GitHub-Guides-引导

# GitHub官网

## 网址

官网：<https://github.com/>

进入我的页面：[https://github.com/**zhaoxiyuan**](https://github.com/zhaoxiyuan)

紧接着添加Repository名字就直接进入该仓库：

[https://github.com/zhaoxiyuan/**MyFirstDubbo**](https://github.com/zhaoxiyuan/MyFirstDubbo)

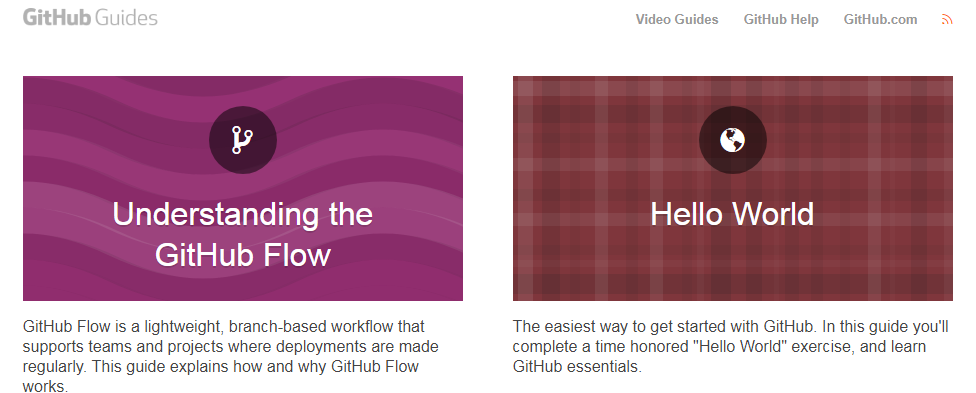
## What is GitHub?

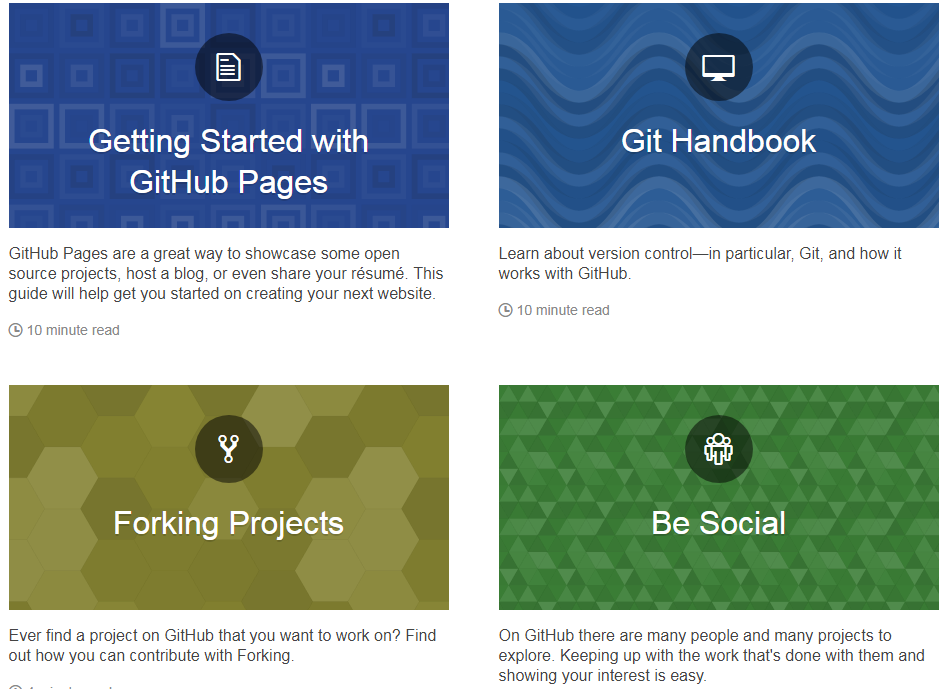
GitHub is **a code hosting platform** for version control and collaboration. It lets you and others work together on projects from anywhere.

