



# Source Control, Git, GitHub

---

.NET

**Version Control** is a component of software configuration management. It is the management of all changes to documents, computer programs, large web sites, and other collections of information, over time.

[HTTPS://EN.WIKIPEDIA.ORG/WIKI/VERSION\\_CONTROL](https://en.wikipedia.org/wiki/Version_Control)

# VCS (Version Control Management)

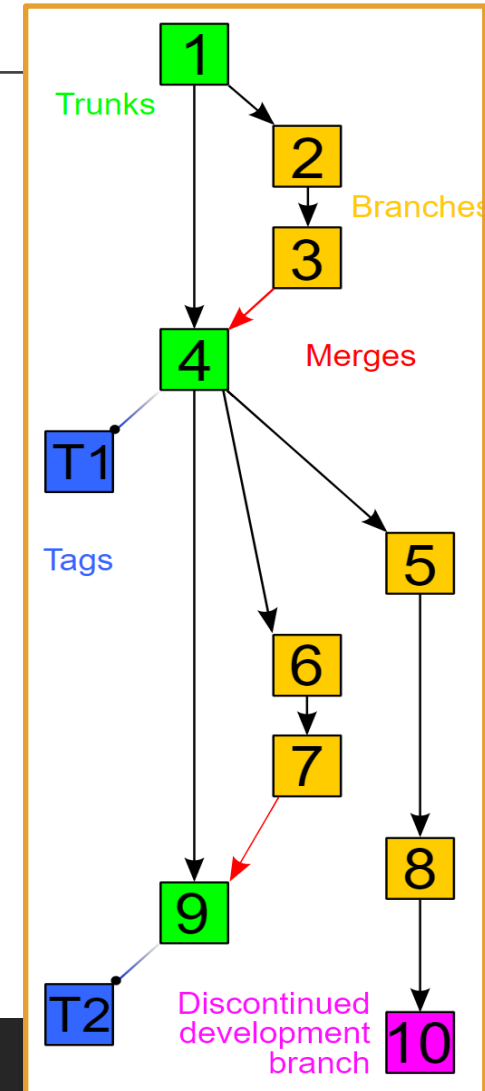
[https://en.wikipedia.org/wiki/Version\\_control](https://en.wikipedia.org/wiki/Version_control)

*Version Control Systems (VCS)* have seen great improvements over the past few decades.

VCSs are sometimes known as:

- *SCM (Source Code Management)* tools or
- *RCS (Revision Control System)*.

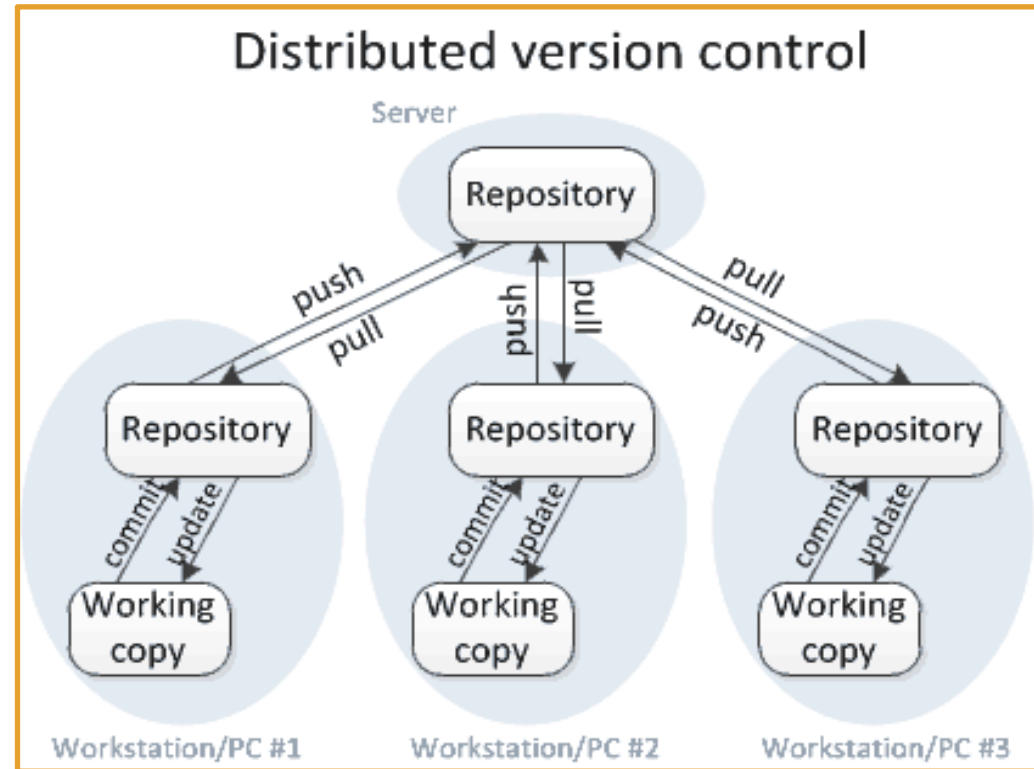
One of the most popular **VCS** tools used today is called **Git**.



