# Git Training - Basics

Created: 29 January, 2019 – Modified: 29 January, 2019

Table of Contents

[Git Training - Basics 1](#_Toc536611171)

[Overview 1](#_Toc536611172)

[What is Git? 1](#_Toc536611173)

[How does Git work? 1](#_Toc536611174)

[Branching 2](#_Toc536611175)

[Editing 2](#_Toc536611176)

[Committing 2](#_Toc536611177)

[Merging 2](#_Toc536611178)

[Installing Git on Mac 3](#_Toc536611179)

[Setup Git 3](#_Toc536611180)

[Using Git 3](#_Toc536611181)

[Getting a Git Repository 4](#_Toc536611182)

[Creating a Branch 5](#_Toc536611183)

[Summary 7](#_Toc536611184)

[References 7](#_Toc536611185)

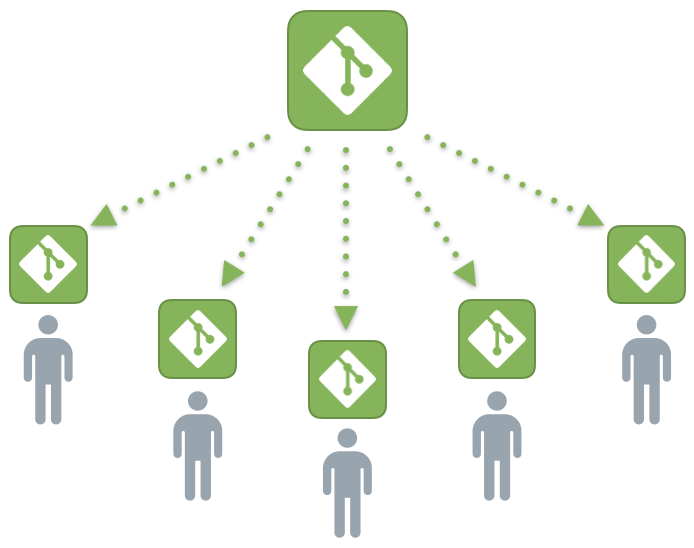
## Overview

This document will cover the history and basics of installing, setting up, and using Git.

## What is Git?

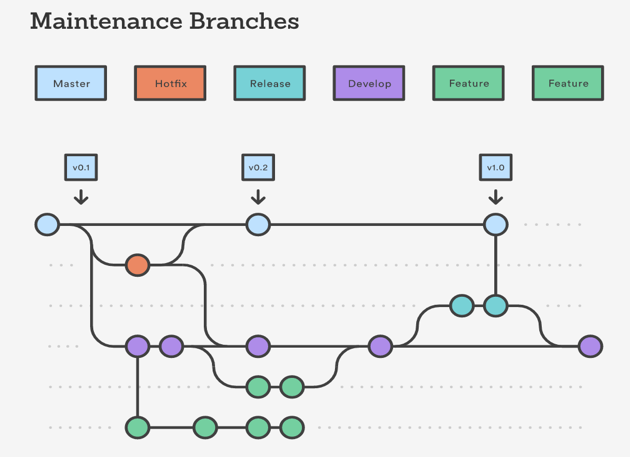
Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

This means that several developers can be working on the same project at the same time without overwriting one another's files. They can view changes made by other developers, merge their changes with those changes, revert back to a prior version of a file, or experiment on the same project without affecting the integrity of the stable version of the project.



## How does Git work?

Git uses a process of branching, editing, committing, pushing, and merging.



### Branching

Branching creates a separate line of development from the main trunk, known as the master trunk, or from another branch. Unlike traditional methods, the files and folders are not copied. Instead the new branch records the changes made to the file which keeps the project both lean and robust.

