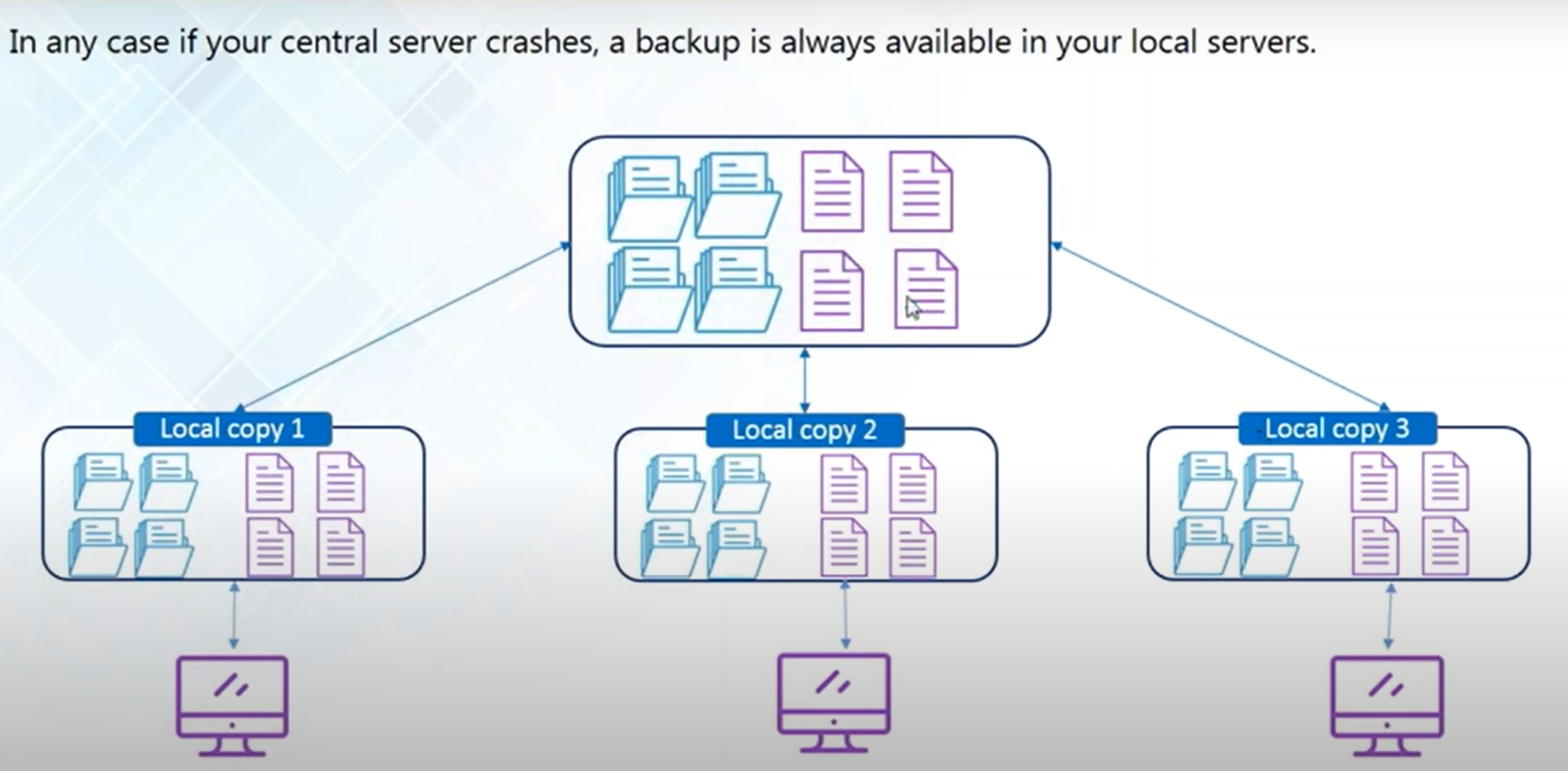
Version Controller System (VCS)

