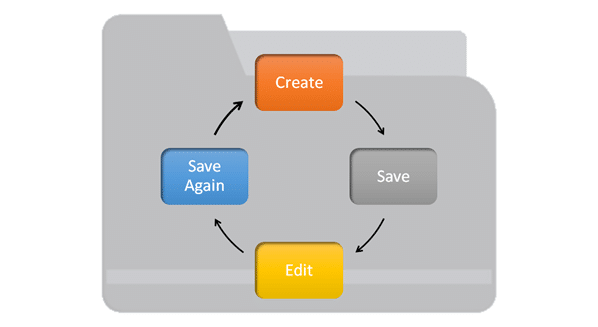
# **Overview of Git**

Be it a Designer, be it a Creator or be it a Developer, we all deal with projects and files every day. Our primary work-cycle revolves around creating a file, saving it, editing or making required changes and saving it again. In this Git Tutorial, we will be focusing on the life of a Developer, and therefore understand **What is Git.**  


# **What is Version Control System?**

**Version Control System** (VCS) is a software that helps software developers to work together and maintain a complete history of their work.

**Following are the goals of a Version Control System.**

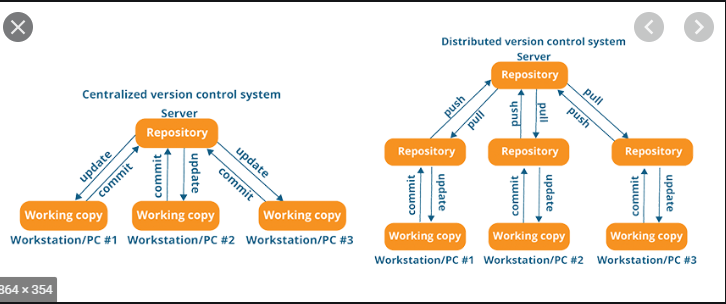
* Allow developers to work simultaneously.
* Do not overwrite each other’s changes.
* Maintain history of every version of everything.

**A VCS (Version Control System) is divided into two categories.**

* Centralized Version Control System (CVCS), and -- SVN
* Distributed/Decentralized Version Control System (DVCS). - GIT

*Note - Subversion (svn) falls under centralized version control system, meaning that it uses central server to store all files and enables team collaboration*

1. First it was local code Repo, where Developers were keeping code but if that laptop is crashed, your code is lost
2. Centralised Repo (CVCS) – where centrally code repo was there – Now if server goes off, code is lost
3. Distributed/Decentralized Version Control System (DVCS) – code is present at Local repo and central repo both



## **What is Git?**

Git helps developers by **keeping track of the history of their codes** files by storing them in versions on its own server repository i.e. GitHub/gitlab. Git has the functionality, performance, security and flexibility that most teams and individual developers need.

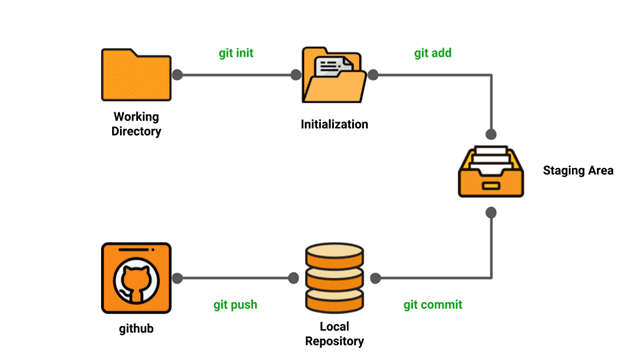
## **Why Git Version Control?**

There are some facts which makes Git so popular:

* **Work offline:** Git provides users the very convenient options of working both online and offline. With other version control systems like SVN or CVS we have to have access to internet to connect to the central repository.
* **Undo your Mistakes:** Git allows you to undo commands for almost every situation. You get to correct the last commit for minor change, also you can revert a whole commit for unnecessary changes.
* **Restore deleted commits:** This feature is very helpful while dealing with large projects, while trying out experimental changes.
* **Secure:** Git provides protection against secret alteration of any files and helps maintaining an authentic content history of the source file.
* **Performance:** Being distributed version control system it has an optimized performance due to its feature like committing new changes, branching, merging and comparing past versions of the source file.
* **Flexibility:** Git supports different kinds of **nonlinear development workflows**, for both small and large projects.

## **Git Lifecycle:**

* Local working directory: First stage of a Git project lifecycle is the local working directory where your project resides, which may or may not be tracked.