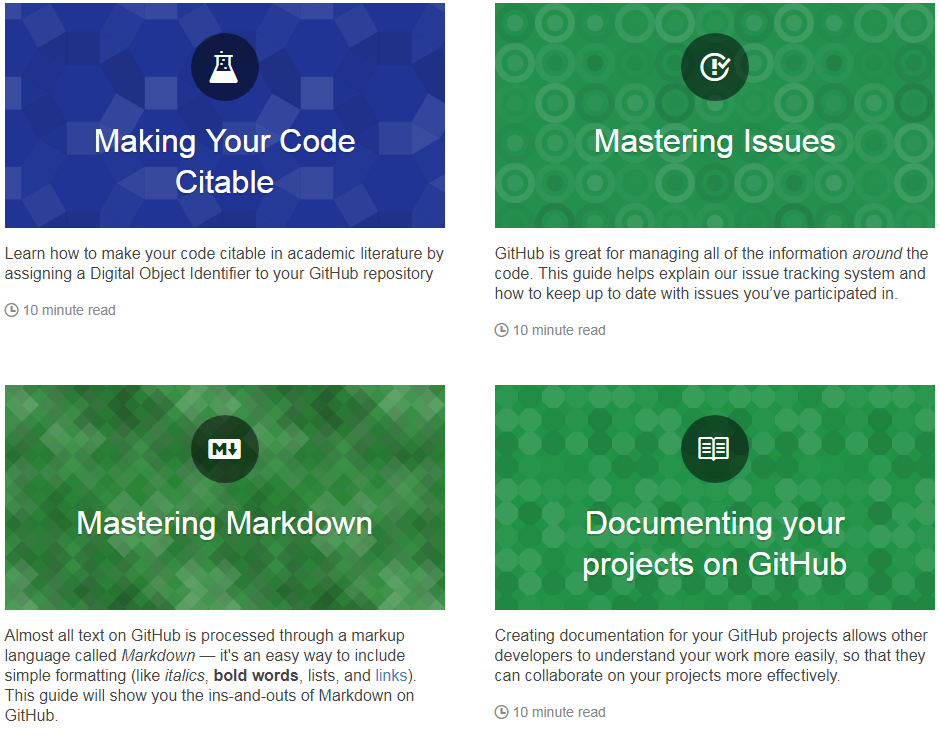
# GitHub-Guides

**GitHub-Guides**：<https://guides.github.com/>

Video：<https://www.youtube.com/githubguides>







# About pull requests

**pull request**是一个request，它的目的是**让别人pull你的东西**。这个词一开始挺费解，不是自己的pull 请求，是请求别人pull自己的东西。pull的执行者是别人，request是自己。pull request 是自己请求别人拉自己的东西。

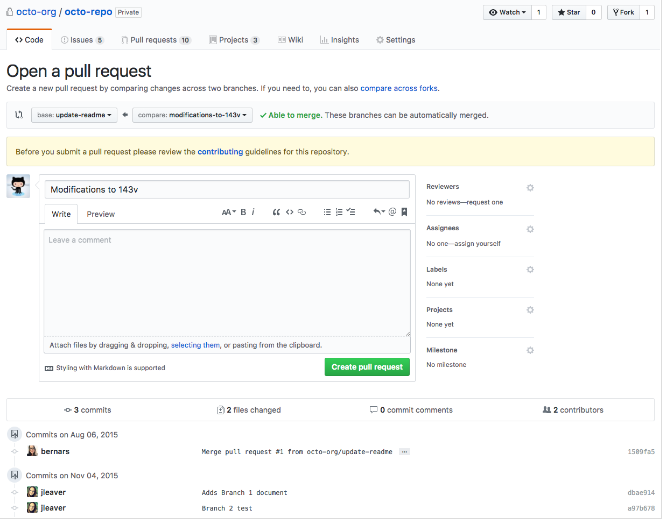
**Pull requests** let you tell others about changes you've pushed to a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before the changes are merged into the repository.

**Note:** When working with pull requests, keep the following in mind:

If you're working in the shared repository model, we recommend that you use a topic branch for your pull request. While you can send pull requests from any branch or commit, with a topic branch you can push follow-up commits if you need to update your proposed changes.

When pushing commits to a pull request, don't force push. Force pushing can corrupt your pull request.

After initializing a pull request, you'll see a review page that shows a high-level overview of the changes between your branch (the compare branch) and the repository's base branch. You can add a summary of the proposed changes, review the changes made by commits, add labels, milestones, and assignees, and @mention individual contributors or teams. For more information, see "**Creating a pull request.**" <https://help.github.com/articles/creating-a-pull-request/>



Once you've created a pull request, you can push commits from your topic branch to add them to your existing pull request. These commits will appear in chronological order within your pull request and the changes will be visible in the "Files changed" tab.

Other contributors can review your proposed changes, add review comments, contribute to the pull request discussion, and even add commits to the pull request.

After you're happy with the proposed changes, **you can merge the pull request**. If you're working in a shared repository model, the proposed changes will be merged from the head branch to the base branch that was specified in the pull request. You can close corresponding issues using a keyword in your pull request or commit message. For more information, see "**Closing issues using keywords.**" <https://help.github.com/articles/closing-issues-using-keywords/>

**merge the pull request** : <https://help.github.com/articles/merging-a-pull-request/>

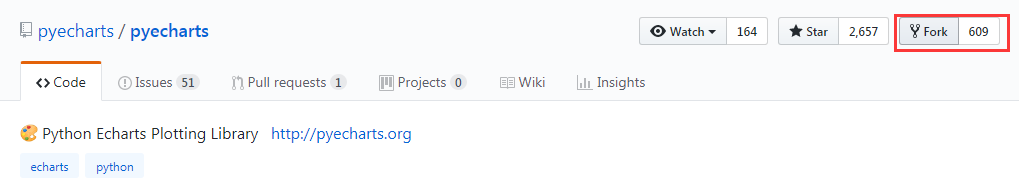
# About forks

**fork** 英 [fɔ:k] 美 [fɔ:rk]

n.餐叉;叉（挖掘用的园艺工具）;叉状物;（自行车或摩托车的）车叉子

vi.**分叉**;分歧 vt.走岔路;叉，耙

GitHub上的**Fork**按钮：



**Fork** your own copy of …… to your account.

说白了：就是把别人的**repository** 复制到自己的账户下。

**A fork is a copy of a repository** that you manage. **Forks** let you make changes to a project **without affecting the original repository**. You can fetch updates from or submit changes to the original repository **with pull requests**.

**Any user or organization on GitHub can fork a repository**. Forking a repository is similar to copying another repository, with two major differences:

### You can use a pull request to suggest changes from your fork to the original repository, also known as the upstream repository.

