



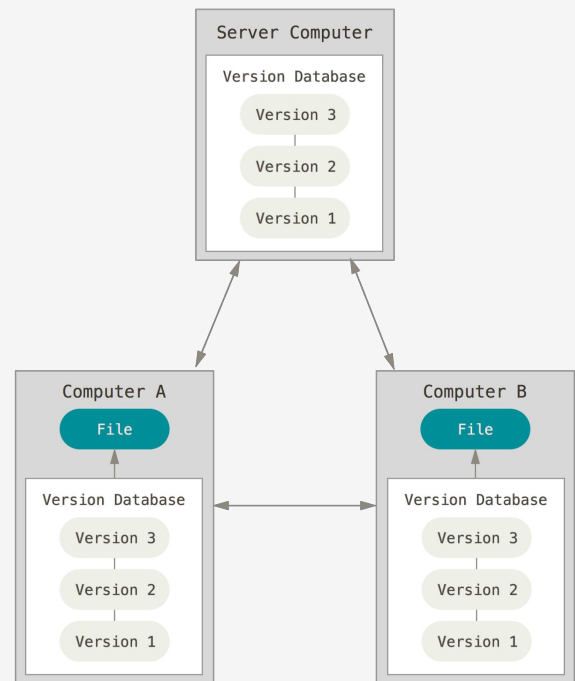
## Getting Familiar With Git

Ethan Brady  
Tom Scallon  
Chengrui Yang

11/17/2015  
English 314  
Vincent Robles

## Introduction

Proper use of version control is one of the most widely needed skills in the tech industry today. According to the official documentation of Git, “Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. This gives programmers the ability to write code while being able to revert back to an older version of their code at any time.” Version control systems allow teams to collaborate easily on programming projects as well as maintain an accessible history of the project’s incremental progress. Since these benefits are so critical to the success of large-scale projects, version control systems are used by virtually all programmers in industry. The image (Figure 1) to the right shows what version control system looks like in team work.



Git is a version control manager that distributes a copy of a project to each collaborator’s machine. Because this setup provides distinct advantages over its competitors, Git is arguably the most popular version control system in the world. These advantages are numerous and extremely beneficial in helping developers manage their codebases. First, local copies allow programmers to work offline. They can make changes to their local repository, and then save these changes to a remote repository once network connectivity becomes available. Next, full copies of repositories allows for data redundancy. If the central remote repository gets deleted, it can be completely restored from one of the distributed copies. Lastly, local copies of repositories allows commands to be executed quickly. Changes to local files are always faster than changes made over the internet. With these advantages, Git has become the industry standard of version control systems.

This document is intended to provide programming students with an introduction to what Git is, how it works, and how to utilize its feature set. Students understanding this information will be more productive and efficient when working on projects and will be better prepared to become successful members of professional teams in the tech industry.

## How it Works

Before delving into the details of using Git, it is important to discuss its functionality at a more abstract level. While hands-on experience is undoubtedly beneficial to learning its idiosyncrasies, rushing into using Git without first creating a solid conceptual understanding of how it works can result in poor decisions regarding its usage. Git is a complex and powerful tool capable of greatly increasing one’s productivity; however, if a user does not fully understand its behavior it can easily do just the opposite.

## Repositories

**Repositories** are the simplest organizational objects used by Git. They are analogous to folders on a personal computer, though they record a plethora of additional information. Among this information is a complete record of all changes made to each file in the repository — including the author and date of the changes — as well as a set of references to endpoints on the web where the repository is replicated.

When working on a project, Git keeps the entire contents of a repository on every machine involved with the project. If a new user joins the project after significant work has already been contributed to the repository, the complete history of the repository is copied to his or her machine. A number of actions are used to keep the various copies of the repository in sync; these will be discussed in the “Commands” section of this description.