* git lifecycle
  + **Initialization:** To initialize a repository we give the command Git init. With this command, we have made the Git aware of the project file in our repository.
  + **Staging area:** Now that our source code files, data files, configuration files are being tracked by Git we will add the files that we want to commit to the staging area by Git add command. This process can be also called as indexing, we can also say that index constitutes of files added to the staging area.
  + **Commit:** Now that our files that need to be committed are ready, we will commit **them using Git commit -m “your message”.**

We have successfully committed files to our local repository. But how does it help in our projects? The answer is, when we will be doing collaborative projects, files may need to be shared with our team members. That is when the next stage of Git lifecycle occurs, which is GitHub, where we publish files from local repository to remote repository

### **Install git**

[root@ip-172-31-80-17 ec2-user]# **yum install git -y**

Loaded plugins: priorities, update-motd, upgrade-helper

Resolving Dependencies

--> Running transaction check

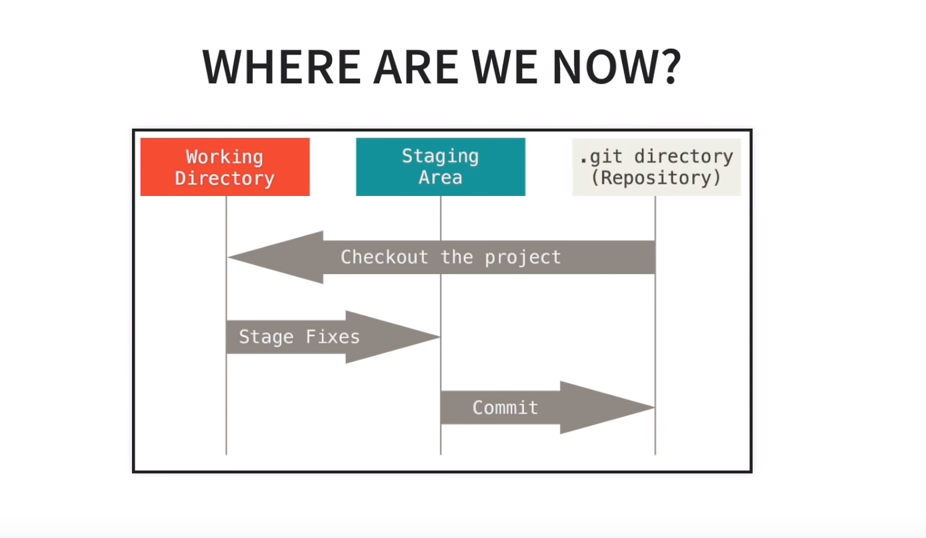
[root@ip-172-31-80-17 ec2-user]# mkdir practice

[root@ip-172-31-80-17 ec2-user]# cd practice/

[root@ip-172-31-80-17 practise]# pwd

/home/ec2-user/practice

# **Creating local repo**



Working Directory – Untrack and modified files will be here. Check all those using git status command and those will come on RED

Staging Directory – We have done a lot of work and created multiple file versions and want to commit only few, we will move those to staging area for commits.

“git init” – This will create a folder .git (local repo) in the folder practice

[root@ip-172-31-80-17 practise]# git init

Initialized empty Git repository in /home/ec2-user/practise/.git/

[root@ip-172-31-80-17 practise]#

[root@ip-172-31-80-17 practise]# git --version

git version 2.14.5

[root@ip-172-31-80-17 practise]# ls -la

total 12

drwxr-xr-x 3 root root 4096 Jan 22 03:41 .

drwx------ 8 ec2-user ec2-user 4096 Jan 22 03:40 ..

drwxr-xr-x 7 root root 4096 Jan 22 03:41 .git

[root@ip-172-31-80-17 practise]#

- This is having everything which is related to repository

-

Help command –

git help config

git config –help

## git config --global –edit (to add email and user name)

[root@ip-172-31-47-114 ~]# git config --global user.name "Rahul Chaudhuri"

[root@ip-172-31-47-114 ~]# git config --global user.email [mailrahulsre@gmail.com](mailto:mailrahulsre@gmail.com)

