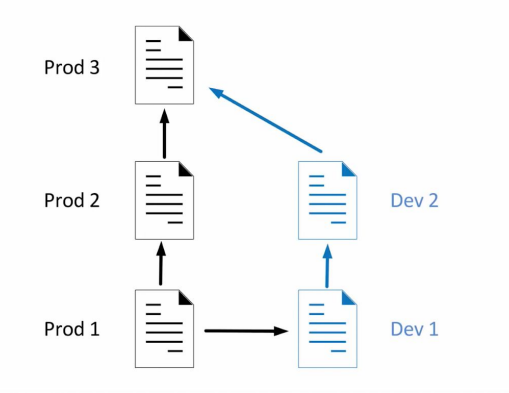
# Introduction  to Git Crore Topics:

## Central Git Concepts:

A Git is a kind of VCS (version control system) that helps us document any modification to our files.Git works well specially with text files.
Tasks that can be done using Git:

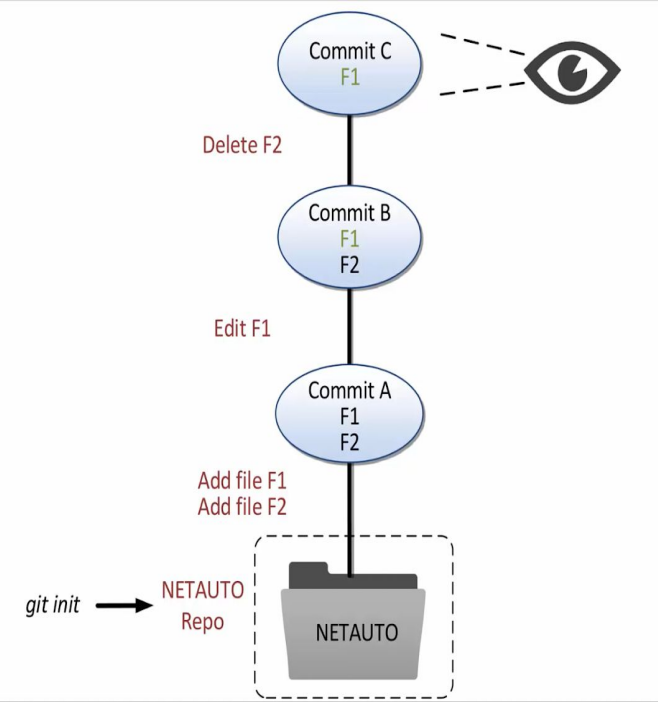
- Take shots of files
- Brings back previous versions of files we made earlier
- Work on multiple versions of files simultaneously for example working on a document and keeping a fresh copy aside and work on another of the same kind. Ultimately when we feel ready, we can merge them into the same file like in the image below.




## Two basic diagrams:

- Git commit graph
    - Enables us to take snap shots of files at any time. These Snap-shots are called commits.
    - In a repository we start by using the git init command.
    - Let's say we added two files F1 and F2 to our directory and then we decided to take our first snapshot .
    - So this is a saved version of the file we are working on and we can always go back to it.
    - Suppose then we edit F1 and then we take another snapshot
    - For example then we decide to delete F2 and we do commit to update the version
    - In this way we can update our versions using comment and restore it back whenever we want. This is very useful for a large projects.

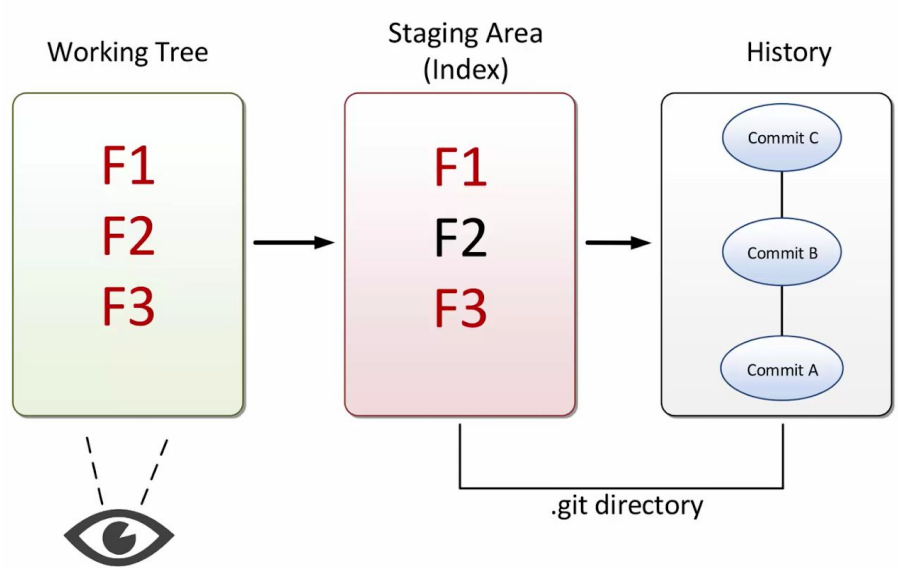
**Commit Graph:**




A commit saves the state of the file at a particular point in time.Usually we make a commit when a logical unit of work is done.

