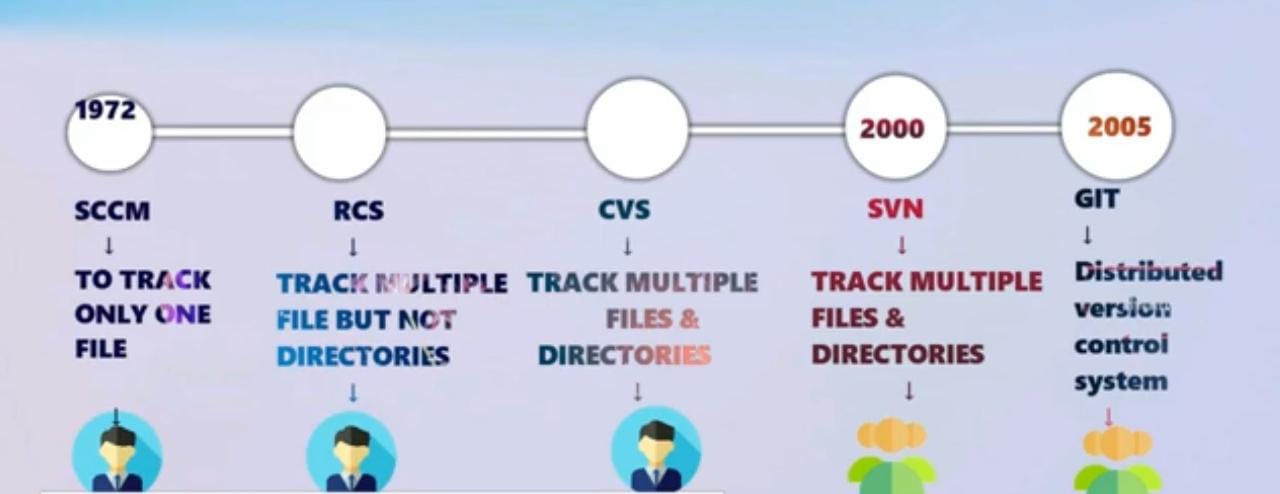
**GIT**

* It is a version control system (VCS) or source code management (SCM).
* It is used for track the changes in files.
* It will maintain multiple versions of same file.
* It is platform independent.
* It is free and open-source.
* They can handle larger projects efficiently.
* They save time and developers can fetch and create pull requests without switching.

**VCS HISTORY**



**Revision Control System**

* is an early version control system (VCS). It is a set of UNIX commands that allow

users to develop and maintain program code or documents. With RCS, users can make their own revisions of a document, commit changes, and merge them.

* It will track only Multiple files but not Directories.
* Allowed for single user only.

**Concurrent Versions System**

* CVS is a version control system, an important component of Source Configuration Management (SCM). Using it, you can record the history of sources files, and documents. It fills a similar role to the free software RCS, PRCS, and Aegis packages. CVS is a production quality system in wide use around the world, including many free software projects.
* Tracks Multiple files and Directories.
* Allowed for single user only.

