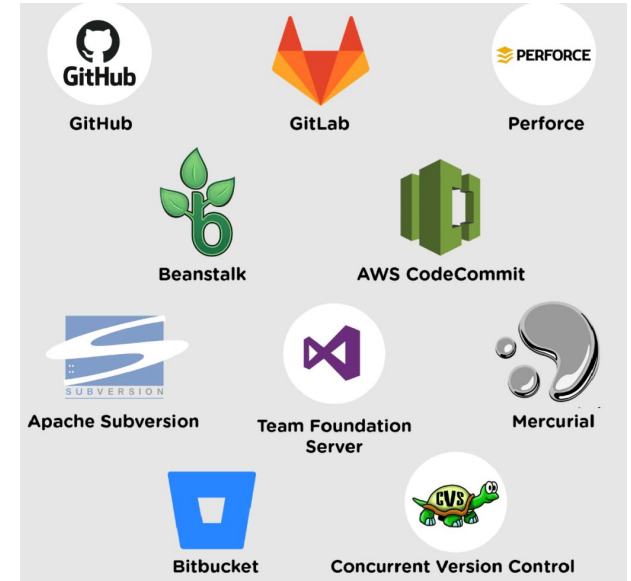


Git

Version control

It also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.



Why Version Control system is so Important?

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So in order to contribute to the product, they made modifications to the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made.

Benefits of the version control system

- Enhances the project development speed by providing efficient collaboration,
- Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
- Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
- Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this **VCS**,
- For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is **Git, Helix core, Microsoft TFS**,

Types of Version Control Systems:

Local Version Control Systems

Centralized Version Control Systems

Distributed Version Control Systems

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems: Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository.