### Editing

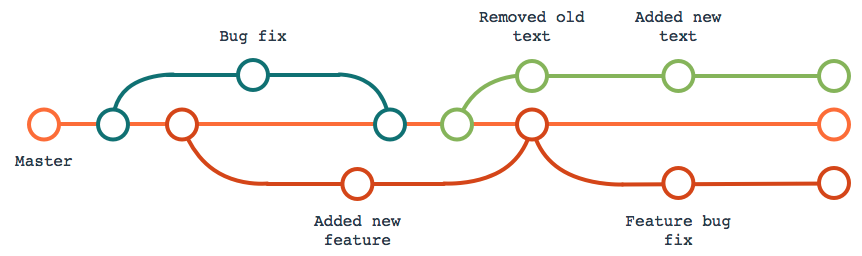
Files are modified the same way they have been traditionally using a text editor of the developer's preference.

### Committing

Once a series a edits have been made and the files saved, they are committed to the branch. This means the changes are recorded to the repository. A commit creates a save point for the state of all the files and folders at the time the commit is made. This save point can be referred to after further edits and commits have been made to the branch. The whole previous commit can be resorted, only certain files can be restored, or only certain chunks of changes can be restored as needed.

### Merging

Once all of the changes have been made for a branch and it is ready for production, it can be merged back into the main trunk or master branch. This is done using a file by file, line by line comparison which shows the difference between the two files. The desired changes are selected to replace or be added to the file in the main branch. Once this process is complete, the development branch should be removed to keep the project clean.



For example, a bug fix needs to be made to the program. Ben creates a branch called 'Bug#132'. He makes the changes and tests the program. Great! It squashed the bug. Ben commits the changes to the branch 'Bug#132'. Then he merges those changes back into the main branch of the project, tests the program again to confirm everything was successful, and then deletes that branch.

At the same time, Joshua is tasked with adding new feature that makes the menu fade in and out. So, he creates a branch off from the main trunk called 'Menu Fade'. He makes the changes, tests them and works out all the bugs. Then he commits those changes to his branch, merges those changes into the main branch, and tests the program again to confirm everything was successful. Uh oh, the bug Ben fixed caused a bug in the new feature. Joshua reverts the main branch back to the previous commit which removes the changes he merged into it. Then he merges the updated main branch into his 'Menu Fade' Branch and proceeds to modify the files accordingly.

After Ben performed the bug fix, Nathan then tasks him with removing some text from the contact page. Ben makes a new branch called, 'Contact Page Text Update'. He removes the text and sends the change to Nathan for review. Nathan reviews it and says, "That just doesn't look right. It should say, 'On the corner of Mead and Douglas' instead." Ben makes the change, commits it to the project, and sends it to the client for review. The branch stays open until final approval after which the merging process is executed again.

This is the basics of how Git works.

## Installing Git on Mac

Git comes preinstalled on Mac. To confirm this, open a terminal window and type 'git --version'. This will display the version of Git installed. If Git is not installed, Mac may prompt you to install it. If it does not, the easiest way to install Git is to download and install the Github Desktop (<https://desktop.github.com/>). There are several GUI options all of which offer slighter different features and usability (<https://git-scm.com/download/gui/mac>).

## Setup Git

The only thing that needs to be set up is your name and email address. This identifies changes with you and provides others with a way to contact you. To set up these values type:

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

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

## Using Git

Git is a command line-based program, but several GUIs have been created to manage the most commonly used features.

### Getting a Git Repository

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can **clone** an existing Git repository from elsewhere.

#### Initializing a Repository in an Existing Directory

If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to navigate to that project’s directory.

$ cd ~/Sites/FullServiceReview

$ git init

This creates a new subdirectory named '.git' that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet.

To start version-controlling existing files you perform an initial commit. This is done using the **add** and **commit** commands.

