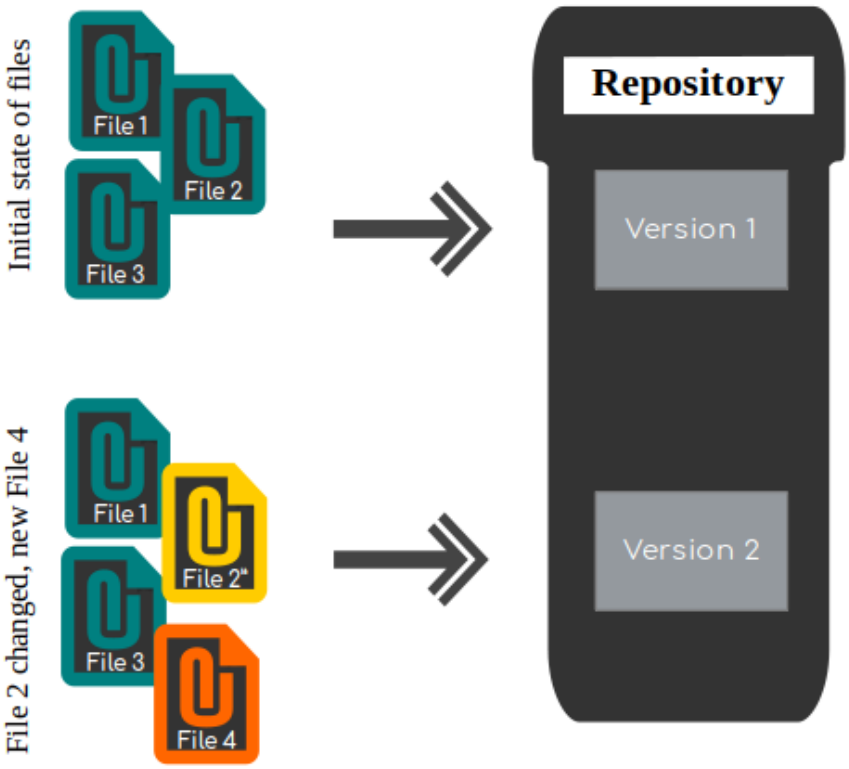
1. Introduction into version control systems

1.1. What is a version control system?

A version control system (VCS) allows you to manage a collection of files and gives access to different versions of these files.

The VCS allows you to capture the content and structure of your files at a certain point in time. You can use the VGS to switching between these versions and you can work on different versions of these files in parallel. The different versions are stored in storage system which is typically called a *repository*. The process of creating different versions (snapshots) in the repository is depicted in the following graphic.



In this example, your repository contains two versions, one with three files and another version with four files, two one them in the same state as in the first version, one modified one and another new one.

VCS are very good in tracking changes in text files. For example, you may track changes in HTML code or Java source code. If is also possible to use VCS for other file types but VCS are not that efficient to trace changes in binary files.

A localized version control system keeps local copies of the files. This approach can be as simple as creating a manual copy of the relevant files.

A centralized version control system provides a server software component which stores and manages the different versions of the files. A developer can copy (checkout) a certain version from the central sever onto their individual computer.

Both approaches have the drawback that they have one single point of failure. In a localized version control systems it is the individual computer and in a centralized version control systems it is the server machine. Both system makes it also harder to work in parallel on different features. To remove the limitations of local and centralized version control systems, distributed version control system have been created.

Version Control System (VCS) is software that helps software developers to work together and maintain a complete history of their work.

Listed below are the functions of a VCS –

- Allows developers to work simultaneously.
- Does not allow overwriting each other's changes.
- Maintains a history of every version.

