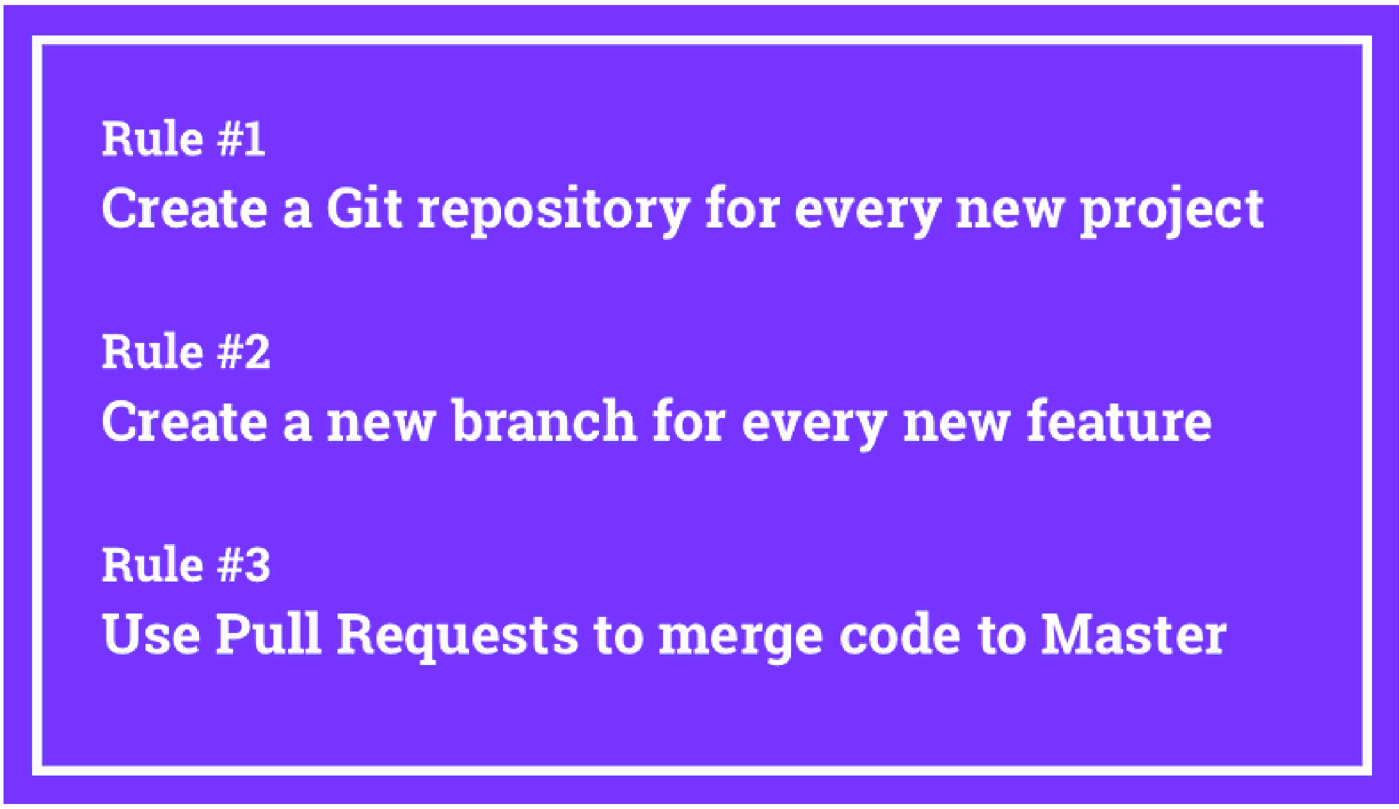
Simple rules to become master in Git and GitHub



Rule #1
Create a Git repository for every new project

Rule #2
Create a new branch for every new feature

Rule #3
Use Pull Requests to merge code to Master

In this article, I will explain why using Git and GitHub every day is so important, especially for those of you who are learning to code. I'll also share and discuss the three simple rules that you can easily follow to become a master Git and GitHub user.

Why are Git and GitHub so important?

If you are learning to code, chances are your most important goal is to eventually get a job as a software developer. In that case, the answer is very simple:

Learning Git and GitHub is incredibly important **because 99% of the companies that can hire you will use Git and GitHub.** Therefore, learning how to work with Git and GitHub make you more hirable and help you differentiate yourself from more junior developers.

What makes senior developers senior is not that they know the syntax of a given language better, but that they have experience working with large and complex projects with real users and business goals.