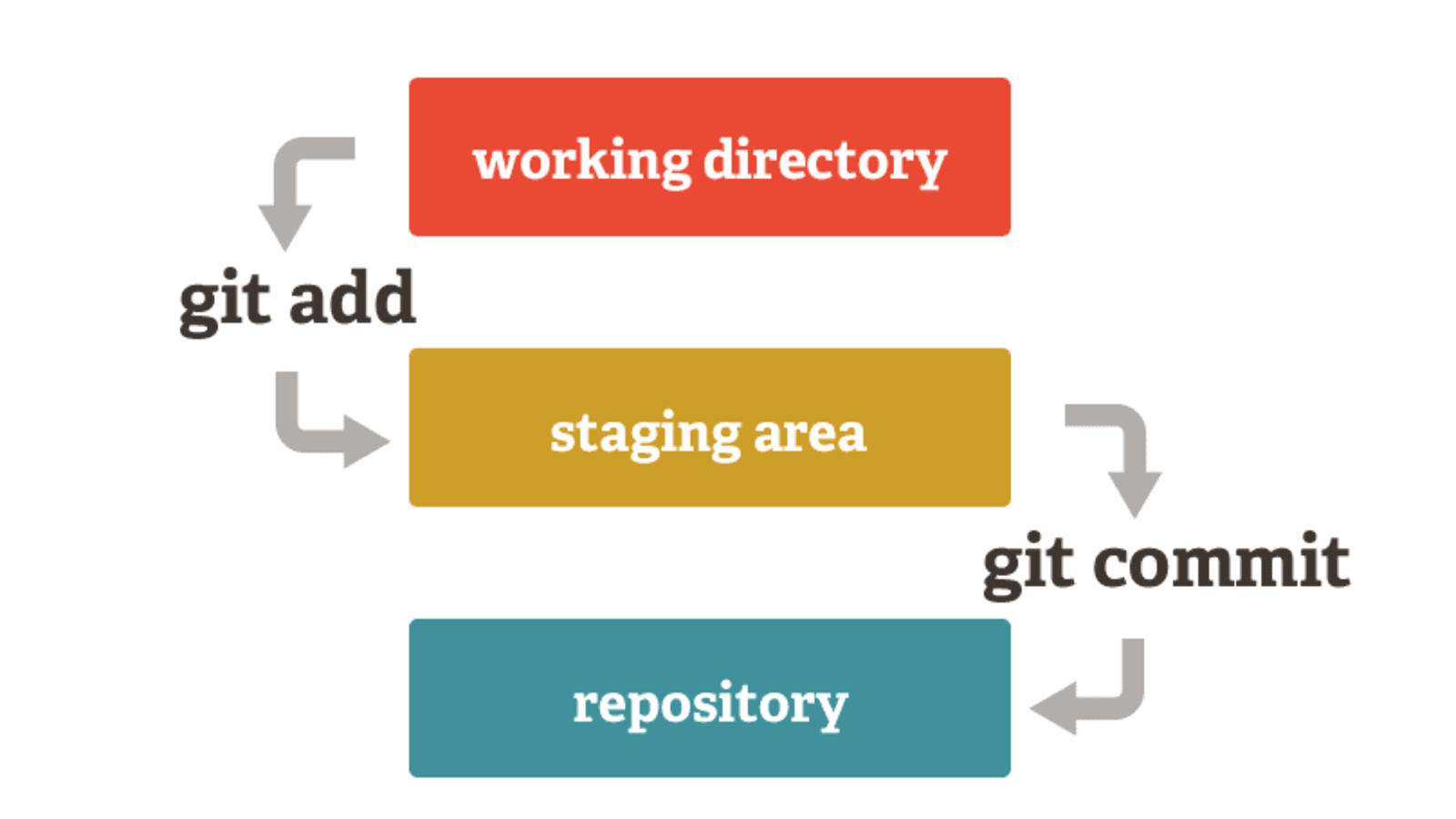
**Subversion**

* SVN is an open-source centralized version control system that is available for everyone at zero cost. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work among programmers. The version control allows you to track and work together with your team members at the same workspace.
* Allowed Multiple users.

FREE : GIT, SVN

PAID : BITBUCKET, P4, STASH

**GIT STAGES**



**WORKING DIRECTORY**

* In this stage git is only aware of having files in the project.
* It will not track these files until we commit those files.

**STAGING AREA**

* The staging area is like arough draft space, it's where you can git add the version of a file or multiple files that you want to save in your next commit.
* In other words, in the next version of your project.

**REPOSITORY**

* Repository in Git is considered as your project folder.
* A repository has all the project-related data.
* It contains the collection of the files and also history of changes made to those files.

**TYPES OF REPOS**

**LOCAL REPO:**

The Local Repository is everything in your .git directory. Mainly what you will see in your Local Repository are all of your checkpoints or commits. It is the area that saves everything (so don’t delete it).

