**<https://www.bitdegree.org/learn/git-basics>**

**Git Basics: Main Concepts and Principles Explained**

**Reading time5 min**

**PublishedApr 11, 2019**

**UpdatedOct 3, 2019**

During the development process programmers often have issues keeping track of changes in their source code. Therefore, version-control systems were created to help developers manage and merge different revisions of projects. Many software creators use Git basic commands to keep track of alterations in the source code and to maintain different versions of projects.

Git can be confused with GitHub, but these two should not be used as synonyms. So, what is GitHub? How is it related to Git? GitHub is a web-based platform, allowing people to edit their files and manage projects online. GitHub is different from the Git tool which you have to install locally instead of using online.

Git was developed in 2005 by Linus Torvalds, the person behind the Linux operating system kernel. It is hard to keep track of your source code as it grows. In Git, changes are versions that are stored in a repository, which catalogs those changes.

Quickly revising different versions and undoing changes with Git basic commands can save your software from crashing. You will be able to know the date and authors of submitted changes. If a group of people contribute ideas to the same project, it’s worth to install Git to coordinate work and improve productivity.

**Contents**

* [**1.** Why You Should Learn Git: Purpose and Use](https://www.bitdegree.org/learn/git-basics#why-you-should-learn-git-purpose-and-use)
* [**2.** Where to Start](https://www.bitdegree.org/learn/git-basics#where-to-start)
* [**3.** What You Will Learn](https://www.bitdegree.org/learn/git-basics#what-you-will-learn)

**Why You Should Learn Git: Purpose and Use**

Git is a tool to make the lives of developers easier. If a compiler fails to compile the source code of a new version, you can quickly retrieve the previous one and find errors. However, Git vs. GitHub is a question that needs to be answered in more detail.

GitHub is a cloud-based, distributed system, handling copies of Git repositories and merging them into various branches. Git does not need any cloud hosting and mostly focuses on source code management.

GitHub is also a platform to share your code. As a result, many open-source projects are posted there. Think of GitHub as a nerdy version of Facebook or Twitter.

Who wins the Git vs. GitHub race depends on you. GitHub has recently released an Electron-based app called GitHub Desktop. Most people that use Git also use GitHub, but you do not have to. If your goal is monitoring changes in your source code, Git is enough. Here we present the reasons why Git can save people time and money:

* **Versions:** always have different versions of your project and retrieve them if necessary.
* **Changes:** all changes are attributed to the developer that made modifications to the project.
* **Forking:** experiment with changes in repository copies without influencing the original.
* **Collaboration:** work with other people on projects and manage everyone’s input.
* **Not only for developers:** use Git to keep track of any project and make sure that no information is lost.
* **Compatibility:** Git works on all operating systems.

Let’s also quickly review the two Git basic commands:

* **Git Delete Branch:** you can delete local or remote branches. If you have already included a local branch in the original project, you can get rid of it with git branch -d branch\_name. If you want to delete a branch which has not been merged with the remote one, you can use the command git branch -D branch\_name. In case you want to delete a remote branch, you should use the git push remote\_name --delete branch\_name.
* **Git Create Branch:** creating a new branch in repository is simple. To create and check a local branch, use git checkout b branch\_name command. If you want to make a remote branch, you can simply push a local one with git push remote\_name branch\_name command.

**Where to Start**

Before you can start taking advantage of Git, you should follow these steps:

* + **Download Git**. Follow [this tutorial](https://www.bitdegree.org/learn/how-to-install-git) and find out how Git is downloaded and installed on different operating systems.
  + Register by providing **username** and **email**. More information on this step can be found [here](https://www.bitdegree.org/learn/git-config).
  + Learn the basics of [repository](https://www.bitdegree.org/learn/what-is-a-git-repository) which stores **different versions** of your projects.
  + Continue to **read our tutorial**. It is very short: you will learn about Git quickly.
  + In case some aspects of Git are still unclear, check out **our interactive and video courses**. They will provide you with more details and features that might not be fully covered in this tutorial.

**What You Will Learn**

After reading our tutorial, you will have no problem understanding Git basics. You will be able to install Git and set up your account. The terms branch and repository will also no longer be strangers: you will know about them. By learning Git basic commands, you can start monitoring your projects and collaborating with other developers.

As we have mentioned before, you are welcome to enroll in our interactive or video courses about Git. This [video course about Git](https://www.bitdegree.org/course/git-commands) is designed for beginners. It will cover such topics as installation and use of Git, important features, cloning with Cloud BitBucket, architecture and command lines

**What Is Git and Why Developers Choose It**

**Reading time4 min**

**PublishedJan 1, 2016**

**UpdatedOct 3, 2019**

Welcome to your first Git tutorial for beginners! As you probably know, Git is a software used for version control. Systems such as Git dedicated to version control, allow managing changes of information in various documents - websites, programs and other files and forms of gathered information. Git commands are easy to use, and that is one of the reasons for Git's popularity among developers. Git can be used for your work, but it is also a very convenient tool for collaboration, especially for the remote one.

Git is a system that works on a distributed version control principle. Unlike centralized version control systems, it does not need to rely on a central server to store different versions of a project. Instead, all the contributors - people who are working on the project - have the locally stored copies of the main repository.

This Git tutorial for beginners introduces the very basics of Git as a version control system, what is GIt, its working and interaction principles. You will see that it is a good idea to learn to operate the Git command line as it is a more powerful way to execute Git commands. You will learn what is Git Bash commands and Git shell and how they help run the commands.

Let's start with Git commands!

**Contents**

* [**1.** Git As a Version Control: Main Tips](https://www.bitdegree.org/learn/what-is-git#git-as-a-version-control-main-tips)
* [**2.** Git Benefits As A VCS](https://www.bitdegree.org/learn/what-is-git#git-benefits-as-a-vcs)
* [**3.** Getting Started - Git Tools](https://www.bitdegree.org/learn/what-is-git#getting-started-git-tools)
* [**3.1.** Git CMD](https://www.bitdegree.org/learn/what-is-git#git-cmd)
* [**3.2.** Git Bash](https://www.bitdegree.org/learn/what-is-git#git-bash)
* [**3.3.** Git GUI](https://www.bitdegree.org/learn/what-is-git#git-gui)
* [**4.** Git As a Version Control: Summary](https://www.bitdegree.org/learn/what-is-git#git-as-a-version-control-summary)

**Git As a Version Control: Main Tips**

* Git is one of the **most popular choices** among developers to track version changes made over time
* There is a number of **additional tools** to use with Git and its already pre-built GUI .
* Even though Git comes with **GUI**, it's a good idea to master its **command line** as it helps fully use all the functions of Git.

**Git Benefits As A VCS**

Why should you use Git and what are its main differences from the other Version Control Systems (VCS)?

All the companies use one or other type of version control system. For example, Local Version Control tool saves changes to the database on the **local** computer. Centralized Version Control tools use a **shared server** to store files, while the Distributed Version Control system **distributes** copies of the files to the clients who use it. This version is preferred by many because:

1. It allows easier file sharing than the Local Version Control.
2. It can eliminate issues that could occur if a server connection is lost under the Centralized Version Control system.
3. All changes are recorded and difficult to lose.

The following tutorials will go into detail about how to get started with Git for keeping track of the applications you develop.

**Getting Started - Git Tools**

There are many ways to use Git and what is more, it has a number of different dedicated tools to use. For example, it might be a good idea to download some third-party Git GUI tools. On the other hand, Git comes with an already-built-in GUI, as well as other different tools. This tutorial will focus mainly on the command line.