# DVCS (Distributed VCS)

<https://www.teamstudio.com/blog/distributed-vs-centralized-version-control-systems-for-lotus-notes>  
<https://homes.cs.washington.edu/~mernst/advice/version-control.html>

- Work on a peer-to-peer model.
- The code base is distributed amongst the individual developers' computers.
- The entire history of the code is mirrored on each system.
- There is still a master copy of the code base kept on a client machine rather than a server.
- There are no locking of parts of the code;
- Developers make changes in their local copy. When they're ready to integrate their changes into the master copy, they issue a request to the owner of the master copy to merge their changes into the master copy.

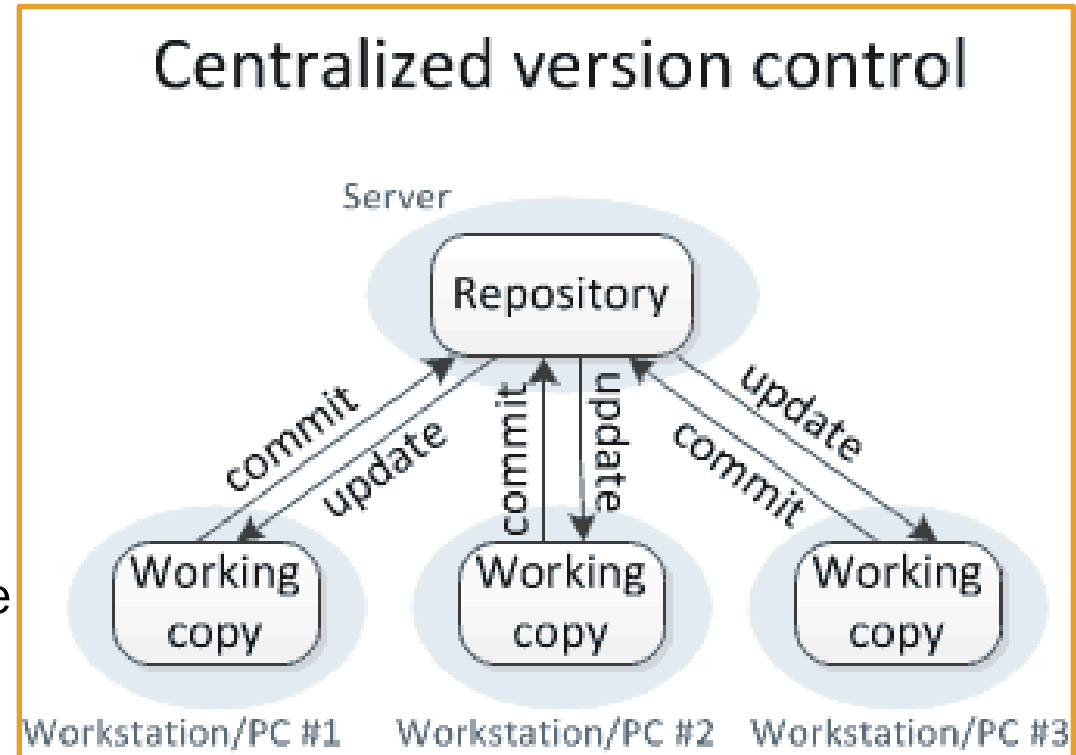


# CVCS (Centralized Version Control System)

<https://www.teamstudio.com/blog/distributed-vs-centralized-version-control-systems-for-lotus-notes>  
<https://homes.cs.washington.edu/~mernst/advice/version-control.html>

