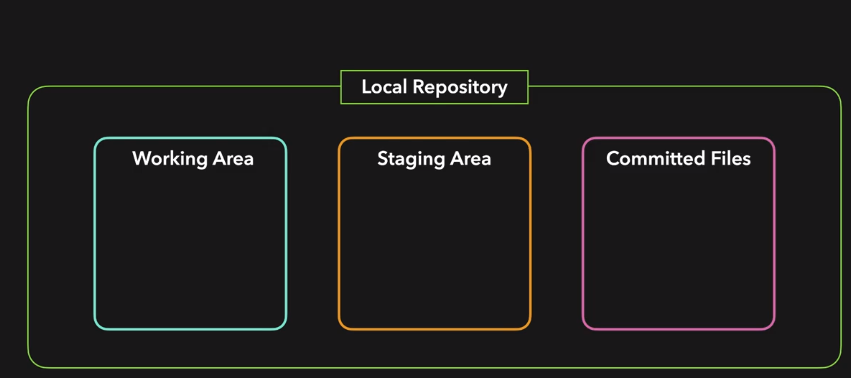
# Git Introduction

1. Local & Remote Repository:
   1. Local repository: resides in local machin
   2. Remote repository: resides in central server
   3. You can **PUSH D**ata from Local to Remote repository or PULL data from remote to local repository.
2. Local repository:
   1. Local repository has three stages:
      1. Working area: contains all files with active changes.
      2. Staging area: contains files that are going to be committed.
      3. Committed Files

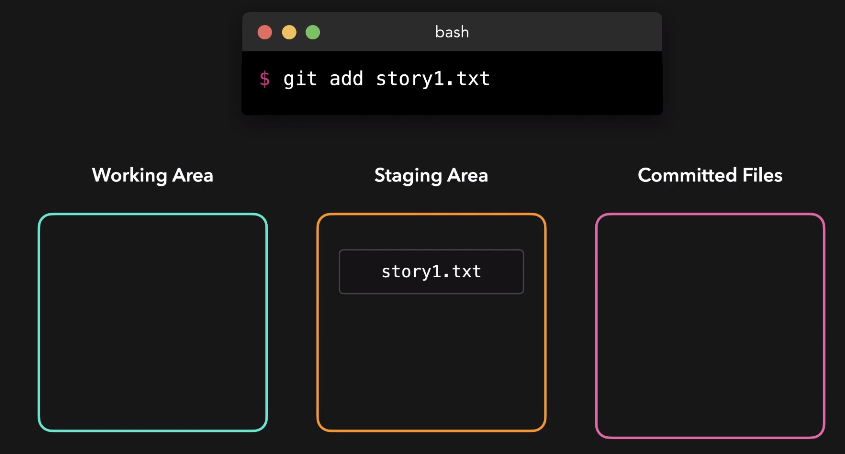
## Lifecycle (status) of a file in git:

Initially a new file (*story1*) is in working area and has **untracked** status/state. After “git add *story1*” command it is cached in staging area and has **staged** status. After “git commit” command, the file is committed and it has **committed** status. When the file is changed again in working area, it has **modified** status, since it has been modified after the last commit. The file is then moved between modified, staged and committed status based on operation.



## git basic operations (steps):

* 1. # git init – initializes a git local repository in the current folder
  2. # git add story1.txt– adds the file story1.txt from working area to staging area



* 1. # git status – check the status of files in current repository
  2. Authenticate yourself as a owner of all future commit: Set git username as sarah and user email as sarah@example.com using the below commands

# git config user.email “sarah@example.com”

# git config user.name “sarah”

* 1. # git commit –m “message” – commits all the files of the staging area in a single operation.

1. # git restore story1.txt – if a file in working area is accidentally changed, you can restore a file to last staged version.
2. # git restore --staged story1 – remove a file (story1) from the staging area to working area. The file status changes from staged to modified state.
3. # git add **.** – stages all files (that have been changed) from working area to staging area in a single operation.
4. # git rm –cached notes.txt – removes the file from git tracking (staging area). Makes the file (notes.txt) untracked state.
5. # git rm –f notes.txt – permanently delete the file (notes.txt) from directory.
6. To ignore a file always from tracking by git, add the file name in .gitignore file.
7. Show all commits

# git log

# git log --oneline

1. **You can list the changed files as well using the --name-only option with the git log command**