**Git CMD**

As its name suggests, this is a command line prompt interpreter. If you currently use or have used Windows, you probably have seen one. If you have installed Git on Windows and are used to using CMD, you can use it for Git commands.

**Git Bash**

Bash in Git is an emulation of a Unix shell for Linux and Mac OS, so you can use it on Windows as well if you are used to Linux. Git Bash commands are run in Linux, while Windows have Git Shell command line.

**Git GUI**

Graphical User Interface (GUI) facilitates the usage of Git. You might not need to touch Git Bash, CMD or use command lines at all.

On the other hand, the built-in GUI has its own limitations. At first sight, it might look quite minimalistic. What is more, different GUI despite their capabilities can support only partial Git functionality. That is why this tutorial focuses on mastering Git command lines.

**Git As a Version Control: Summary**

* Developers often choose Git when they need to **track** and **control** version changes.
* Git has a number of **third** **parties** **tools** that can be implemented to and used in the system.
* Git has itw own GUi but it's **not recommended** to rely solely on it.

**How to Install Git on Windows, Mac OS and Linux**

**Reading time7 min**

**PublishedJan 2, 2016**

**UpdatedOct 3, 2019**

Now that you know that Git is widely used for controlling version changes, it is time to learn how to install Git and start using it. Just like many other products for developers out there, Git is an open source and a free to use the system; therefore, it does not require complex actions to install it.

How you install Git depends on what operating system you have: Windows, Mac OS or Linux. The guidelines for each of the operating system differ a little bit, so you should always follow the guidelines for your system. You can install Git either manually or by using a package manager which automates some processes. In any case, installing Git shouldn't take long and is easy to complete.

Even though you already have Git installed on your computer, it might also be a good idea to check which version you have. Git is constantly updated, and new features are added or the bugs fixed. Therefore, it is a good idea to download the latest Git version and updated your Git.

In this tutorial, you will be guided through the installation process for each of the operating systems and shown several most common options to do it.

So start reading below!

**Contents**

* [**1.** How to Install Git: Main Tips](https://www.bitdegree.org/learn/how-to-install-git#how-to-install-git-main-tips)
* [**2.** Git Install for Windows](https://www.bitdegree.org/learn/how-to-install-git#git-install-for-windows)
* [**3.** Git Install for Mac OS](https://www.bitdegree.org/learn/how-to-install-git#git-install-for-mac-os)
* [**4.** Git Install for Linux](https://www.bitdegree.org/learn/how-to-install-git#git-install-for-linux)
* [**5.** Git Tools](https://www.bitdegree.org/learn/how-to-install-git#git-tools)
* [**5.1.** Git CMD](https://www.bitdegree.org/learn/how-to-install-git#git-cmd)
* [**5.2.** Git Bash](https://www.bitdegree.org/learn/how-to-install-git#git-bash)
* [**5.3.** Git GUI](https://www.bitdegree.org/learn/how-to-install-git#git-gui)
* [**6.** How to Install Git: Summary](https://www.bitdegree.org/learn/how-to-install-git#how-to-install-git-summary)

**How to Install Git: Main Tips**

* There are **many versions** of Git available **online** for all the operating systems.
* There are several ways of how to **install Git**: you can install **Git tools** either through package managers, or other installers or source code compilation on the computer.
* After completing the installation, you always **have to configure** the information of a user - set a **username and an email.**

**Git Install for Windows**

There are a few ways of how to get Git on Windows:

* Using **Git download file** for Windows.
* Using a **package manager** for Windows.
* Choosing to install it together with your favorite **Git GUI**.

The first way is quite straightforward: you can get the official Git version for Windows at certain websites that provide the download files. You can get the latest release of Git for Windows [here](https://git-scm.com/download/win) or [here](http://gitforwindows.org/).

After downloading the installation file from one of the listed websites, you will have to continue Git installation by yourself. If you are not so sure how to do it (or let's be honest, don't have the time), you can also have it automated by enabling a package manager. Git [Chocolaty](https://chocolatey.org/packages/git) is a good solution to use. Chocolaty is a package monitor for Windows run by the community.

Using automated installation is easy. After installing Chocolaty, just type this line into the command window choco install git -params '"/GitAndUnixToolsOnPath"'.

Another way to get Git on your computer is by installing [GitHub](https://desktop.github.com/)for Windows or any other Git GUI - [Sourcetree](https://www.sourcetreeapp.com/" \t "_blank), [GitKraken](https://www.gitkraken.com/" \t "_blank), etc. Keep in mind, though that even if it is tempting, it is not the most recommended version. Software based on graphic user interface usually has its limitations. Learning how to operate a command line might be a more efficient option as it will be easier for you to switch between the systems.

More information on how to set up the system after the installation will be provided later.

**Git Install for Mac OS**

There are several ways of how you can install Git on Mac OS. The list of them looks like the following:

1. Install the Xcode Command Line Tools.
2. Use download file, a binary installer.
3. Use a package manager.

The first way is one of the easiest. All you have to do is to install the Xcode Command Line Tools. If you already have it, newer versions of Mac OS may already have Git installed on your computer.

To check, type $ git --version in the terminal. This command will show the version of Git you have on your machine:

$ git --version  
git version 2.15.1

If you see that there is no Git on your Mac, you will be asked to run its installation. You can do this by running this command - git in the terminal.

If you want a more up-to-date Git version, you can install it via installer from special download source pages. For that, you will need to download the installation files for Mac. You can do this by visiting [Git website](http://git-scm.com/download/mac) or other websites like [SourceForge.](https://sourceforge.net/projects/git-osx-installer/files/" \t "_blank)

Like with Windows, you can install Git indirectly via package managers. You can do this by using [Homebrew](https://brew.sh/)or [MacPorts.](https://www.macports.org/" \t "_blank)

To start the installation with Homebrew type the following line to the terminal:  
$ brew install git

If you want to use MacPorts, firstly search for the most recent Git ports and options available. You can do that by typing in the terminal the following lines:

$ port search git  
$ port variants git

After that, type in the terminal:

$ sudo port install git +svn +doc +bash\_completion +gitweb

Finally, to finish your Git installation you can use an alternative to command line as well - GUI (Graphic User Interface) app like [GitHub](http://mac.github.com/)for Mac. However, it is important to have in mind that Git commands remain the same across different platforms, while GUIs might differ depending on the platform.

**Git Install for Linux**

The easiest way to get Git on Linux is through the binary installer. You can do this by using the general package-management tools that come with your Linux distribution. You have to type specific commands in your shell in order to begin Git installation.

For Fedora use **dnf / yum:**

$ sudo dnf install git-all

If you are working on distribution based on Debian, for instance, Ubuntu, Git packages are available via [apt:](https://wiki.debian.org/Apt)

$ sudo apt-get install git-all

You can find more options and instructions on Git installation for other Unix-like systems [here](http://git-scm.com/download/linux).

You can check how successful the installation was by typing the following command into the shell:

$ git --version

**Git Tools**

There are many ways to use Git. First of all, there is a number of **third-party tools** available to download and use for Git. Even though Git comes with GUI, it is highly recommended not to rely exceptionally on it but nail the command line, too. After you have mastered the command line Git version, you will relatively easily figure out how to use the GUI version.