Git

Software

Version control

Maintained by Linux

Open-Source

No user management

Locally installed

Minimal external tool
configuration

Little to no competition



GitHub

Service

Git repository hosting

Maintained by Microsoft

Free or paid membership

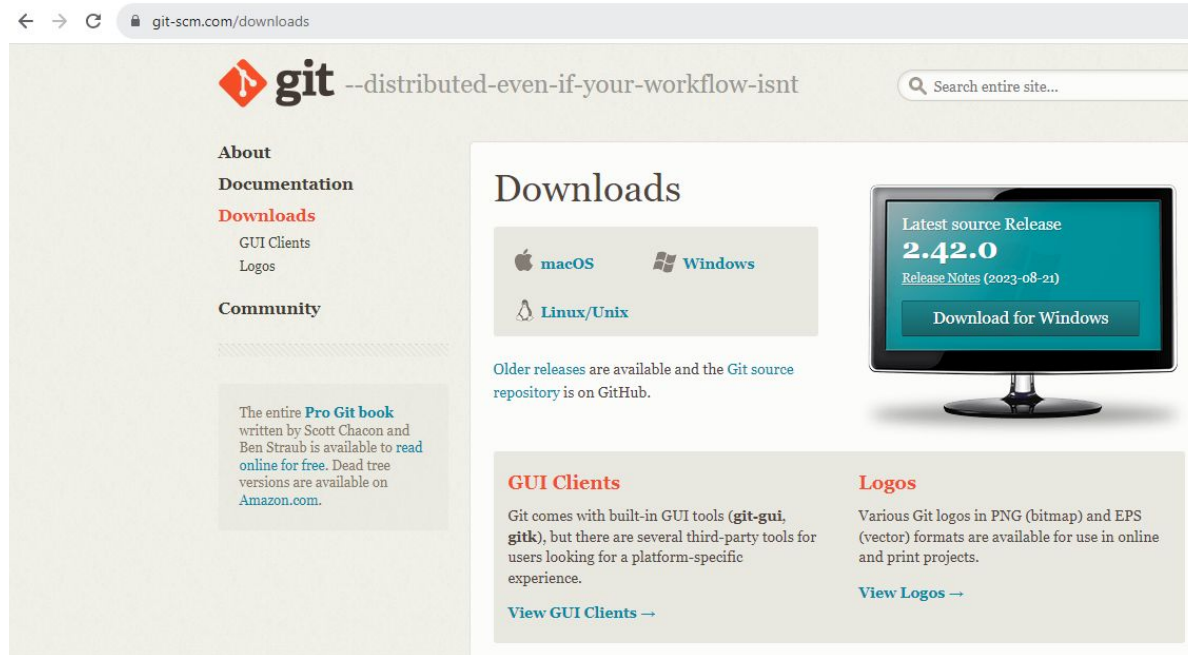
Built-in user management

Hosted on the web

Active marketplace for
tool integration

High competition

How to install Git



The screenshot shows the 'Downloads' page of the Git SCM website. The browser address bar displays 'git-scm.com/downloads'. The page features a navigation menu on the left with links to 'About', 'Documentation', 'Downloads' (highlighted), 'GUI Clients', 'Logos', and 'Community'. The main content area is titled 'Downloads' and includes a search bar. It lists download links for macOS, Windows, and Linux/Unix. A prominent monitor graphic displays the 'Latest source Release 2.42.0' with a 'Download for Windows' button. A text box mentions that older releases are on GitHub. At the bottom, there are sections for 'GUI Clients' and 'Logos' with links to view more options.

git --distributed-even-if-your-workflow-isnt

Search entire site...

About
Documentation
Downloads
GUI Clients
Logos
Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to read [online for free](#). Dead tree versions are available on [Amazon.com](#).

Downloads

macOS Windows Linux/Unix

Latest source Release
2.42.0
[Release Notes \(2023-08-21\)](#)
[Download for Windows](#)

Older releases are available and the Git source repository is on GitHub.

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.
[View GUI Clients →](#)

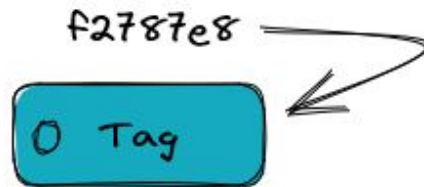
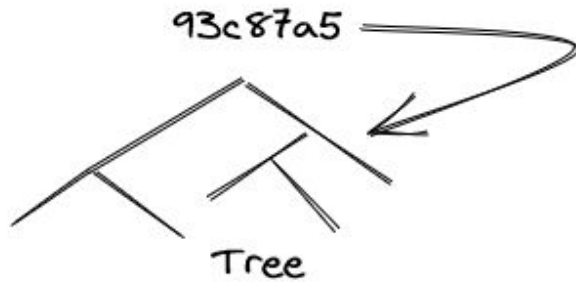
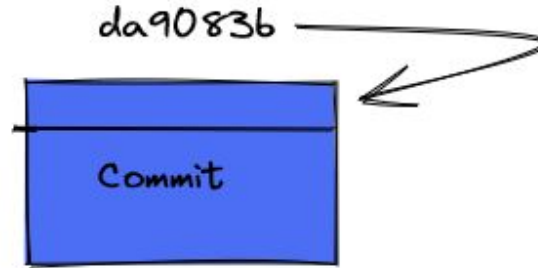
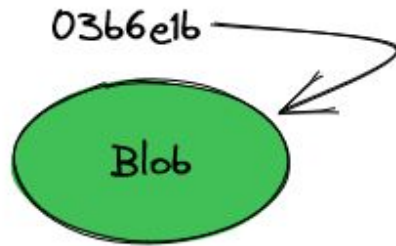
Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.
[View Logos →](#)

Type of Git objects



LOVELY
PROFESSIONAL
UNIVERSITY



Git Blob

A Git blob (binary large object) is the object type used to store the contents of each file in a repository. The file's SHA-1 hash is computed and stored in the blob object.

Git blobs are built up in a *memory buffer* (a location in your computer's memory that Git's code can access) by Git's code in the following format:

blob <size-of-blob-in-bytes>\0<file-binary-data>

When does Git create blobs?

Git creates new blobs in the following circumstances:

Adding an untracked file to the staging area with `git add`

Adding a modified, tracked file to the staging area with `git add`

Git merges that result in changing an existing file's content (not all merges do this)

Can Git reuse blobs across commits?