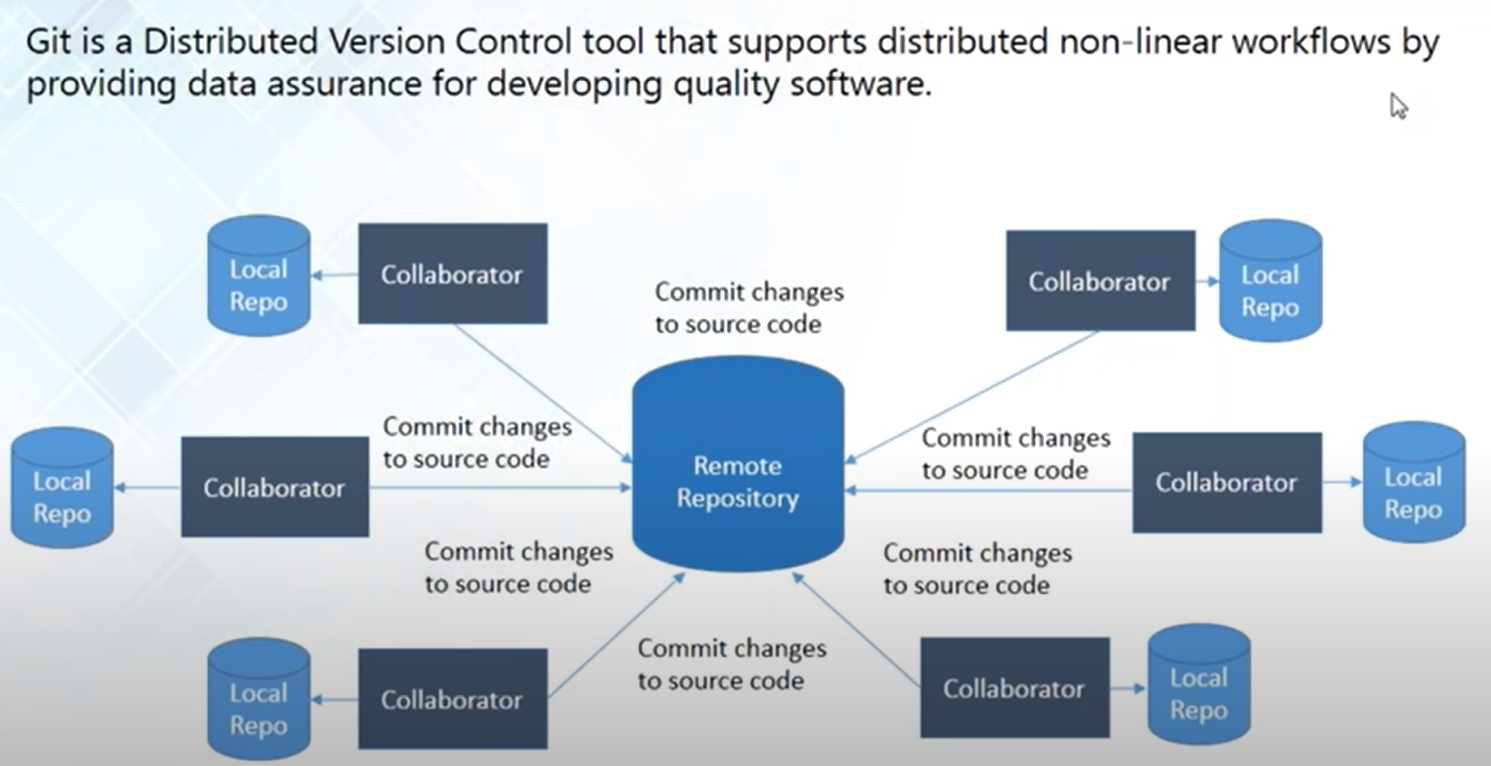
It’s a version control system so every time you change in system it creates the snapshot (entire project backup) for the version. It stores versions in local system as well as git storage.



**Version control tools:**

* GIT (Global Information Tracker): local as well as remote copy
* SVN (Apache Subversion): no local copy
* CVS (Concurrent Versions System): no local copy
* Mercurial: similar to git

**GIT**

****

**Why GIT?**

* Version control system to maintain the code.
* Collaboration to work with team.
* Shared workplace.
* Who made changes in large project?
* Version compares.
* Analyze of your project and timeline.

Git features:

* **Distributed**: Located in different systems.
* **Compatible**: cross platform and backward compatible also can be easily work with other VCS.
* **Non-Liner**: it does not progress or develop smoothly from one stage to the next in a logical way. Instead, it makes sudden changes, or seems to develop in different directions at the same time.
* **Branching**: you can create different branch for the development/production/testing etc.
* **Lightweight**: designed to have small memory footprint, are easy to implement, and/or have minimalist syntax and features.
* **Speed**: Improve your work speed by stopping manually transfer data.
* **Open** **source**: Can be used by anyone without owing the license.
* **Reliable**: you don’t have to worry about system failure or other data lost issues.
* **Secure**: Pen drive share could cause you virus.

Repository: A directory or storage space where your project can live. It can be a local folder in your computer or GitHub storage on the cloud.

Files in GitHub are stored in compressed format when you pull them, they get decompressed in your system automatically.