**Git CMD**

As its name suggests, it is a command line **prompt interpreter**. If you use or have used Windows, you probably have seen one. If you work with Windows and are accustomed to using CMD, you can use it for git commands.

**Git Bash**

**Bash** in Git is an emulation of a Unix shell for Linux and Mac OS, so you can use it on Windows if you are used to working with Linux.

**Git GUI**

It is a Graphical User Interface. This allows to use Git without touching Git Bash or CMD and **avoid using command lines**. However, built-in GUI has its **limitations**. On the one hand, it is very minimalistic. On the other hand, many GUI with different capabilities only can implicate a partial Git functionality. This why this tutorial focuses on using the command line.

**How to Install Git: Summary**

* A number of different versions exist online. They can be dowloaded and installed depending on what operating system - **Windows, Mac OS or Linux** - is used.
* How to install Git depends on the user preferences, operating sytem and knowledge: you can use different installers, such as package managers, source code or **both**.
* In order to start using Git, you will always have to set up **user information**, such as **username and password.**

**How to Start Using Git and Set up Your Username Data**

**Reading time4 min**

**PublishedJan 3, 2016**

**UpdatedOct 3, 2019**

Is your chosen Git version well installed on your Windows, Mac OS or Linux system? If so, it is time to move on and start Git setup. The first step to do so is by setting an account of the user who is going to be in charge of a specific Git account.

It is a common practice among systems, and Git setup is no exception. You will have to choose a username and indicate an email which you will use to connect to the system for the first time and later on at Git login.

Don't worry, all this information is easy to change. Git username and email that you choose during Git setup can be seen and changed in the settings of Git, also known as Git config. What is more, once you learn how to manage Git config section, you will be able to see and manage all the possible options there.

Git setup begins with Git username and email, but there is more to do. For example, you will also be able to set a text editor of your choice. That is the very beginning of setting up your work environment at Git by using Git setup. All these actions are covered in the tutorial below, so start reading now.

**Contents**

* [**1.** Setup Git: Main Tips](https://www.bitdegree.org/learn/git-config#setup-git-main-tips)
* [**2.** Setting up Your Personal Information](https://www.bitdegree.org/learn/git-config#setting-up-your-personal-information)
* [**3.** How to Access Git Settings by One Code Line](https://www.bitdegree.org/learn/git-config#how-to-access-git-settings-by-one-code-line)
* [**4.** Git Setup: Summary](https://www.bitdegree.org/learn/git-config#git-setup-summary)

**Setup Git: Main Tips**

* The first thing you should do after Git installation is to create **Git login** by setting values for your Git config username and email.
* The personal details you will enter to the system will be used in revising the information you change - Git uses it for **commits**.
* Git has a **configuration tool** where you can check or change your user information, also start customizing your Git environment.

**Setting up Your Personal Information**

After the installation, there are a couple of things you must do on any operating system, not exceptionally Git - to set up your **username** and **email address**. Your username will be often referred to as Git config username and your email as Git config email.

This information is essential to **identify the user** which makes the changes or creates versions of the information. Git config username and email will be used with so-called commits or revisions which are individual changes to a file.

In order to set up your git username, you might want to type this command line into the console, only of course, with your own username and email address:

$ git config --global user.name "My Name"  
$ git config --global user.email myemail@example.com

**How to Access Git Settings by One Code Line**

You might also find yourself wondering how to **check settings**.

It's easy: if you want to see your Git settings information, just run the command below:

Git config --list

This command shows you the settings information.

Git config is a configuration (or settings) section where you can see and change your user information as well as to customize your work environment. One of such changes that you should do is to configure the default text editor. Git will use it to send messages to you. This step is optional, but if you choose not to set, Git will use the default system editor.

If you would like to use another text editor (for example Geany, Emacs, etc.) for Git, you can set it by typing:

git config --global core.editor geany

If you want, you can also check specific key values. For this, type a chosen config key. For example, let's say we want to find the Git username that you have chosen:

git config user.name

You might find it useful to type in git config to the console. By this command, you will be able to see all the possible configuration options.

**Git Setup: Summary**

* Starting using Git requires you to set a **username** and **an email** which in the system are names Git config username and Git config email.
* The username and email will be used for **it login** as well as to create commits and identify changes in the files you will later make.
* You can see and change Git setup information in the **Git config** section. All the other configuration options can be found there.

**Branches as a Way to Develop Changes**

**Reading time5 min**

**PublishedJan 12, 2018**

**UpdatedOct 3, 2019**

Chances are you have heard the term "branching" before. Why? Because it is a widespread feature among version control systems (VSC). Git branch is vital in development because it allows you to create a number of different development lines at the same time.

Imagine, if several developers are working on the same source code, they probably are not responsible for the same tasks. Each of them is devoted to solving a certain issue. By using the Git branch, they can isolate themselves from each other and work on their own field of focus or develop different versions of the code. When they work on distributed development, such as, for example, Git branch, later they can merge different versions. The only condition is that Git branch must belong to the same repository.

Git branch is quite significant among other version control systems. First of all, unlike many other version control systems out there, it actually encourages developers to use the Git merge branch. It is definitely worth learning how to master it because Git branch and Git merge are the features that are exceptionally fast and created for multiple renewals.

In this tutorial, you will learn how to take advantage of Git branch, and its features, not only such as Git merge branch, but Git create branch and Git switch branch as well. Keep reading the tutorial below!

**Contents**

* [**1.** Git Branch: Main Tips](https://www.bitdegree.org/learn/git-branch#git-branch-main-tips)
* [**2.** Local Access and Git Compression Techniques](https://www.bitdegree.org/learn/git-branch#local-access-and-git-compression-techniques)
* [**3.** Moving Around the Branches](https://www.bitdegree.org/learn/git-branch#moving-around-the-branches)
* [**4.** Creating Multiple Branches in Non-Linear Development](https://www.bitdegree.org/learn/git-branch#creating-multiple-branches-in-non-linear-development)
* [**5.** Git Branch: Summary](https://www.bitdegree.org/learn/git-branch#git-branch-summary)

**Git Branch: Main Tips**

* Git branches are **movable pointers of commits** that have been done by that time.
* Branching is a process when developers create **separate branches** from the same code but remain in the same repository.
* Essentially, it is an **independent** development **line**.

**Local Access and Git Compression Techniques**

A **branch** is a term used to describe a unit in which Git stores data about changes made throughout the development or a representation of pointers to snapshots of changes.

That is because Git doesn't store changes as a list of file-based changes; instead, it considers data to be a set of snapshots of a particular files system. As a result, each time you save changes of a project in Git, it immediately takes a picture of what all the files look like.

Normally, one might think that the local repository of a project would take a lot of space of the machine because of Git saving files locally. However, Git provides an **effective solution** to this issue. Git uses certain **compression techniques** so that the files are stored on your local disk occupying less space than they would normally. After a snapshot is taken, a reference point is saved, so only the information about the changes is stored. Meanwhile, no action is taken towards the files that have already been stored but haven't experienced any changes so far.

For most operations, Git uses locally stored resources and saves files **locally**. This allows you to work and save changes offline as well (locally). The whole history of a project is accessible in the local database, and every new change can be saved at a local disk.

Git create branch is a simple command, similar to many other Git commands:

git branch mybranch01

It is also important to note that there are different types of branches - **local** and **remote** branches. You will learn how to work with them through code later.