## Hosting and Remotes

As mentioned before, each copy of a repository on a contributor’s machine (called a **local clone** or **local repository**) contains a set of references to online copies of the repository. These references are called **remote repositories** (or **remotes** for short), and they enable project contributors to sync their work with one another. Most often, remotes are **hosted** on a site such as GitHub ([Github.com](https://github.com)), whose sole purpose is to provide Git users with an online location for their projects. By configuring their local clones to use the same remote site (ie, GitHub), a member of a team can work locally before syncing his or her changes to the remote repository. The remote repository then ensures other users are aware of the changes and prevents other members from creating conflicting edits.

Git is a versatile software; as such, it can be implemented easily on a large-scale site like GitHub or a personal server created by a service like Amazon Web Services. Virtually any network endpoint can be configured to host a Git repository, at which point configuring the endpoint as a remote for a local repository takes less than a minute.

## Local File Management

Managing Git files locally requires an understanding of three fundamental concepts. The first of these is the **working directory**, which essentially refers to the state of a local file system. Any changes made to a file occur in the working directory, much like the working draft of a report.

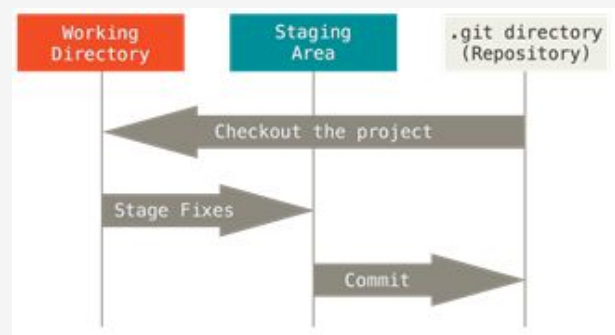


Figure 2

Next is the **staging area**. Changes one wishes to save to a repository are first **staged**, simply meaning they are added to the staging area. One can add or remove (stage or unstage, in Git terminology) changes to or from the staging area at any time; however, one must do so explicitly. Git will not automatically stage changes.

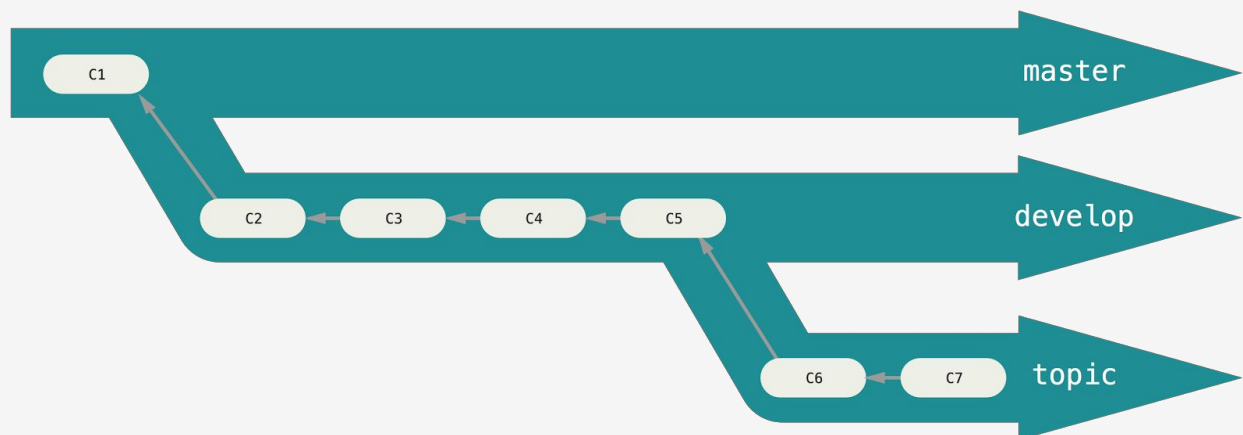


The last concept to know is the **.git directory**. This is a special folder containing the entire contents of the repository and stored on a collaborator's local machine. When the collaborator updates the local repository to reflect the latest information stored in a remote (using one of the commands described in the “Commands” section), those changes are stored in the .git directory. Similarly, when a set of changes currently in the staging area needs to be recorded, a collaborator adds them to the .git directory. This action is called **committing**, and is described in greater detail below.

