How To Use Git: A Reference Guide

Posted October 10, 2018 13.7k GIT OPEN SOURCE DEVELOPMENT

16

By: Lisa Tagliaferri

Git Cheat Sheet

Introduction

Teams of developers and open-source software maintainers typically manage their projects through Git, a distributed version control system that supports collaboration.

This cheat sheet-style guide provides a quick reference to commands that are useful for working and collaborating in a Git repository. To install and configure Git, be sure to read "How To Contribute to Open Source: Getting Started with Git."

How to Use This Guide:

- This guide is in cheat sheet format with self-contained command-line snippets.
- Jump to any section that is relevant to the task you are trying to complete.
- When you see `highlighted text` in this guide's commands, keep in mind that this text should refer to the commits and files in *your own* repository.

Set Up and Initialization

Check your Git version with the following command, which will also confirm that Git is installed.

```
$ git --version
```

You can initialize your current working directory as a Git repository with `init`.

```
$ git init
```

To copy an existing Git repository hosted remotely, you'll use `git clone` with the repo's URL or server location (in the latter case you will use `ssh`).

```
$ git clone https://www.github.com/username/repo-name
```

Show your current Git directory's remote repository.

```
$ git remote
```

For a more verbose output, use the `-v` flag.

```
$ git remote -v
```

Add the Git upstream, which can be a URL or can be hosted on a server (in the latter case, connect with `ssh`).

```
$ git remote add upstream https://www.github.com/username/repo-name
```

Staging

When you've modified a file and have marked it to go in your next commit, it is considered to be a staged file.

Check the status of your Git repository, including files added that are not staged, and files that are staged.

```
$ git status
```

To stage modified files, use the `add` command, which you can run multiple times before a commit. If you make subsequent changes that you want included in the next commit, you must run `add` again.

You can specify the specific file with `add`.

```
$ git add my_script.py
```

With `.` you can add all files in the current directory including files that begin with a `.`.

```
$ git add .
```

You can remove a file from staging while retaining changes within your working directory with `reset`.

```
$ git reset my_script.py
```

Committing

Once you have staged your updates, you are ready to commit them, which will record changes you have made to the repository.

To commit staged files, you'll run the `commit` command with your meaningful commit message so that you can track commits.

```
$ git commit -m "Commit message"
```

You can condense staging all tracked files with committing them in one step.

```
$ git commit -am "Commit message"
```

If you need to modify your commit message, you can do so with the `--amend` flag.

```
$ git commit --amend -m "New commit message"
```

Branches

A branch in Git is a movable pointer to one of the commits in the repository, it allows you to isolate work and manage feature development and integrations. You can learn more about branches by reading the Git documentation.

List all current branches with the `branch` command. An asterisk (`*`) will appear next to your currently active branch.

```
$ git branch
```

Create a new branch. You will remain on your currently active branch until you switch to the new one.

```
$ git branch new-branch
```

Switch to any existing branch and check it out into your current working directory.

```
$ git checkout another-branch
```

You can consolidate the creation and checkout of a new branch by using the `-b` flag.

```
$ git checkout -b new-branch
```

Rename your branch name.

```
$ git branch -m current-branch-name new-branch-name
```

Merge the specified branch's history into the one you're currently working in.

```
$ git merge branch-name
```

Abort the merge, in case there are conflicts.

```
$ git merge --abort
```

You can also select a particular commit to merge with `cherry-pick` with the string that references the specific commit.

```
$ git cherry-pick f7649d0
```

When you have merged a branch and no longer need the branch, you can delete it.

```
$ git branch -d branch-name
```

If you have not merged a branch to master, but are sure you want to delete it, you can **force** delete a branch.

```
$ git branch -D branch-name
```

Collaborate and Update

To download changes from another repository, such as the remote upstream, you'll use `fetch`.

```
$ git fetch upstream
```

Merge the fetched commits.

```
$ git merge upstream/master
```

Push or transmit your local branch commits to the remote repository branch.

```
$ git push origin master
```

Fetch and merge any commits from the tracking remote branch.

```
$ git pull
```

Inspecting

Display the commit history for the currently active branch.

```
$ git log
```

Show the commits that changed a particular file. This follows the file regardless of file renaming.

```
$ git log --follow my_script.py
```

Show the commits that are on one branch and not on the other. This will show commits on `a-branch` that are not on `b-branch`.

```
$ git log a-branch..b-branch
```

Look at reference logs (`reflog`) to see when the tips of branches and other references were last updated within the repository.

```
$ git reflog
```

Show any object in Git via its commit string or hash in a more human-readable format.

```
$ git show de754f5
```

Show Changes

The `git diff` command shows changes between commits, branches, and more. You can read more fully about it through the Git documentation.