**Moving Around the Branches**

Normally, after some time of working with Git, you will have more than one or two branches. It becomes important then not to get lost among them. There is a useful command for that, **Git switch branch**. Just like the name says, it will allow you to move from one branch to another.

For Git switch branch, use this code line by including the name of your Git branch:

**git checkout <branch name>**

You can go back to the main branch by typing **git checkout master**. Master is your main project branch. At any moment, after you have made a few branches you can check where you are by using **git status**command**.**

$ git status  
On branch mybranch01

You might want to use **git branch**to view all the branches you have created by now:

$ git branch  
master  
\* mybranch01

**Creating Multiple Branches in Non-Linear Development**

Branching is an integral part of the Git workflow. One of the reasons why Git stands out from other source code managers (SCM) is its supported non-linear code development. This type of development is supported by branching.

The master branch is the main Git branch. When something is changed, **new branches** are immediately created, despite the extent of the change. All the branches are indispensable to each other. Such structure protects the work that is being done in the master branch.

Before the progress is merged or stored, everything is check-summed by using SHA-1 Hash mechanism. It ensures that all the changes will be detected by Git, even if the information is lost or damaged during the transit.

All in all, Git branch and its development process is a fast and effortless content moving and switching way. It allows developers to experiment with the code base and use Git merge to integrate the desired changes.

**Git Branch: Summary**

* Git branch system tracks the changes that have been done **simultaneously**.
* It allows to create changes **independent** from the main line.
* Branching allows to **experiment with the code** and remain in control of it at the same time.

**How to Create and Manage Your First Git Repository**

**Reading time7 min**

**PublishedJan 16, 2018**

**UpdatedOct 3, 2019**

Wherever you work, whether on your local computer or cloud, the files and changes must be saved somewhere. That is the reason why Git repository is one of the basic and most popular functions to use. Git repository, just like the name indicates, serves as a storage for all the changes and files that relate to a certain project. That is the shortest answer to the question of what is a repository.

Initializing Git repository is quite simple. However, if you want to learn how to work with Git repository properly, you must understand how and why files are stored there. Some Git repositories can be local, placed directly at your local computer. You can use an already existing directory as a repository for your Git files or create a brand new one.

If you work in a team or you are invited to make changes to a particular code, chances are you will need to access a remote Git repository. Git clone command or Git clone repository are the names for a command that create a local version for you of that remote Git repository so that you could create your own changes without any damage to the remote version.

In this tutorial, you will be explained how to clone a Git repository and how to initiate one from the very beginning.

Keep reading below!

**Contents**

* [**1.** Git Repository: Main Tips](https://www.bitdegree.org/learn/what-is-a-git-repository#git-repository-main-tips)
* [**2.** The Basics of Repository](https://www.bitdegree.org/learn/what-is-a-git-repository#the-basics-of-repository)
* [**3.** How to Access Git Repository](https://www.bitdegree.org/learn/what-is-a-git-repository#how-to-access-git-repository)
* [**4.** How to Initialize Git Repository](https://www.bitdegree.org/learn/what-is-a-git-repository#how-to-initialize-git-repository)
* [**5.** Creating Git Repository in an Already Existing Directory](https://www.bitdegree.org/learn/what-is-a-git-repository#creating-git-repository-in-an-already-existing-directory)
* [**6.** Adding Files to Git Repository](https://www.bitdegree.org/learn/what-is-a-git-repository#adding-files-to-git-repository)
* [**7.** Getting to Know Your Git Repository](https://www.bitdegree.org/learn/what-is-a-git-repository#getting-to-know-your-git-repository)
* [**8.** Cloning Remote Git Repository](https://www.bitdegree.org/learn/what-is-a-git-repository#cloning-remote-git-repository)
* [**9.** Git Repository: Summary](https://www.bitdegree.org/learn/what-is-a-git-repository#git-repository-summary)

**Git Repository: Main Tips**

* Git has a place called **repository**, sometimes shorten as **repo**, where your Git **projects** are **stored**.
* All **changes** in a **project** and versions of **saved** **files** are in its repository.
* Git **clone command** is used to create an identical copy of remote Git repository, but it can also be placed locally, on developer's computer .

**The Basics of Repository**

What is a repository? To put it simply, it is a place where all the files of a certain project are placed. It can be both, either a **remote** storage space or a **locally accessible** directory. For instance, storage space in the cloud, an online host or local folder on your computer all can serve as repositories. In this particular folder, sometimes also abbreviated as a repo, Git saves all the files and project-related information, such as changes history.

There are two types of repositories - **local**and **remote**. Correspondingly, there are two options to work with Git: to start an entirely new project or join an already existing one.

**How to Access Git Repository**

One of the easiest ways to get a Git repository is to use a local directory. It must be free, i.e., not yet used for a version control system work.

Most of the work with Git is done locally. However, it is important to keep in mind that none of the files are tracked until a user **asks Git to start** doing so - that will make the files version-controlled. Initialising a repository is exactly a way to ask Git to enable version control on the files held in that repository.

**How to Initialize Git Repository**

To begin with Git repository, you have to go to the project's that you intend to control directory. Here you have to run 'git init' command. This process is called **initializing a repository**.

$ git init

This command creates a new Git repository. In general, it is used to initialize an empty new repository or convert an existing unversioned project into a Git repo.

You need a folder in which you will hold all the files of your project. One way to do it is simply manually. First of all, choose a location where you want to create a folder. In this example you can see how the "Git lessons" folder is created on the desktop:

That is done by typing this command into a new terminal window:

$ cd Desktop/Git lessons/  
$ git init

Another way to do the same is by right-clicking your mouse, opening a Git bash and typing Git init.

You should see a message similar to this one with Git lesson folder created on Desktop:

Initialized empty Git repository in .../Desktop/Git lessons/.git/

**Creating Git Repository in an Already Existing Directory**

Instead of creating a brand new folder for your Git repository, you can **overtake** an already existing directory. How to initiate changes to that directory depends on which operating system you are using. In this particular example a directory is created here: user/my\_new\_project.

**If you are using Windows**, type:

$ cd c/user/my\_new\_project

**For Mac OS**:

$ user/username/my\_new\_projectcd

**For Linux**:

$ cd /home//user/my\_new\_project

You have indicated which directory you want to use. Now type in this command to initiate Git repository:

$ git init

With the last command line, you make a new Git repository existing. Its subdirectory **.git** will have all the repository files. No changes are tracked yet, though - that will be covered in the other tutorial.

**Important:** If you want to start a version-control directory which was not empty before, first you need to start to track it and only then do an initial commit.

**Adding Files to Git Repository**

Adding more files to the Git repository you have created might be quite useful.

In order to do that you have to create a new Git repo with the command, you have just learned: $git init. After that you have to execute two other commands: add and commit. The full code should look like the following:  
$ git init  
$ git add --all  
$ git commit -m 'this message shows I have initiated a commit'

**Note:**It is important to leave messages when saving your files.

**Getting to Know Your Git Repository**

It is useful to know where you are while moving around the directories. You can check your location by typing this command into the terminal or Git Bash:

$ pwd

The result should be something similar to this location, depending on where your folders have been saved:

User/Desktop/Git lessons

It is also essential to be able to check the **current status** of your repository. Running the following command in a newly created repository should provide you with similar answers from Git:

$ git status  
On branch master  
No commits yet  
nothing to commit (create/copy files and use "git add" to track)

**Cloning Remote Git Repository**

**Git clone** repository is another useful command that developers often use at Git. It is quite common that you work on a shared project and the repository of it is placed somewhere remote. In this case to contribute to the code you might need to get your own clone of the repository. You will be explained here how to clone a Git repository to your local computer.