## Commits and Branches

A **commit** is a set of changes saved to the repository. When a collaborator directs Git to store a group of staged changes in the .git directory, a commit is created to encapsulate the changes, along with information such as the date and author of the changes. The action of adding staged changes to the .git directory is called **committing** those changes.

Every time a commit is created, it is also configured to reference the last commit that was made. In this way, the commits form a chain of changesets which effectively detail the development of a project over time. Collaborators also have the option to split this chain into multiple chains in order to work without affecting the progress of other collaborators. These sub-chains are called **branches**, and the action of splitting the chain into multiple branches is called **branching**.



## Commands

With a vivid conceptual understanding of Git established, the details regarding how to utilize the version control system's broad feature-set can be discussed. Interaction with Git is achieved through the use of a predefined set of commands. With a thorough understanding of these commands, users can utilize Git to its full potential. As such, an overview of the most popular Git commands is given below.



## Init

The **init** command is used for creating a new Git repository. It is often the first command used. Once the command is executed, a `.git` directory will be created at the root of the project directory, which contains all of the necessary files for a repository. No files in the initial project will be altered.

## Clone

The **clone** command is used for copying a remote Git repository to a local machine. This should be used when the project and repository do not yet exist on the local machine, but exist remotely.

## Add

The **add** command saves changes made to files in the working directory into the staging area. It is necessary for changes to be in the staging area before they can be committed and ultimately **pushed** to a remote repository.

## Commit

The **commit** command saves changes that have been **added** to the staging area into the local Git repository. This must be done before changes can be **pushed** to a remote repository.

## Pull

The **pull** command retrieves the latest changes from another repository for the current branch. These changes are merged into the project's current working directory.

If a file has been updated in both the remote repository as well as in the local working tree, then there will be a merge conflict. In this scenario, both updates of the file will be present, and it will need to be manually edited in order to decide which changes should remain and which changes should be removed.

## Push

The **push** command copies **commits** from the local Git repository to a specified remote repository. If the remote repository has changes that are not reflected in the local repository, the **push** command will not work. This is to ensure that a collaborator's commits to the project are not overwritten. In this scenario, it is necessary to **pull** remote changes into the local copy before new changes can be pushed.

## Branch

The **branch** command allows for all interactions with branches on the local Git repository. This includes creating, listing, renaming, and deleting branches.