Following are the types of VCS –

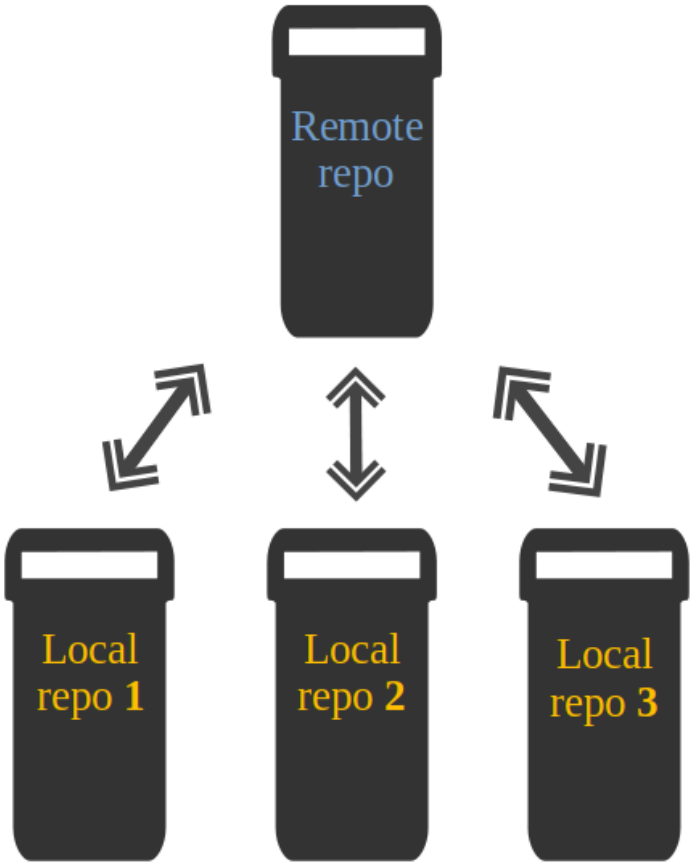
- Centralized version control system (CVCS).
- Distributed/Decentralized version control system (DVCS).

1.2. Distributed version control systems

In a distributed version control system each user has a complete local copy of a repository on his individual computer. The user can copy an existing repository. This copying process is typically called *cloning* and the resulting repository can be referred to as a *clone*.

Every clone contains the full history of the collection of files and a cloned repository has the same functionality as the original repository.

Every repository can exchange versions of the files with other repositories by transporting these changes. This is typically done via a repository running on a server which is, unlike the local machine of a developer, always online. Typically, there is a central server for keeping a repository but each cloned repository is a full copy of this repository. The decision which of the copies is considered to be the central server repository is pure convention.



2. Introduction into Git

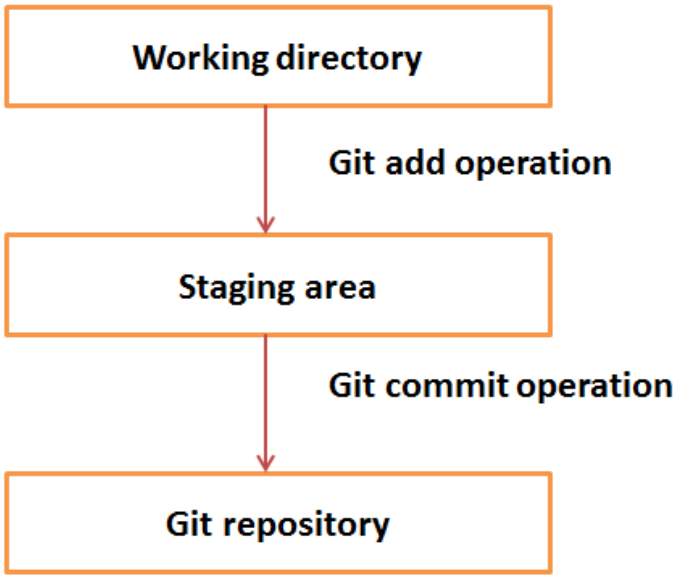
The following description gives you a very high-level overview of the Git version control system.

2.1. What is Git?

Git is the leading distributed version control system.

Git originates from the Linux kernel development and was founded in 2005 by **Linus Torvalds**. Nowadays it is used by many popular open source projects, e.g., Visual Studio Code from Microsoft, Android from Google or the Eclipse developer teams, as well as many commercial organizations.

The core of Git was originally written in the programming language *C*, but Git has also been re-implemented in other languages, e.g., Java, Ruby and Python



Suppose you modified two files, namely “sort.c” and “search.c” and you want two different commits for each operation. You can add one file in the staging area and do commit. After the first commit, repeat the same procedure for another file.

2.1. What is Git?

Git is the leading distributed version control system.

Git originates from the Linux kernel development and was founded in 2005 by Linus Torvalds. Nowadays it is used by many popular open source projects, e.g., Visual Studio Code from Microsoft, Android from Google or the Eclipse developer teams, as well as many commercial organizations.

