Git basics

Git is a free and open source version control system, originally created by Linus Torvalds in 2005. Unlike older centralized version control systems such as SVN and CVS, Git is distributed: every developer has the full history of their code repository locally. This makes the initial clone of the repository slower, but subsequent operations such as commit, blame, diff, merge, and log dramatically faster.

Git also has excellent support for branching, merging, and rewriting repository history, which has lead to many innovative and powerful workflows and tools. Pull requests are one such popular tool that allow teams to collaborate on Git branches and efficiently review each others code. Git is the most widely used version control system in the world today and is considered the modern standard for software development.

How Git works

Here is a basic overview of how Git works:

1. Create a "repository" (project) with a git hosting tool (like Bitbucket)
2. Copy (or clone) the repository to your local machine
3. Add a file to your local repo and "commit" (save) the changes
4. "Push" your changes to your master branch
5. Make a change to your file with a git hosting tool and commit
6. "Pull" the changes to your local machine
7. Create a "branch" (version), make a change, commit the change
8. Open a "pull request" (propose changes to the master branch)
9. "Merge" your branch to the master branch

Git configuration

The first thing we need to know is that there's three places that Git stores configuration information and depends on how widely we want those configurations to apply. The first and largest is System level configuration, that is configurations that ought to apply to every user of this computer. Now each user, of course, can overwrite it with their own, but these are going to be default configurations. Now in truth you won't use this very often. It's much easier and much better to set it up on a per user basis, but I just want you to know that it exists.

Now the most useful place to store configurations is going to be User level configurations. These are going to apply to a single user, which most of us, most of the time are working on a single-user machine and so we can have our single-user configuration. On Unix, that's going to be in your home directory, inside a file called .gitconfig.

And then the third place that we can store configurations is on a project-by-project basis. So in a single project we can have configurations that apply only to that project. Now most configurations, you are probably don't want to use from project-to-project, and you want to put them in the User configuration.

Git gives us some commands that we can use to make editing these configurations easy. For all three of them, it's going to be git config, followed by a modifier that tells at what level we want to do the configuration, and then followed by the configuration itself that we want to do. So if we want to do a system-wide configuration, then it's --system at the end, if it's User level then that's --global, don't let that throw you, global doesn't mean system, it means global to the user, and then if we don't have any modifier then it's just on a single project basis.

So that's what the command looks like, and that tells it how to direct it, what are the kinds of configurations that we can set? Well, let's set a few. So here I am in my command line. You'll want to make sure that you're there as well, it doesn't matter where we're located because we are going to be doing configuration that's global. So as long as we are logged in as a user, we will be making edits to our global user file. So we can call git config and then --global, and then whatever we want to configure. Well, the first thing we need to configure is our user.name, so user.name, space, and then in double quotes put in your name.

So obviously you've used your name and not my name and then when you are done hit Return and Git added it to the config file.

**git config -- global user.name “Your Name”**

Let's add another one, git config --global user.email, space, and then you can put in your email address there. I am going to put in just a fake one here rather than give out my real email address, someone@nowhere.com but you will put in your real one.

**git config -- global user.email “your@email.com”**

Now if you want to see these configurations, you can say git config --list, and it will then pair it back to the list of configurations that it has set for you, or if you want to look at a specific one, you can say user.name, and it returns just that one, same thing, user.email, and it returns the email address.

**git config – list**

**git config user.name**

**git config user.email**

So we can take a look, we have the ability to set them, and we have the ability to retrieve them and look at what they are.

One good way to take a look at it real quick with Unix is just to use the cat command, cat .gitconfig.

**cat .gitconfig**

That's what it actually looks like inside there. So this is the minimum that you need to configure and to start working with git. You will come back to this file and add other configurations over time.