When you are learning to code, it's hard to get that kind of experience. However, a simple way of getting real-world experience is by using the tools and methodologies used in real-world projects. Git and GitHub are an example of those.

Other things you can do are remote pair programming, contributing to open source, and building professionally-designed websites for your portfolio.

Even if you agree that mastering Git and GitHub will help you get a job, you might still be wondering:

"Why are Git and Github so important for companies?"

The short answer is that Git allows teams to efficiently and effectively contribute code to the same project in an asynchronous way. This empowers teams to collaborate better and thus allows them to solve bigger and more complex problems.

Git, which is a distributed version control system, also provides mechanisms to revert changes, create branches of code, solve merge conflicts, and so on. Those are very useful features that solve specific and common problems that every software team faces every day. And Git is the dominant solution nowadays.

GitHub, on the other hand, is an added layer on top of Git that provides solutions to other specific and common problems such as code reviews, pull requests, issue management/bug tracking, and so on.

Quick note: Even though Git is the go-to version control solution for most companies, GitHub has some strong competitors such as GitLab and Bitbucket. However, if you know how to use GitHub, you won't have any problem working with GitLab or Bitbucket.

Now that you know why it's so important to master Git and Github, it's time to tell you the three simple rules to follow to easily become a professional Git and Github user while you are still learning to code.

How to master Git and Github with 3 simple rules

Just for some additional context, I’m the founder of Microverse, a school for remote software developers that is completely free until you get a job. As part of our 22-week program, we not only teach our students how to code, but we also give them plenty of guidance and structure for them to get real-world experience while in the program.

One of the things we ask our students to do is to follow the three rules you will find below in order to become professional Git and Github users. By the end of the training, working with Git, GitHub, branches, pull requests and code reviews becomes second nature for our students.

Before I go ahead and discuss the three simple rules for mastering Git and Github, please consider completing the following tasks:

1. **If you are not familiar with Git or GitHub yet**, you should complete this awesome tutorial from HubSpot.
2. **If you don't know what the GitHub Flow is**, you should learn about Github Flow since we will use it below.

And now, without much further ado, the three simple rules to master Git and Github while learning how to code…

- **Rule #1**: Create a Git repository for every new project
- **Rule #2**: Create a new branch for every new feature
- **Rule #3**: Use Pull Requests to merge code to Master

Even if you are working on small and simple projects, and even if you are working alone, following those three rules every time you code will make you a Git and GitHub master user very quickly.

Let's briefly break down each one of the rules so you understand what you are supposed to do and why each rule is important.

Rule #1: Create a Git repository for every new project

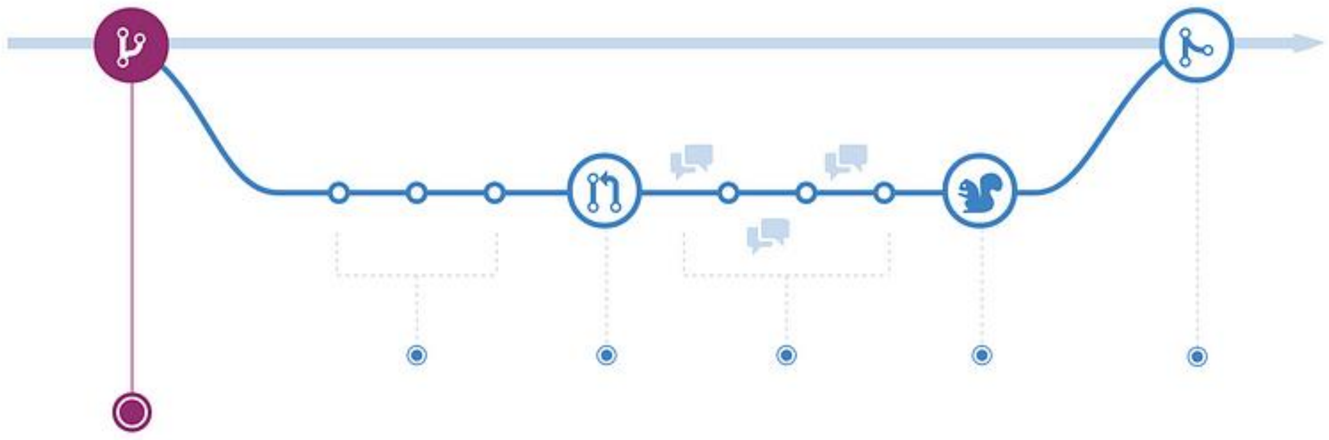
This first rule is quite straightforward, but making a habit out of it is very important. Every time you start working on something new — your portfolio, a learning project, a solution to a coding challenge, and so on — you should create a new Git repository and push it to GitHub.

