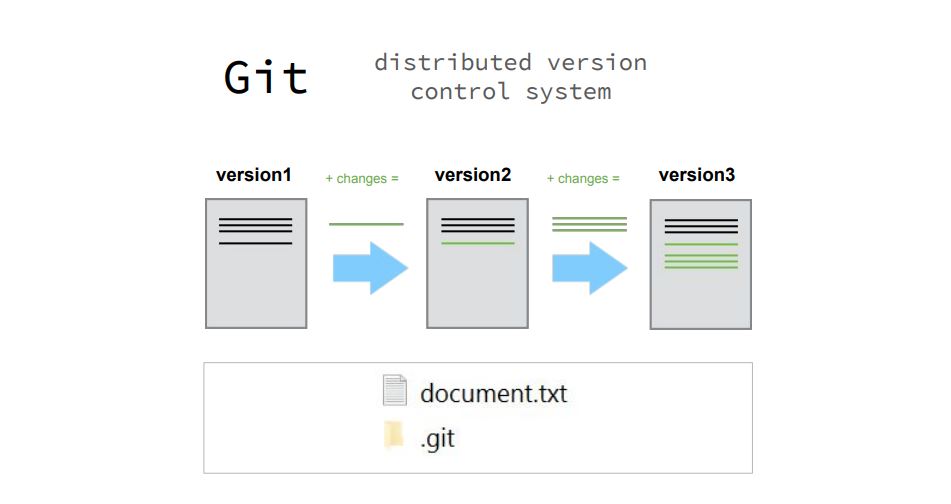
**Git and Github**

Git is a postmodern version control system that offers the familiar capabilities of CVS or Subversion, but doesn’t stop at just matching existing tools. Git stretches the very notion of version control systems (VCS) by its ability to offer almost all of its features for use offline and without a central server. It is the brainchild of Linus Torvalds, with the first prototype written in a vitriolic two-week response to the “BitKeeper debacle” of 2005. Today, developers everywhere are migrating in droves to this exciting platform. Users reference its blistering performance, usage flexibility, offline capabilities, and collaboration features as their motivation for switching. Let’s get started with Git. You’ll be using it like a master in no time at all.



**Step1: Installing Git**

Before you start using Git, you have to make it available on your computer. Even if it’s already installed, it’s probably a good idea to update to the latest version. You can either install it as a package or via another installer, or download the source code and compile it yourself.

### Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution. If you’re on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use dnf:

$ sudo dnf install git-all

If you’re on a Debian-based distribution, such as Ubuntu, try apt:

$ sudo apt install git-all

For more options, there are instructions for installing on several different Unix distributions on the Git website, at <http://git-scm.com/download/linux>.

### **Installing on Mac**

There are several ways to install Git on a Mac. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run git from the Terminal the very first time.

$ git --version

If you don’t have it installed already, it will prompt you to install it.

If you want a more up to date version, you can also install it via a binary installer. A macOS Git installer is maintained and available for download at the Git website, at <http://git-scm.com/download/mac>.



Figure 7. Git macOS Installer.

You can also install it as part of the GitHub for Mac install. Their GUI Git tool has an option to install command line tools as well. You can download that tool from the GitHub for Mac website, at [http://mac.github.com](http://mac.github.com/).

### Installing on Windows

There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to <http://git-scm.com/download/win> and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to <https://git-for-windows.github.io/>.

To get an automated installation you can use the [Git Chocolatey package](https://chocolatey.org/packages/git). Note that the Chocolatey package is community maintained.

Another easy way to get Git installed is by installing GitHub Desktop. The installer includes a command line version of Git as well as the GUI. It also works well with Powershell, and sets up solid credential caching and sane CRLF settings. We’ll learn more about those things a little later, but suffice it to say they’re things you want. You can download this from the [GitHub Desktop website](https://desktop.github.com/).

**Step 2: Identify Yourself**

The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it’s immutably baked into the commits you start creating:

$ git config --global user.name "John Doe"

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

Again, you need to do this only once if you pass the --global option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the --global option when you’re in that project.

Many of the GUI tools will help you do this when you first run them.

**Step 3: Setting Github Account**

## 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.

Create your account on Github with the same email as you have written in the Git configuration.

## Step 4: 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**. :tada:

**Step 5: 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.

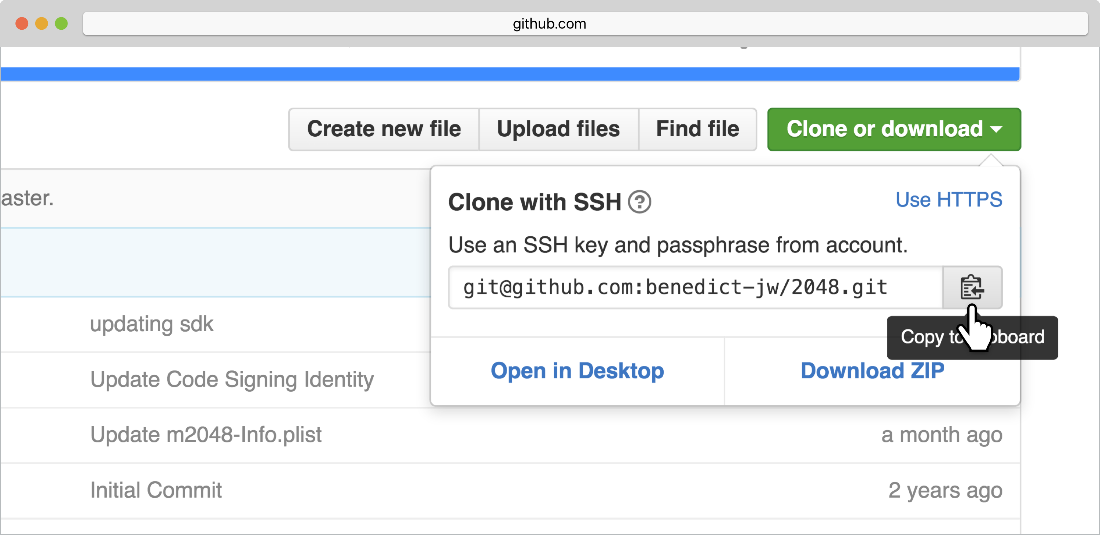
#### Make and commit changes

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.



These changes will be made to just the README file on your master branch.

**Step 6: Clone the repo and make some changes**



Basic Workflow:

git clone / pull

git add

git commit

git push

|  |  |  |
| --- | --- | --- |
| 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()" |

For More Details: <https://www.youtube.com/watch?v=SWYqp7iY_Tc>