### You can bring changes from the upstream repository to your local fork by synchronizing your fork with the upstream repository.

**Deleting a fork does not delete the original upstream repository**. In fact, you can make any changes you want to your fork--add collaborators, rename files, generate GitHub Pages--with no effect on the original.

In open source projects, forks are often used to iterate on ideas or changes before they are offered back to the upstream repository for everyone to benefit from. When you make changes in your fork and open a pull request that compares your work to the upstream repository, **you can give anyone with push access to the upstream repository permission to push changes to your pull request branch.** This speeds up collaboration by allowing repository maintainers the ability to make commits or run tests locally to your pull request branch from a fork before merging.

**Private forks inherit the permissions structure of the upstream or parent repository.** For example, if the upstream repository is private and gives **read/write** access to a team, then the same team will have read/write access to any forks of the private upstream repository. This helps owners of private repositories maintain control over their code.

# Creating a pull request

<https://help.github.com/articles/creating-a-pull-request/>

Create a pull request to propose and collaborate on changes to a repository. These changes are proposed in a branch, which ensures that the **master** branch only contains finished and approved work.

**Pull requests** can only be opened if there are differences between your branch and the upstream branch. You can specify which branch you'd like to merge your changes into when you create your pull request.

If you don't have write access to the repository where you'd like to create a pull request, you must create a fork, or copy, of the repository first. For more information, see "**Creating a pull request from a fork**" and "**About forks**."

<https://help.github.com/articles/creating-a-pull-request-from-a-fork/>

<https://help.github.com/articles/about-forks/>

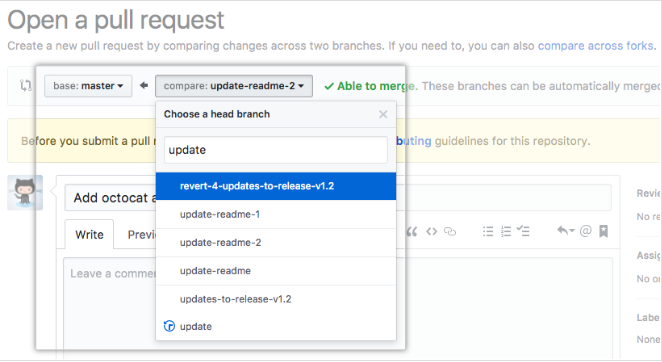
**Note**: To open a pull request in a public repository, you must have write access to the head or the source branch or, for organization-owned repositories, you must be a member of the organization that owns the repository to open a pull request.

You can close corresponding issues using a keyword in your pull request or commit message. For more information, see "**Closing issues using keywords**."

## Changing the branch range and destination repository

**By default, pull requests are based on the parent repository's default branch.**

If the default parent repository isn't correct, you can change **both the parent repository and the branch with the drop-down lists**. You can also swap your head and base branches with the drop-down lists to establish diffs between reference points. References here must be branch names in your GitHub repository.



When thinking about branches, remember that the base branch is where changes should be applied, the head branch contains what you would like to be applied.

When you change the base repository, you also change notifications for the pull request. Everyone that can push to the base repository will receive an email notification and see the new pull request in their dashboard the next time they sign in.

When you change any of the information in the branch range, the Commit and Files changed preview areas will update to show your new range.

Tips:

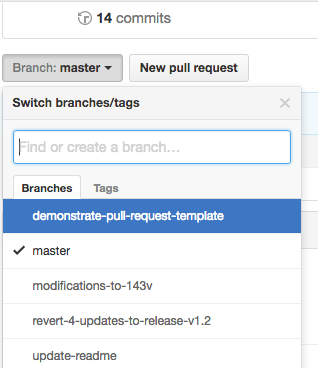
**Using the compare view, you can set up comparisons across any timeframe.**

Project maintainers can add a pull request template for a repository. Templates include prompts for information the project's maintainers prefer to see in the body of a pull request.

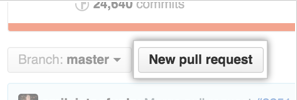
## Creating the pull request

### On GitHub, navigate to the main page of the repository.

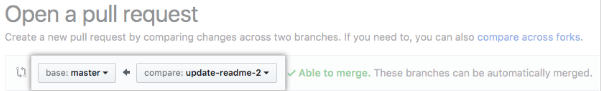
### In the "Branch" menu, choose the branch that contains your commits.



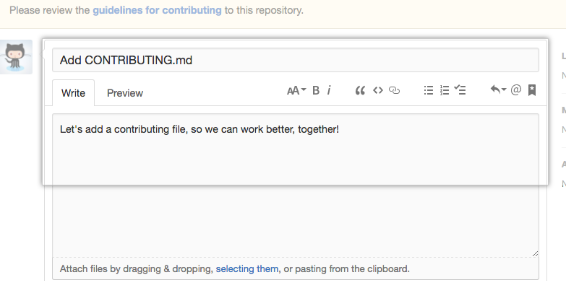
### To the right of the Branch menu, click New pull request.



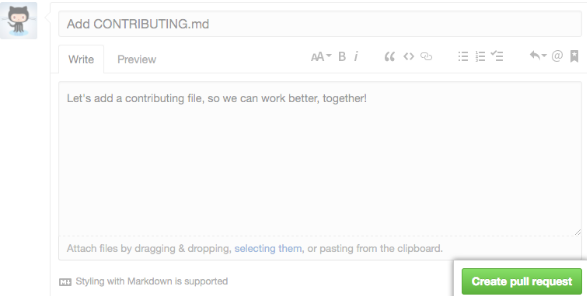
### Use the base branch dropdown menu to select the branch you'd like to merge your changes into, then use the compare branch drop-down menu to choose the topic branch you made your changes in.