The core of Git was originally written in the programming language C, but Git has also been re-implemented in other languages, e.g., Java, Ruby and Python.

2.2. Git repositories and working trees

A Git repository manages a collection of files in a certain directory. A Git repository is file based, i.e., all versions of the managed files are stored on the file system.

A Git repository can be designed to be used on a server or for an user:

bare repositories are supposed to be used on a server for sharing changes coming from different developers. Such repositories do not allow the user to modify locally files and to create new versions for the repository based on these modifications.

non-bare repositories target the user. They allow you to create new changes through modification of files and to create new

versions in the repository. This is the default type which is created if you do not specify any parameter during the clone operation.

A local non-bare Git repository is typically called local repository.

Git allows the user to synchronize the local repository with other (remote) repositories.

Users with sufficient authorization can send new versions of their local repository to the remote repositories via the push operation. They can also integrate changes from other repositories into their local repository via the fetch and pull operation.

Every local repository has a working tree. The files in the working tree may be new or based on a certain version from the repository. The user can change and create files or delete them.

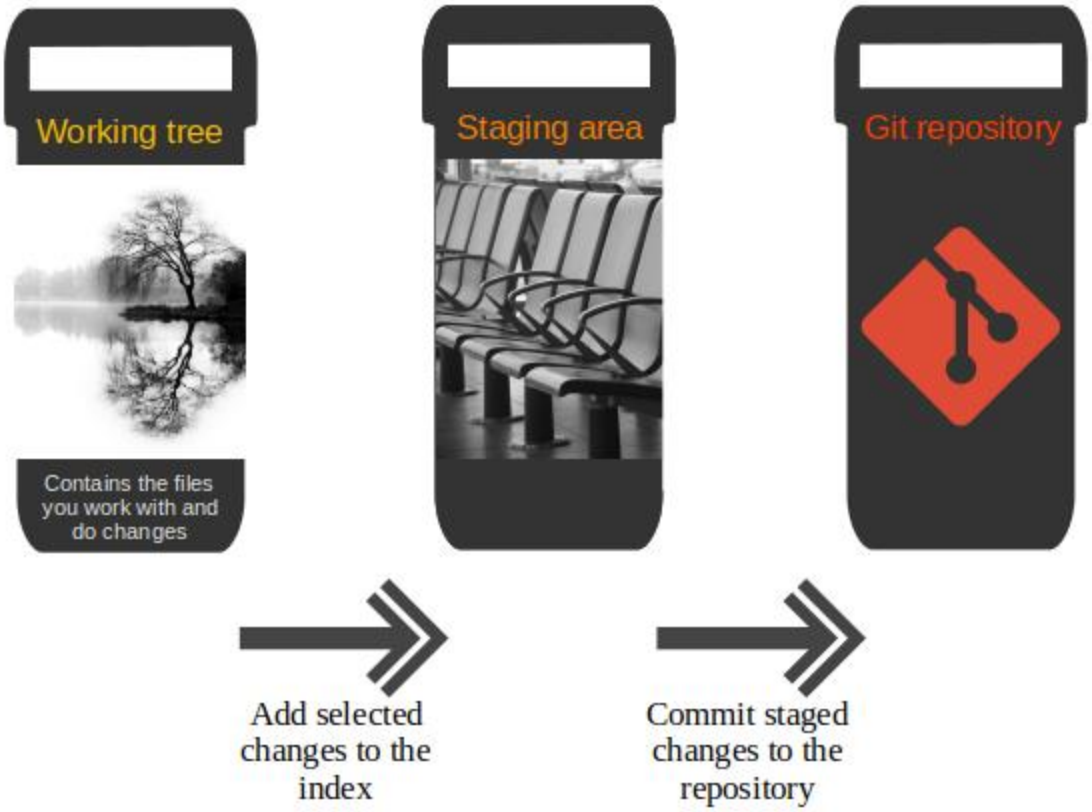
After doing changes in the working tree, the user can capture new versions of the files in the Git repository. Or the user can restore files to a state already captured by Git.