Having a dedicated repo is the first step to being able to use version control for every line of code you write. Using version control is how you will work once you join a company and start working on real-world projects. Learn this early and make it a habit.

Quick Note: if using the terminal becomes a hassle and makes you less likely to use Git for all your projects, consider using the Github Desktop app.

Rule #2: Create a new branch for every new feature

Let's say you are working on your portfolio and you want to build a new "Contact me" section/component. Create a dedicated branch for this new feature, give it a meaningful name (e.g. *contact-me-section*), and commit all the code to that specific branch.



If you don't know what branches are, go back to the Github Flow reading that I recommended before.

Working with branches allows you and your team members to work on different features in a parallel way while keeping the specific code for each feature isolated from the rest. This makes it harder for unstable code to get merged into the main code base.

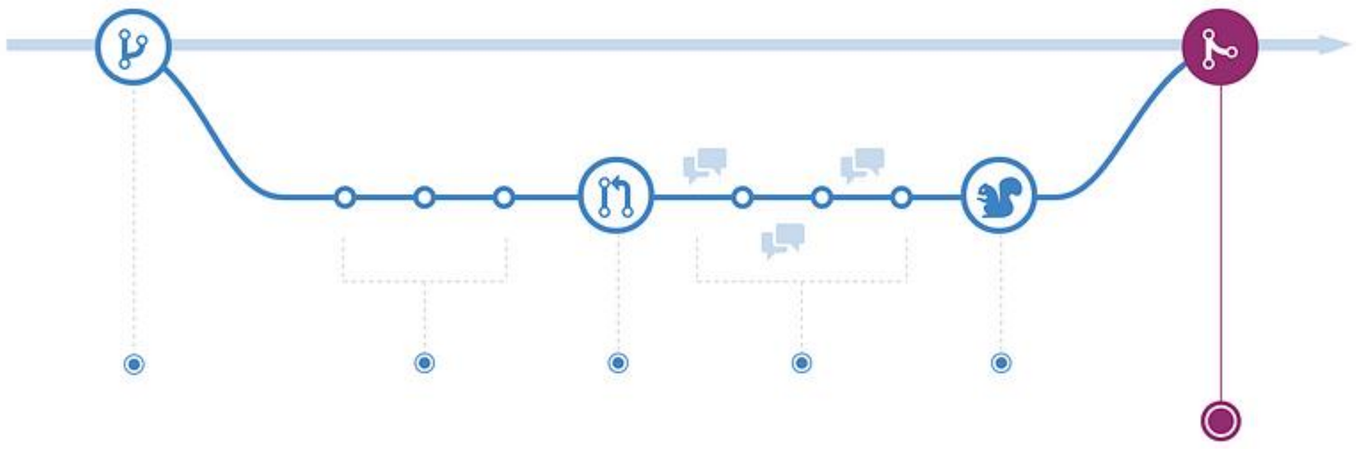
Even if you are the only person on your team, getting used to using feature branches will make the Github Flow process a breeze once you join a real job.

Rule #3: Use Pull Requests to merge code to Master

Every repository starts with a master branch by default. **You should never push changes directly to the master branch**. Instead, you should use feature branches as described above, and open a new Pull Request to merge the feature branch code with the master branch code.

In a real job, someone will look at your Pull Request and do a code review before approving it. GitHub will even run automated tests to your code and let you know if there is an issue with it. You will also be notified if there is any merge conflict between your code and the code in the master branch. This can happen, for example, if another developer pushed a change to the master branch that affects a file that you also modified.

After your code has been reviewed, tested, and approved, your reviewer will give you thumbs up for you to merge the Pull Request, or they will directly merge your pull request.



Even if you are working alone, get used to creating Pull Requests as a way to merge your changes to the master branch. This, by the way, is the basic workflow used by almost every open source project. If you ever contribute to one (you should!), understanding this three rules will make it really easy for you to get your contribution accepted without any problem.

— AWS Community Builder | DevOps Engineer | Docker | Linux | Jenkins | AWS | Git | Terraform | Docker | kubernetes

Empowering communities via open source and education.

Connect with me over linktrr: https://linktr.ee/biswaraj333

Git

Github

Repositories