### Type a title and description for your pull request.



### Click Create pull request.



# Deleting and restoring branches in a pull request—删除和恢复分支

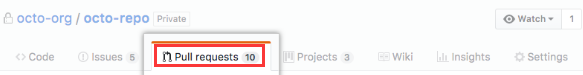
<https://help.github.com/articles/deleting-and-restoring-branches-in-a-pull-request/>

## Deleting a branch

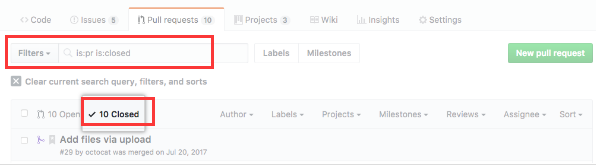
If you have write access in a repository, **you can delete branches that are associated with closed or merged pull requests**. You cannot delete branches that are associated with open pull requests.

### On GitHub, navigate to the main page of the repository.

### Under your repository name, click  **Pull requests**.

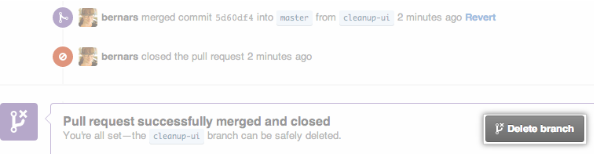


### Click Closed to see a list of closed pull requests.



### In the list of pull requests, click the pull request that's associated with the branch that you want to delete.

### Near the bottom of the pull request, click ****Delete branch****.

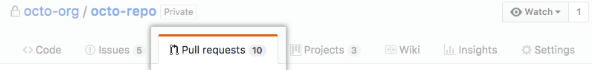


## Restoring a deleted branch

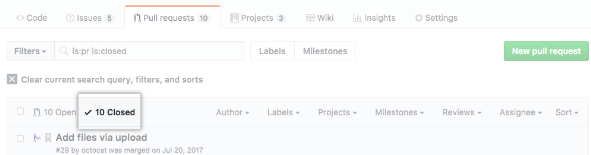
You can restore the head branch of a closed pull request.

### On GitHub, navigate to the main page of the repository.

### Under your repository name, click Pull requests.



### Click Closed to see a list of closed pull requests.



### In the list of pull requests, click the pull request that's associated with the branch that you want to restore.

### Restore deleted branch buttonNear the bottom of the pull request, click Restore branch.



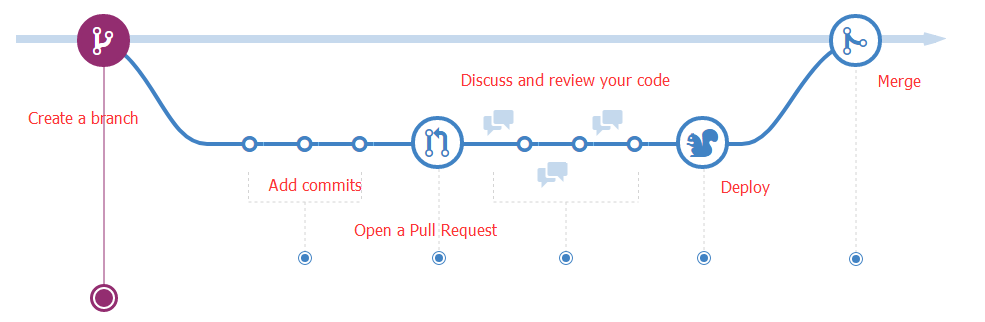
# GitHub-Guides- Understanding the GitHub Flow

## Understanding the GitHub Flow

GitHub Flow is a **lightweight**, **branch-based** workflow that supports teams and projects where deployments are made regularly. This guide explains how and why GitHub Flow works.

## 六步

Create a branch🡺**Add commits**🡺**Open a Pull Request**🡺**Discuss and review your code**🡺**Deploy**🡺**Merge**



### Create a branch

When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.

When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the **master** branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

**ProTip**

Branching is a core concept in Git, and the entire GitHub Flow is based upon it. There's only one rule: anything in the master branch is always deployable.

Because of this, it's extremely important that your new branch is created off of master when working on a feature or a fix. Your branch name should be descriptive (**e.g., refactor-authentication, user-content-cache-key, make-retina-avatars**), so that others can see what is being worked on.

### Add commits

Once your branch has been created, it's time **to start making changes**. Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature branch.

**Commits also create a transparent history of your work** that others can follow to understand what you've done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. **Furthermore, each commit is considered a separate unit of change.** This lets you roll back changes if a bug is found, or if you decide to head in a different direction.

**ProTip**

Commit messages are important, especially since Git tracks your changes and then displays them as commits once they're pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.

### Open a Pull Request

**Pull Requests initiate discussion about your commits.** Because they're tightly integrated with the underlying Git repository, anyone can see exactly what changes would be merged if they accept your request.

You can open a Pull Request at any point during the development process: when you have little or no code but want to share some screenshots or general ideas, when you're stuck and need help or advice, or when you're ready for someone to review your work. 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 ten time zones away.