In order to clone Git repository you have to use **git clone [url]** command. For this example, a linkable Git repository is at GitHub. If you want to clone the linkable library, navigate to the directory where you want to have the copied Git repository and run this command with your folders names:

$ git clone https://github.com/folder/folder

**Git Repository: Summary**

* Git system stores and saves the files to a directory called **repository**, also abbreviated as **repo**.
* Git repository is used to **save the information** about the changes in the project.
* Repository can be created **locally**, or it can be a clone Git repository which is a copy of a **remote Git repo**.

**How to Create Remote Repository for Collaboration**

**Reading time4 min**

**PublishedJan 31, 2018**

**UpdatedOct 3, 2019**

So far, you have learned how to operate and manage your work in your own environment. You know how to initiate Git repositories and what processes lie behind saving Git files. However, if you plan to work in a team of several developers, you must have a remote directory for your collaboration. That is where you need to learn how to set up a Git server.

Setting up Git server is relatively easy - you only need to know a particular command which you will have to run. Your remote repository will be a collaboration point for the developers who work on the same project. It is convenient as you will be able to access it even if your device is offline. That adds more reliability to work.

In this tutorial, you will learn not only how to set up Git server, but also how to obtain public SSH keys for every user. This is a common method among various systems to authenticate the users and provide or deny access to remote places. Getting SSH keys is essential so that each of them could access the remote repository after setting up Git server.

All the necessary information is below, so let's dive in!

**Contents**

* [**1.** Setting up Git Server: Main Tips](https://www.bitdegree.org/learn/git-server#setting-up-git-server-main-tips)
* [**2.** Initiating .Git Repository](https://www.bitdegree.org/learn/git-server#initiating-git-repository)
* [**3.** Git Server Configuration and SSH Keys](https://www.bitdegree.org/learn/git-server#git-server-configuration-and-ssh-keys)
* [**4.** Setting up Git Server: Summary](https://www.bitdegree.org/learn/git-server#setting-up-git-server-summary)

**Setting up Git Server: Main Tips**

* For your own work you can use **local** repositories.
* For **collaboration** you will need a **remote repository** where all the common work will be saved.
* In order to have a remote repository, you will need to **set up Git server** and obtain **SSH public key**.

**Initiating .Git Repository**

Technically, it is possible to save your collaborative work in one of the private repositories; however, it is **severely recommended** not doing so. It is easy to confuse the repositories and accidentally destroy the work that has been done. Therefore, it is a common practice to run a Git server for a **remote repository.**

This particular repository is bare as it doesn't have any working directory in it - it simply is a **collaboration point** for your team. Such repositories have the extension **.git.** To initialize such a repository, you have to run the following commands:

$ pwd  
$ mkdir my\_repo.git  
$ cd my\_repo.git  
$ ls  
$ git --bare init

You can check the status of the process by the command **$ ls**. If you successfully created a repository, the received response should be as the following, only with your location information:

$ ls  
Initialized empty Git repository in /home/gituser-m/my\_repo.git/

**Git Server Configuration and SSH Keys**

In order to have your remote repository placed somewhere accessible remotely, you must **run a Git server**. The Git server setup is not hard at all - you need to do some configuration, and each user of the Git server must obtain **public SSH keys**.

Before you begin to setup Git server, it might be a good idea to check if you have a key in general. The keys are usually placed in the **~/.ssh** directory. To check it, you should run the following command:

$ cd ~/.ssh  
$ ls  
authorized\_keys2 id\_dsa known\_hosts  
config id\_dsa.pub

If you have a file with **.pub** ending, this is your public key. The file above is a private key; usually, they are named **id\_rsa** or **id\_dsa**.

If you do not have a key, you will see a message that no such directory exists. If this is the case, then each user of the system have to **generate it**. To be able to obtain the keys, run this command:

$ ssh-keygen

If executed successfully, you will be informed that a pair of **rsa** keys are being generated. One of them is public and the second one is private. After that you will be told where the keys have been placed and their fingerprint information.

Each user has to generate **their own** key in order to be authenticated by the Git server. They should send their public key to you or the person who is in charge of setup Git server and its administration. To do that is easy: they have to copy the contents of the **.pub file** and send this information to the administrator.

**Setting up Git Server: Summary**

* Local repositories are the most convenient for your **own work**.
* It is a common practice to have a **remote repository** for the **collaborative work**.
* **Git server setup** is the primary step for a remote repository, each user of it also must get **SSH** public keys.

**A Useful Git Cheat Sheet: Learn About File Stages and Common Commands**

**Reading time4 min**

**PublishedJan 31, 2018**

**UpdatedOct 3, 2019**

This Git cheat sheet includes information necessary for managing and updating your projects.

**Contents**

* [**1.** Git Cheat Sheet: Main Tips](https://www.bitdegree.org/learn/git-cheat-sheet#git-cheat-sheet-main-tips)
* [**2.** Git: Best Practices](https://www.bitdegree.org/learn/git-cheat-sheet#git-best-practices)
* [**3.** The Life Cycle of Git Files](https://www.bitdegree.org/learn/git-cheat-sheet#the-life-cycle-of-git-files)
* [**4.** Three Stages of Edited Files](https://www.bitdegree.org/learn/git-cheat-sheet#three-stages-of-edited-files)
* [**5.** Version Control System](https://www.bitdegree.org/learn/git-cheat-sheet#version-control-system)
* [**6.** How Git Works: Step by Step](https://www.bitdegree.org/learn/git-cheat-sheet#how-git-works-step-by-step)
* [**7.** Git Cheat Sheet: Summary](https://www.bitdegree.org/learn/git-cheat-sheet#git-cheat-sheet-summary)

**Git Cheat Sheet: Main Tips**

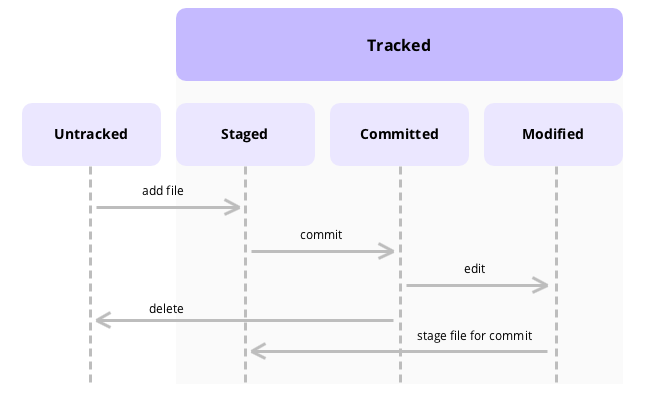
* Files in Git have **several states**: **modified, staged**, or **committed**.
* Git is a **version control system** meaning that it helps users keep track and manage changes of projects. It is also possible to **revert** to older versions.

**Git: Best Practices**

When using Git for your projects, it is best to set **certain rules** and **limitations** for managing your work. Here are a few best **practices of Git** that you and your contributors should follow:

* Submit commits that address a **particular issue or change**. When you are updating one specific section of your project, do not make other modifications.
* Organize your work and **make commits often**. Commits make sure that nothing gets lost.
* Every commit should be **described clearly**. Indicate the **changes** you made to make sure that other contributors can follow up easily.
* **Do not change the history** of commits unless specific conditions require you to do it.
* Test your changes **before committing**. If your commit fixes a bug, make sure that it does.

