You can use Git CLI and GitHub Desktop version independently. Both are installed in my PC

**Git User name: BB-Cyber**

**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.

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.

**No coding necessary**

To complete this tutorial, you need a [GitHub.com account](http://github.com) 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.

**Step 1. 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 hello-world repository can be a place where you store ideas, resources, or even share and discuss things with others.

**To create a new repository**

1. In the upper right corner, next to your avatar or identicon, click and then select **New repository**.
2. Name your repository hello-world.
3. Write a short description.
4. Select **Initialize this repository with a README**.



Click **Create repository**.

**Step 2. Create a Branch**

**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:

* story.txt
* story-joe-edit.txt
* story-joe-edit-reviewed.txt

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**

1. Go to your new repository hello-world.
2. Click the drop down at the top of the file list that says **branch: master**.
3. Type a branch name, readme-edits, into the new branch text box.
4. Select the blue **Create branch** box or hit “Enter” on your keyboard.



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**

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.



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](https://help.github.com/articles/about-writing-and-formatting-on-github/#text-formatting-toolbar) 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*

| **Step** | **Screenshot** |
| --- | --- |
| Click the **Pull Request** tab, then from the Pull Request page, click the green **New pull request** button. | [pr-tab](https://guides.github.com/activities/hello-world/pr-tab.gif) |
| In the **Example Comparisons** box, select the branch you made, readme-edits, to compare with master (the original). | [branch](https://guides.github.com/activities/hello-world/pick-branch.png) |
| Look over your changes in the diffs on the Compare page, make sure they’re what you want to submit. | [diff](https://guides.github.com/activities/hello-world/diff.png) |
| When you’re satisfied that these are the changes you want to submit, click the big green **Create Pull Request** button. | [create-pull](https://guides.github.com/activities/hello-world/create-pr.png) |
| Give your pull request a title and write a brief description of your changes. | [pr-form](https://guides.github.com/activities/hello-world/pr-form.png) |

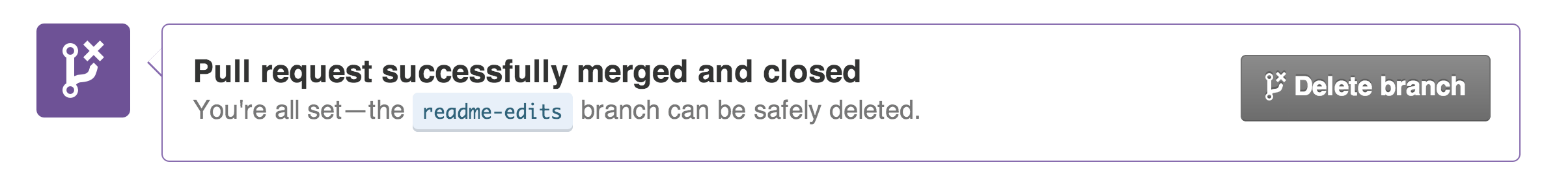
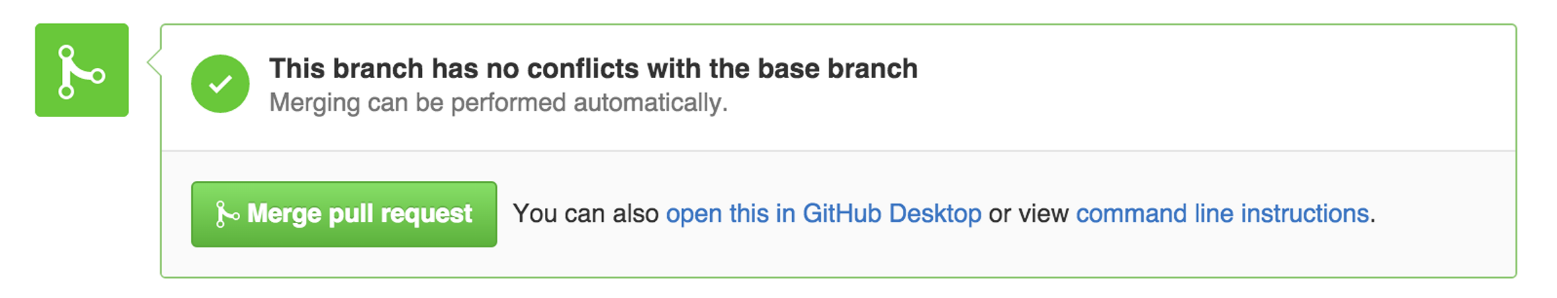
When you’re done with your message, click **Create pull request**!