**ProTip**

Pull Requests are useful for contributing to open source projects and for managing changes to shared repositories. If you're using a Fork & Pull Model, Pull Requests provide a way to notify project maintainers about the changes you'd like them to consider. If you're using a Shared Repository Model, Pull Requests help start code review and conversation about proposed changes before they're merged into the master branch.

### Discuss and review your code

Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation.

You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. GitHub will show your new commits and any additional feedback you may receive in the unified Pull Request view.

**ProTip**

Pull Request comments are written in Markdown, so you can embed images and emoji, use pre-formatted text blocks, and other lightweight formatting.

### Deploy

With GitHub, you can deploy from a branch for final testing in production before merging to master.

Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them in production. If your branch causes issues, you can roll it back by deploying the existing master into production.

### Merge

Now that your changes have been verified in production, it is time to merge your code into the master branch.

**Once merged**, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.

**ProTip**

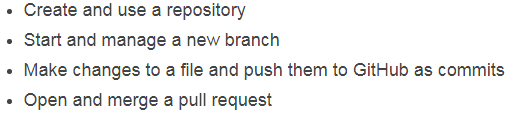
By incorporating certain keywords into the text of your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issues are also closed. For example, entering the phrase Closes #32 would close issue number 32 in the repository. For more information, check out our help article.

# GitHub-Guides-Hello World

<https://guides.github.com/activities/hello-world/>

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

**GitHub essentials** : **repositories**, **branches**, **commits**, and **Pull Requests**

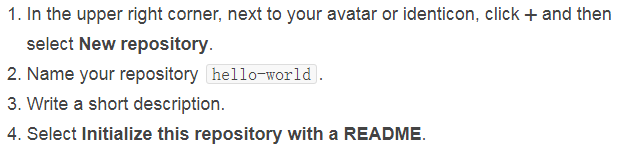


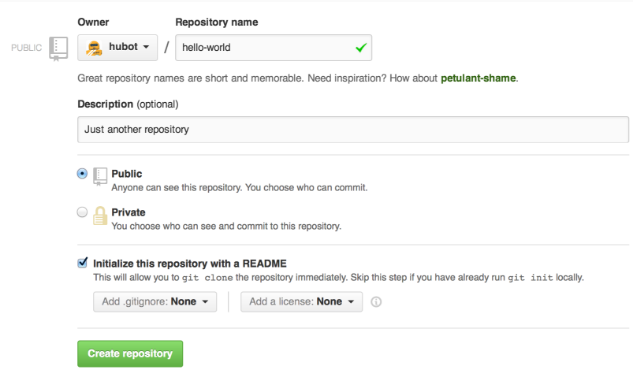
## Step 1. Create a Repository:创建一个仓库

**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.

**To create a new repository**

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





Click **Create repository**.

## Step 2. Create a Branch：创建一个分支

### 分支介绍

**Branching** is the way to work on different versions of a repository at one time.

By default your repository has one branch named **master** which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to **master**.

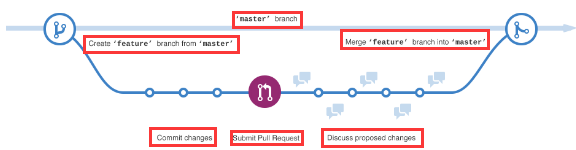
When you create a branch off the **master** branch, you’re **making a copy, or snapshot**, of master as it was at that point in time. If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.

This diagram shows:

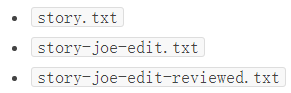
The **master** branch

A new branch called **feature** (because we’re doing ‘feature work’ on this branch)

The journey that **feature** takes before it’s merged into **master.**



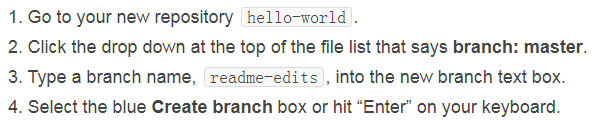
Have you ever saved **different versions of a file**? Something like:

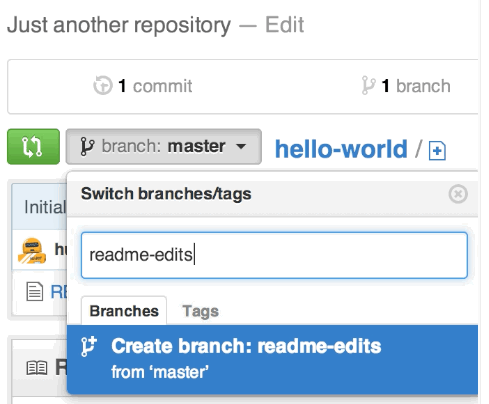


**Branches** accomplish similar goals in GitHub repositories.

Here at GitHub, our developers, writers, and designers use branches for keeping bug fixes and feature work separate from our **master** (production) branch. When a change is ready, they merge their branch into **master**.

### To create a new branch





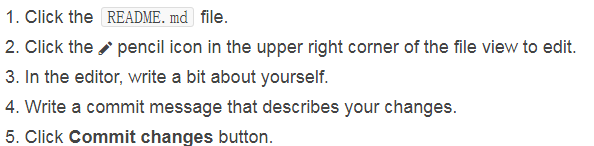
Now you have two branches, **master** and **readme-edits**. They look exactly the same, but not for long! Next we’ll add our changes to the new branch.