**# git log –name-only**

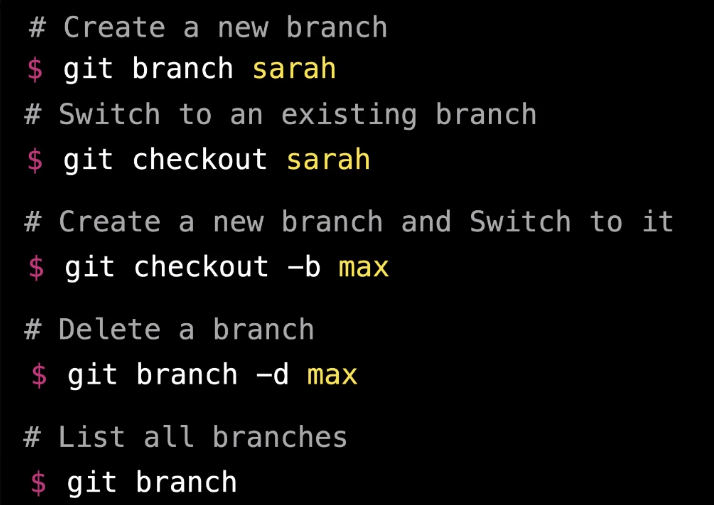
1. You may list the last 3 commits alone using the --max-count option like this