A **Centralized Version Control** system works on a **client-server model**. There is a single, (centralized) master copy of the code base, and pieces of the code that are being worked on are typically locked (“checked out”), so that only one developer is allowed to work on that part of the code at any one time.

Access to the code base and locking is controlled by the server. When the Software Developer checks their code back in, the lock is released so it’s available for others to check out.



# Git - A History

<https://git-scm.com/>

<https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git>

Git officially began in 2005 but is the result of 15 years of experience with another VCM application. Linus Torvalds (creator of Linux) was instrumental in creating a new VCM tool that was distributed, fast and simple, supported branching, and could handle large projects. Out of those goals came Git.

Git is a free, open-source, distributed **Version Control System** designed to handle small and large projects with speed and efficiency.

Many different **VCS** websites leverage **Git**. The primary and most popular is [www.github.com](https://www.github.com). Most open-source software is hosted, for free, on **GitHub**.





# How Git works

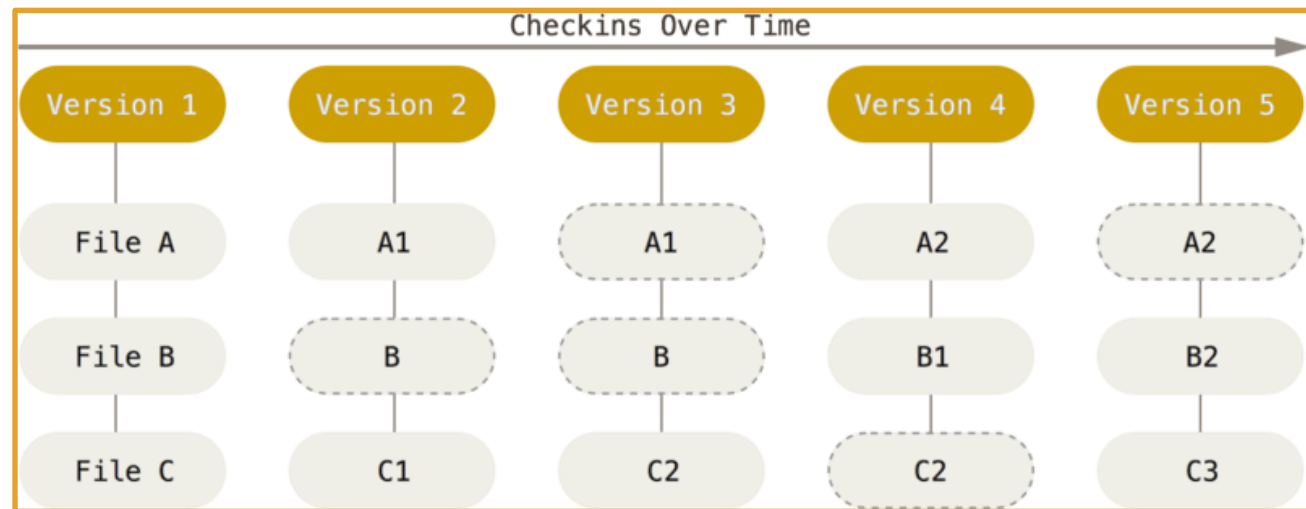
<https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>

**Git** thinks of its data like a series of snapshots of a filesystem. Every time you **commit** (save the state of your project), **Git** snapshots (documents) what your files look like and stores a reference to that snapshot. Only changed files are re-snapshotted.

