# 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.

Everybody understands adding, committing, pushing and pulling; but this is about as far as Git’s simplicity goes. Past this point, Git is shrouded by fear, uncertainty and doubt. Once you start talking about branching, merging, rebasing, multiple remotes, remote-tracking branches, detached HEAD states… Git becomes less of an easily-understood tool and more of a feared deity.

Complex systems like 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.

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

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. I’ve seen case after case of devel­opers who love Git after they finally understand it, but getting to that point is often somewhat painstaking.

This book is aimed at the developer who does not particularly like Subversion, Perforce or whatever SCM system they are currently using, has heard good things about Git, but doesn’t know where to start or why it’s so wonderful. It is meant to explain Git as simply as possible in a clean, concise, easily readable volume. My goal is to help you understand Git internals as well as usage at a fundamental level by the time you finish this book.

To accomplish this, I’m starting the book out (after the introduction) with a section about what Git actually **does**, rather than how to use it. I found that I didn’t really understand Git and had many problems using it until I understood what it was actually doing at a low level, rather than thinking of it as a different, weird SVN-like system.

## 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:

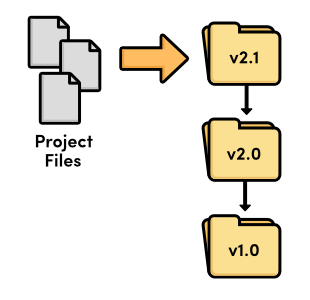
* The ability to undo changes. You can recover an earlier version of you work
* A complete history of all the changes
* Documentation of why changes are made. Often it is hard to remember why a change was made
* Multiple streams of history.

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

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

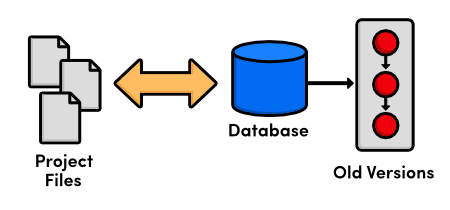
### Files and Folders

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.