[root@ip-172-31-47-114 ~]# git config --list

user.name=Rahul Chaudhuri

user.email=mailrahulsre@gmail.com

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

# This is Git's per-user configuration file.

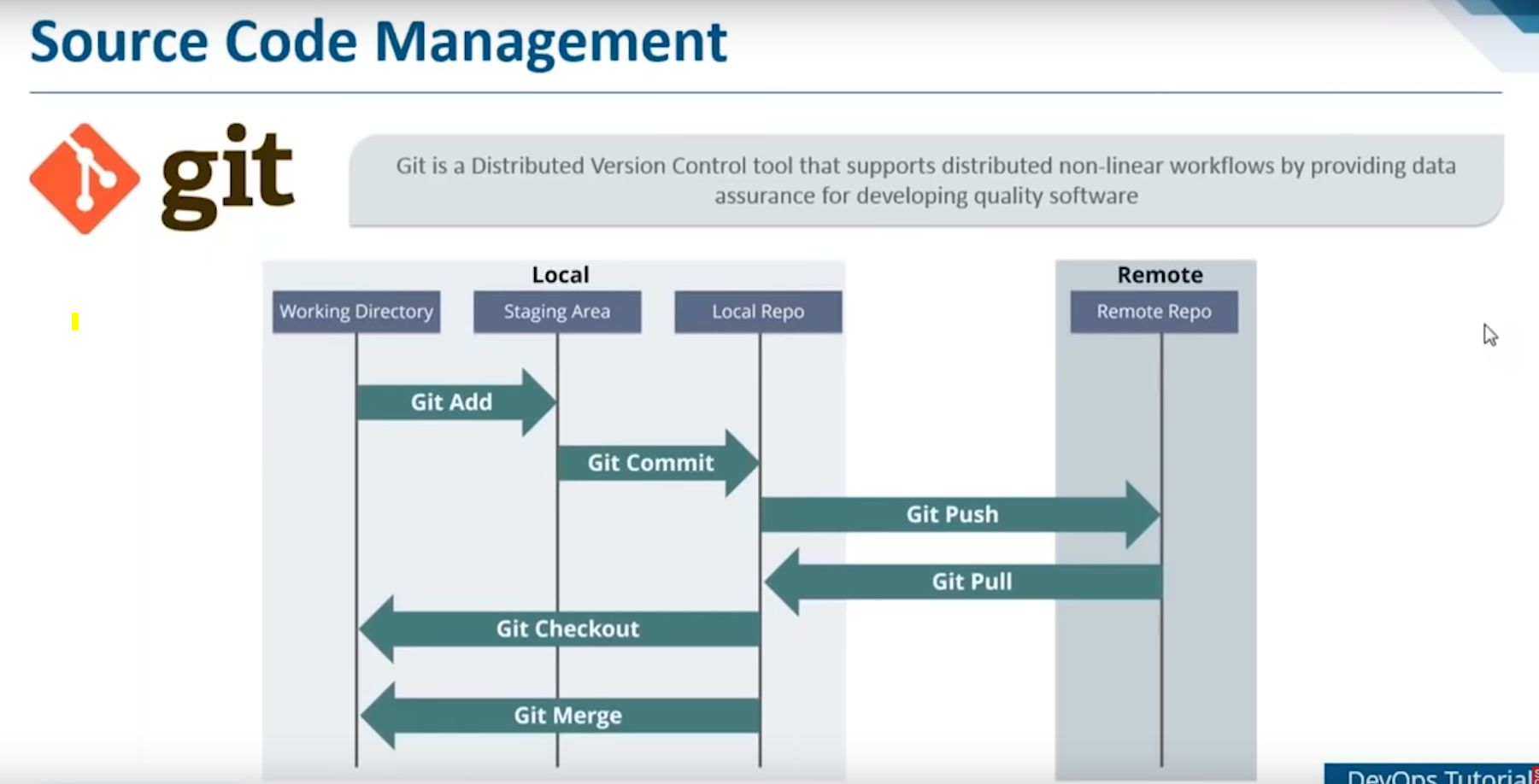
[user]

# Please adapt and uncomment the following lines:

name = rahul

email = rahul.office87@gmail.com

# **Commiting code**



## Create a file and commit.

* Before commit there will be no BRANCH which will be there in the GIT

#### git status

[root@ip-172-31-80-17 practise]# touch jan19.txt

[root@ip-172-31-80-17 practise]# git status

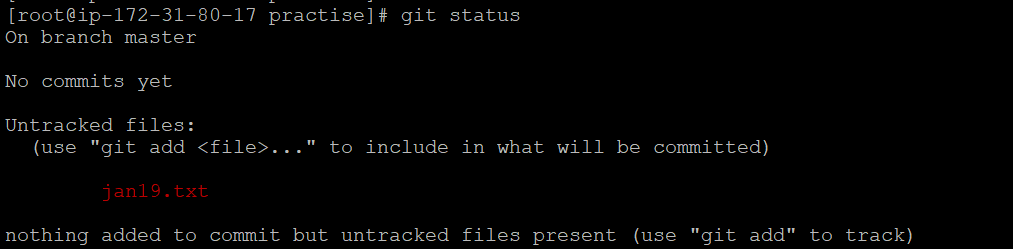
On branch master

No commits yet

**Untracked files:**

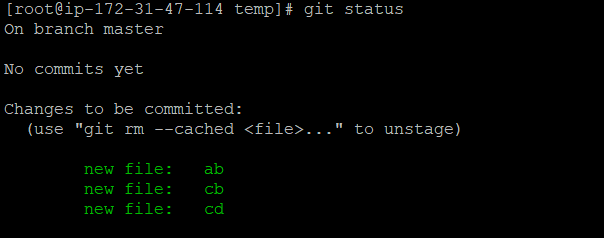
(use "git add <file>..." to include in what will be committed)

jan19.txt  
nothing added to commit but untracked files present (use "git add" to track)