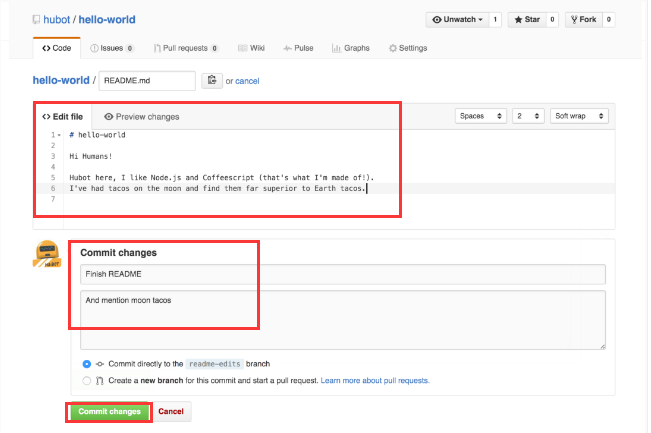
## Step 3. Make and commit changes

Bravo! Now, you’re on the code view for your **readme-edits** branch, which is a copy of master. Let’s make some edits.

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





These changes will be made to just the README file on your **readme-edits branch**, so now this branch contains content that’s different from **master**.

## Step 4. Open a Pull Request

Nice edits! Now that you have changes in a branch off of **master**, you can 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.**

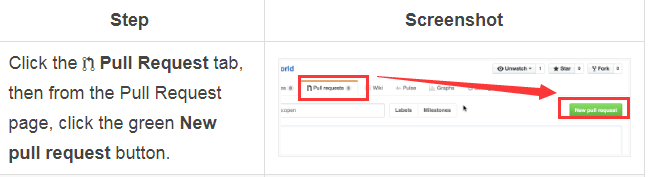
As soon as you make a commit, you can open a pull request and start a discussion, even before the code is finished.

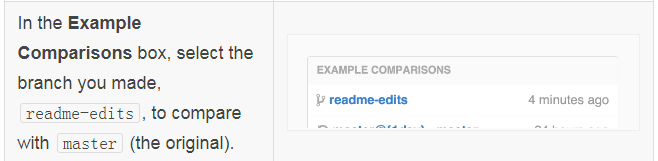
By using GitHub’s **@mention system** in your pull request message, you can ask for feedback from specific people or teams, whether they’re down the hall or 10 time zones away.

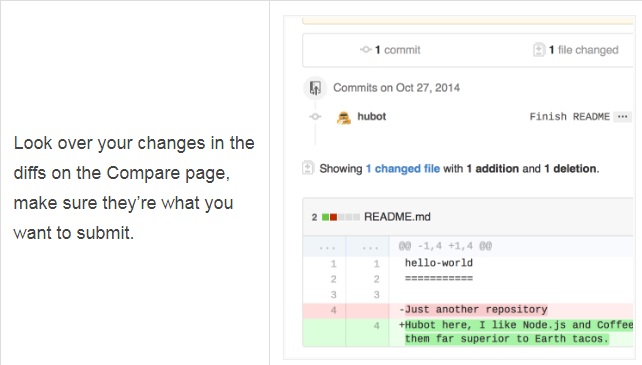
You can even open pull requests in your own repository and merge them yourself. It’s a great way to learn the GitHub Flow before working on larger projects.

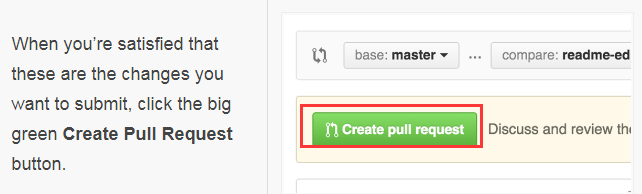
### Open a Pull Request for changes to the README

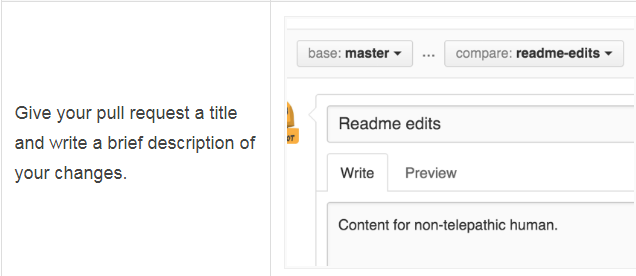
Click on the image for a larger version









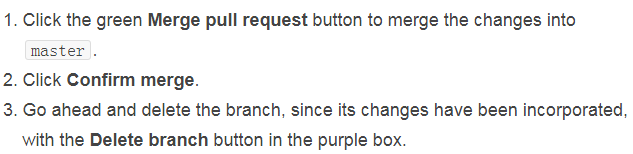


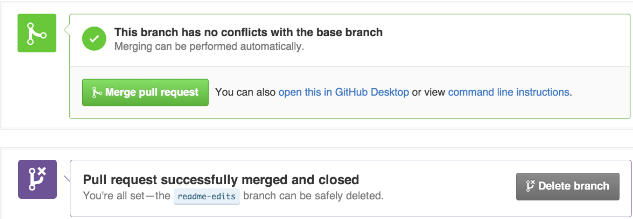
When you’re done with your message, click **Create pull request**!