- Three conceptual areas for files:
    - There are three areas, The working tree, the staging area or index and the history.
    - When we add delete or edit files we do that in the working tree
    - They get history is similar to the commit graph we did above.This history is kept in a hidden directory .git. It contains our object database and metadata that make our repo. If we happen to send our .git directory to someone else then that person would have complete access to our full git project and its history including all versions of the file and commits.
    - As we are working on a project we make changes in the working tree. Git Gives us full control over what changes from our working tree we are putting into our next commit. For example if we edit three files F1 F2 and F3 files how is the ever we only want the new versions from those files to be taken snapshots of for our next commit.This control is through staging area
    - We could add the two files we want to the staging area and when it is right we make a commit. The file that was left out can be included in our next/future commits.

**3 area diagram:**




## Hands-on - create a repo:

After successful installation of the gate documentation, we start with a standard directly on our file system
The first go to a new directory and add a file name ,say S1 . For this we can use any text editor like Emacs, vi etc.The command will be ***vi S1***. Then add some text data to indicate few variables for the network switch S1. Save it and exit. Then we have a single file S1 in our new directory. It can hold a git project with the ***git init*** command.

According to the diagram if we see in the working tree we have our first file S1. After running the git init command, It creates a .git  sub directory in our main directory.

Before this it is important to configure our username and email whenever we make a commit includes name ,email and a timestamp with a commit. It is used to track if any changes are made to the project and Are made by which member of the project.

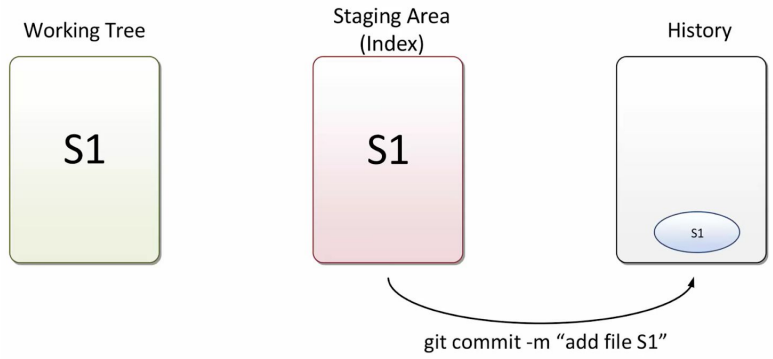
For this we do,
***— global user.name "My name"***
***— global user.email "My@some-email"***
***git config —list***
This ***git config —list*** shows us our name and email that we just set. As we used —global flag with the commands, our name and email will be used in future and we would not have to repeat it.

Coming back to the diagram,As we know that git tracks changes to files over time but for now it is not tracking S1. Soon after we add S1 to our staging area it will be tracked.

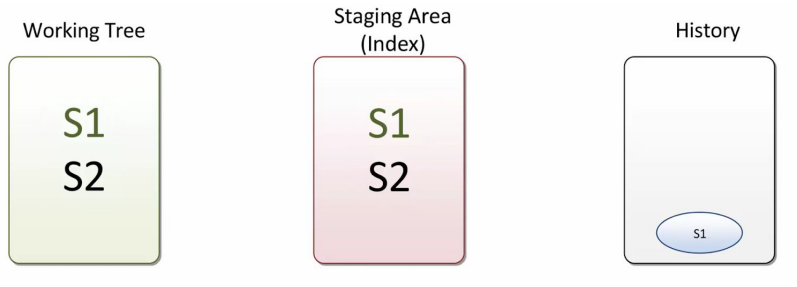
Use:
***git status***
Which will Tell us how things stand in our working tree and in our staging area. We also notice that we are on branch master. It also guide us how to get S1 into the staging area. We stage S1 with a git add command

***git add S1***

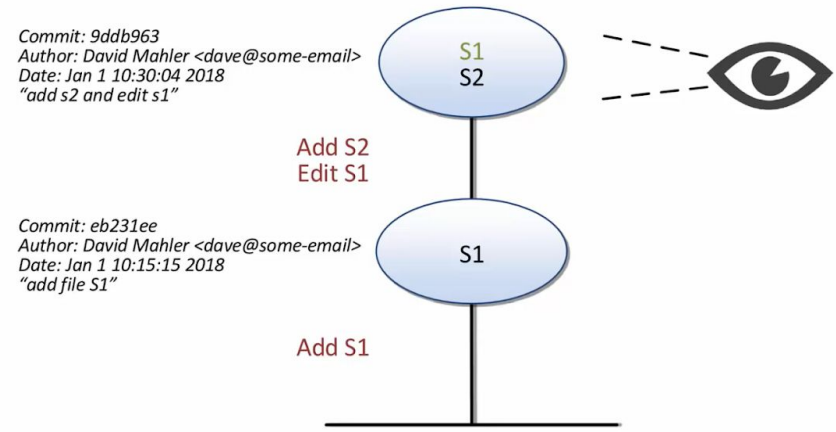
After this command it moved as one into the staging area as desired.We can check it by running its we can check it by running ***git status*** command. The “changes to be committed” indicates that we are in the staging area.
Now git is tracking S1.
It’s time to make our first commit using:
***git commit -m “add file S1”***
This command creates a commit with whatever is in the staging area. The –m gives a short message as a description of what has been changed.
If we check again by ***git status*** command we can see that the working directory is clean which means there is nothing new in our main directory. Every commit has a unique SHA -1 hash.