## Checkout

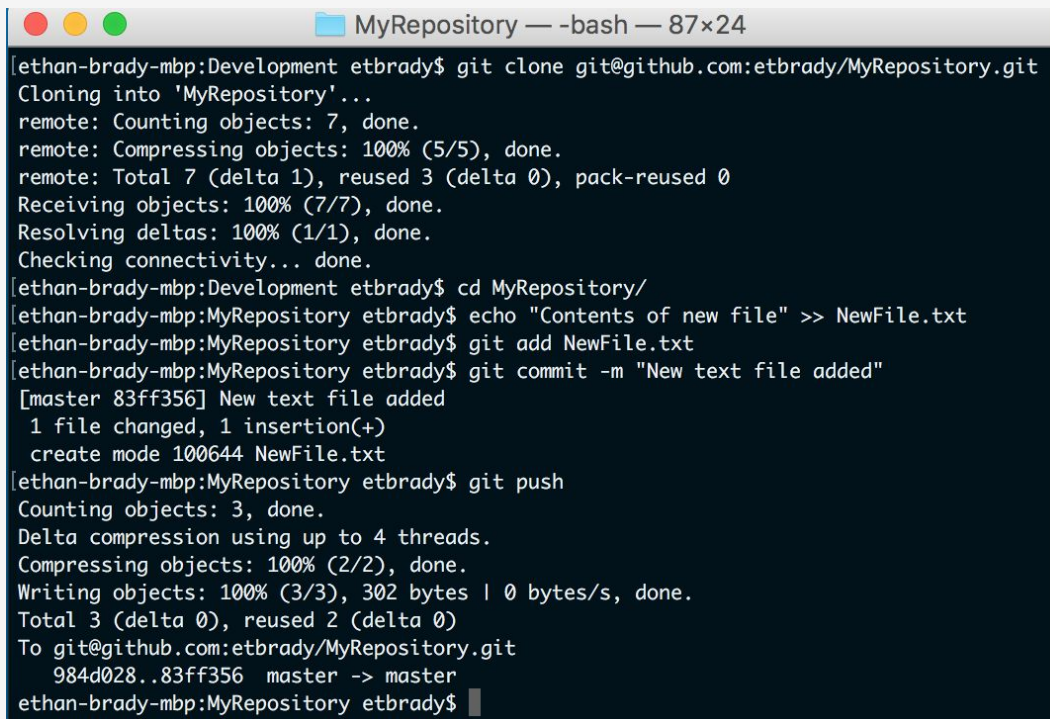
The **checkout** command selects which branch of a Git repository should be used in the working directory. Upon execution of the command, all files in the current working directory are updated to reflect what is saved in the repository for the selected branch.

## Interacting with Git

### Command Line

One of the primary ways to execute commands in Git is to use a command line interface. Git can either be used through any Unix-based terminal, or users can download a special command line interface through Git's official webpage (<https://git-scm.com/downloads>). Once the terminal is open, the user can use it to navigate to the working directory of their local project. Then, the user can enter the desired Git commands into the terminal. The exact syntax for these commands is outside of the scope of this document. Visit (<https://git-scm.com/docs>) to read up on command line syntax for Git.

The image below illustrates a potential use case of Git on the command line. In this example, the user is cloning a remote repository, adding a new file to the staging area, committing the changes to the local repository, and then pushing this change to the remote repository.



```
ethan-brady-mbp:Development etbrady$ git clone git@github.com:etbrady/MyRepository.git ]
Cloning into 'MyRepository'...
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 1), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (1/1), done.
Checking connectivity... done.
[ethan-brady-mbp:Development etbrady$ cd MyRepository/ ]
[ethan-brady-mbp:MyRepository etbrady$ echo "Contents of new file" >> NewFile.txt ]
[ethan-brady-mbp:MyRepository etbrady$ git add NewFile.txt ]
[ethan-brady-mbp:MyRepository etbrady$ git commit -m "New text file added" ]
[master 83ff356] New text file added
 1 file changed, 1 insertion(+)
 create mode 100644 NewFile.txt
[ethan-brady-mbp:MyRepository etbrady$ git push ]
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 2 (delta 0)
To git@github.com:etbrady/MyRepository.git
 984d028..83ff356 master -> master
ethan-brady-mbp:MyRepository etbrady$
```

Figure 1



## Graphical User Interface

In addition to using the command line to issue Git commands, there are several third-party applications that provide a Graphical User Interface (GUI) for interacting with Git. These provide visual representations of the state of Git repositories as well as provide buttons for issuing Git commands. Thus, GUI applications are beneficial for abstracting and simplifying the command line interface approach to Git. Users of these applications no longer need to remember and type syntactically correct commands; the application will easily guide the user to the correct button.

There are a number of different GUI applications for interacting with Git. Among the most popular are SourceTree, GitHub for Windows/Mac, Tower, and Gitbox. A full list of the third-party applications endorsed by Git can be viewed at (<https://git-scm.com/downloads/guis>).