Yes! Since Git uses a content-addressable database, it is able to detect whether a file's content has changed based on the calculated SHA-1 of the file's blob. If a file is not changed between commits, or is changed but to a known previous state, the existing blob for that file will be reused in future commits.

This saves a lot of hard-drive space since Git doesn't need to store the same object multiple times across commits.

A Git tree object creates the hierarchy between files in a Git repository. You can use the Git tree object to create the relationship between directories and the files they contain. These endpoints allow you to read and write [tree objects](#) to your Git database on GitHub.

A tree in Git associates blobs with their actual file path/name and permissions.

What is the Git tree format?

A tree in Git is built up by Git's code in a [memory buffer](#) in the following format:

```
tree <size-of-tree-in-bytes>\0 <file-1-mode> <file-1-path>\0<file-1-blob-hash>  
  <file-2-mode> <file-2-path>\0<file-2-blob-hash> ... <file-n-mode>  
  <file-n-path>\0<file-n-blob-hash>
```

The tree format starts with the object's type, which is just the string 'tree', followed by the size of the tree object in bytes.

Where are Git trees stored?

Trees are stored in Git's repository, also known as the **object store**. This is located at the path **.git/objects/** in your project root directory.

Does Git delete trees?

No. Git doesn't delete trees, unless the commit tied to that tree becomes orphaned. If this does happen, Git's garbage collector could delete the tree once a certain period of time elapses.

Can Git reuse trees across commits?

