about viewing YAML metadata in your documents.

Rendering Tabular Data

GitHub supports rendering tabular data in the form of `.csv` (comma-separated) and `.tsv` (tab-separated) files.

Read more about rendering tabular data.

Rendering PDF

GitHub supports rendering PDF:

Read more about rendering PDF.

Revert a Pull Request

After a pull request is merged, you may find it does not help anything or it was a bad decision to merge the pull request.

You can revert it by clicking the **Revert** button on the right side of a commit in the pull request page to create a pull request with reverted changes to this specific pull request.

Read more about reverting pull requests

Diffs

Rendered Prose Diffs

Commits and pull requests, including rendered documents supported by GitHub (e.g. Markdown), feature *source* and *rendered* views.

Click the “rendered” button to see the changes as they’ll appear in the rendered document. Rendered prose view is handy when you’re adding, removing, and editing text:

Read more about rendered prose diffs.

Diffable Maps

Any time you view a commit or pull request on GitHub that includes geodata, GitHub will render a visual representation of what was changed.

Read more about diffable maps.

Expanding Context in Diffs

Using the *unfold* button in the gutter of a diff, you can reveal additional lines of context with a click. You can keep clicking *unfold* until you've revealed the whole file, and the feature is available anywhere GitHub renders diffs.

Read more about expanding context in diffs.

Diff or Patch of Pull Request

You can get the diff of a Pull Request by adding a `.diff` or `.patch` extension to the end of the URL. For example:

```
https://github.com/tiimgreen/github-cheat-sheet/pull/15
https://github.com/tiimgreen/github-cheat-sheet/pull/15.diff
https://github.com/tiimgreen/github-cheat-sheet/pull/15.patch
```

The `.diff` extension would give you this in plain text:

```
diff --git a/README.md b/README.md
index 88fcf69..8614873 100644
--- a/README.md
+++ b/README.md
@@ -28,6 +28,7 @@ All the hidden and not hidden features of Git and GitHub. This cheat sheet was i
 - [Merged Branches](#merged-branches)
 - [Quick Licensing](#quick-licensing)
 - [TODO Lists](#todo-lists)
+- [Relative Links](#relative-links)
 - [.gitconfig Recommendations](#gitconfig-recommendations)
     - [Aliases](#aliases)
     - [Auto-correct](#auto-correct)
@@ -381,6 +382,19 @@ When they are clicked, they will be updated in the pure Markdown:
 - [ ] Sleep

(...)
```

Rendering and diffing images

GitHub can display several common image formats, including PNG, JPG, GIF, and PSD. In addition, there are several ways to compare differences between versions of those image formats.

Read more about rendering and diffing images.

Hub

Hub is a command line Git wrapper that gives you extra features and commands that make working with GitHub easier.

This allows you to do things like:

```
$ hub clone tiimgreen/toc
```

Check out some more cool commands Hub has to offer.

Contributing Guidelines

Adding a `CONTRIBUTING` file to the root of your repository will add a link to your file when a contributor creates an Issue or opens a Pull Request.

Read more about contributing guidelines.

Octicons

GitHubs icons (Octicons) have now been open sourced.

Read more about GitHub's Octicons

GitHub Resources

Title	Link
GitHub Explore	https://github.com/explore
GitHub Blog	https://github.com/blog
GitHub Help	https://help.github.com/
GitHub Training	https://training.github.com/
GitHub Developer	https://developer.github.com/
Github Education (Free Micro Account and other stuff for students)	https://education.github.com/

GitHub Talks

Title	Link
How GitHub Uses GitHub to Build GitHub	https://www.youtube.com/watch?v=qyz3jkOBbQY
Introduction to Git with Scott Chacon of GitHub	https://www.youtube.com/watch?v=ZDR433b0HJY
How GitHub No Longer Works	https://www.youtube.com/watch?v=gXD1ITW7iZI

Git and GitHub Secrets	https://www.youtube.com/watch?v=Foz9yvMkvlA
More Git and GitHub Secrets	https://www.youtube.com/watch?v=p50xsL-iVgU

Git

Remove All Deleted Files from the Working Tree

When you delete a lot of files using `/bin/rm` you can use the following command to remove them from the working tree and from the index, eliminating the need to remove each one individually:

```
$ git rm $(git ls-files -d)
```

For example:

```
$ git status
On branch master
Changes not staged for commit:
    deleted:    a
    deleted:    c

$ git rm $(git ls-files -d)
rm 'a'
rm 'c'

$ git status
On branch master
Changes to be committed:
    deleted:    a
    deleted:    c
```

Previous Branch

To move to the previous branch in Git:

```
$ git checkout -
# Switched to branch 'master'

$ git checkout -
# Switched to branch 'next'

$ git checkout -
# Switched to branch 'master'
```

Read more about Git branching.