Compare modified files that are on the staging area.

```
$ git diff --staged
```

Display the diff of what is in `a-branch` but is not in `b-branch`.

```
$ git diff a-branch..b-branch
```

Show the diff between two specific commits.

```
$ git diff 61ce3e6..e221d9c
```

Stashing

Sometimes you'll find that you made changes to some code, but before you finish you have to begin working on something else. You're not quite ready to commit the changes you have made so far, but you don't want to lose your work. The `git stash` command will allow you to save your local modifications and revert back to the working directory that is in line with the most recent `HEAD` commit.

Stash your current work.

```
$ git stash
```

See what you currently have stashed.

```
$ git stash list
```

Your stashes will be named `stash@{0}`, `stash@{1}`, and so on.

Show information about a particular stash.

```
$ git stash show stash@{0}
```

To bring the files in a current stash out of the stash while still retaining the stash, use `apply`.

```
$ git stash apply stash@{0}
```

If you want to bring files out of a stash, and no longer need the stash, use `pop`.

```
$ git stash pop stash@{0}
```

If you no longer need the files saved in a particular stash, you can `drop` the stash.

```
$ git stash drop stash@{0}
```

If you have multiple stashes saved and no longer need to use any of them, you can use `clear` to remove them.

```
$ git stash clear
```

Ignoring Files

If you want to keep files in your local Git directory, but do not want to commit them to the project, you can add these files to your `.gitignore` file so that they do not cause conflicts.

Use a text editor such as nano to add files to the `.gitignore` file.

```
$ nano .gitignore
```

To see examples of `.gitignore` files, you can look at GitHub's `.gitignore` template repo.

Rebasing

A rebase allows us to move branches around by changing the commit that they are based on. With rebasing, you can squash or reword commits.

You can start a rebase by either calling the number of commits you have made that you want to rebase (`5` in the case below).

```
$ git rebase -i HEAD~5
```

Alternatively, you can rebase based on a particular commit string or hash.

```
$ git rebase -i 074a4e5
```

Once you have squashed or reworded commits, you can complete the rebase of your branch on top of the latest version of the project's upstream code.

```
$ git rebase upstream/master
```

To learn more about rebasing and updating, you can read How To Rebase and Update a Pull Request, which is also applicable to any type of commit.

Resetting

Sometimes, including after a rebase, you need to reset your working tree. You can reset to a particular commit, and **delete all changes**, with the following command.

```
$ git reset --hard 1fc6665
```

To force push your last known non-conflicting commit to the origin repository, you'll need to use `--force`.

> **Warning**: Force pushing to master is often frowned upon unless there is a really important reason for doing it. Use sparingly when working on your own repositories, and work to avoid this when you're collaborating.

```
$ git push --force origin master
```

To remove local untracked files and subdirectories from the Git directory for a clean working branch, you can use `git clean`.

```
$ git clean -f -d
```

If you need to modify your local repository so that it looks like the current upstream master (that is, there are too many conflicts), you can perform a hard reset.

> **Note**: Performing this command will make your local repository look exactly like the upstream. Any commits you have made but that were not pulled into the upstream **will be destroyed**.

```
$ git reset --hard upstream/master
```

Conclusion

This guide covers some of the more common Git commands you may use when managing repositories and collaborating on software.

You can learn more about open-source software and collaboration in our Introduction to Open Source tutorial series:

- How To Contribute to Open Source: Getting Started with Git
- How To Create a Pull Request on GitHub
- How To Rebase and Update a Pull Request
- How To Maintain Open-Source Software Projects

There are many more commands and variations that you may find useful as part of your work with Git. To learn more about all of your available options, you can run:

```
$ git --help
```

To receive useful information. You can also read more about Git and look at Git's documentation from the official Git website.

By: Lisa Tagliaferri

Tutorial Series

An Introduction to Open Source

Open-source projects that are hosted in public repositories benefit from contributions made by the broader developer community, and are typically managed through Git. This tutorial series will guide you through selecting an open-source project to contribute to, making a pull request to a Git repository through the command line, and taking steps to follow up on your pull request.

Show Tutorials

Related Tutorials

How To Rebase and Update a Pull Request

How To Create a Pull Request on GitHub

How To Ensure Code Quality with SonarQube on Ubuntu 18.04

How to Manually Set Up a Prisma Server on Ubuntu 18.04

How To Display Data from the DigitalOcean API with React

0 Comments