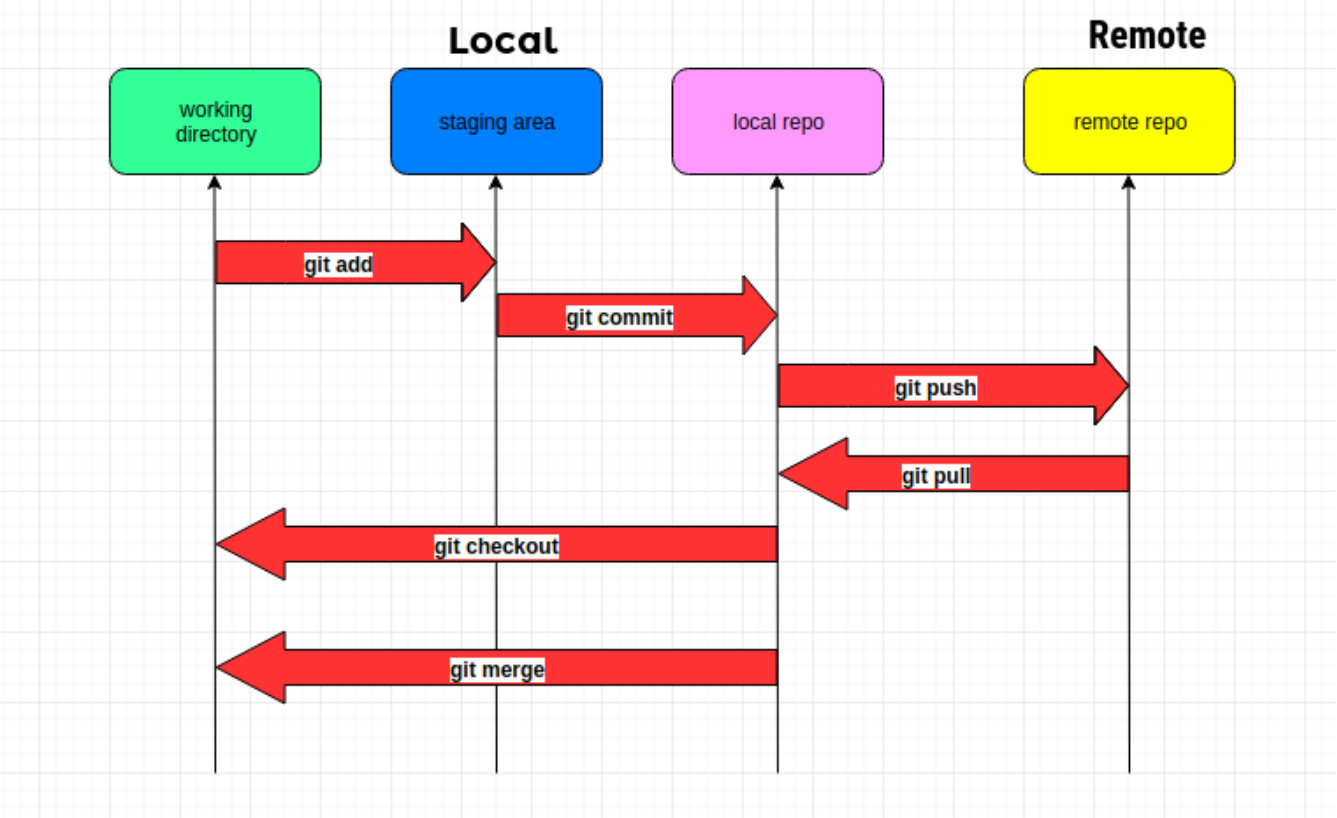
2.3. Adding a new version of the files to a Git repository

After modifying files in your working tree you need to perform two steps to add them to your local repository.

- mark the desired file changes as relevant for the next commit; this operation is called staging
- instruct Git to create a new version of the managed files via the commit operation, the new created version is called commit.

This process is depicted in the following graphic.


Working tree
Contains the files you work with and do changes
Staging area
Git repository
Add selected changes to the index
Commit staged changes to the repository



During the stage operation, copies of the specified files are added to a persisted storage called the staging area (sometimes it is also called index). This allows you to do further modifications to the same file without including these modifications in the next commit. You can repeat the staging operation until you are satisfied and continue with the commit operation.

The commit operation creates a new persistent snapshot called commit object (short form: commit) of the managed files in the Git repository. A commit object, like all objects in Git, is immutable

Some Git Basic commands :

- **Branch**

  A branch is a unique set of code changes with a unique name. That means each repository can have one or more branches.