# git log -n 3

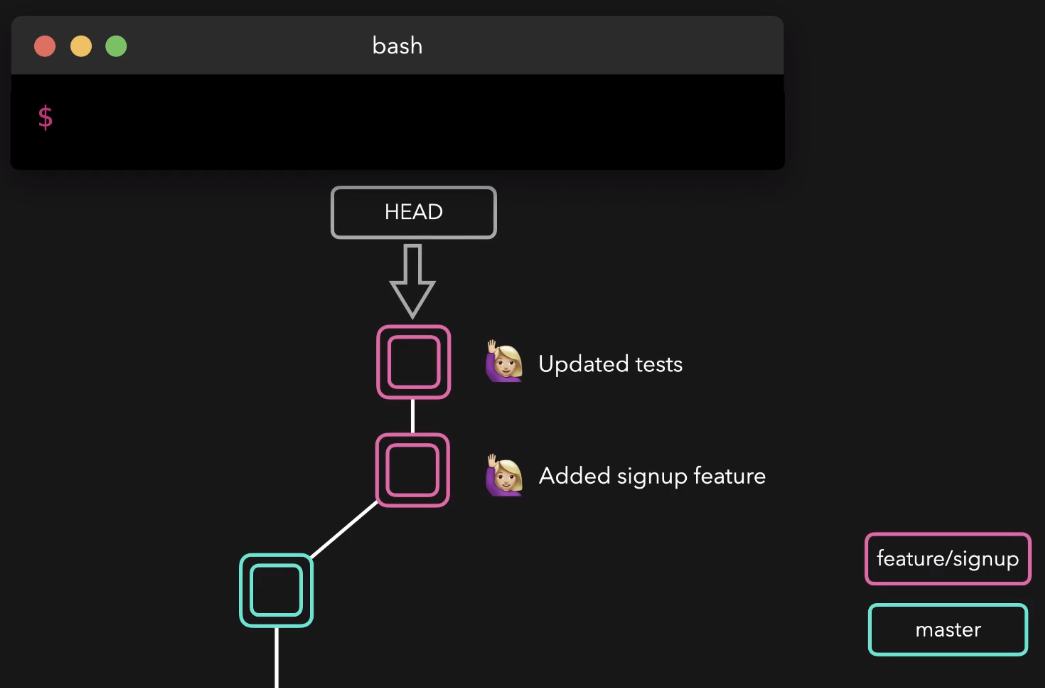
# git log --max-count 3

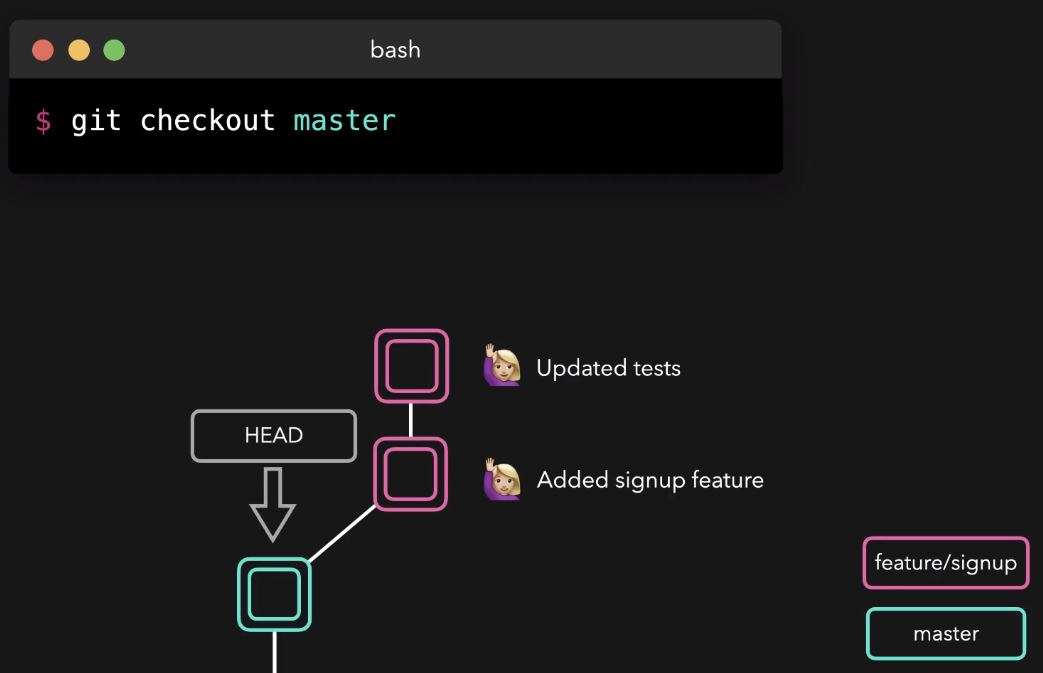
# Git Branches

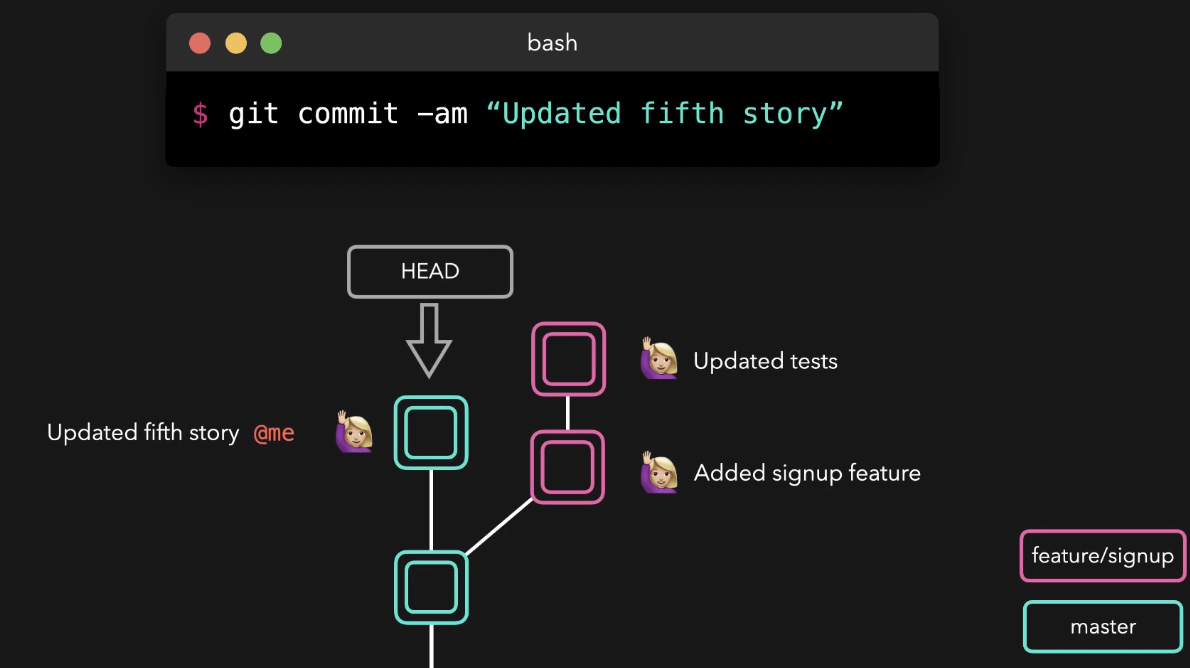
1. Branch is nothing but a pointer to a certain commit.
2. Checkout means switch to a branch

