# Introduction

It is an introduction to Git that is currently the most widely used version control system in the world, mostly thanks to GitHub. It’s also the most misunderstood version control system in the world. Git become much easier to understand once you figure out how they really work. The goal of this guide is to shed some light on how Git works under the hood.

It took me a pretty long time to really get Git. As I’ve continued to use Git more and more where I work, I’ve found myself trying to teach people what it is and why we use it over and over again, and the reality is that Git generally has a pretty steep learning curve compared to many other systems.

Git is an open source distributed version control system created in 2005 to manage the entire Linux kernel. Instead storing file information in a central repository, Git gives every developer a full copy of the repository

## History of Revision Control

A version control system is a software designed to keep track of the changes made to files over time. There are a number of benefits to using VCS including the following:

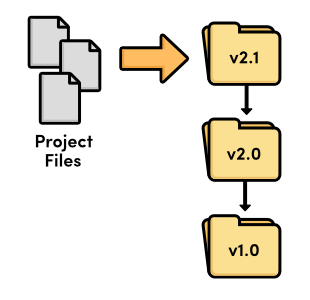
* The ability to undo changes. You can recover an earlier version of you work
* A complete history of all the changes
* Documentation of why changes are made. Often it is hard to remember why a change was made
* Multiple streams of history.

Working on a team, VCS provides a number of additional benefits

* The ability to resolve conflicts
* Independent streams of history.

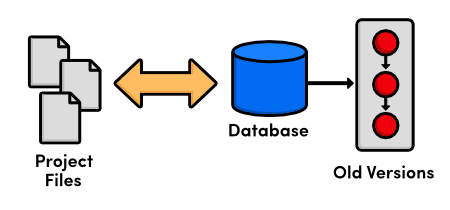
### Files and Folders

The simplest form of version control.



Revision control with files and folders

### Local VCS



Local version control

At this point, versioning only took place on the developer’s *local* computer—there was no way to efficiently share code between several programmers

### Centralized VCS



Centralized version control

### Distributed VCS



Distributed version control

- There was no longer a central repository, everyone could develop at their own pace, store the updates locally, and put off merging conflicts until their convenience.

- The local nature of DVCSs also made development much faster, since you no longer had to perform actions over a network. And, since each user had a complete copy of the project, the risk of a server crash, a corrupted repository, or any other type of data loss was much lower than that of their CVCS predecessors.

## The Birth of Git

As a source code manager for the entire Linux kernel, Git had several unique constraints, including:

* Reliability
* Efficient management of large projects
* Support for distributed development
* Support for non-linear development

### More specifically, Git is a distributed version control system which means that everyone working with a project, not just the current state of the files.

### Github is platform where you can upload a copy of your Git repository. But more than than a centralized location to share your repository, it allows you to collaborate much more easily with other people on your projects. It does that by providing a web interface to manage it

Though originally used for just the Linux kernel, the Git project spread rapidly, and quickly became used to manage a number of other Linux projects,

## Installation On Windows

For Windows users, Git installation will install a special command shell called *Git Bash*. To test your installation, open a new command prompt and run

$ git --version.

Simply download the exe file from the “downloads list”:http://code.google.com/p/msysgit/downloads/list, execute it and follow the on-screen instructions.



At any time, the central server contains well defined revisions of file sets which can be consecutively numbered.



* Each individual repository has its own history
* each object is identified by a SHA1 hash consisting of 40 hexadecimal values
* there are more than 1048 different SHA1 hashes
* often the first seven hex digits are sufficient for identification

# What is version control.

Version control systems are a category of software tools that keep track of every modification to source code over time. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. This history should also include the author, date and written notes on the purpose of each change. Version control protects source code from both catastrophe and the casual degradation of human error and unintended consequences.

One of the most popular VCS tools in use today is called Git. Git is a Distributed VCS, a category known as DVCS, more on that later. Git is free and open source.

# What is Git

By far, Git is the most used modern version control system in the world. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel

Having a distributed architecture, Git is an example of a DVCS (hence Distributed Version Control System). Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN), in Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.

Git, however, is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits

Unlike some version control software, Git is not fooled by the names of the files when determining what the storage and version history of the file tree should be, instead, Git focuses on the file content itself.

Git has been designed with the integrity of managed source code as a top priority. The content of the files as well as the true relationships between files and directories, versions, tags and commits, all of these objects in the Git repository are secured with a cryptographically secure hashing algorithm called SHA1. This protects the code and the change history against both accidental and malicious change and ensures that the history is fully traceable.

With Git, you can be sure you have an authentic content history of your source code.

Some other version control systems have no protections against secret alteration at a later date. This can be a serious information security vulnerability for any organization that relies on software development.

One common criticism of Git is that it can be difficult to learn. Nevertheless, Git is very capable and provides a lot of power to its users. Learning to use that power can take some time, however once it has been learned, that power can be used by the team to increase their development speed.

Unlike centralized version control systems, Git branches are cheap and easy to merge. Feature branches provide an isolated environment for every change to your codebase. This ensures that the master branch always contains production-quality code.

======================

In this section, we will go over what Git was built for and how it works, hopefully laying the groundwork to properly understand what it is doing when we run the commands.

Git is a stupid content tracker. Git tracks content – files and directories. It is at its heart a collection of simple tools that implement a tree history storage and directory content management system. It is simply used as an SCM, not really designed as one.

When most SCMs store a new version of a project, they store the code delta or diff. When Git stores a new version of a project, it stores a new *tree* – a bunch of blobs of content and a collection of point­ers that can be expanded back out into a full directory of files and subdirectories. If you want a diff between two versions, it doesn’t add up all the deltas, it simply looks at the two trees and runs a new diff on them.

This is what fundamentally allows the system to be easily distributed – it doesn’t have issues figuring out how to apply a complex series of deltas, it simply transfers all the directories and content that one user has and another does not have but is requesting. It is efficient about it – it only stores identical files and directories once and it can com­press and transfer its content using delta-compressed packfiles – but in concept, is a very simple beast. Git is at it’s heart very stupid-simple.

* Non-Linear Development

Git is optimized for cheap and efficient branching and merging

* Distributed Development

Git is built to make distributed development simple. No repository is special or central in Git – each clone is basically equal and could generally replace any other one at any time. It works completely offline or with hundreds of remote repositories

* Efficiency

Most operations are local, which reduces unnecessary network overhead

* A Toolkit Design

Git is not really a single binary, but a collection of dozens of small specialized programs, which is sometimes annoying to people trying to learn Git, but is pretty cool when you want to do anything non-standard with it. Git is less a program and more a toolkit that can be combined and chained to do new and interesting things.

The tools can be more or less divided into two major camps, often referred to as the *porcelain* and the *plumbing*. The plumbing is not really meant to be used by people on the command line, but rather to do simple things flexibly and are combined by programs and scripts into porcelain programs.