  We can create a branch using 'git branch' command: **$ git branch branch-name**

- ***Commit***

  We are using the "commit" command to save your changes to the local repository.

  'git commit' command: **$ git commit -m "commit message"**

- **Push**

  For uploading local changes to a master repository, 'push' is most commonly used.

  'git push' command: **$ git push origin master**

- **Pull**

  To download and integrate remote changes, we are using the 'pull' command.

  'git pull' command: **$ git pull origin master**

- **Merge**

  To integrate changes from another branch, we use "merge" command.

  'git merge' command: **$ git merge branch-name**

- **Clone**

  The 'clone' command is used to a copy of an existing Git repository.

  'git clone' command: **$ git clone repository.**

Git Tools

To explore the robust functionality of Git, we need some tools. Git comes with some of its tools like Git Bash, Git GUI to provide the interface between machine and user. It supports inbuilt as well as third-party tools.

Git comes with built-in GUI tools like **git bash**, **git-gui**, and **gitk** for committing and browsing. It also supports several third-party tools for users looking for platform-specific experience.

Git Package Tools

Git provides powerful functionality to explore it. We need many tools such as commands, command line, Git GUI. Let's understand some essential package tools.

GitBash

Git Bash is an application for the Windows environment. It is used as Git command line for windows. Git Bash provides an emulation layer for a Git command-line experience. Bash is an abbreviation of **Bourne Again Shell**. Git package installer contains Bash, bash utilities, and Git on a Windows operating system.

Bash is a standard default shell on Linux and macOS. A shell is a terminal application which is used to create an interface with an operating system through commands.

By default, Git Windows package contains the Git Bash tool. We can access it by right-click on a folder in Windows Explorer.

Git Bash Commands

Git Bash comes with some additional commands that are stored in the **/usr/bin** directory of the Git Bash emulation. Git Bash can provide a robust shell experience on Windows. Git Bash comes with some essential shell commands like **Ssh**, **scp**, **cat**, **find**.

Git Bash also includes the full set of Git core commands like **git clone, git commit, git checkout, git push**, and more.

Git GUI

Git GUI is a powerful alternative to Git BASH. It offers a graphical version of the Git command line function, as well as comprehensive visual diff tools. We can access it by simply right click on a folder or location in windows explorer. Also, we can access it through the command line by typing below command.

GitLab

What is Gitlab?

Before we dive into definition for Gitlab, first we need to understand few terminologies. We often come across these terms like Git, Gitlab, GitHub, and Bitbucket. Let's see definiton of all these as below –

Git - It is a source code versioning system that lets you locally track changes and push or pull changes from remote resources.

GitLab, GitHub, and Bitbucket - Are services that provides remote access to Git repositories. In addition to hosting your code, the services provide additional features designed to help manage the software development lifecycle. These additional features include managing the sharing of code between different people, bug tracking, wiki space and other tools for 'social coding'.

- **GitHub** is a publicly available, free service which requires all code (unless you have a paid account) be made open.

Anyone can see code you push to GitHub and offer suggestions for improvement. GitHub currently hosts the source code for tens of thousands of open source projects.

- **GitLab** is a github like service that organizations can use to provide internal management of git repositories. It is a self-hosted Git-repository management system that keeps the user code private and can easily deploy the changes of the code.

History

GitLab was found by *Dmitriy Zaporozhets* and *Valery Sizov* in October 2011. It was distributed under MIT license and the stable version of GitLab is 10.4 released in January 22, 2018.

Why to use GitLab?

GitLab is great way to manage git repositories on centralized server. GitLab gives you complete control over your repositories or projects and allows you to decide whether they are public or private for free.

Features

- GitLab hosts your (private) software projects for free.
- GitLab is a platform for managing Git repositories.
- GitLab offers free public and private repositories, issue-tracking and wikis.
- GitLab is a user friendly web interface layer on top of Git, which increases the speed of working with Git.

- GitLab provides its own *Continuous Integration* (CI) system for managing the projects and provides user interface along with other features of GitLab.

Advantages

- GitLab provides *GitLab Community Edition* version for users to locate, on which servers their code is present.
- GitLab provides unlimited number of private and public repositories for free.
- The *Snippet* section can share small amount of code from a project, instead of sharing whole project.

Disadvantages

- While pushing and pulling repositories, it is not as fast as GitHub.
- GitLab interface will take time while switching from one to another page.