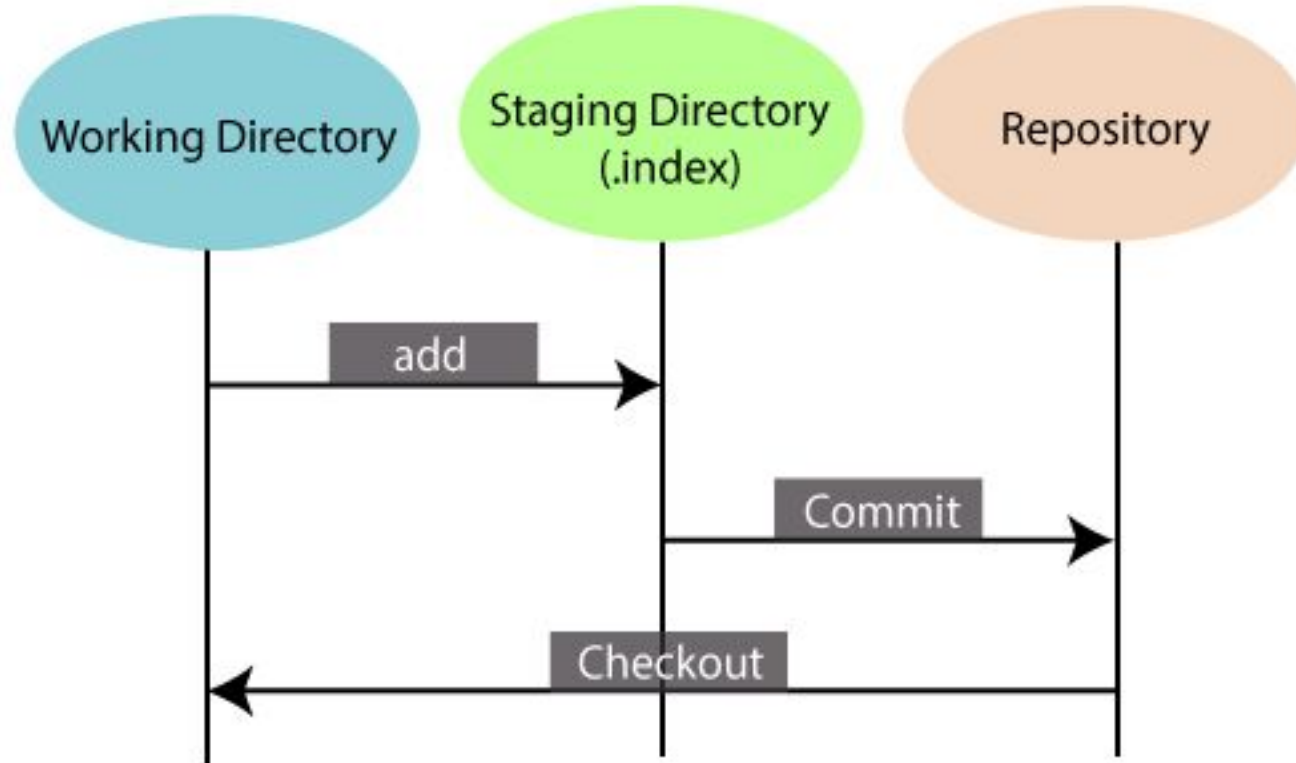


Yes! Git will never create the same object twice in your object database! So if multiple commits represent snapshots of the exact same set of files - with unchanged content - Git will just reuse the existing tree that represents that set.

Git architecture



LOVELY
PROFESSIONAL
UNIVERSITY



Git workflow is as follows –

You clone the Git repository as a working copy.

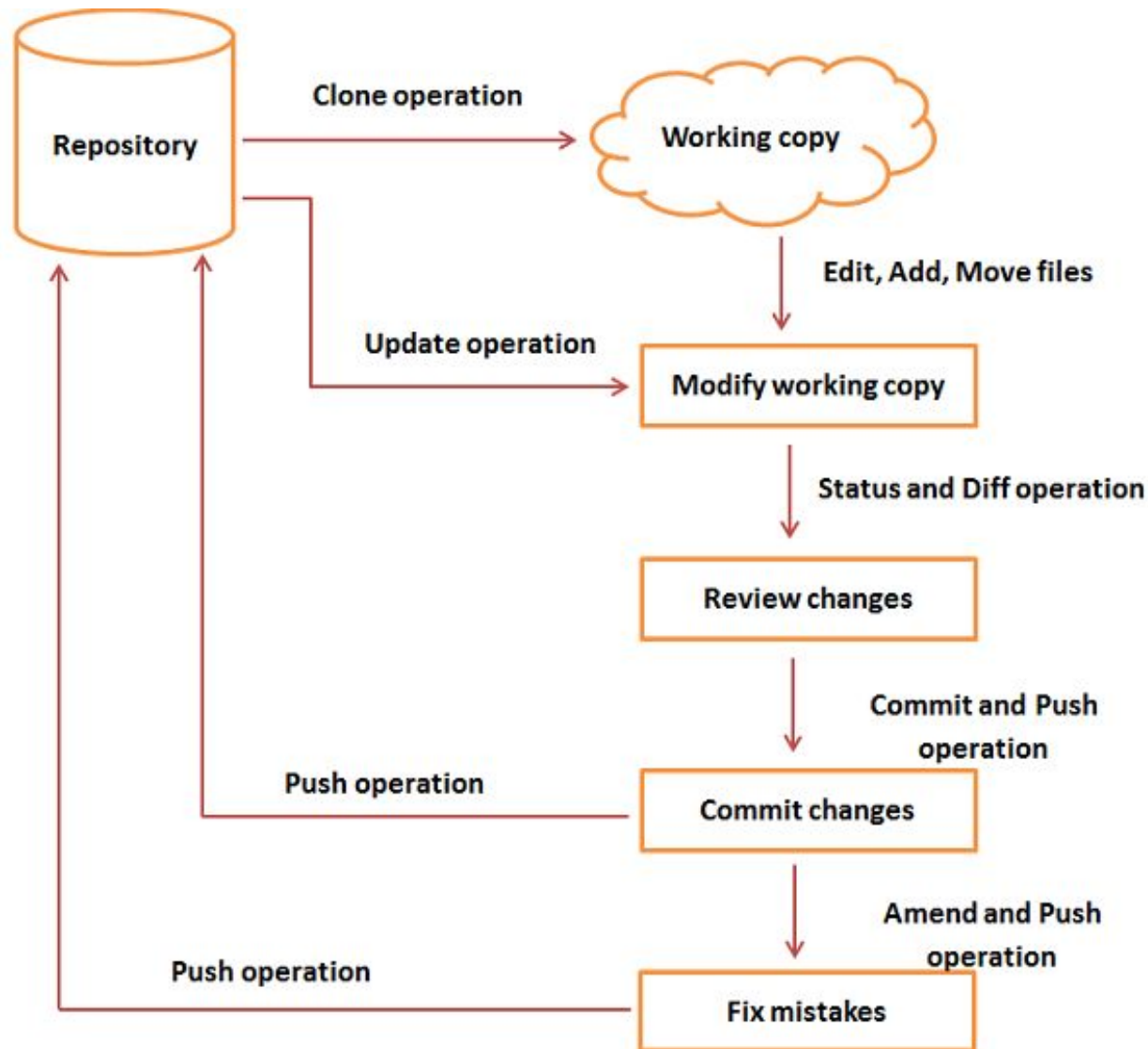
You modify the working copy by adding/editing files.