Most operations in **Git** need only local files and resources to operate.

For example, **Git** doesn't need to go out to the server to get the history of a file and display it. All that data is kept locally.

**Git** can look up the state of a file a month ago and do a local difference calculation to show the differences.



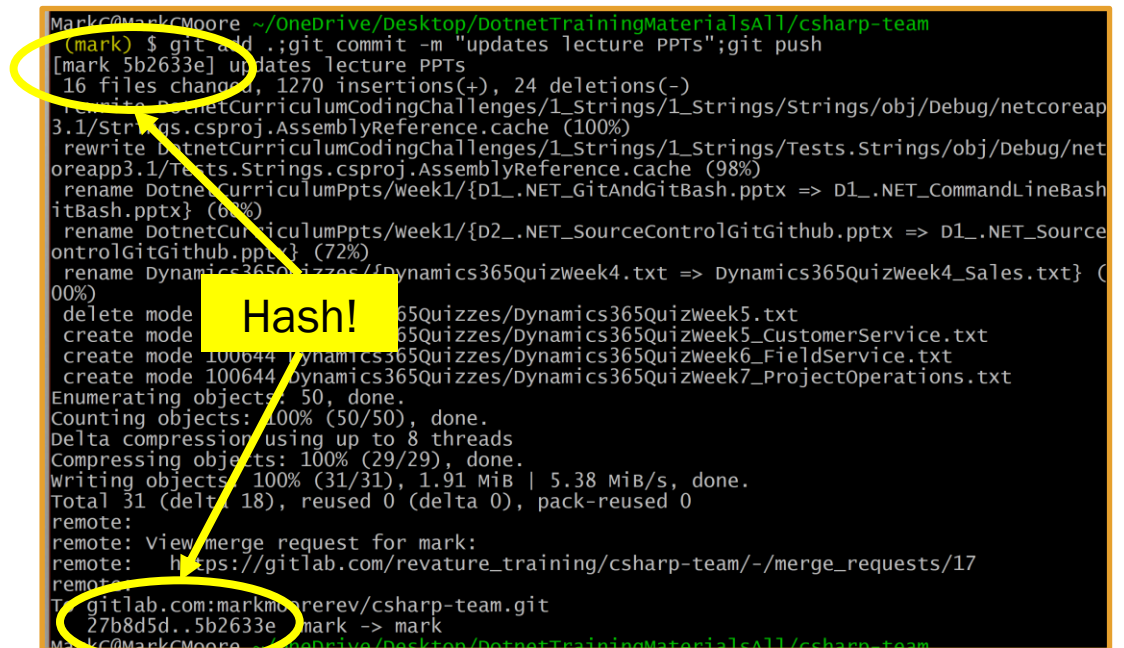
# How Git works

<https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>

Every time you **commit**, each changed file is **checksummed** before it is stored. That moment in its individual history is then referred to by that 40-character checksum **hash**.

In the future, that file can be **reverted** to that point in its history using the assigned **hash**.