**The Life Cycle of Git Files**



When working with Git repositories, you will **add**, **delete**, and **edit** files (with **init, add** and **commit** commands).

You need to learn that files in your working directory can be either **tracked** or **untracked**. Here is an explanation of how does Git files work:

* Tracked files are the ones that Git has been informed about. They can be **modified, unmodified**, or **staged**.
* **Untracked files** refer to files that are not in the **staging area** and not in the **latest snapshot**.**Three Stages of Edited Files**

In general, Git sees the files you edit as **modified**. It is standard that you **stage** the modified files and then **commit** the changes you have made.

Here is a short explanation of each state:

* **Modified** — the file has been **changed**, but **not yet committed** to the database.
* **Staged** — the **current version** of the modified file has been proposed by you to commit.
* **Committed** — means that the **changes to data** have been stored in the database.

**Version Control System**

The Git version control consists of **making snapshots** of projects. In other words, version control **keeps track** of all your changes and allows **reverting** in case the changes bring issues.

**How Git Works: Step by Step**

This list shows **how does Git work**:

* You begin with **cloning Git repository** as a working copy.
* By editing or adding files, you **modify** the working copy.
* You **update** the working copy by taking over the changes other developers made.
* You **review** the changes and **decide** if and which ones to commit.
* You check if there are no mistakes and move the changes to the **repository** with the Git **commit** command.
* If you notice mistakes or issues, you can always **correct** the last commit and update the changes.

To describe this Git workflow cheat sheet in a more terms-filled manner, it would look like this:

* You modify or change files in a **working tree** (also called **working directory** or area).
* Select and stage only the files that you want to commit. Those particular changes are added to the **index** (a technical term for **staging area**).
* You commit a snapshot of those changed files, and they land to the **repository** (**.git directory** or the **HEAD**).
* Do not be confused over the use of **commit** and **push**. **Commit** command adds changes to the **local repository**, while **push** does the same but sends commits to **remote servers**.

**Git Cheat Sheet: Summary**

* Best practices of Git include **committing often**, **clearly describing changes**, and **focusing on one issue in one commit**.
* There are several file **stages**. Make sure that Git saves the files and changes to them.

**Steps of Saving Changes: Git Commit Command, Staging and Pushing**

**Reading time3 min**

**PublishedJan 30, 2018**

**UpdatedOct 3, 2019**

Git commit command is the main function for saving changes to local repository safely. Commit takes a snapshot of the project and treats it as a version.

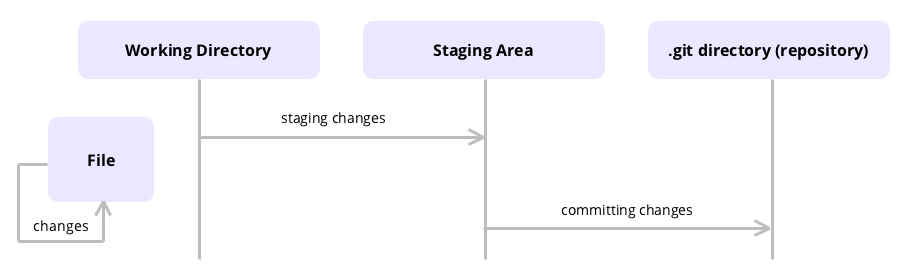
**Contents**

* [**1.** Git Commit Command: Main Tips](https://www.bitdegree.org/learn/git-commit-command#git-commit-command-main-tips)
* [**2.** Staging and Committing](https://www.bitdegree.org/learn/git-commit-command#staging-and-committing)
* [**2.1.** Staging Changes](https://www.bitdegree.org/learn/git-commit-command#staging-changes)
* [**2.2.** Committing Changes](https://www.bitdegree.org/learn/git-commit-command#committing-changes)
* [**3.** Pushing Changes to Remote Servers](https://www.bitdegree.org/learn/git-commit-command#pushing-changes-to-remote-servers)
* [**4.** Git Commit Command: Summary](https://www.bitdegree.org/learn/git-commit-command#git-commit-command-summary)

**Git Commit Command: Main Tips**

* In Git, **commit** is the term used for **saving changes**.
* Git **does not add changes** to a commit automatically. You need to indicate which file and changes need to be saved **before** running the Git commit command.
* The **commit** command does not save changes in **remote servers**, only in the local repository of Git.

**Staging and Committing**



Staging in Git refers to a **phase** which includes all the changes you want to include in the **next commit**. Here are the steps leading to staging:

* You make **changes** to a file in the working directory.
* You use the Git **add** command to move those changes from the working directory to the **staging area**.
* Git does not save changes yet. You need to run the Git **commit** command to move changes from the staging area to the **local repository**.

Additionally, you may use **git status** command to check the status of the files and the staging area. If changes are not staged for commit, they **won't be saved**.

**Note:** do not confuse **git add** with **svn** **add** command. **svn add**creates a Git clone from any repository that belongs to Subversion while Git commit command finalizes the changes.

**Staging Changes**

The Git **add** command moves changes to the **staging area**.

Here is a basic example of using **add**:

git add <file>

Git moves all changes of **<file>** in the staging area to wait for the next commit.

This example adds the entire **<directory>** to the staging area:

git add <directory>

During the Git **add session**, you can pick the changes you would like to **commit**. Before that, you need to start an **interactive session**:

git add -p

* Use **y** to stage a specific portion.
* Use **n** to ignore a specific portion.
* Use **s** to divide the portion into smaller parts.
* Use **e** to edit the portion manually.
* Use **q** to exit the interactive session.

**Committing Changes**

Git **commit** command takes a **snapshot** representing the staged changes.

git commit

After running the Git commit command, you need to type in the description of the commit in the **text editor**.

This Git commit example shows how you set the **description** with the **commit** function:

git commit -m "<message>"

The following example shows how to save a snapshot of changes done in the whole working directory. This code only works for **tracked files**.

git commit -a

**Pushing Changes to Remote Servers**

Git **push** command **moves** the changes from the local repository to a **remote server**.

This example creates a **local branch** in the remote repository, including all specified commits and objects:

git push <name of remote server> <branch name>

The following example **pushes changes** even if it does not end in a non-fast-forward merge. Beginners should not use this option:

git push --force

This example adds all local branches to the indicated remote repository:

git push --all

The following example pushes all local tags to the remote:

git push --tags

**Note:** the Git **push** command makes sure that you share your changes with remote colleagues.

**Git Commit Command: Summary**

* The Git committing process requires **several steps**: moving changes to the staging area and saving them with the **commit** command.
* Once you move the changes to the **local repository** by using Git **commit** command, you can use Git **push** to transfer them to a remote repository.

**Moving, Renaming and Removing Files at Git Repositories**

**Reading time4 min**

**PublishedJan 31, 2018**

**UpdatedOct 3, 2019**

After getting familiar with the previous tutorials, you can now create files, Git add folder and save changes in repositories. For this, you always use specific commands. However, mistakes happen and sometimes not all the files end up placed where they are supposed to be. Sometimes you will find yourself needing to move a file from one repository to others or simply to delete it.

How does Git move files? How to remove a file from being tracked and exchange it with another one? These are common questions to ask; however, in the language of Git, it would sound more like how to undo files from the staging area. You can remove a file by using Git rm command and add a new one or do the same action by executing Git move files command abbreviated as Git mv. The result is the same: a file will be removed from the repository.