Similarly we work on a second file S2 using the same steps.After adding it in the working tree we will see that S2 is on track but then S1 is being tried since we previously committed it. We can confirm it by running get status command.
A git diff command shows the difference between tracked files in the working area and the Staging area
To add git files we could do
***git add S1 S2***
Alternatively we could also do
***git add .***
The doubts indicate adding all the new and modified files to our staging area.
To double check it we can always use ***git status*** command.




This looks good, so let us commit using
***git commit -m "add file S2 and edit S1"***
Now we notice that we got a unique hash for the 2nd commit as well




We notice that our most recent comment is at the top and the previous one is below that.

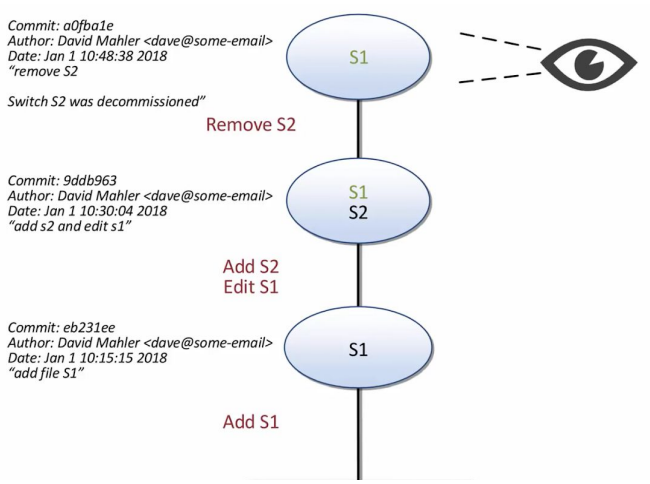
## How to remove a file:

***git rm*** command helps us remove.
We can remove S2 with ***git rm S2***

This command does two things at once:

- It removed S2 from our working tree
- And also staged this removal therefore S2 is removed from the staging area as well

We can then make a commit and check. We will notice that our last commit will have S2 removed from our directory in our commit graph.




***git log*** can be used to see our recent commits.

## How to Undo a working tree change:

- Lets make some changes to our S1
- Then when we look at our 3 area diagram, we notice that S1 has changed in the working tree but not yet updated in the staging area
- ***git checkout --S1***
- This will bring back S1 to how it was previously.
- We can double check using ***git diff*** command which will show that there are no modifications as our working tree is clean.

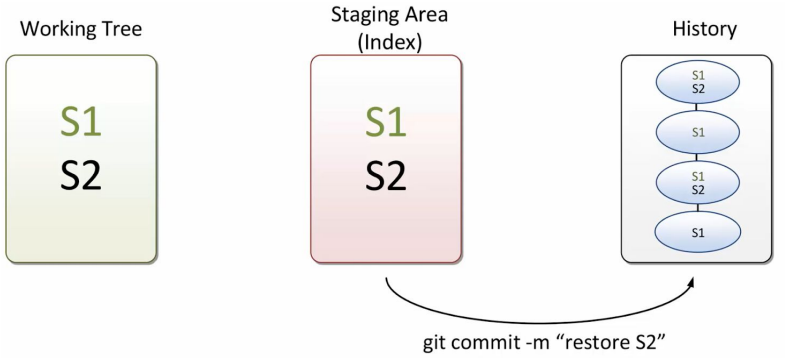
## How to Undo staging of files:

- Edit S1 and stage the change
- ***git diff*** shows our changes
- Perform ***git add S1*** and ***git diff*** will not show anything which means our working tree and staging area match.
- Our last commit does not have the snapshot of our S1 so we can use ***git diff --stage*** that will show us the difference between the staging area and our latest commit.
- Then we can do ***git status*** which will indicate us the latest changes made which can be now committed.
- The ***git reset HEAD S1*** will help us unstage S1.
- We can restore the working tree by using ***git checkout --S1*** command.

## How to restore from an earlier commit:

- As S2 is not in our working tree or staging area anymore, let us try getting S2 back from commit before it was deleted.

- We can notice commits affecting our file by using ***git log -- S2*** . W e can notice from our commit messages of where we added S2.
- Use ***git checkout commit hash -- S2***
- We can check by doing ***ls***
- Now make a commit git ***commit -m “restore S2”***




## .gitignore

- There are git files which we do not need.
- For say there are files like *logs/* and *myapp.pyc* which we want git to ignore
- Use ***.gitignore***
- Use ****.pyc***
- When you check by running ***git status*** , we will notice that it is no more *logs/* and *myapp.pyc* but there is a ***.gitignore file***
- Use ***git add.***
- And ***git commit -m “add .gitignore file”***