Stripspace

Git Stripspace:

- Strips trailing whitespace
- Collapses newlines
- Adds newline to end of file

A file must be passed when calling the command, e.g.:

```
$ git stripspace < README.md
```

Read more about the Git `stripspace` command.

SSH keys

You can get a list of public ssh keys in plain text format by visiting:

```
https://github.com/{user}.keys
```

e.g. https://github.com/tiimgreen.keys

Read more about accessing public ssh keys.

Checking out Pull Requests

Pull Requests are special branches on the GitHub repository which can be retrieved locally in several ways:

Retrieve a specific Pull Request and store it temporarily in `FETCH_HEAD` for quickly `diff` ing or `merge` ing:

```
$ git fetch origin refs/pull/[PR-Number]/head
```

Acquire all Pull Request branches as local remote branches by refspec:

```
$ git fetch origin '+refs/pull/*/head:refs/remotes/origin/pr/*'
```

Or setup the remote to fetch Pull Requests automatically by adding these corresponding lines in your repository's `.git/config` :

```
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = git@github.com:tiimgreen/github-cheat-sheet.git
```

```
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = git@github.com:tiimgreen/github-cheat-sheet.git
    fetch = +refs/pull/*/head:refs/remotes/origin/pr/*
```

For Fork-based Pull Request contributions, it's useful to `checkout` a remote branch representing the Pull Request and create a local branch from it:

```
$ git checkout pr/42 pr-42
```

Or should you work on more repositories, you can globally configure fetching pull requests in the global git config instead.

```
git config --global --add remote.origin.fetch "+refs/pull/*/head:refs/remotes/origin/pr/*"
```

This way, you can use the following short commands in all your repositories:

```
git fetch origin
```

```
git checkout pr/42
```

Read more about checking out pull requests locally.

Empty Commits

Commits can be pushed with no code changes by adding `--allow-empty`:

```
$ git commit -m "Big-ass commit" --allow-empty
```

Some use-cases for this (that make sense), include:

- Annotating the start of a new bulk of work or a new feature.
- Documenting when you make changes to the project that aren't code related.
- Communicating with people using your repository.
- The first commit of a repo, as the first commit cannot be rebased later: `git commit -m "init repo" --allow-empty`.

Styled Git Status

Running:

```
$ git status
```

Produces:

By adding `-sb`:

```
$ git status -sb
```

This is produced:

Read more about the Git `status` command.

Styled Git Log

Running:

```
$ git log --all --graph --pretty=format:'%Cred%h%Creset -%C(auto)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative
```

Produces:

Credit to Palesz

This can be aliased using the instructions found here.

Read more about the Git `log` command.

Git Query

A Git query allows you to search all your previous commit messages and find the most recent one matching the query.

```
$ git show :/query
```

Where `query` (case-sensitive) is the term you want to search, this then finds the last one and gives details on the lines that were changed.

```
$ git show :/typo
```

Press `q` to quit.

Git Grep

Git Grep will return a list of lines matching a pattern.

Running:

```
$ git grep aliases
```

will show all the files containing the string *aliases*.

Press `q` to quit.

You can also use multiple flags for more advanced search. For example:

- `-e` The next parameter is the pattern (e.g. regex)
- `--and`, `--or` and `--not` Combine multiple patterns.

Use it like this:

```
$ git grep -e pattern --and -e anotherpattern
```

Read more about the Git `grep` command.

Merged Branches

Running:

```
$ git branch --merged
```

Will give you a list of all branches that have been merged into your current branch.

Conversely:

```
$ git branch --no-merged
```

Will give you a list of branches that have not been merged into your current branch.

Read more about the Git `branch` command.

