# Git training manual

pwd – print working directory

cd – change directory

cd .. – move to the directory before the current one

cd "directory name" – to move to a particular directory

ls – print directory elements

ls -l – print directory elements line by line

ls -a – print all directory elements including hidden filles

ls -la – print all directory elements including hidden files line by line

mkdir – create a directory

touch – create an empty file

mv – move a file

rm – remove a file

rm -r – remove a folder

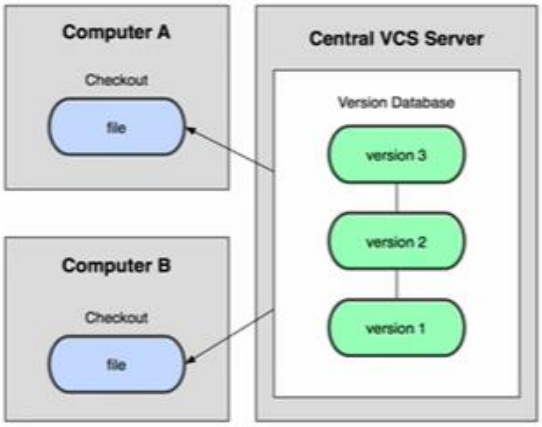
cp – to copy file

git init – to create a local repository

**Features of version control systems:**

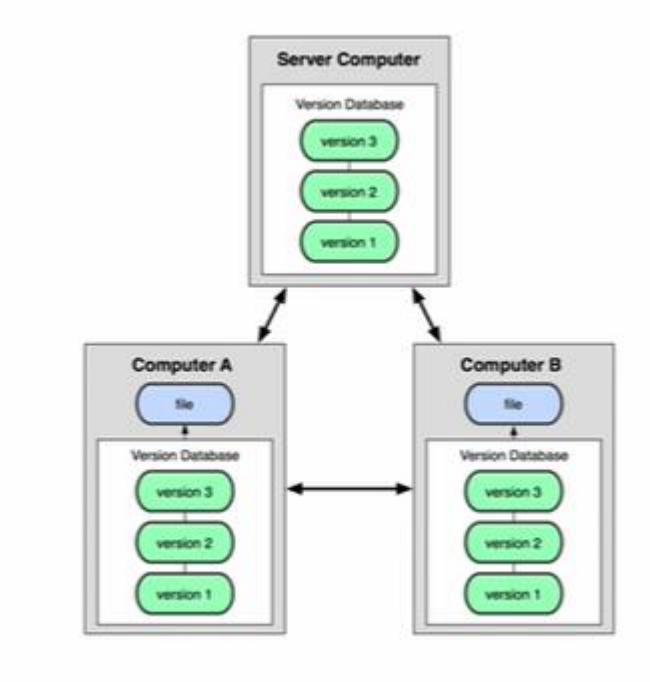
- File change history tracking
- Ability to restore any version of the file
- Synchronization of changes introduced by various authors in a distributed environment

1. **Centralized version control systems e.g., concurrent version system CVS**



Server and client are present, download the update from the server (main repository) to the local computer, after changes commit to the main repository (source code) from your local computer. Any lost to the history of the central server is a risk.

2. **Distributed version control systems e.g GIT**



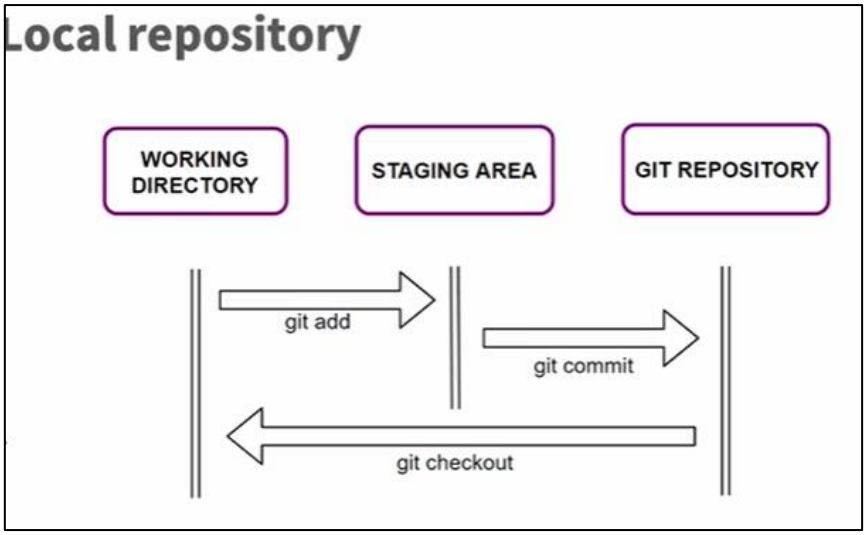

Client will have a copy of the version of the repository, clients clone the entire history to their hard disk, make changes, commit the changes to their local hard disk, then to align them with the remote repository, they must push the changes to the remote repository.