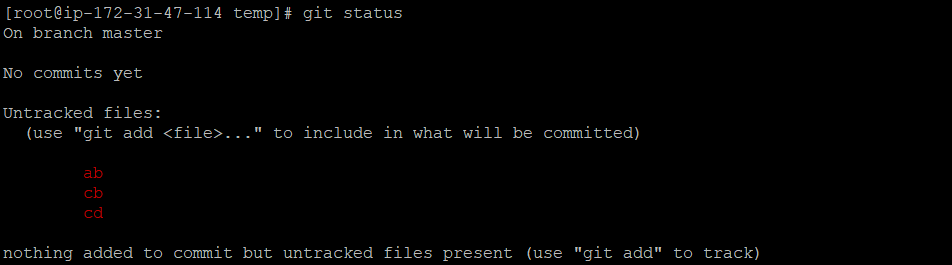
#### Adding to staging area

[root@ip-172-31-80-17 practise]# git add . >> Adding files ffrom Working directory to Staging Area (It should turn to GREEN)



#### Removing from Staging area

git reset



#### commit

[root@ip-172-31-80-17 practise]# git commit . (This is without comment)

[master (root-commit) a40e4d4] first commmit

Committer: root <root@ip-172-31-80-17.ec2.internal>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly. Run the

following command and follow the instructions in your editor to edit

your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 jan19.txt

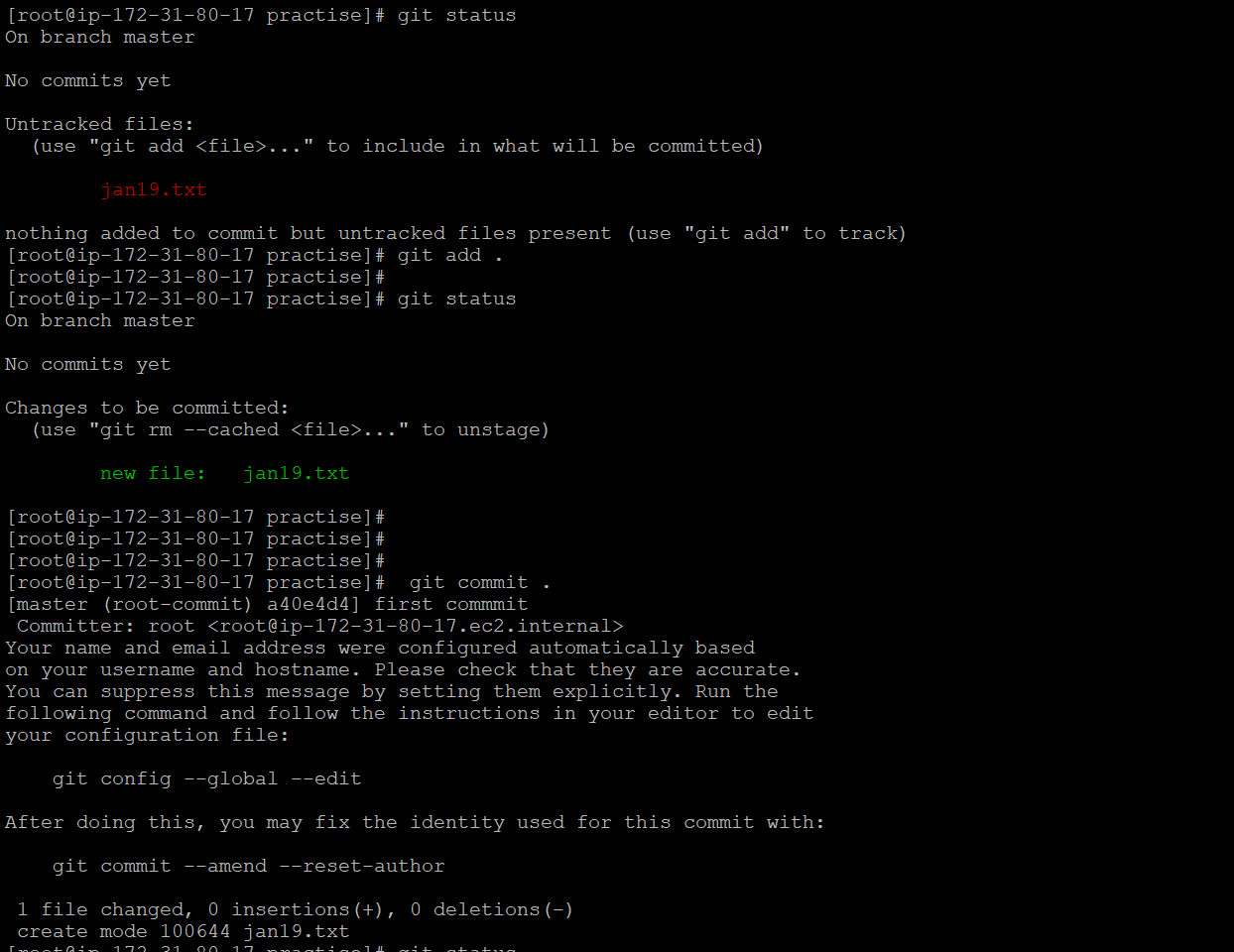
**git commit -a -m “this is for skipping staging area”**