* Central repository
* Local repository

**Git vs Git hub:**

* Git is a software -> GitHub is a service
* Git can be installed locally on the system -> GitHub is hosted on the web
* Provides a desktop interface called git GUI -> Provides a desktop interface called GitHub Desktop.
* It does not support user management features -> Provides built-in user management

**Fork**

The fork is the process when a copy of the repository is made. It's usually experimentation in the project without affecting the original project. They’re used to advise changes or take inspiration from someone else’s project.

Create repository

* Using GitHub website or GitHub API.
* GitHub API are https rest api’s which help you to gain more control on git operations (<https://docs.github.com/en/rest>)

Git Init: (initialize local git repository)

…or create a new repository on the command line

echo "# sample" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin <git link>

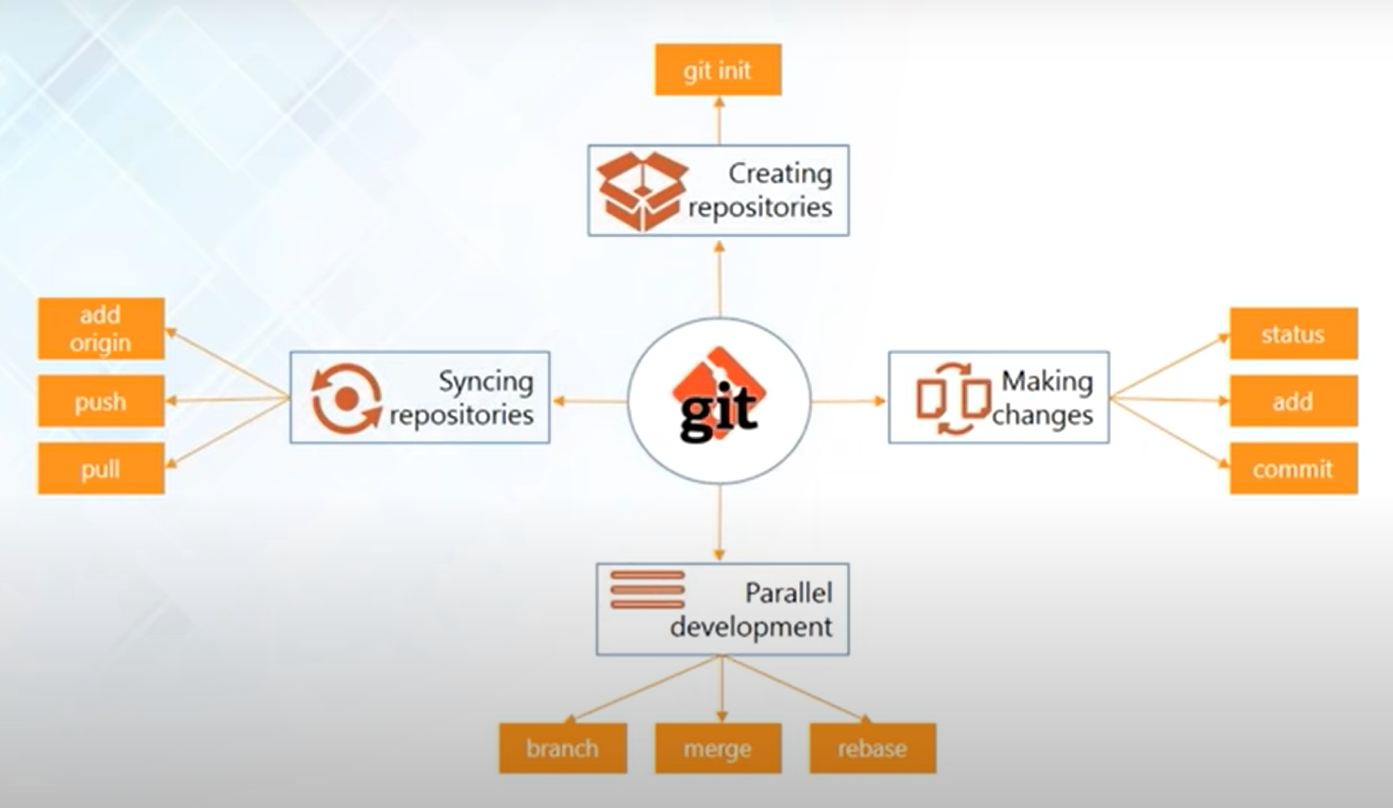
git push -u origin main

…or push an existing repository from the command line

git remote add origin <git link>

git branch -M main

git push -u origin main



**GIT Token (git password support deprecated) & Config**

* Go to settings in git hub.
* On the left side menu, you will fine developer settings in the bottom.
* Left side menu, select personal access token.
* Click on generate new token.
* Select repo.
* Click generate.
* Copy the key. This will not be visible again to make a copy of it before you reload the page.
* Using CMD: git config --global user.email "you@example.com"
* Using CMD: git config --global user.name "Your Name"