**Tip**: You can use [emoji](https://help.github.com/articles/basic-writing-and-formatting-syntax/#using-emoji) and [drag and drop images and gifs](https://help.github.com/articles/file-attachments-on-issues-and-pull-requests/) 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.

1. Click the green **Merge pull request** button to merge the changes into master.
2. Click **Confirm merge**.
3. Go ahead and delete the branch, since its changes have been incorporated, with the **Delete branch** button in the purple box.



**Celebrate!**

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](https://help.github.com/articles/viewing-contributions)!

To learn more about the power of Pull Requests, we recommend reading the [GitHub flow Guide](http://guides.github.com/overviews/flow/). You might also visit [GitHub Explore](http://github.com/explore) and get involved in an Open Source project.

**Tip**: Check out our other [Guides](http://guides.github.com), [YouTube Channel](http://youtube.com/githubguides) and [On-Demand Training](https://services.github.com/on-demand/) for more on how to get started with GitHub.

Configure Git

After you [install Git](https://www.linode.com/docs/development/version-control/how-to-install-git-on-mac-and-windows), configure it for first time use using git config, a built-in tool that obtains and sets configuration variables. These configuration variables are located in three different places on a GNU/Linux system:

* /etc/gitconfig - stores the configuration information for all system users and their respective repositories.
* ~/.gitconfig - stores user-specific configuration files on the system.
* .git/config - this is the configuration file of your current working repository.

For a Windows system, the .gitconfig file is located in the $HOME directory of the user’s profile. The full path is C:\Document and Settings\$USER or C:\Users\$USER

After installing Git make sure your username and email address are set correctly. To verify, use the command:

git config --list

If your name and email are not listed in the output, use the following commands to set them manually, replacing examplename and user@example.com:

git config --global user.name examplename

git config --global user.email user@example.com

Set your default text editor, replacing editor-name with your desired editor:

git config --global core.editor editor-name

The output of git config --list should show echo the information you inputted:

$ git config --list

user.name=exampleuser

user.email=user@email.com

core.editor=editor-name

Use Git with a Local Repository

A *repository*, or repo, is a collection of files and folders and the history of their changes. Changes are tracked through *commits*, which are like snapshots of a file at various points in the file’s history. These commits are not automatic, you need to manually stage a commit after each of series of file changes. Commits allow you to refer or revert back to a place in the file’s timeline if there are bugs or errors in your code.

If you have an new or existing project and you want to start using Git to keep track of its changes, run git initfrom the existing project’s directory:

git init

git init creates a new .git subdirectory in the current directory. This is where Git stores your configurations. The git add command tells Git to add a file to the repository and track that file’s changes:

git add filename

After you have added the file, stage a commit and leave a commit message. Commit messages serve as a reminder of the changes that were made to a file:

git commit -m "Initialized a Git repository for this project. Tracking changes to a file."

**Note**

It’s good practice to provide clear and descriptive commit messages for every commit you stage, as this helps collaborators to understand what a commit encompasses.

There may be files or folders in your project directory that you do not wish to include in your Git repository. You can include these files in a .gitignore file, and Git will ignore them. A sample .gitignore file might look like the following:

**.gitignore**

|  |  |
| --- | --- |
| 1  2  3 | .DS\_Store  \*.zip  \_\_doNotInclude\_\_/ |

Basic Git Commands

This table lists basic commands, a description, and an example of the command in use:

| Command | Description | Example |
| --- | --- | --- |
| git add | Add a file to a repository. | git add filename |
| git rm | Remove a file from a repository. | git rm filename |
| git mv | Move or rename a tracked file, directory, or symlink. | git mv file\_from file\_to |
| git branch | List all the local and remote branches. | git branch branchname |
| git commit | Commit all staged objects. Optionally, you can append a message with the -m flag. | git commit -m "updates" |
| git pull | Download all changes from the remote repo and merge them in a specified repo file. | git pull repo refspec |
| git push | Publish the changes to the remote repo. | git push repo |

Branches

*Branches* are used for editing files without disturbing the working portions of a project. The main branch is normally named master and is usually reserved for clean, working code. When making changes to your code, it’s customary to create a new branch and name it after the issue being fixed or the feature being implemented. Because Git keep tracks of file changes, you can jump from branch to branch without overwriting or interfering with other branches in the repo.