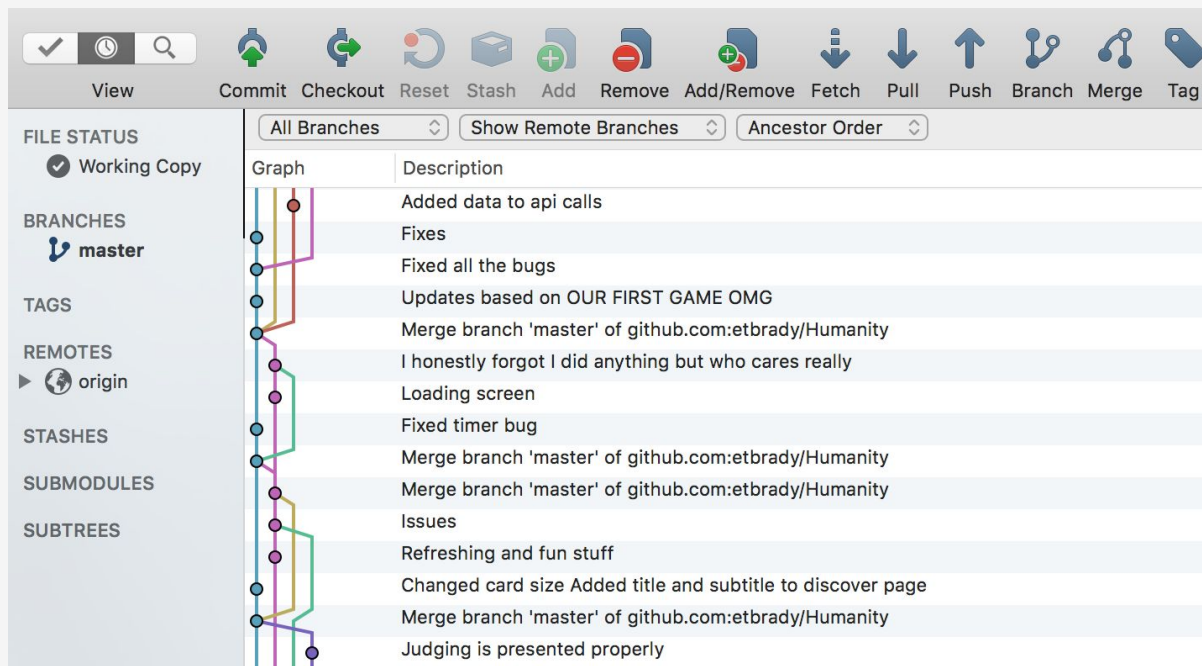


Figure 2

## Conclusion

Now that an understanding of how Git works and how to utilize its commands has been established, students can more effectively manage their programming projects and collaborate with other developers. Proper use of version control systems is a crucial aspect of professional software development, and having knowledge of a tool as popular as Git is a great advantage to students searching for careers in development.

While this document has provided a solid foundation of the fundamentals of Git, there is much more to learn. Git is a highly complex tool with numerous features and idiosyncrasies. Located below are a number of resources that explore the internal operations of Git in greater detail. It is highly recommended everyone



interested in a career involving software development use these resources to further solidify their expertise of Git.

## Sources

Git. <i>Git documentation</i> . [Online]. Available: <a href="https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control">https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control</a> [Accessed: 9- Nov- 2015]	
Atlassian. <i>Git tutorial</i> . [Online]. Available: <a href="https://www.atlassian.com/git/tutorials/">https://www.atlassian.com/git/tutorials/</a> [Accessed: 12- Nov- 2015]	
Vogella. <i>Git - Tutorial</i> . [Online]. Available: <a href="http://www.vogella.com/tutorials/Git/article.html">http://www.vogella.com/tutorials/Git/article.html</a> [Accessed: 12- Nov- 2015]	
Figure 1 and Figure 2	<i>Git. Git documentation</i>
Figure 3	Screenshot from Ethan's personal computer library
Figure 4	Screenshot from Ethan's personal file