Fixup and Autosquash

If there is something wrong with a previous commit (can be one or more from HEAD), for example `abcde`, run the following command after you've amended the problem:

```
$ git commit --fixup=abcde
$ git rebase abcde^ --autosquash -i
```

Read more about the Git `commit` command.
Read more about the Git `rebase` command.

Web Server for Browsing Local Repositories

Use the Git `instaweb` command to instantly browse your working repository in `gitweb`. This command is a simple script to set up `gitweb` and a web server for browsing the local repository.

```
$ git instaweb
```

Opens:

Read more about the Git `instaweb` command.

Git Configurations

Your `.gitconfig` file contains all your Git configurations.

Aliases

Aliases are helpers that let you define your own git calls. For example you could set `git a` to run `git add --all`.

To add an alias, either navigate to `~/.gitconfig` and fill it out in the following format:

```
[alias]
  co = checkout
  cm = commit
  p = push
  # Show verbose output about tags, branches or remotes
  tags = tag -l
  branches = branch -a
  remotes = remote -v
```

…or type in the command-line:

```
$ git config --global alias.new_alias git_function
```

For example:

```
$ git config --global alias.cm commit
```

For an alias with multiple functions use quotes:

```
$ git config --global alias.ac 'add -A . && commit'
```

Some useful aliases include:

Alias	Command	What to Type
`git cm`	`git commit`	`git config --global alias.cm commit`
`git co`	`git checkout`	`git config --global alias.co checkout`
`git ac`	`git add . -A` `git commit`	`git config --global alias.ac '!git add -A && git commit'`
`git st`	`git status -sb`	`git config --global alias.st 'status -sb'`

`git tags`	`git tag -l`	`git config --global alias.tags 'tag -l'`				
`git branches`	`git branch -a`	`git config --global alias.branches 'branch -a'`				
`git cleanup`	`git branch —merged	grep -v '*'	xargs git branch -d`	`git config —global alias.cleanup "!git branch —merged	grep -v '*'	xargs git branch -d"`
`git remotes`	`git remote -v`	`git config --global alias.remotes 'remote -v'`				
`git lg`	`git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --`	`git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --"`				

Some Aliases are taken from @mathiasbynens dotfiles: https://github.com/mathiasbynens/dotfiles/blob/master/.gitconfig

Auto-Correct

If you type `git comit` you will get this:

```
$ git comit -m "Message"
# git: 'comit' is not a git command. See 'git --help'.

# Did you mean this?
#   commit
```

To call `commit` when `comit` is typed, just enable auto-correct:

```
$ git config --global help.autocorrect 1
```

So now you will get this:

```
$ git comit -m "Message"
# WARNING: You called a Git command named 'comit', which does not exist.
# Continuing under the assumption that you meant 'commit'
# in 0.1 seconds automatically...
```

Color

To add more color to your Git output:

```
$ git config --global color.ui 1
```

Read more about the Git `config` command.

Git Resources

Title	Link
Official Git Site	http://git-scm.com/
Official Git Video Tutorials	http://git-scm.com/videos
Code School Try Git	http://try.github.com/
Introductory Reference & Tutorial for Git	http://gitref.org/
Official Git Tutorial	http://git-scm.com/docs/gittutorial
Everyday Git	http://git-scm.com/docs/everyday
Git Immersion	http://gitimmersion.com/
Ry's Git Tutorial	http://rypress.com/tutorials/git/index.html
Git for Designers	http://hoth.entp.com/output/git_for_designers.html
Git for Computer Scientists	http://eagain.net/articles/git-for-computer-scientists/
Git Magic	http://www-cs-students.stanford.edu/~blynn/gitmagic/
GitHub Training Kit	http://training.github.com/kit
Git Visualization Playground	http://onlywei.github.io/explain-git-with-d3/#freeplay

Git Books

Title	Link
Pragmatic Version Control Using Git	http://www.pragprog.com/titles/tsgit/pragmatic-version-control-using-git
Pro Git	http://git-scm.com/book
Git Internals PluralSight	https://github.com/pluralsight/git-internals-pdf
Git in the Trenches	http://cbx33.github.com/gitt/
Version Control with Git	http://www.amazon.com/Version-Control-Git-collaborative-development/dp/1449316387
Pragmatic Guide to Git	http://www.pragprog.com/titles/pg_git/pragmatic-guide-to-git
Git: Version Control for Everyone	http://www.packtpub.com/git-version-control-for-everyone/book