The basic options used with the git branch command are:

| Option | Description |
| --- | --- |
| -r | List the remote branches |
| -a | Show both local and remote branches |
| -m | Rename an old branch |
| -d | Delete a branch |
| -r -d | Delete a remote branch |

Example Usage

Consider an application with a single master branch. The author of the application wants to develop a new search feature. They would add a new feature branch:

git branch new-search-feature

Then, they would switch to that branch using the checkout command:

git checkout new-search-feature

Now they can safely develop and commit their changes to this feature branch without altering the working code of the master branch. At any time they could switch back to the master branch:

git checkout master

A shortcut for creating a branch and switching to that branch is to use the -b flag with the checkout command:

git checkout -b new-search-feature

Once the new search feature is finalized, the author of the application can merge the new-search-feature branch into the master branch:

git checkout master

git merge new-search-feature

Now the master branch has the new search feature.

Use Git with a Remote Repository

[GitHub](https://github.com/), [GitLab](https://gitlab.com/), and [Bitbucket](https://bitbucket.org/) all provide ways to store Git repositories remotely and facilitate collaboration. Many of these services also include a number of other features that are vital to content development, including pull requests, continuous integration / continuous delivery pipelines (CI/CD), wikis, and webhooks. If you’d rather use a self-hosted solution, GitLab and [Gogs](https://gogs.io/) offer free locally hosted versions of their software that can easily be managed on a Linode. Check out our guides on [installing GitLab](https://www.linode.com/docs/development/version-control/install-gitlab-on-ubuntu-18-04/) and [installing Gogs](https://www.linode.com/docs/development/version-control/install-gogs-on-debian/) for more information on hosting your own remote repository software. GitHub and Bitbucket also offer paid enterprise versions of their software for local hosting. When discussing remote repositories, usually one of the aforementioned services is being referenced.

This section provides some basic information on navigating remote Git repositories.

To copy every file from a remote repository to your local system, use git clone followed by the remote repo’s URL:

git clone https://github.com/linode/docs.git

You can typically find a remote repository’s URL by clicking on the *Clone* or *Download* buttons of a remote repository’s user interface.

To check the status of the files within the current branch of your repository, use status:

git status

The output of the status command will tell you if any tracked files have been modified.

Use remote to view which remote servers are configured:

git remote

The remote command will display the short names of your remote repositories. If your repository was cloned, you will see a repository called origin. The default name origin comes from the cloned repository. To view more information about the remote repositories, use the command:

git remote -v

Git Remote Repository Commands

Below are some basic commands for working with remote repositories:

| Command | Description |
| --- | --- |
| git remote add [remote-name] [url] | Add a new remote repository. |
| git fetch [repository [refspec]] | Gather all the data from a remote project that you do not have yet. |
| git pull | Obtain and merge a remote branch into your current branch. |
| git push [remote-name] [branch-name] | Move your data from your branch to your server. |
| git remote show [remote-name] | Display information about the remote you specified. |
| git remote rename [old-name] [new-name] | Rename a remote. |
| git remote rm [name] | Remove the remote you specified. |

Example Usage

Continuing the example posed in the previous section, let’s say that someone wants to add a functionality to an application in a remote repository. The first step would be to *fork* that repository. This feature, like cloning, is usually available in the user interface of the remote repository software. Forking creates a remote repository, which is a copy of the forked repository, on the user’s own account. From their the user would clone the fork locally use the clone command discussed above.

git clone https://github.com/username/Spoon-Knife.git

Once the repository is cloned, the user would create a branch for the new feature using either the branch or checkout -b command.

git checkout -b new-special-feature

After the branch is made, the user would make the necessary updates or changes to the codebase, and commit them.

git commit -m "A new special feature"

With the commit staged, the user would then push their changes to their fork of the remote repository:

git push origin new-special-feature

The result of the push command is that now the new feature branch exists on the user’s fork. To contribute those changes to the initial repository, known as the upstream repository, the user must now submit a *pull request*(PR). A PR is a feature that most remote repository software has that allows the user to safely contribute a commit, or series of commits, to an upstream repository. This is accomplished by navigating through the user interface to the pull requests page, selecting the master branch of the upstream repository as the target, and selecting the origin repository’s feature branch as the source. A pull request will be created and the code will be up for review before being merged into the master branch of the upstream repository.

While creating PRs is a healthy standard within the development community, it is not the only way to contribute to a remote repository. You could simply push your commits directly to the upstream repository:

git push my-repo new-special-feature

The downside of this approach is that there is no option to hold the pushed commits for review. However, if you are the only contributor to a project then obviously there will be no one to review your changes, so creating a fork and contributing PRs is not necessary.

If you are collaborating with another developer, it is necessary to be able to retrieve their work. To do so, issue the pull command:

git checkout new-special-feature

git pull

Git will grab the new code from the chosen remote repository branch and merge it into your local branch.