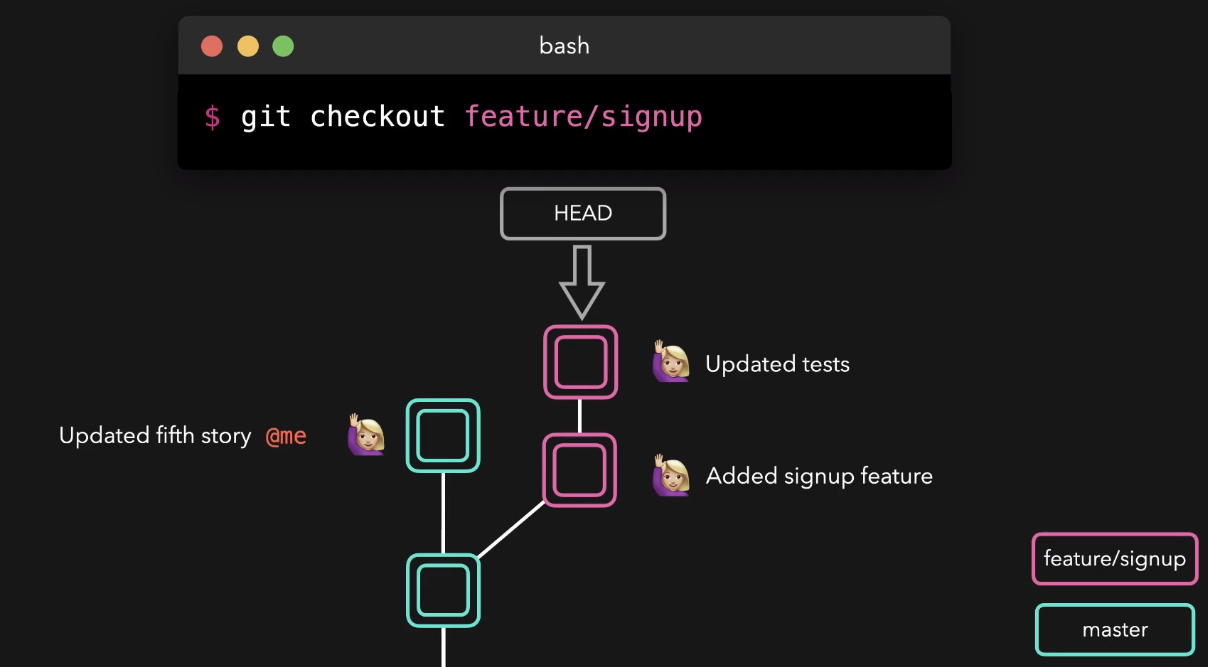


1. A HEAD is where you are right now in the git repository. A HEAD points to the last commit in the branch you are currently on. When you switch branches the head moves with you.









1. what branch was the feature/signout branch created from?

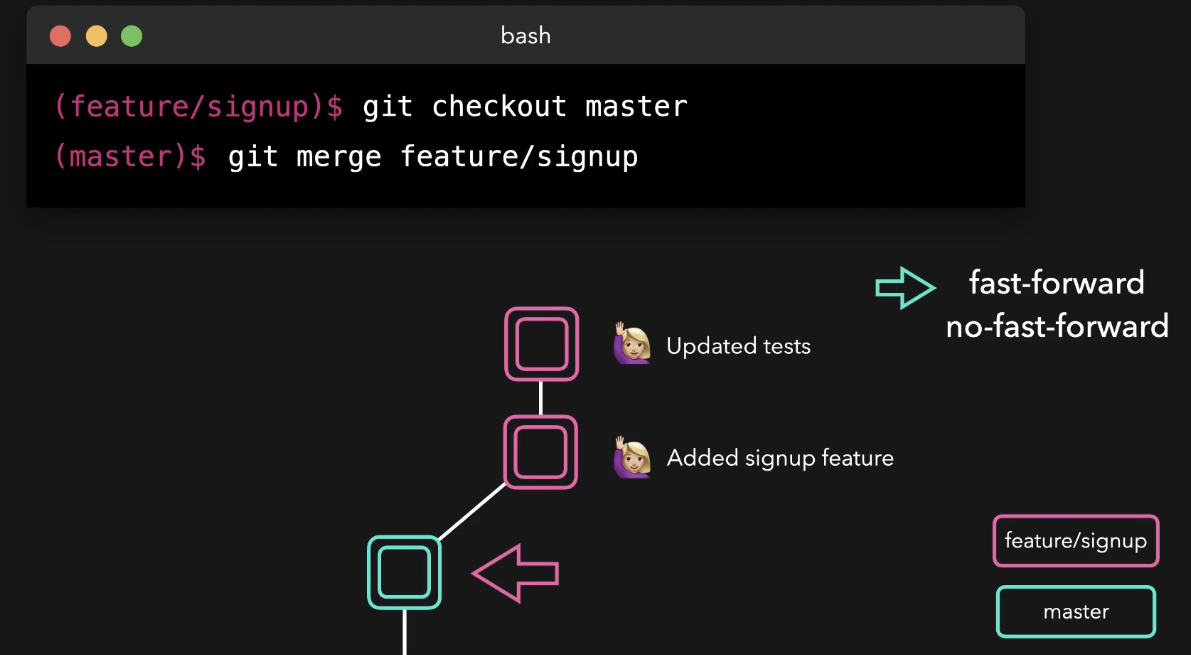
Ans: use the command git log --graph --decorate to see previous commit history along with the branch they were committed on.