**Tip**: You can use **emoji** and **drag and drop images and gifs** onto comments and Pull Requests.

## Step 5. 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**.





**Celebrate!**

## 总结:Conclusions

By completing this tutorial, you’ve learned **to create a project and make a pull request on GitHub!**

Here’s what you accomplished in this tutorial:

### Created an open source repository

### Started and managed a new branch

### Changed a file and committed those changes to GitHub

### Opened and merged a Pull Request

### Take a look at your GitHub profile and you’ll see your new contribution squares!

To learn more about the power of Pull Requests, we recommend reading the **GitHub Flow Guide**. You might also visit **GitHub Explore** and get involved in an Open Source project :octocat:

Tip: Check out our other ***Guides***, **YouTube Channel** and **On-Demand Training** for more on how to get started with GitHub.

***Guides:*** [***https://guides.github.com/***](https://guides.github.com/)

**YouTube Channel** : <https://youtube.com/githubguides>

**On-Demand Training:** [**https://services.github.com/on-demand/**](https://services.github.com/on-demand/)

# GitHub-Guides-Git Handbook

## What is Version Control?

### What’s a version control system?

**A version control system, or VCS**, tracks the history of changes as people and teams collaborate on projects together. As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time. Developers can review project history to find out:

Which changes were made?

Who made the changes?

When were the changes made?

Why were changes needed?

### What’s a distributed version control system?

Git is an example of a distributed version control system (DVCS) commonly used for open source and commercial software development. DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes. Unlike once popular centralized version control systems, DVCSs like Git don’t need a constant connection to a central repository. Developers can work anywhere and collaborate asynchronously from any time zone.

Without version control, team members are subject to redundant tasks, slower timelines, and multiple copies of a single project. To eliminate unnecessary work, Git and other VCSs give each contributor a unified and consistent view of a project, surfacing work that’s already in progress. Seeing a transparent history of changes, who made them, and how they contribute to the development of a project helps team members stay aligned while working independently.

### Why Git?

According to the latest **Stack Overflow developer survey**, more than 70 percent of developers use Git, making it **the most-used VCS in the world**. Git is commonly used for **both open source and commercial software development**, with significant benefits for individuals, teams and businesses.

**Git lets developers see the entire timeline of their changes, decisions, and progression of any project in one place**. From the moment they access the history of a project, the developer has all the context they need to understand it and start contributing.

**Developers work in every time zone. With a DVCS like Git, collaboration can happen any time while maintaining source code integrity.** Using branches, developers can safely propose changes to production code.

**Businesses using Git can break down communication barriers between teams and keep them focused on doing their best work.** Plus, Git makes it possible to align experts across a business to collaborate on major projects.

## Repositories--What’s a repository?

**A repository, or Git project**, encompasses the entire collection of files and folders associated with a project, along with each file’s revision history. The file history appears as snapshots in time called commits, and the commits exist as a linked-list relationship, and can be organized into multiple lines of development called branches. **Because Git is a DVCS, repositories are self-contained units and anyone who owns a copy of the repository can access the entire codebase and its history.** Using the command line or other ease-of-use interfaces, a git repository also allows for: interaction with the history, cloning, creating branches, committing, merging, comparing changes across versions of code, and more.

**Working in repositories keeps development projects organized and protected.** Developers are encouraged to fix bugs, or create fresh features, without fear of derailing mainline development efforts. Git facilitates this through the use of topic branches: lightweight pointers to commits in history that can be easily created and deprecated when no longer needed.

Through platforms like GitHub, Git also provides more opportunities for project transparency and collaboration. Public repositories help teams work together to build the best possible final product.

## Basic Git--Basic Git commands

To use Git, developers use specific commands to **copy, create, change, and combine code**. These commands can be executed directly from the command line or by using an application like **GitHub Desktop** or Git Kraken. Here are some common commands for using Git:

**GitHub Desktop ：** [**https://desktop.github.com/**](https://desktop.github.com/)

### git init

***git init*** initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.

### git clone

***git clone*** creates a local copy of a project that already exists remotely. The clone includes all the project’s files, history, and branches.

### git add

***git add*** stages a change. Git tracks changes to a developer’s codebase, but it’s necessary to stage and take a snapshot of the changes to include them in the project’s history. This command performs staging, the first part of that two-step process. Any changes that are staged will become a part of the next snapshot and a part of the project’s history. Staging and committing separately gives developers complete control over the history of their project without changing how they code and work.

### git commit

***git commit*** saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that’s been staged with git add will become a part of the snapshot with git commit.

### git status

***git status*** shows the status of changes as untracked, modified, or staged.

### git branch

***git branch*** shows the branches being worked on locally.

### git merge

***git merge*** merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the master branch for deployment.

### git pull

***git pull*** updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.

### git push

***git push*** updates the remote repository with any commits made locally to a branch.

Explore more Git commands

Learn more from **a full reference guide to Git commands.** <https://git-scm.com/docs>

***Explore more Git commands***

For a detailed look at Git practices, the videos below show how to get the most out of some Git commands.