If you simply delete a file from your working directory, it will be regarded as a change which has not been committed. You will learn in this tutorial how to make these changes permanent and how to use Git move files, and Git mv commands in order to freely operate your Git files.

Start reading below!

**Contents**

* [**1.** Git Move Files: Main Tips](https://www.bitdegree.org/learn/git-mv#git-move-files-main-tips)
* [**2.** Removing Git Files](https://www.bitdegree.org/learn/git-mv#removing-git-files)
* [**3.** Moving Git Files and the Process Behind It](https://www.bitdegree.org/learn/git-mv#moving-git-files-and-the-process-behind-it)
* [**4.** Git Move Files: Summary](https://www.bitdegree.org/learn/git-mv#git-move-files-summary)

**Git Move Files: Main Tips**

* Just like you can use Git add folder or files, so can you **delete them**, too.
* You can use **git rm** command to remove the files from a repository and **git mv** command to remove and instantly add a new file instead of the old one.

**Removing Git Files**

**git rm** command is usually used when a developer wants to remove a certain file from the working tree and the index, **not only** from the working directory. This process is called **Git remove** or **Git remove file**. It means that after running this command on certain files, changes in them will not be staged and you will not see it as a tracked file the next time you perform a status check.

There might be cases when you want to keep a file on your hard drive, but want that Git would **stop tracking** it or you want to remove one of them from the staging so that you wouldn't commit it accidentally. To do this, add **--cached** option:

$ git rm --cached text.txt

Just like you stage multiple files, you can do the same only in reverse, by removing batches of files:

$ git rm --\\*a

Git remove or Git remove file command will eliminate all the files with 'a' name ending. If you use, for example, .db, it will remove all the files with db extensions.

**Note:**Add a **/** forward slash for the directory, e.i. **myproject/**. There is a backslash in an example above **\** in front of**\*** because Git has its own file extension.

**Moving Git Files and the Process Behind It**

Paradoxically, even though Git is a file version control system, it doesn't track the movement of the files. It means that no metadata is saved if you rename or move a file. However, **Git mv** command facilitates the process as you don't need to manually delete a file and add a new one. You can use it as in the following example:

.$ git mv from\_\_this\_file to this \_file

The same can be done by **removing a file and adding a new file** with the same content. You can use other ways to rename files as well, only that it will require you to run more commands as you still will have to add the file and remove the other version by using **rm***:*

$ mv file\_name1 file\_name2  
$ git rm name1  
$ git add file\_name2

**Git Move Files: Summary**

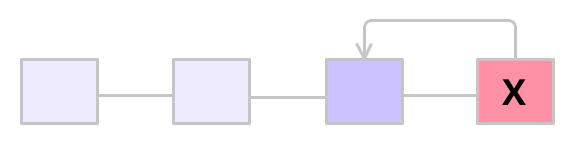
* Files are **flexible** to move from one place to another or to be deleted completely.
* **Git rm** and **Git mv** are two main commands to execute these actions.

**How to Revert Undesired Changes**

**Reading time5 min**

**PublishedJan 31, 2018**

**UpdatedNov 6, 2019**



It would be hard to work with any version control system, including Git if you could not return any changes to their original state. That would make every mistake very pricey as you would have to start the same work over and over again until you master the workflow to perfect. Luckily, the Git revert commit function and its variations allow to undo the changes and suggest several smart ways of how to secure them with a new commit.

Git undo commit options have been designed for different actions while working with Git files. With them, you can remove files from the staging area, edit recent commits, see and edit the timeline of commits or even invert changes and finalize them further with a new Git revert commit.

Learning how to use Git revert commit and its variations, such as Git undo commit, or Git revert is essential if you want to work with your Git files freely. You should note as well that some functions, for example, Git checkout are very broad and can be used for different purposes. In this tutorial, you will learn how to use it and other commands for undoing the undesired changes.

Keep reading!

**Contents**

* [**1.** Git Revert Commit: Main Tips](https://www.bitdegree.org/learn/git-revert-commit#git-revert-commit-main-tips)
* [**2.** Editing the Recent Commit](https://www.bitdegree.org/learn/git-revert-commit#editing-the-recent-commit)
* [**3.** Unstaging Files](https://www.bitdegree.org/learn/git-revert-commit#unstaging-files)
* [**4.** Git Checkout Command](https://www.bitdegree.org/learn/git-revert-commit#git-checkout-command)
* [**5.** Cleaning Untracked Files from Repository](https://www.bitdegree.org/learn/git-revert-commit#cleaning-untracked-files-from-repository)
* [**6.** Git Undo Last Commit](https://www.bitdegree.org/learn/git-revert-commit#git-undo-last-commit)
* [**7.** Securing Change with New Commit](https://www.bitdegree.org/learn/git-revert-commit#securing-change-with-new-commit)
* [**8.** Git Revert Commit: Summary](https://www.bitdegree.org/learn/git-revert-commit#git-revert-commit-summary)

**Git Revert Commit: Main Tips**

* There are **several options** of how to edit changes and Git revert to previous commit in the system.
* Some of them, for example, **Git checkout**, can perform a number of actions.
* Git revert commands or Git undo commit is a convenient way to **cancel the undesired changes**.

**Editing the Recent Commit**

You can begin the changing process by editing a recent commit. Git has a specific command for it:

$ git commit --amend

It will allow you to add more or **edit the changes** of the recent commit. It can be used for alterations as little as editing the commit message.

**Unstaging Files**

The purpose of this command is to **unstage the staged files** in such a case if you staged a file and changed your mind or realized you chose a wrong file. Essentially, it performs **Git undo add** of the files you were planning to stage but changed your mind.

An exampe of Git undo add looks as the following:

$ git reset <file>

**Git Checkout Command**

Git checkout is used to **go back** to the file version of the **previous commit** - to the previous set of changes committed. Git checkout is run by this line:

$ git checkout -- <file>

In general, this command switches branches or restores files in the working tree. It can be used to discard changes in the **working directory**, for a particular file that you specify when you use this command. Git checkout is a broad command that can be used for actions other than undo.

Sometimes it might be useful to see the **entire timeline** of commits made. You can do that by running **git log**. By running it, you can select a specific commit you would like to go back to.

**Cleaning Untracked Files from Repository**

Git clean command is used for untracked changes, i.e., changes that are in the working directory but haven't been added to staging by Git add. It is a common practice to remove the untracked files from repositories.

For this purpose, **git clean** is used. You can use various options of it, for example:

$ git clean -n

This command will show you a **list of files** that are about to be removed. It is a good idea to run this list before actually removing the files just to be sure you have chosen the right ones.

Then, to initiate the actual cleaning, you can force it by using this line:

$ git clean -f

or

$ git clean --force

This command will remove the untracked files, except those marked with **.gitignore** suffix and the ones specified in the **ignore** file.

You can also specify the path to a specific file you would like to remove by this command:

$ git clean -f <filepath>

**Git Undo Last Commit**

Sometimes you might need the simple plain **undo version** of Git revert. For that, you can run this command:

$ git reset --soft HEAD~1

As you can see, it will bring the HEAD branch to a certain revision, in this particular case - one revision back. Note that **--soft** indicates that the changes in undone revisions are maintained. So basically, this function is useful for developers who need to Git undo last commit.