1. what's the best graphical representation of the branches in this repository?

Ans: Checkout each branch and then use the command git log --graph --decorate to see previous branch.

1. To view your remote branches, simply pass the -r flag to the git branch command.

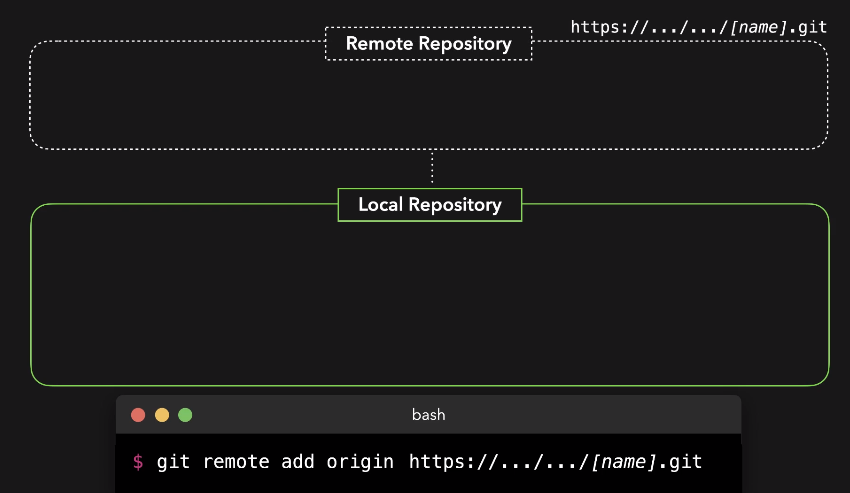
## Merging branch

1. two types of merging:
   1. Fast-forward
   2. No-fast-forward
2. 

# Initializing Remote Repositories

## Initialize remote repository

1. When a remote repository is created, a connection string is generated that you can use to access the remote repository.



**here “origin” is the alias of remote repository connection URL/string**)

1. Create a new connection to a remote repository.

# git remote add <name> <connection url>

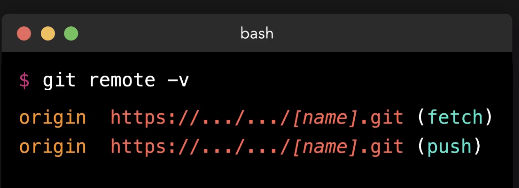
1. Remove the connection to the remote repository called ＜name＞

# git remote rm <name>

1. Rename a remote connection from ＜old-name＞ to ＜new-name

# git remote rename <old-name> <new-name>

1. List all remote repositories



1. give detailed output on the configuration of a remote.

# git remote show name

## Pushing to Remote Repository:

1. # git push <remote repo alias> <remote branch>

e.g. # git push origin master

1. Forcefull push to remote branch even there is a merge conflict

# git push –f origin master

## Clone a remote repository to local machine

1. # git clone [ssh/http link of remote repository]
   1. for https, # git clone https://HOSTNAME/USERNAME/REPONAME.git

e.g. git clone https://github.com/kamruddinch/nd064\_course\_1.git

* 1. for ssh, # git clone [git@HOSTNAME:USERNAME/REPONAME.git](mailto:git@HOSTNAME:USERNAME/REPONAME.git)

e.g. # git clone git@github.com:kamruddinch/nd064\_course\_1.git

N.B 1: git clone, by default creates a folder locally with the same name as the remote repository. It copies the entire git history and files of the project to the folder.