$ git add \* //adds all the files and folders in the directory to the repository

$ git commit -m 'initial project version' //-m records a succinct explanation of the commit

#### Cloning an Existing Repository

Cloning is used to get an existing repository housed on a remote server. The two most popular Git servers are Bitbucket (<https://bitbucket.org>) and Github (<https://github.com/>). Baseline Creative stores its repositories at Bitbucket because private repositories are available at no extra charge. Before cloning a repository:

1. Create an account with Bitbucket (<https://bitbucket.org/account/signup/>)
2. Be added to the BaselineCreative group

To clone a repository onto your local machine:

$ git clone https://baseline\_creative-ben@bitbucket.org/baseline\_creative/fullservicereview.git ~/Sites/FullServiceReview/

The source URL can be found by logging into your bitbucket account, clicking on the repository, selecting source, and clicking 'clone'.



### Creating a Branch

Before making any changes you want to create a branch. To do this type

$ git checkout -b Bug#132

This creates a new branch called 'Bug#132' and switches the project on your local machine to it. To switch back to master or to a different branch:

$ git checkout master

#### Saving Changes to the Local Repository

Now you are ready to make changes. Once you are at a spot where you want to take a snapshot of your changes type:

$ git add -u //this add changed files that are being tracked

$ git commit -m 'bug fix before testing' //this creates a commit point labeled 'bug fix before testing'

Git will list the files that it is tracking that have changed. If one or more files there were changed are not listed, you will need to tell Git to track them. This is done by:

$ git add assets/logo-baseline-creative.png //adds a specific file

$ git add assets/page-headers/\* //adds all the files in the page-headers directory

I run the project on the server that is on my machine. If you want to run the files on a remote server, you will have to upload them using an FTP client.

#### Saving Changes to the Remote Server

At the end of the day or when you want to back up your repository to the Bitbucket server:

$ git push //this pushes the changes to the remote server

#### Merging Files Back into the Master Branch

You've made the changes, tested them, and the changes are ready to be pushed to the production server. Common practice is to merge the offshoot branch into the master trunk. However, it is better practice to merge the master trunk into the offshoot branch, resolve any conflicts, test the program after the merge, and then merge the offshoot branch into the master trunk. This avoids having to roll back to a pervious commit point should the changes break the production program. Which option you choose depends on if other commits or merges have occurred on the main trunk since you created the offshoot branch and the changes you made. To merge one branch back into another:

$ git checkout Bug#132 //switches to the Bug132 branch

$ git merge master //merges the master branch into Bug#132

If there are conflicts it cannot resolve, you will be notified. You will need to open the file in a text editor. In side it will list the conflicts as follows:

<<<<<<< Bug#132:index.html

<div id="footer">contact: webmaster@baselinecreative.com</div>

=======

<div id="footer">

please contact us at webmaster@baselinecreative.com

</div>

>>>>>>> master:index.html

You will remove one chunk of code or the other, or Frankenstein the two together. When you are finished resolving the conflict, only valid code should remain:

<div id="footer">contact: webmaster@baselinecreative.com</div>

Then you will need to follows the same steps in the section 'Saving Changes to the Local Repository' to notify Git that the conflicts have been resolved. Now you can test the merge to verify the program is still working as expected. Finally, merge the update branch back into master.

$ git checkout master //switches to the master branch

$ git merge Bug#132 //merges the offshoot branch into the master branch

$ git push //copies the changes to the remote server

The remote server can be set up to, in turn, push the files from the master branch automatically to the production server when they are updated. If this is not set up, then the files on the production server will need to be manually updated using an FTP client.

#### Deleting a Branch

After final testing to confirm the program is working successfully on the production server, you can delete the offshoot branch. All the commits made on the offshoot branch are merged along with the final commit and are still retained on the master branch. To delete the offshoot branch:

$ git branch -d Bug#132

## Summary

This is the basic workflow of using Git. All of these functions can be performed through the Atom text editor (https://atom.io/).

## References

* <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
* <https://www.atlassian.com/git/tutorials>
* <https://blog.cpanel.com/git-version-control-series-what-is-git/>
* <https://bitbucket.org>
* <https://atom.io/>
* <https://desktop.github.com/>
* <https://en.wikipedia.org/wiki/Git>