**Git** stores everything in its own database by the hash value. This functionality is built into **Git** at the lowest levels, so you'll never get file corruption without **Git** being able to detect it.



```
MarkC@MarkC Moore ~/OneDrive/Desktop/DotnetTrainingMaterialsAll/csharp-team
(mark) $ git add .;git commit -m "updates lecture PPTs";git push
[mark 5b2633e] updates lecture PPTs
16 files changed, 1270 insertions(+), 24 deletions(-)
rewrite DotnetCurriculumCodingChallenges/1_Strings/1_Strings/Strings/obj/Debug/netcoreap
3.1/Strings.csproj.AssemblyReference.cache (100%)
rewrite DotnetCurriculumCodingChallenges/1_Strings/Tests.Strings/obj/Debug/net
oreapp3.1/Tests.Strings.csproj.AssemblyReference.cache (98%)
rename DotnetCurriculumPpts/Week1/{D1_.NET_GitAndGitBash.pptx => D1_.NET_CommandLineBash
itBash.pptx} (60%)
rename DotnetCurriculumPpts/Week1/{D2_.NET_SourceControlGitGithub.pptx => D1_.NET_Source
ontrolGitGithub.pptx} (72%)
rename Dynamics365Quizzes/{Dynamics365QuizWeek4.txt => Dynamics365QuizWeek4_Sales.txt} (
00%)
delete mode 100644 Dynamics365Quizzes/Dynamics365QuizWeek5.txt
create mode 100644 Dynamics365Quizzes/Dynamics365QuizWeek5_CustomerService.txt
create mode 100644 Dynamics365Quizzes/Dynamics365QuizWeek6_FieldService.txt
create mode 100644 Dynamics365Quizzes/Dynamics365QuizWeek7_ProjectOperations.txt
Enumerating objects: 50, done.
Counting objects: 100% (50/50), done.
Delta compression using up to 8 threads
Compressing objects: 100% (29/29), done.
Writing objects: 100% (31/31), 1.91 MiB | 5.38 MiB/s, done.
Total 31 (delta 18), reused 0 (delta 0), pack-reused 0
remote:
remote: View merge request for mark:
remote: https://gitlab.com/revature_training/csharp-team/-/merge_requests/17
remote:
T gitlab.com:markmoorerev/csharp-team.git
27b8d5d..5b2633e mark -> mark
MarkC@MarkC Moore ~/OneDrive/Desktop/DotnetTrainingMaterialsAll/csharp-team
```



# How Git works

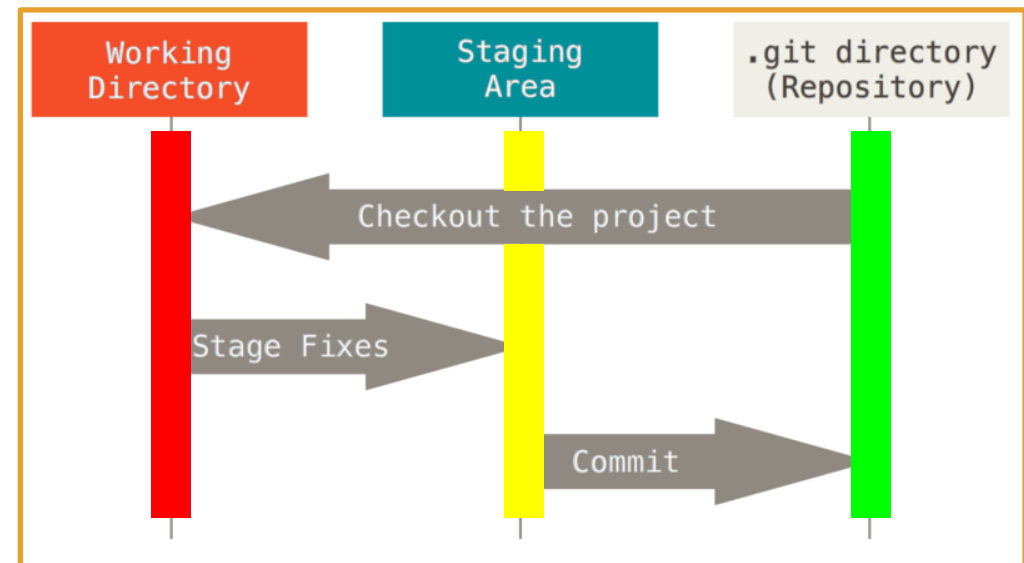
<https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>

The **Git** lifecycle has three main stages that a file could be in:

- **Modified** - you have changed a file but have not committed it yet.
- **Staged** - you have marked a modified file in its current version to go into your next commit.
- **Committed** - the data is safely stored in your local database.

A **Git** project has three main sections:

- **working tree** - a single checkout of one version of the project.
- **staging area** - a file contained in your **Git** directory that stores what will go into your next commit.
- **Git directory** - where **Git** stores the metadata and object database for your project.



The three **Git** stages and three Git project sections correlate.

# Git Setup

<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>

---

# Git Basic Commands

---

# Simple (NO-CONFLICTS) Github Workflow