First time we should edit this

[root@ip-172-31-80-17 practise]# git status

On branch master

nothing to commit, working tree clean



#### Commit with comment

[root@ip-172-31-80-17 practise]# **git commit . -m "this is test with comment**"

[master f7a9469] this is test with comment

1 file changed, 1 insertion(+)

## **Log file of commit**

**[root@ip-172-31-80-17 practise]# git log**

commit f7a9469a8b21286f77b2b4de9ee22149587d84c9 (HEAD -> master)

Author: rahul <rahul.office87@gmail.com>

Date: Tue Jan 22 05:50:10 2019 +0000

this is test with comment

commit b34e42ff516ca829e132231c9c39868d0b68a165

Author: rahul <rahul.office87@gmail.com>

Date: Tue Jan 22 05:35:46 2019 +0000

new commit

**[root@ip-172-31-8-148 shellscript]# git log --stat**

commit 347e1e6d668e71d163f58aa01ebe51b1b85c7d43 (HEAD -> master)

Author: Rahul Chaudhuri <mailrahulsre@gmail.com>

Date: Thu Sep 24 03:12:48 2020 +0000

var added

java\_install.sh | 10 ++++++++--

1 file changed, 8 insertions(+), 2 deletions(-)

commit 780d8b6182d01a2eb10d9fceddea15ab4695c4fd

Author: Rahul Chaudhuri <mailrahulsre@gmail.com>

Date: Thu Sep 24 03:02:11 2020 +0000

java\_install.sh file updated

java\_install.sh | 8 ++++++++

1 file changed, 8 insertions(+)

## **git ls-files >>** Listing commited files or files in staging area of git

mkdir test

git ls-files >>nothing should come

touch testfile

git ls-files >> still nothing should come

git add .

[root@ip-172-31-47-114 test]# git ls-files

testfile

# Gitignore file

[root@ip-172-31-47-114 temp]# cat .gitignore

c\*

.git\*

[root@ip-172-31-47-114 temp]# ls

ab cb cd

[root@ip-172-31-47-114 temp]# git status

On branch master

No commits yet

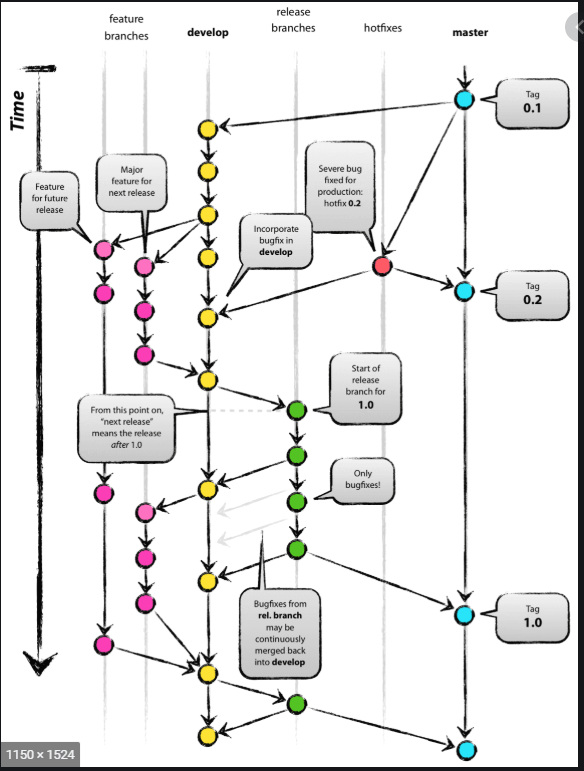
Untracked files:

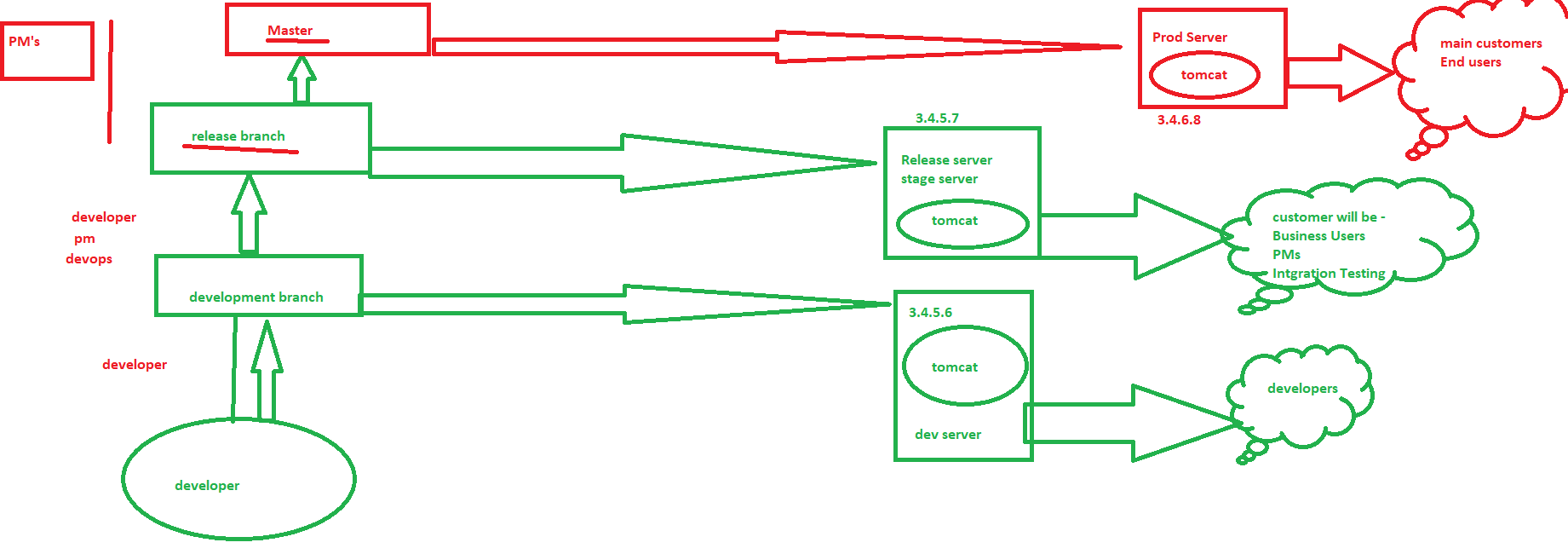
(use "git add <file>..." to include in what will be committed)

ab

nothing added to commit but untracked files present (use "git add" to track)

# **Branch**





[root@ip-172-31-80-17 practise]# git branch

\* master

Note – this will be created when first time you have committed the code (default)

## Create branch

[root@ip-172-31-80-17 practise]# git branch january

## List branches

[root@ip-172-31-80-17 practise]# git branch

january

\* master

## **Checkout the branch**

[root@ip-172-31-80-17 practise]# git branch

january

\* master

[root@ip-172-31-80-17 practise]# git checkout january

Switched to branch 'january'

**git checkout -b issue1 (Create a new branch and jump there)**

[root@ip-172-31-80-17 practise]# git branch

\* january

Master

[root@ip-172-31-80-17 practise]# git ls-files

jan19.txt

new

test/aaaah

### Note – So for first time when we create and switch the branches – it will inherit the files from where we have switched **(the last checkout branch)**

* It will not inherit local unix file system but the last branch git repo (we have checked and confirmed it)
* If we want the new files of the local FS in the new branch, you should **commit first the local FS files to current branch and then switch.**

[root@ip-172-31-80-17 practise]# git branch

\* feb

january

master

[root@ip-172-31-80-17 practise]# git ls-files

jan19.txt

new

test/aaaah

[root@ip-172-31-80-17 practise]# pwd

/home/ec2-user/practise

[root@ip-172-31-80-17 practise]# ls

jan19.txt new test

[root@ip-172-31-80-17 practise]# rm -f new jan19.txt

[root@ip-172-31-80-17 practise]# ls

test

[root@ip-172-31-80-17 practise]# git branch mar

[root@ip-172-31-80-17 practise]# git ls-files mar

[root@ip-172-31-80-17 practise]# git checkout mar

D jan19.txt >> D – deleted

D new

Switched to branch 'mar'

[root@ip-172-31-80-17 practise]# git ls-files

jan19.txt

new

test/aaaah

[root@ip-172-31-80-17 practise]# git branch

feb

january

\* mar

master

[root@ip-172-31-80-17 practise]# git checkout new (To get the file from git repo to local)

**[root@ip-172-31-80-17 practise]# ls -ltr**

**total 8**

**drwxr-xr-x 2 root root 4096 Jan 22 05:56 test**

**-rw-r--r-- 1 root root 6 Jan 22 06:30 new**

#### Good checkout testing –

This testing will show that all branches will have their own repo files and contents and we will get different list of files for each branches.