Basic Git commands

|  |  |  |
| --- | --- | --- |
| Git task | Notes | Git commands |
| [**Tell Git who you are**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-config) | Configure the author name and email address to be used with your commits.  Note that Git [strips some characters](http://stackoverflow.com/questions/26159274/is-it-possible-to-have-a-trailing-period-in-user-name-in-git/26219423#26219423) (for example trailing periods) from user.name. | git config --global user.name "Sam Smith"  git config --global user.email sam@example.com |
| [**Create a new local repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-init) |  | git init |
| [**Check out a repository**](https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone) | Create a working copy of a local repository: | git clone /path/to/repository |
| For a remote server, use: | git clone username@host:/path/to/repository |
| [**Add files**](https://www.atlassian.com/git/tutorials/saving-changes#git-add) | Add one or more files to staging (index): | git add <filename>  git add \* |
| [**Commit**](https://www.atlassian.com/git/tutorials/saving-changes#git-commit) | Commit changes to head (but not yet to the remote repository): | git commit -m "Commit message" |
| Commit any files you've added with git add, and also commit any files you've changed since then: | git commit -a |
| [**Push**](https://www.atlassian.com/git/tutorials/syncing#git-push) | Send changes to the master branch of your remote repository: | git push origin master |
| [**Status**](https://www.atlassian.com/git/tutorials/inspecting-a-repository#git-status) | List the files you've changed and those you still need to add or commit: | git status |
| [**Connect to a remote repository**](https://www.atlassian.com/git/tutorials/syncing#git-remote) | If you haven't connected your local repository to a remote server, add the server to be able to push to it: | git remote add origin <server> |
| List all currently configured remote repositories: | git remote -v |
| [**Branches**](https://www.atlassian.com/git/tutorials/using-branches) | Create a new branch and switch to it: | git checkout -b <branchname> |
| Switch from one branch to another: | git checkout <branchname> |
| List all the branches in your repo, and also tell you what branch you're currently in: | git branch |
| Delete the feature branch: | git branch -d <branchname> |
| Push the branch to your remote repository, so others can use it: | git push origin <branchname> |
| Push all branches to your remote repository: | git push --all origin |
| Delete a branch on your remote repository: | git push origin :<branchname> |
| [**Update from the remote repository**](https://www.atlassian.com/git/tutorials/syncing) | Fetch and merge changes on the remote server to your working directory: | git pull |
| To merge a different branch into your active branch: | git merge <branchname> |
| View all the merge conflicts:  View the conflicts against the base file:  Preview changes, before merging: | git diff  git diff --base <filename>  git diff <sourcebranch> <targetbranch> |
| After you have manually resolved any conflicts, you mark the changed file: | git add <filename> |
| **Tags** | You can use tagging to mark a significant changeset, such as a release: | git tag 1.0.0 <commitID> |
| CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using: | git log |
| Push all tags to remote repository: | git push --tags origin |
| [**Undo local changes**](https://www.atlassian.com/git/tutorials/undoing-changes) | If you mess up, you can replace the changes in your working tree with the last content in head:  Changes already added to the index, as well as new files, will be kept. | git checkout -- <filename> |
| Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this: | git fetch origin  git reset --hard origin/master |
| **Search** | Search the working directory for foo(): | git grep "foo()" |

GIT HUB

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

This tutorial teaches you GitHub essentials like repositories, branches, commits, and Pull Requests. You’ll create your own repository and learn GitHub’s Pull Request workflow, a popular way to create and review code.

No coding necessary

To complete this tutorial, you need a GitHub.com account and Internet access. You don’t need to know how to code, use the command line, or install Git (the version control software GitHub is built on).

Tip: Open this guide in a separate browser window (or tab) so you can see it while you complete the steps in the tutorial.

Create a Repository

A repository is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a README, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. It also offers other common options such as a license file.

Your repository can be a place where you store ideas, resources, or even share and discuss things with others.

To create a new repository

In the upper right corner, next to your avatar or identicon, click and then select New repository.

Name your repository hello-world.

Write a short description.

Select Initialize this repository with a README.



Click Create repository.

Make and commit changes

On GitHub, saved changes are called commits. Each commit has an associated commit message, which is a description explaining why a particular change was made. Commit messages capture the history of your changes, so other contributors can understand what you’ve done and why.

Make and commit changes

1. Click the README.md file.
2. Click the pencil icon in the upper right corner of the file view to edit.
3. In the editor, write a bit about yourself.
4. Write a commit message that describes your changes.
5. Click Commit changes button.



Open a Pull Request

Pull Requests are the heart of collaboration on GitHub. When you open a pull request, you’re proposing your changes and requesting that someone review and pull in your contribution and merge them into their branch. Pull requests show diffs, or differences, of the content from both branches. The changes, additions, and subtractions are shown in green and red.

As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

By using GitHub’s @mention system in your pull request message, you can ask for feedback from specific people or teams, whether they’re down the hall or 10 time zones away.

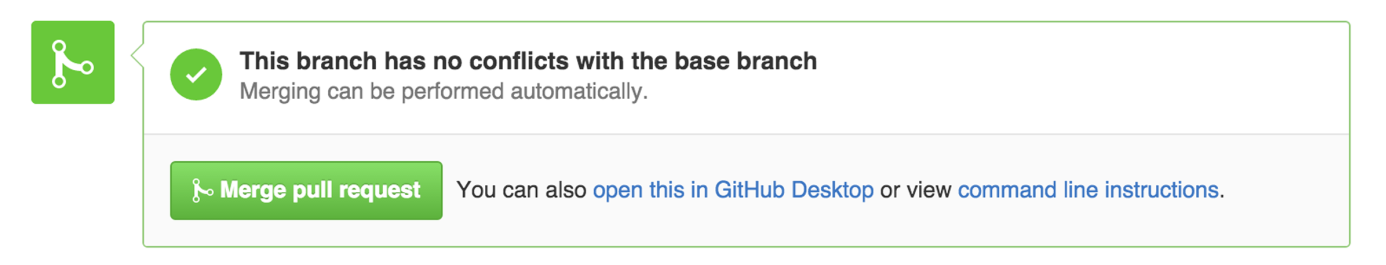
You can even open pull requests in your own repository and merge them yourself. It’s a great way to learn the GitHub Flow before working on larger projects.

Merge your Pull Request

In this final step, it’s time to bring your changes together – merging your readme-edits branch into the master branch. Click the green Merge pull request button to merge the changes into master.

Click Confirm merge.

Go ahead and delete the branch, since its changes have been incorporated, with the Delete branch button in the purple box.



More reference you can find on <https://guides.github.com/features/wikis/>