Working locally：<https://www.youtube.com/watch?v=rBbbOouhI-s&index=2&list=PLg7s6cbtAD17Gw5u8644bgKhgRLiJXdX4>

git status：<https://www.youtube.com/watch?v=SxmveNrZb5k&list=PLg7s6cbtAD17Gw5u8644bgKhgRLiJXdX4&index=3>

Two-step commits： <https://www.youtube.com/watch?v=Vb0Ghkkc2hk&index=4&list=PLg7s6cbtAD17Gw5u8644bgKhgRLiJXdX4>

**git pull and git push**: <https://www.youtube.com/watch?v=-uQHV9GOA0w&index=5&list=PLg7s6cbtAD17Gw5u8644bgKhgRLiJXdX4>

## GitHub

### How GitHub fits in

**GitHub** is **a Git hosting repository** that provides developers with tools to ship better code through command line features, issues (threaded discussions), pull requests, code review, or the use of a collection of free and for-purchase apps in the GitHub Marketplace. With collaboration layers like the GitHub flow, a community of 15 million developers, and an ecosystm with hundreds of integrations, GitHub changes the way software is built.

### How GitHub works

**GitHub** builds collaboration directly into the developement process. Work is organized into repositories, where devlopers can outlines requirements or direction and set expectations for team members. Then, **using the GitHub flow, developers simply create a branch to work on updates, commit changes to save them, open a pull request to propose and discuss changes, and merge pull requests once everyone is on the same page.**

### The GitHub flow

The GitHub flow is a lightweight, branch-based workflow built around core Git commands used by teams around the globe—including ours.

The GitHub flow has six steps, each with distinct benefits when implemented:

**Create a branch**: Topic branches created from the canonical deployment branch (usually master) allow teams to contribute to many parallel efforts. Short-lived topic branches, in particular, keep teams focused and results in quick ships.

**Add commits**: Snapshots of development efforts within a branch create safe, revertible points in the project’s history.

**Open a pull request**: Pull requests publicize a project’s ongoing efforts and set the tone for a transparent development process.

**Discuss and review code**: Teams participate in code reviews by commenting, testing, and reviewing open pull requests. Code review is at the core of an open and participatory culture.

**Merge:** Upon clicking merge, GitHub automatically performs the equivalent of a local ‘git merge’ operation. GitHub also keeps the entire branch development history on the merged pull request.

**Deploy**: Teams can choose the best release cycles or incorporate continuous integration tools and operate with the assurance that code on the deployment branch has gone through a robust workflow.

**Learn more about the GitHub flow**

Developers can find more information about the GitHub flow in the resources provided below.

**Interactive guide**：<https://guides.github.com/introduction/flow/>

**GitHub Flow video**：

<https://www.youtube.com/watch?v=47E-jcuQz5c&index=1&list=PLg7s6cbtAD17Gw5u8644bgKhgRLiJXdX4>

### GitHub and the command line

For developers new to the command line, the GitHub Training team has put together a series of tutorials on Git commands to guide the way. Sometimes, a series of commands can paint a picture of how to use Git:

**Example: Contribute to an existing repository**

# download a repository on GitHub.com to our machine

**git clone https://github.com/me/repo.git**

# change into the `repo` directory

**cd repo**

# create a new branch to store any new changes

**git branch my-branch**

# switch to that branch (line of development)

**git checkout my-branch**

# make changes, for example, edit `file1.md` and `file2.md` using the text editor

# stage the changed files

**git add file1.md file2.md**

# take a snapshot of the staging area (anything that's been added)

**git commit -m "my snapshot"**

# push changes to github

**git push --set-upstream origin my-branch**

**Example: Start a new repository and publish it to GitHub**

# create a new directory, and initialize it with git-specific functions

**git init my-repo**

# change into the `my-repo` directory

**cd my-repo**

# create the first file in the project

**touch README.md**

# git isn't aware of the file, stage it

**git add README.md**

# take a snapshot of the staging area

**git commit -m "add README to initial commit"**

# push changes to github

**git push --set-upstream origin master**

**Example: contribute to an existing branch on GitHub**

# assumption: a project called `repo` already exists on the machine, and a new branch has been **pushed to GitHub.com since the last time changes were made locally**

# change into the `repo` directory

**cd repo**

# update all remote tracking branches, and the currently checked out branch

git pull

# change into the existing branch called `feature-a`

**git checkout feature-a**

# make changes, for example, edit `file1.md` using the text editor

# stage the changed file

**git add file1.md**

# take a snapshot of the staging area

**git commit -m "edit file1"**

# push changes to github

**git push**

### Models for collaborative development

There are two primary ways people collaborate on GitHub:

**Shared repository**

**Fork and pull**

With a shared repository, individuals and teams are explicitly designated as contributors with **read, write, or administrator access**. This simple permission structure, combined with features like protected branches and Marketplace, helps teams progress quickly when they adopt GitHub.

For an open source project, or for projects to which anyone can contribute, managing individual permissions can be challenging, but a fork and pull model allows anyone who can view the project to contribute. A fork is a copy of a project under an developer’s personal account. Every developer has full control of their fork and is free to implement a fix or new feature. Work completed in forks is either kept separate, or is surfaced back to the original project via a pull request. There, maintainers can review the suggested changes before they’re merged. See **the Forking Projects** Guide for more information. <https://guides.github.com/activities/forking/>

### Learn at your own pace

The GitHub team has created a library of **educational videos and guides** to help users continue to develop their skills and build better software.

Beginner projects to explore：

<https://github.com/showcases/great-for-new-contributors>

GitHub video guides：

<https://youtube.com/githubguides>

GitHub on-demand training：

<https://services.github.com/on-demand/>

GitHub training guides：

<https://guides.github.com/>

GitHub training resources：

<https://services.github.com/on-demand/resources/>