[root@ip-172-31-47-114 test]# git branch

january

mar

\* master

new

[root@ip-172-31-47-114 test]# git ls-files

testfile

[root@ip-172-31-47-114 test]# ls

testfile

[root@ip-172-31-47-114 test]# touch abn cbd

[root@ip-172-31-47-114 test]# git add .

[root@ip-172-31-47-114 test]# git commit -m "3rd commit"

[master 0847a92] 3rd commit

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 abn

create mode 100644 cbd

[root@ip-172-31-47-114 test]# git ls-files

abn

cbd

testfile

[root@ip-172-31-47-114 test]# git checkout new > Switching a Branch

Switched to branch 'new'

[root@ip-172-31-47-114 test]# git branch

january

mar

master

\* new

[root@ip-172-31-47-114 test]# git ls-files

testfile

[root@ip-172-31-47-114 test]# git checkout master

Switched to branch 'master'

[root@ip-172-31-47-114 test]# git ls-files

abn

cbd

testfile

there is a file Rahul.txt and we will change it using vi tool.

[root@ip-172-31-47-114 test]# git log

commit 479b1ef6886c64a5c6840a44a81dfcf82fb86c09 (HEAD -> master)

Author: Rahul Chaudhuri <mailrahulsre@gmail.com>

Date: Tue Dec 31 08:23:07 2019 +0000

changed rahul

commit f4de9e124bde38bda71a64520b757830efee1a5a

Author: Rahul Chaudhuri <mailrahulsre@gmail.com>

Date: Tue Dec 31 08:06:41 2019 +0000

rahul

commit 0847a92aa7f1ac4272d16ba15df27218ecc48eae

Author: Rahul Chaudhuri <mailrahulsre@gmail.com>

Date: Tue Dec 31 06:25:37 2019 +0000

3rd commit

commit 9d0304e44a2f79f9ce4ef93421bef3d62028651e (mar)

Author: Rahul Chaudhuri <mailrahulsre@gmail.com>

Date: Tue Dec 31 06:12:09 2019 +0000

first commit

[root@ip-172-31-47-114 test]# git checkout f4de9e124bde38bda71a64520b757830efee1a5a

Note: checking out 'f4de9e124bde38bda71a64520b757830efee1a5a'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b <new-branch-name>

HEAD is now at f4de9e1... Rahul

[root@ip-172-31-47-114 test]# cat Rahul.txt

test

[root@ip-172-31-47-114 test]#

[root@ip-172-31-47-114 test]# git checkout 479b1ef6886c64a5c6840a44a81dfcf82fb86c09

Previous HEAD position was f4de9e1... rahul

HEAD is now at 479b1ef... changed rahul

[root@ip-172-31-47-114 test]# cat Rahul.txt

Rahul

### Deleting the branch

[root@ip-172-31-80-17 practise]# git branch -d january

Deleted branch january (was b34e42f).

