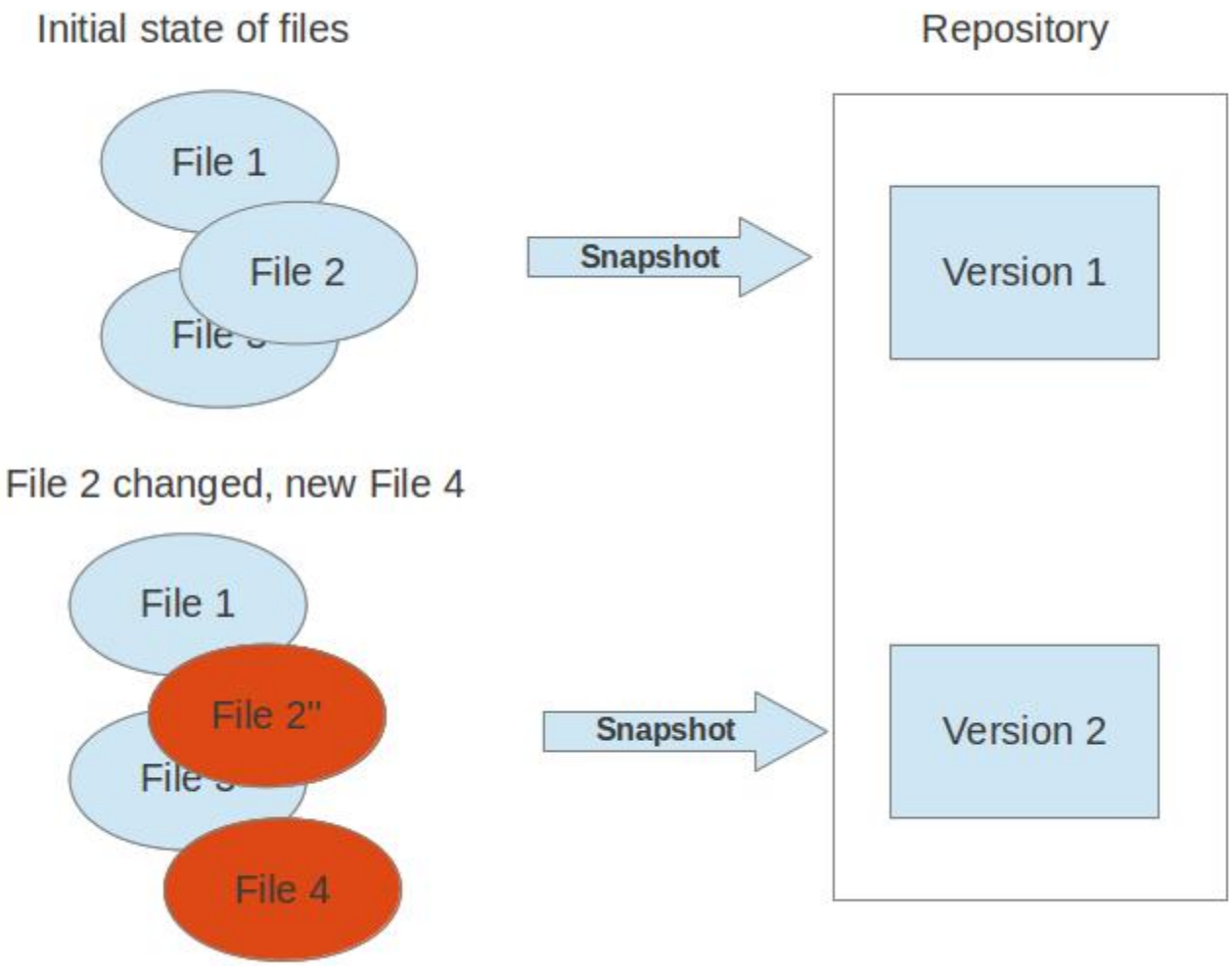
Git is basically a version control system and code sharing and publishing.services.

1.1. What is a version control system?

Ans:- A version control system allows you to track the history of a collection of files and includes the functionality to revert the collection of files to another version. Each version captures a snapshot of the files at a certain point in time. The collection of files is usually*source code* for a programming language but a typical version control system can put any type of file under version control.

The collection of files and their complete history are stored in a *repository*.

The process of creating different versions (snapshots) in the repository is depicted in the following graphic. Please note that this picture fits primarily to Git. Other version control systems like *Concurrent Versions System* (CVS) don't create snapshots but store file deltas.



These snapshots can be used to change your collection of files. You may, for example, revert the collection of files to a state from 2 days ago. Or you may switch between versions for experimental features.