N.B 2: When you clone a repository with git clone, it automatically creates a remote connection named origin pointing back to the cloned repository. This is why most Git-based projects call their central remote repository as **origin**.

## Pull Requests or PR (merging a remote branch to remote master e.g. merging origin/sarah branch to origin/master) steps:

1. Clone a project from remote repository to local repository
2. Add required new files to cloned project.
3. commit all the files to new branch. (Create a new local branch and switch to the branch and commit).
4. Push the files to new remote branch.
5. Create a Pull Request(PR) to merge the remote branch into the remote master branch
6. Another user (user 2) will Review the Pull Request(PR).
7. Another user (user 3) will Merge the Pull Request(PR) to make the story available in the master branch.

## Fetching and Pulling

1. Fetching operation fetches all the history and changes from remote repository to local repository, but does not immediately merge them.

# git fetch *remote\_repository\_alias branch\_name*

*e.g.* git fetch origin master

1. After fetching, we need to merge them to get them in our currenty local repository

# git merge *remote\_repository\_name/branch\_name*

# git merge origin/master

1. Git pull is actually the combination of two commands git fetch and git merge. It fetches and merges immediately

# git pull origin master

## Fork

1. Forking means creating our own copy of original project

## Typical steps in a Remote ropositoy:

1. Fork a remote repository
2. Clone the forked repository to local machine.
3. Add files to to cloned repo.
4. Push the repo to remote repo.
5. Create a Pull Request (PR) to merge it to original repo.
6. Another user will approve the PR.
7. Owner will merge the PR.

# Rebasing

## Rebasing

1. Rebasing means putting one branch on top of another one. It is a way to get all the changes that happened to another branch. When rebasing git history is changed.
2. To Rebase sarah branch on top of master branch first checkout to sarah branch. Then type

# git rebase *new\_base-name*

## Interactive Rebasing

1. We can change the history of a git branch with an interactive rebase.
2. Interactively rebase last 4 commits

# git rebase –I HEAD~4

## Cherry Picking

1. If we want to copy a specific commit of a specific branch to another branch we can use cherry picking.
2. Cherry pick uses hash value of the commit that we want to pick
3. First checkout to the branch where the commit will be absorbed. Then type the following command:

# git cherry-pick commit hash of the commit

# Resetting and Reverting

## Resetting and Reverting

## Revert

1. Revert operation creates a new commit that literally reverts all the changes the we made on the specified commit. The reverted commit contains all the opposite changes.

# git revert *commit\_hash*

1. Revert operation undo changes and keep those changes in the history.

## Reset

1. Reset operation undo the last n number of commits.
2. Tow types of reset
   1. If we still want to keep all the changes we made (still have access to the changes done by the commit)

# git reset –soft HEAD~1

* 1. If we want to lose all the changes we made

# git reset –hard HEAD~1

## Stashing

1. The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy.
2. stash is local to your Git repository; stashes are not transferred to the server when you push.
3. Stash is like a stack. (last-in first-out). We can keep on pushing to the stash and the cnages will simply pile up.
4. Popping your stash removes the changes from your stash and reapplies them to your working copy.
5. Do a stash

# git stash

1. See all the stashes that are currently on the stash

# git stash list

1. See content of a certain stash

# git stash show stash@{2}

1. Popping all the stash

# git stash pop

1. Popping a specific stash

# git stash pop stash@{1}

1. .

# Most useful commands

## For local repository operation

1. # git version
2. # git init
3. # git add *file1.txt*
4. #git commit –m “*message*”
5. # git config user.email “sarah@example.com”
6. # git config user.name “sarah”
7. # git status
8. # git log
9. **# git log –name-only (list the changed files)**
10. # git branch
11. # git branch –r (check remote branch)
12. # git checkout *branch\_name*

## For remote repository operation

1. # git clone https://HOSTNAME/USERNAME/REPONAME.git
2. # git remote add repo\_name <remote\_repository\_url>
3. # git remote –v
4. # git remote show *remote\_repo\_name/alias*
5. # git push <remote\_repo\_alias> <branch-name>
6. # git pull *repository\_name branch\_name*
7. .