**Sync**

* Creating repository: using git web app.
* Linking repository: use git remote add origin <link>
* git pull origin master (name of repo)
* git pull <remote> <branch>
* If you wish to set tracking information for this branch you can do so with:
  + git branch --set-upstream-to=origin/<branch> master
* If you are on cmd, by default, git create branch name master.
* If you want to change it, just create new branch then switch to that branch and then use git push --set-upstream origin <new\_branch\_name>

**changes**

* git status: to check the file status.
* git add .: add file to local repository index.
* git commit -m “comment”: add file to local repository.
* git push origin master: push your project to central repo.
* Untracked files: the files which are not in the index. To fix it just use git add .
* git log: to check commit logs and related data like date and author.

**Delete**

* git branch -d <name\_of\_branch>
* git branch -D <name\_of\_branch> (force delete including unmerge changes)

**Parallel Development**

**Branches:**

* Branches are pointer to a specific commit.
* Branches are two types
  + Local Branch
  + Remote Tracking Branch

**Create branch:**

* git branch <name of branch> (from default branch
* git checkout -b <name\_of\_parent\_branch> <name\_of\_branch> (from specific branch)

**Switch branch:**

* git checkout <name of the branch>
* switch to a new branch or master branch to make changes accordingly.

**git stash:** breakup your branch and store it differently. And if you want to go into that use **git stash apply** and reattach those files back to your branch.

Use case: Suppose if you have made some changes in file and the same file had been changed and pushed by your team member for the same branch. If you will pull that branch it will show an error that you have unmerge changes. You can make your changes to separate then pull the code after that you can recover using git stash apply.

**Merging**

* it’s a way to combine the work of different branches together.
* Allows to branch off develop a new feature & combination it back in.
* Merge is use to merge your branches to your master branch (could be production or development as per requirement)

**How to Merge**

* Switch to the branch you want to get merge with like master/production/parent branch
* git checkout <production/master/parent>
* git merge <name\_of\_branch\_whose\_work\_you\_want\_to\_merge>
* git push (send your updated branch to central repo)

**Rebase**

* This is also a way to combining the work between different branches.
* It can be used to make a linear sequence or commit.
* All the commit that you did in your branch will be reattached to head in the parent branch. So suppose you have a branch which is not required anymore so you can rebase it with it’s parent branch. And remove that branch. (removing is not compulsory)

**How to Rebase**

* Switch to the branch you want to get merge with like master/production/parent branch
* git checkout <production/master/parent>
* git rebase <name\_of\_branch\_whose\_work\_you\_want\_to\_rebase>
* git push (send your updated branch to central repo)

**Expert Branch System**

Master/Main Branch

* The updated branch and can be directly use in production aka production branch.

Development Branch

* The branch with latest comments from all the other branches and tested properly then we can update its status to master branch once everything seems fine.

Team branch:

* Can be user specific or work specific for particular changes and once the code it verified this can be marge with development branch then main branch after testing.

All commands:

* git --bare init

Some useful error:

**fatal: not a git repository (or any of the parent directories)**

* Command
  + git status
* **Cause**: The error indicates that the command cannot be executed because the current directory is not a Git directory.
* **Reason**:
  + You either forgot to initialize the repository as a Git repository using git init
  + or you are probably in the wrong directory.
* Solution:
  + Use git init
  + Check directory path.

**fatal: refusing to merge unrelated histories**

* Command
  + git pull
  + git merge
* **Cause**: The error indicates that the command cannot be executed because the .git directory has some unrelated files.
* **Reason**:
  + The .git directory got corrupted. This hidden directory contains information relevant to your project that is being version controlled (for example, information about commits, remote repositories, etc.). If the directory is corrupted, git is unaware of the local history and thus the error will be reported when trying to push or pull to/from a remote repository.
  + Most common reason is that you just made commit by using git add and git commit and then you are trying to pull the same directory. Now git will say that I am not aware of those changes.
* **Solution**:
  + git pull origin master --allow-unrelated-histories
  + or you are probably in the wrong directory. You can check your current work directory using pwd. It’s quite common for people to forget to change the directory when cloning a Git repository.
* git branch <name> : creating new branch without any commit will give you error.
  + fatal: Not a valid object name: 'master'
* git push for newly created branch: The current branch <name> has no upstream branch. To push the current branch and set the remote as upstream, use
  + git push --set-upstream origin <branch\_name>
  + to verify upstream branch use -> git branch -vv