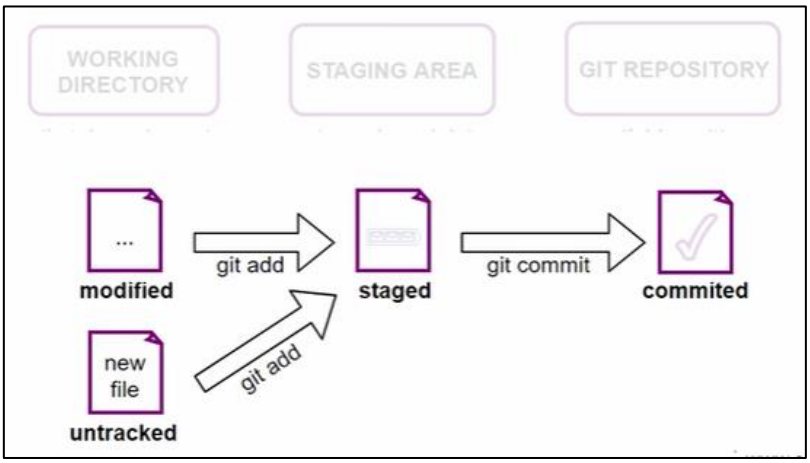
**Advantages:**

- you can make changes without connecting to a remote server
- speed of work - unlike centralized systems, you don't have to communicate with a remote server with every command
- data security - each of the developers has a local copy of the repositories

Git is a distributed version control system which helps us to manage the source code by offering:

- Speed
- Efficient work with large projects
- Full dispersion




**git status** – to check the status of the git repository

vim "filename.txt" then press i – to edit the file

esq : w q – to exit vim file.txt

**git add** – to add a file to the git repository for tracking

**git add .** – if adding more than one file

**git commit -m "first commit – adding "filename""** – to commit the file that was added

**git config** – global user.email "your email"

**git config** – global user.name "your name"

**git config** --list – print current git configuration

**git reset** – rolls back files from stagging area to working directory

**git checkout/restore** – restores the state of the file to the form saved by git

**git rm** – removes specified file/directory

**git log** – allows you to list the commits in order from newest to oldest

**git log --patch** – displays the changes made in each commit

**git log --stat** – displays the statistics e.g., the number of lines changed

**git log --oneline** – displays each commit on one separate line

**git log --author=”authorname”** – displays the changes made by the author only.

Also you can use git log --author=”authorname” --oneline, git log --author=”authorname” --stat, git log --author=”authorname” –patch, or git log --author=”authorname” --oneline --patch or git log --author=”authorname” --oneline --stat.

**git log -n** – to display n number of commits

**git revert <commit>** - allows reversing changes introduced to a specific commit

**git remote -v** – to check the current remote repository.

**.gitignore file** - allows you to define the patterns that git looks for in the file path

**git branch** – to display the available branches

**git pull origin** - update your local version to synchronize with the associated remote branch

**git push <remote_name> <branch_name>** - update a remote repository with the changes you've made locally. It is used to push your local branch commits to a remote repository

**git push origin HEAD** – This pushes the changes from the current local branch to a branch with the same name on the remote repository.

**git branch <branch_name>** - to create a new branch

**git checkout <branch_name>** - to select branch name to work on

**git checkout -b <branch_name>** – to create and switch to a new branch

**cat <filename>** - to print the content of a file

**git branch -d <branch_name>** - to delete the selected branch or

**git branch -D <branch_name**> - to delete the selected branch when not fully merged.

**git merge –no-ff –no-commit/--commit <branch_name)** – to avoid automatic fast-forward when merging branches

**git merge --abort** – to abort merging branches

**git rebase <branch_name>** - allow you to change the base of the branch we are on to the branch given in the command

**git rebase -I <branch_name>** - launches the interactive rebase creator. One can change commits in it, including joining them together, changing their messages.

**to create a file and commit it:** touch "filename" then git add "filename" then git commit -m "comment on the process or changes"

**to remove file and restore it:** git rm "filename" then git reset HEAD"filename" then git checkout "filename"

**to stash a WIP file to go back to it later:** on the working branch(e.g test branch), edit file, go back to master branch, use git stash to save the WIP file, got back to test branch, git status → git stash apply → git stash pop → git add . → git commit -m "comment" → go back to master branch → create an experimental branch using git checkout -b experimental → git stash list → git status → git stash pop → git stash list (it will show empty).

git fetch origin master:tmp
git rebase tmp
git push origin HEAD:master
git branch -D tmp

**To change the url on git:**
git remote set-url --push origin url (git url)
git push -u origin master
git push -u –all
**git pull origin master**

.gitignore:

To exclude directories: directoryName/* (e.g., drinks/*)

To exclude a file extension: *fileExtention (e.g., *.bat)

To exclude the gitignore file: .gitignore

# Advantages and disadvantages of rebase

**Disadvantages:**

- Collecting several commits into one can distort the context of work.
- Performing a rebase in a public repository can be dangerous if you work as a team.
- A little more work - you need to use rebase to keep branch updated.

```
Commands:
p, pick <commit> = use commit
r, reword <commit> = use commit, but edit the commit message
e, edit <commit> = use commit, but stop for amending
s, squash <commit> = use commit, but meld into previous commit
f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
                   commit's log message, unless -C is used, in which case
                   keep only this commit's message; -c is same as -C but
                   opens the editor
x, exec <command> = run command (the rest of the line) using shell
b, break = stop here (continue rebase later with 'git rebase --continue')
d, drop <commit> = remove commit
l, label <label> = label current HEAD with a name
t, reset <label> = reset HEAD to a label
m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
        create a merge commit using the original merge commit's
        message (or the oneline, if no original merge commit was
        specified); use -c <commit> to reword the commit message
u, update-ref <ref> = track a placeholder for the <ref> to be updated
                      to this position in the new commits. The <ref> is
                      updated at the end of the rebase

These lines can be re-ordered; they are executed from top to bottom.

If you remove a line here THAT COMMIT WILL BE LOST.

However, if you remove everything, the rebase will be aborted.
```

# Stash

| Command | |
|---|---|
| **git stash** | allows you to 'put' introduced changes aside, without having to commit them. |
| **git stash list** | lets you see a list of saved changes in the stash. |
| **git stash apply** | allows you to re-apply the changes you have recently stashed. This option only integrates changes, they will still be listed in the stash list. |
| **git stash pop** | applies recently stashed changes and then removes them from the stash list. |