If necessary, you also update the working copy by taking other developer's changes.

You review the changes before commit.

You commit changes. If everything is fine, then you push the changes to the repository.

After committing, if you realize something is wrong, then you correct the last commit and push the changes to the repository.



What is a Git workflow?



Identifying a single Git workflow is a necessary step in ensuring rapid delivery. Software development teams encompass contributors from various backgrounds and experiences, and they're likely to feel comfortable with a workflow they've used previously. Without a single workflow, a team's development workflow could be chaotic and slow down cycle time. [Git workflows empower teams](#) to determine roles and responsibilities, set boundaries, and identify areas of improvement.

Centralized Git workflow



A centralized Git workflow enables all team members to make changes directly to the main branch (sometimes called the master branch or default branch), with every change logged in a running history. A centralized workflow involves every contributor committing to the main branch without using any other branch. This strategy works well for small teams, because team members can communicate so that multiple developers aren't contributing to the same piece of code simultaneously. Centralized workflow can be seamless if team members communicate well, but there are limitations. If multiple developers commit to the same branch, it's challenging to find a stable moment to release changes. Consequently, developers must keep unstable changes local until they're ready for release.

Trunk-based development Git workflow



Trunk-based development facilitates concurrent development on a single branch called trunk. When developers are ready to push changes to the central repository, they'll pull and rebase from it to update the working copy of the central branch. Successful trunk-based development requires a developer to resolve merge conflicts locally. Regularly updating the local branch reduces the impact of integration changes, because they're spotted when they're still small, avoiding merge hell.

Personal branching Git workflow



Personal branching is similar to feature branching, but rather than have a single branch per feature, it's per developer. This approach works well if team members work on different features and bugs. Every user can merge back to the main branch whenever their work is done.



A forking approach to version control starts with a complete copy of the repository. Forking effectively creates a local copy of a Git repository and provides the ability to create a new collaboration structure. In other words, every developer in the team has two repositories: a local workspace and a remote repository.

This workflow is popular for projects that have multiple developers contributing to it, particularly open source projects. After all, keeping track and providing privileges to collaborate to a repository with thousands of contributors is difficult to maintain. If a maintainer enables contributors to try their changes on their forked copy, managing change proposals is easier and safer.

With GitFlow, the main branch should always be releasable to production, and there should never be untested or incomplete code on the main branch. When using this Git workflow, no one commits to the main branch but rather uses a develop branch with feature branches. When the develop branch is ready to go to production, a contributor creates a release branch where testing and bug fixing occur before being merged back to the develop branch. The release branch makes the code review process easier, because there's a dedicated place to resolve conflicts when merging into the main branch. With this strategy, the main branch always reflects production.

Working with Remote Repository



Repositories in GIT contain a collection of files of various different versions of a Project. These files are imported from the repository into the local server of the user for further updations and modifications in the content of the file.

Local Repository: Git allows the users to perform work on a project from all over the world because of its Distributive feature. This can be done by cloning the content from the Central repository stored in the GitHub on the user's local machine. This local copy is used to perform operations and test them on the local machine before adding them to the central repository.

Remote Repository: Git allows the users to sync their copy of the local repository to other repositories present over the internet. This can be done to avoid performing a similar operation by multiple developers. Each repository in Git can be addressed by a shortcut called **remote**.