**Securing Change with New Commit**

Git revert command is a smart option of how to execute a process, similar to "undo" in many other systems. Instead of simply undoing the commit, it returns to the changes represented by a commit and **secures a new commit** with the redefined changes.

This command is useful for simple changes by referring to a certain commit in the past. That's why it is called Git revert commit sometimes:

$ git revert

To execute Git revert, the tree HEAD commit must be without any recent modifications.

A simple usage of this command is typing in **git revert HEAD** to go back to the last commit before the current one.

**Git Revert Commit: Summary**

* Git revert to previous commit and undoing changes in Git is a **well-developed function**.
* **Git checkout** command among others have a broader range of functions than a simple "undo".
* Git revert commands are **the most convenient** way to undo the changes.

**How to Untrack Files and Examine Repository**

**Reading time5 min**

**PublishedJan 31, 2018**

**UpdatedOct 3, 2019**

After working with Git for a while, cloning a repository or making a couple of commits, you will soon want to Git untrack file or to check the status of the processes that have been made in your repository. There might be a number of reasons why you want to Git untrack file. Not all the files need to be tracked - some of them can be easily created again, or they occupy the workflow. In most cases, files falling to Git untrack file category include various machine generated files.

Another thing that might be important to you is how to check Git commit command history - of your own or other users. In the previous tutorials, you have learned to use one command to check the information at the working directory: git status. However, this command, even though it shows all the untracked, unstaged and staged files, it does not include the information about the committed project history. There exist another powerful function which enables you to perform a careful investigation on your repository: git log. It can perform several actions that help you examine your repository, for example, to search the repository according to Git commit message.

All the information about how to Git untrack file and check the status of a repository is below, so start reading now!

**Contents**

* [**1.** Git Untrack File: Main Tips](https://www.bitdegree.org/learn/git-log#git-untrack-file-main-tips)
* [**2.** Untracking and Ignoring Files](https://www.bitdegree.org/learn/git-log#untracking-and-ignoring-files)
* [**3.** Investigating Repository History](https://www.bitdegree.org/learn/git-log#investigating-repository-history)
* [**4.** Git Untrack File: Summary](https://www.bitdegree.org/learn/git-log#git-untrack-file-summary)

**Git Untrack File: Main Tips**

* You can use **.gitignore** to untrack files.
* To check Git add, Git add all and Git commit history in a repository you can use **git log** command.
* By **adding** an **option** to **command,** you can narrow down or widen the range of an operation.

**Untracking and Ignoring Files**

There might be cases when you don't want to start tracking some files. Then you can add all of them to **.gitignorefile** and basically execute Git untrack file. First of all, you should create a file **.ignore**. Secondly, open it in a text editor and list all the files on a separate line. **'#'** is used for comments and **'\*'** to describe the type of the files. For example:

# Files to ignore  
\*.dex  
\*.class  
bin/  
gen/

Note that if you are on Windows, you will not be allowed to create an empty file with .gitignore. Fortunately, Git has a **tool** that you can use to create such a file. You should type this into a terminal:

$ touch .gitignore

This command will create the file .gitignore. If you run status, you will see that there is an untracked file:

Untracked files:  
(use "git add <file>..." to include in what will be committed)

.gitignore

The next step is to **add and commit** the .ignore file. After executing *add*command, you will see git ignore as a new file and after *commit* it will disappear. If you rerun Git status, you will not be able to see that file anymore.

To see all the files that reside in a repository, run **ls -a.** This command will show all the tracked and untracked files. This command should show your .gitignore file:

$ls -a  
./ ../ .got/ /gitignore firstfile.txt

**Investigating Repository History**

The other command that you can use is a more elaborated tool to view the history of Git commit command and inspect the working directory with the staging area. You can try to inspect the history of the project by typing this line:

$ git log

It will display the **entire Git commit history**. If no arguments are added, **git log** by default will list all of the commits with its ID - [SHA - 1](https://en.wikipedia.org/wiki/SHA-1)[checksum](https://en.wikipedia.org/wiki/Checksum)from the latest to the oldest. Information about the commit's author's name and email, date, and a message will be displayed.

There is a vast variety of options available with Git log command. If you want not only to see the history of commits but also to create changes in the files, you should use this command: **git log --stat.**

Check the example below:

commit e8353493b09b6ffa095cc10568a7fca5ab6428ce

Author: Your Name &lt;your@mail.com&gt;

Date: Mon Jan 22 15:18:42 2018 +0200

edited the version

.gitignore | 2 ++

first\_file.text | 3 ++-

2 files changed , 4 insertions(+), 1 deletion(-)

As you can see, this option shows the commits and changes to the files - how many files were changed and the number of added or deleted lines in them.

Another helpful option is **-p/--patch,**which provides the most detailed view of the project history. The command for this is - **git log -p**or **git log --patch.**Not only will it show the number of the changed lines but also the changes themselves.

If you have to look at many commits, you can see a more compact report by adding an option **git log --oneline:**

$git log --oneline

9f6f817 (HEAD -> master) Committed few things for example

0519da5 A small way around the staging area

As you can see, this will print all the commits in one line.

Similarly, you can use the option to narrow down and limit what log information to print on your screen. You can reduce the number of commits shown with **git log -n <limit>.**You can search for commits in a particular time interval with **git log <since>...<until>.**

If you work with other people, it might be useful for you to check the Git log of a single user commits sometimes. You can find all the commits done by a certain author by typing **git log --author="author\_name".**You can also list all the commits of a specific file by using **<file>** option or find a certain file by a Git commit message.

**Git Untrack File: Summary**

* **.gitignore** is the command used to Git untrack file.
* **git log** command is used when you want to check Git add, Git add all and Git commit history made by you or other user.
* It is possible to **widen an operation** by adding an option to the command.

## Getting started with JupyterLab

The [*installation guide*](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html) contains more detailed instructions

### **Installation with mamba or conda**

JupyterLab can be installed with mamba and conda:

mamba install -c conda-forge jupyterlab

or

conda install -c conda-forge jupyterlab

Note: If you have not installed mamba or conda yet, you can get started with the [miniforge](https://github.com/conda-forge/miniforge" \l "mambaforge) distribution.

### **Installation with pip**

If you use pip, you can install it with:

pip install jupyterlab

If installing using pip install --user, you must add the user-level bin directory to your PATH environment variable in order to launch jupyter lab. If you are using a Unix derivative (FreeBSD, GNU / Linux, OS X), you can achieve this by using export PATH="$HOME/.local/bin:$PATH" command.

### **Run JupyterLab**

Once installed, launch JupyterLab with:

jupyter-lab

## Getting started with the classic Jupyter Notebook

### **Installation with mamba or conda**

The classic notebook can be installed with mamba and conda:

mamba install -c conda-forge notebook

or

conda install -c conda-forge notebook

### **Installation with pip**

If you use pip, you can install it with:

pip install notebook

Congratulations, you have installed Jupyter Notebook! To run the notebook, run the following command at the Terminal (Mac/Linux) or Command Prompt (Windows):

jupyter notebook

See [Running the Notebook](https://jupyter.readthedocs.io/en/latest/running.html#running) for more details.

## Getting started with Voilà

### **Installation with mamba or conda**

If you use mamba or conda, you can install it with:

mamba install -c conda-forge voila

or

conda install -c conda-forge voila

For more detailed instructions, consult the [installation guide](https://voila.readthedocs.io/en/stable/install.html).

### **Installation with pip**

If you use pip, you can install it with:

pip install voila

Top of Form