### How to restore a file

[root@ip-172-31-47-114 test]# cat abn

This is wrong modified

[root@ip-172-31-80-17 practise]# git branch <branch\_nam>

This is to move to the right branch name

[root@ip-172-31-47-114 test]# git checkout abn

[root@ip-172-31-47-114 test]#

[root@ip-172-31-47-114 test]# cat abn

Original content

To restore the file from git branch

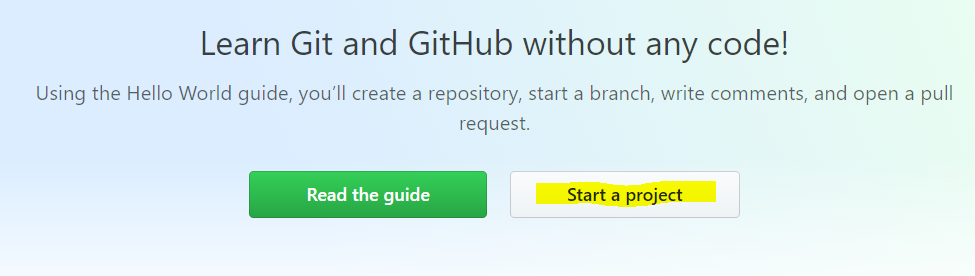
# git reset

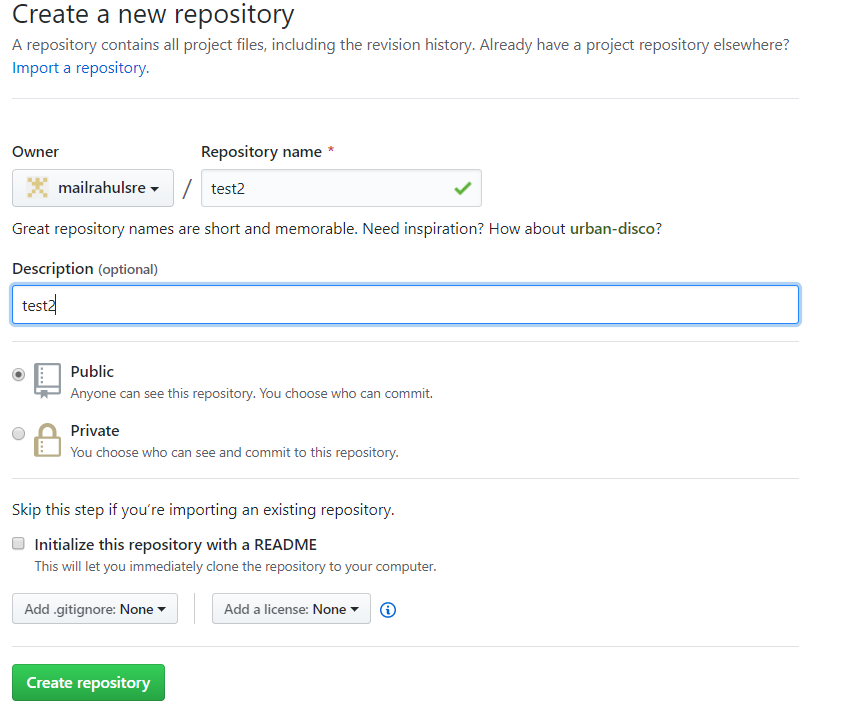
**Usage: git reset –hard [SOME-COMMIT]**  
We use this command to **return** the entire working tree to the last committed state.

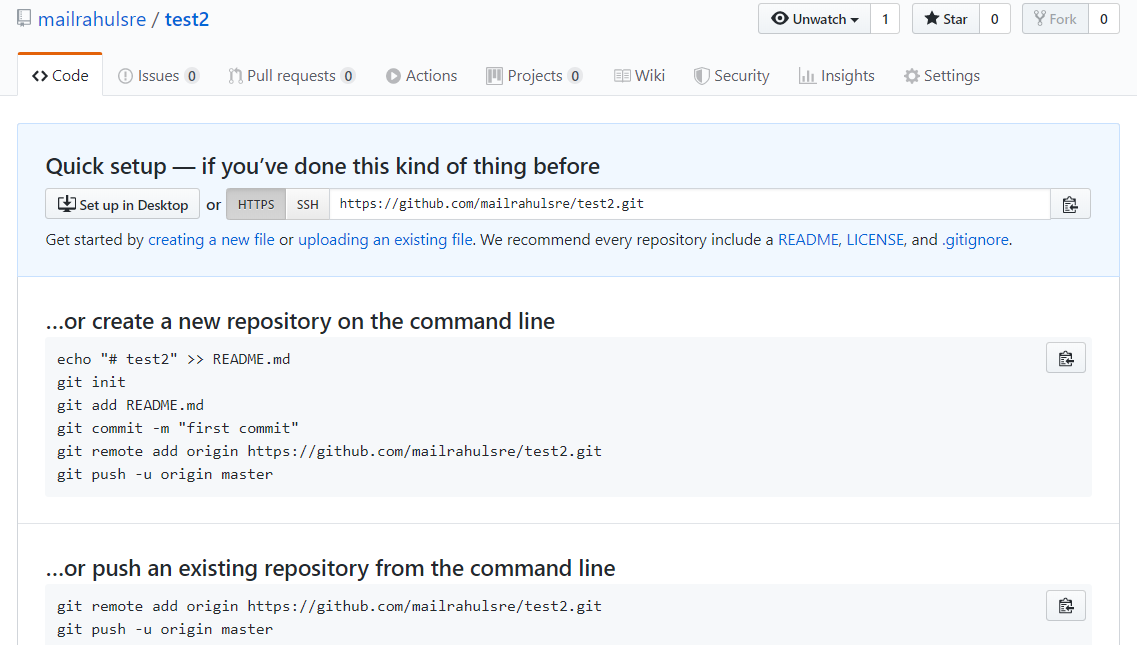
# **Git Remote command**

### Create a GitHub account

* 1. Go to <https://github.com/> and create an account
  2. Start a new Project



* 



### **Environment Setup:**

* Create a GitHub account -🡪 <https://Github.com> 🡪 Create a account and repo
* Configuring Git
* Creating a local repository

# **Git PULL & PUSH**

## **Pull**

Note – please check there should not be anything for commit or new change and tree should be clear

git pull origin master

## **Push**

…or create a new repository on the command line

echo "# test1" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/mailrahulsre/test1.git

git push -u origin master

…or push an existing repository from the command line

git remote add origin <https://github.com/mailrahulsre/test1.git>

git push -u origin master