**fatal: rebase does not get aborted but is still in progress**

* Command
  + git rebase
* **Cause**: The error indicates that the command cannot be executed because the rebase directory is not properly understanding each other.
* **Reason**:
  + Your local directory has some changes not updated into your central library.
  + Your child directory is not able to merge with the parent library.
* **Solution**:
  + To solve the issue first merge then commit then force rebase.
  + git merge --allow-unrelated ORIGINAL\_BRANCH\_THAT\_WAS\_MERGED --no-commit
  + git commit -C ORIGINAL\_MERGE\_COMMIT
  + git rebase –continue

**fatal: Not a valid object name: 'master'**

* Command
  + git branch <name>
* **Cause**: The error indicates that the command cannot be executed because you are creating a branch in new project without any commit.
* **Reason**:
  + Creating new branch without any commit will give you the error.
* **Solution**:
  + To solve the issue first git add then git commit then git branch <name>.

**fatal: The current branch <name> has no upstream branch**

* Command
  + git push
* **Cause**: The error indicates that the command cannot be executed because you are pushing local branch to central repository without telling them the link.
* **Reason**:
  + Push newly created local branch to central repo.
* **Solution**:
  + git push --set-upstream origin <branch\_name>
  + to verify upstream branch use -> git branch -vv

**Basic GIT Commands**

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

shortlog Show summary of logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

revert send you back to the previews commit.

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch (fetch+merge)

push Update remote refs along with associated objects

**Advance Git Command**

* **git bisect**: The Git bisect command helps you to find bad commits. With Git bisect you can narrow down the broken code within a few minutes. Suppose you have an application where some functionality was working a week ago but now it’s not working. In that case git bisect can help you.
  + git bisect start : To start the git bisect
  + git bisect good a123 : let git bisect know about a good commit
  + git bisect bad z123: let git bisect know about a bad commit
* How to use it ?
  + git log (to check all the commits log)
  + git bisect start
  + if current commit is not working mark it (git bisect bad)
  + git log
  + check which commit is working.
  + mark that (git bisect good)
* **git cherry-pick**: Git cherry-pick is a helpful command. It's a robust command and allows you to pick any commit from any branch and apply it to any other branch.
  + Git cherry-pick doesn’t modify the history of a repository; instead, it adds to the history.
* How to use it?
  + git cherry-pick <commit-hash>
* **git archive**: Git archive command will combine multiple files into a single file. It's like a zip utility, so it means you can extract the archive files to get individual files.
  + It will create a zip archive of the current revision.
* How to use it?
  + git archive --format zip HEAD > archive-HEAD.zipcherry-pick <commit-hash>
* **git pull --rebase**: Most of the time, you need to do rebase (and no merge) when you use Git pull.
  + it will help you to keep the history clean. Also, you can avoid multiple merges.
* How to use it?
  + git pull --rebase
* **git blame**: If you need to examine the content of any file line by line, you need to use git blame. It helps you to determine who made the changes to a file.
* How to use it?
  + git blame <file\_name>
* **git tag**: In Git, tags are helpful, and you can use them to manage the release. You can think of a Git tag like a branch that will not change. It is significantly more important if you are making a public release.
* How to use it?
  + git tag -a v1.0.0
* **git verify-commit**: The git verify-commit command will check the gpg signature. GPG or “GNU Privacy Guard” is the tool used in sign files and contains their signatures.
* How to use it?
  + git verify-commit <commit>
* **git verify-tag**: In the same way, you can confirm a tag.
* How to use it?
  + git verify-tag <tag>
* **git diff**: Most of the time, you need to compare two git files or branches before you commit or push. Here is a handy command to do that.
* How to use it?
  + To compare the working directory with the local repo:
    - git diff HEAD <filename>
  + to compare two branches:
    - git diff <source branch> <target branch>
* **git citool**: Git citool is a graphics alternative of the Git commit.
* How to use it?
  + git citool
* **git mv**: To rename a git file. It will accept two arguments, source and target file name.
* How to use it?
  + git mv <old-file-name> <new-file-name>
* **git clean**: You can deal with untracked files by using the Git clean command. You can remove all the untracked files from your working directory by using this command. In case you want to deal with tracked files you need to use the Git reset command.
* How to use it?
  + git clean
* **git help**: There are many commands in Git, and if you need more help with any command, you can use git help at any time from the terminal.
* How to use it?
  + git help
* **git whatchanged**: This command does the same thing as git log but in a raw form. And it’s in the git because of historical reasons.
* How to use it?
  + git whatchanged