**REMOTE REPO:**

**The remote repository is a Git repository that is stored on some remote computer.**

**CENTRAL REPO:**

This will be present in our GITHUB.

**GIT INSTALLATION**

* yum install git -y
* git init .

**STEPS TO COMMIT A FILE**

1. Create a file : touch filename
2. Now add that file : git add . (Dot represents current directory)
3. commit the file with message : git commit -m "commit message you want" filename
4. To see details of that file : git log

Now all those things will be done under root user

if you want to done by another user or as under your name we need to configure it.

**GIT ADD:**

* Git add command is a straight forward command. It adds files to the staging area.
* We can add single or multiple files at once in the staging area.
* Every time we add or update any file in our project, it is required to forward updates to the staging area.
* The staging and committing are co-related to each other

**GIT COMMIT:**

* It is used to record the changes in the repository.
* It is the next command after the git add.
* Every commit contains the index data and the commit message.

**GIT STATUS:**

* The git status command is used to display the state of the repository and staging area.
* It allows us to see the tracked, untracked files and changes.
* This command will not show any commit records or information.

**CONFIGURATION OF USER**

if you want to give your username and E-mail id to those commits then,

* git config user.name "username"
* git config user.email "userxyz@gmail.com"

now give the git log command to see changes, it won't work because after configure we haven't done any thing

Now create a file and commit that file and give git log you will see changes as you configure.

**IGNORING CONTENT**

It will be useful when you don't want to track some specific files then we use a file called .gitignore

create some text files and create a directory with "jpg" files.

* vi .gitignore
* \*.txt

now all the txt files will be ignored

**GIT CLONING**

It means having same files in another folder.

To clone a git repo we need to have a repository and also check our present working directory.

* git clone repurl

now we just cloned the files in repo-A to repo-B.

But before cloning we need to add and commit our files.

**GIT STASH:**

Using the git stash command, developers can temporarily save changes made in the working directory. It allows them to quickly switch contexts when they are not quite ready to commit changes. And it allows them to more easily switch between branches.

Generally, the stash's meaning is "**store something safely in a hidden place**."

* Touch file1 & git stash & git stash save “message” & git stash list

**GIT BRANCHES**

A branch represents an independent line of development.

The git branch command lets you create, list, rename, and delete branches.

The default branch name in Git is master.

* To see current branch : git branch
* To add new branch : git branch branch-name
* To switch branches : git checkout branch-name
* To create and switch at a time : git checkout -b branch-name
* To rename a branch : git branch -m old new
* To clone a specific branch : git clone -b branch-name repo-URL
* To delete a branch : git branch -d <branch>

The -d option will delete the branch only if it has already been pushed and merged with the remote branch. Use -D instead if you want to force the branch to be deleted, even if it hasn't been pushed or merged yet. The branch is now deleted locally.

Now all the things you have done is on your local system.

Now we will go to GIT HUB.

**GIT-HUB**

* Github is a web-based platform used for **version control**.
* it simplifies the process of working with other people and makes it easy to collaborate on projects.
* Team members can work on files and easily merge their changes in with the master branch of the project.

Now if you want to pull your code to Github

**git remote add origin url**

**git push -u origin branch-name**

go to Github and check the files that you have pushed.

**GIT MERGE**

* If you want to merge branch-1 with branch-2 switch to branch-1 first and give command

git merge branch-2

* now that command had merged the content of branch-1 to branch-2
* Whatever the content in branch-1 will be seen in branch-2 now.

**GIT FORK**

* A fork is **a rough copy of a repository**. Forking a repository allows you to freely test and debug with changes without affecting the original project

**Advantages**

* Speed
* Simplicity
* Fully Distributed
* Excellent support for parallel development, support for hundreds of parallel branches.
* Integrity

**DISADVANTAGES**

* Windows support issue.
* Entire download of the project history may be impractical and consume more disk space if the project has long history.