1.2. What is a distributed version control system?

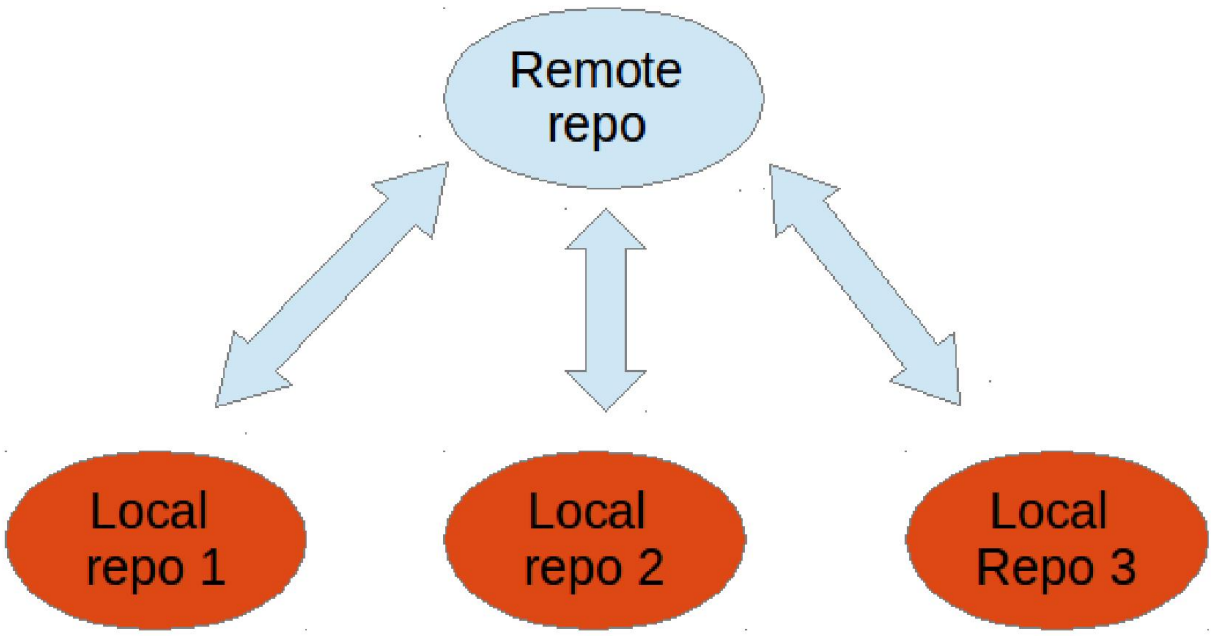
A distributed version control system does not necessarily have a central server which stores the data.

The user can copy an existing *repository*. This copying process is typically called *cloning* in a distributed version control system and the resulting repository can be referred to as a *clone*.

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 and not tied to the capabilities of the distributed version control system itself.

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.



1.3. What is Git?

Git is a distributed version control system.

Git originates from the Linux kernel development and is used by many popular open source projects, e.g., the Android 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.

Reference table with important Git terminology

The following table provides a summary of important *Git* terminology.

Table 1. Important Git terminology

Term	Definition
Branch	A *branch* is a named pointer to a commit. Selecting a branch in Git terminology is called *to checkout a branch*. If you are working in a certain branch, the creation of a new commit advances this pointer to the newly created commit. Each commit knows their parents (predecessors). Successors are retrieved by traversing the commit graph starting from branches or other refs, symbolic references (for example: HEAD) or explicit commit objects. This way a branch defines its own line of descendants in the overall version graph formed by all commits in the repository. You can create a new branch from an existing one and change the code independently from other branches. One of the branches is the default (typically named *master*). The default branch is the one for which a local branch is automatically created when cloning the repository.
Commit	When you commit your changes into a repository this creates a new *commit object* in the Git repository. This*commit object* uniquely identifies a new revision of the content of the repository. This revision can be retrieved later, for example, if you want to see the source code of an older version. Each commit object contains the author and the committer, thus making it possible to identify who did the change. The author and committer might be different people. The author did the change and the committer applied the change to the Git repository. This is common for contributions to open source projects.
HEAD	*HEAD* is a symbolic reference most often pointing to the currently checked out

Term	Definition
	branch. Sometimes the *HEAD* points directly to a commit object, this is called *detached HEAD mode*. In that state creation of a commit will not move any branch. If you switch branches, the *HEAD* pointer points to the branch pointer which in turn points to a commit. If you checkout a specific commit, the *HEAD* points to this commit directly.
Index	*Index* is an alternative term for the *staging area*.
Repository	A *repository* contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository. If the repository is not a bare repository, it allows you to checkout revisions into your working tree and to capture changes by creating new commits. Bare repositories are only changed by transporting changes from other repositories. This book uses the term *repository* to talk about a non-bare repository. If it talks about a bare repository, this is explicitly mentioned.
Revision	Represents a version of the source code. Git implements revisions as *commit objects* (or short *commits*). These are identified by an SHA-1 hash.
Staging area	The *staging area* is the place to store changes in the working tree before the commit. The *staging area* contains a snapshot of the changes in the working tree (changed or new files) relevant to create the next commit and stores their mode (file type, executable bit).
Tag	A *tag* points to a commit which uniquely identifies a version of the Git repository. With a tag, you can have a named point to which you can always revert to. You can revert to any point in a Git repository, but tags make it easier. The benefit of tags is to mark the repository for a specific reason, e.g., with a release. Branches and tags are named pointers, the difference is that branches move when a new commit is created while tags always point to the same commit. Tags can have a timestamp and a message associated with them.
URL	A URL in Git determines the location of the repository. Git distinguishes

Term	Definition
	between *fetchurl* for getting new data from other repositories and *pushurl* for pushing data to another repository.
Working tree	The *working tree* contains the set of working files for the repository. You can modify the content and commit the changes as new commits to the repository.