**Main Branch: Trunk :** It should have always tested code , ready to deploy in production. We should not directly work on Main branch.

**Branch:** create another parallel branch from main branch to work.

* Merge the branch into the master once the development/testing in Done. Then from the master a **Tag** will be created for the deployment.

**Installation and Set up:**

Step 1. Install Git  
Step 2. **Config**: Create a project folder (new folder).  
Step 3. Right Click on the folder and Select **Git Bash Here**Step 4. Execute :  
***git config –global user.name NKUM17*** (this is used to identify the user)  
***git config –global user.email*** [***nitin.kumar@nike.com***](mailto:nitin.kumar@nike.com)

To see the config values:  
***git config –global –list***

By using GitHub’s [@mention system](https://help.github.com/articles/about-writing-and-formatting-on-github/#text-formatting-toolbar) in your pull request message, you can ask for feedback from specific people or teams

**There are 3 levels, the config can be done:**1. –system : for all users on the system  
2. –global : for all projects for the user  
3. –local : for the specific and default repository

File types:  
 1. Tracked  
 2. Untracked (create .gitignore file)

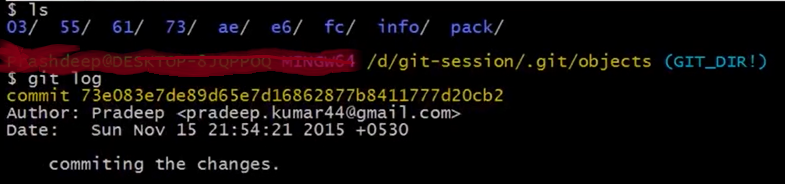
Git life cycle:

Untracked - > tracked/unmodified -> modified -> Staged

|  |
| --- |
|  |

**Commands:**

1. git --version: tells the git version
2. git rm <filename> removes the file from the repository
3. rm <filename> removes the file if the file is not yet tracked or added to the git repository
4. git rm --cached <file>..." to unstage
5. ls -all
6. clear
7. git init (creates the repository, creates a .git folder)
8. cd .git (to see inside the .git foldercd, use cd. command)
9. cd(space).. to exit from the current dir (cd ..)
10. touch filename.txt creates a new file (untracked file)
11. git status
12. git add filename.txt
13. vi filename to edit the file
14. :wq to save and come out of the file
15. git add filename.txt adds the file to the staged
16. git commit -m “some message” this will commit that file into the **local repository** along with the comment
17. git checkout – fileone.txt if want to roll back the changes made in the file
18. git commit -am “some message” this will skip the staging part and directly saves the file into Git repository.
19. git log : fire this command inside the .git/objects directory



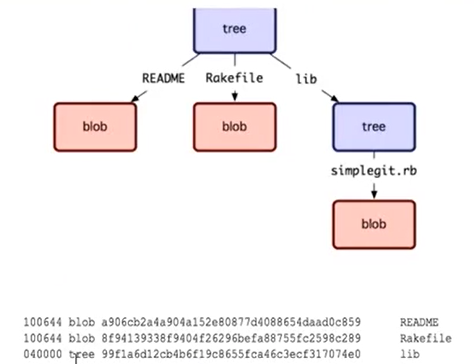
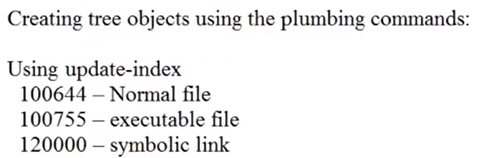
Letters in yellow Commit ---------- are Hash code. First 2 letters denote the folder where the object in. Each commit creates a repository(folder)

Git log –oneline : give hashcode and commit message for the commit.

1. git cat-file -p <hashcode> to see the content of the file, it’s a key value pair formed by Git for each commit.
2. Similar to unix file structure. And only 2 data types are in Git:

Blob : leaf level file

Tree: folder/directory

1. Git diff :- gives the difference in the file
2. Git diff --staged
3. git add, git add -a, git add –u, git add -all
4. Local Repository and Remote repository (there is no 3rd or central repository).

Git remote

1. Connect local git repository to remote github repository:

git remote add <some\_short\_name> <Repository URL from the Github.com “HTTPS clone URL”>

1. git remote :this will give the remote github repository short name as used when created the link to remote repository
2. Git remote show <repository name>

Will display the details of the remote Git, and we can use the short\_name instead everytime the whole URL.

1. To push the changes from the local to remote repository

git push origin<repository name> master<optional branch name>

1. To pull the files from the remote repository to local repository

git pull <repo name> master<branch name “optional”>

1. There is another method to pull the code from repository to local base  
   git fetch
2. To Merge the change   
   git merge
3. What is the difference between Git Pull and Git fetch.  
   Git Pull = Git fetch + git merge  
   git fetch: it extracts the files from repository and keeps it into the temp folder and later manually merge the temp folder with local repository.  
   git pull: it automatically runs the pull and merge, and forces developer to merge the code immediately when the pull command is executed.
4. To create a local repository of the already existing remote repository  
   create a local folder, open git-bash and run the following command

git clone < Repository URL from the Github.com “HTTPS clone URL”>

1. Git remote : after cloning to the existing repository, this command will return the repository name
2. Rename the file int the repository  
   git mv <old name> <new file name>
3. To ignore the files from the git tracking, create .gitignore file and all those file names in this file.  
   touch .gitignore

git add .gitignore  
git commit -m ”committing .gitignore file”

note: contents inside .gitignore file will be ignored but the .gitignore file itself will be tracked.

1. Head : points to the latest commit. There will always be only 1 head.
2. Origin: is a logical name for the default repository and alias to the entire repository URL
3. git help <command\_name> : it provides the details of the command.
4. A commit can not be deleted or altered, and history cannot be modified.
5. Only the current commit can be amended.
6. git commit -amend : is the command to amend the current commit.
7. git branch : to list all the branches available in the local repository. \* on the branch name shows the current branch it is pointing to.
8. git branch --all: to list all the branches available in the remote repository. \* on the branch name shows the current branch it is pointing to.
9. git branch <branch\_name> :create a new branch from the master branch.
10. git checkout <branch\_name> :to change the branch point to. Now git branch will show all the branches of the repository but will point now to <branch\_name> branch.
11. git checkout -b <branch\_name> : creates a new branch ands checks out too.
12. To merge the changes in branch to master branch  
    switch to master branch and execute  
    git merge <branch\_name>

And then “git push origin master” to push the changes to repository.

1. **Light weight branches :** create to work on features/bug fixes and then delete after merging the changes to master branch.
2. Delete branch: safe delete  
   git branch -d <branch\_name> : Notifies if there is any uncommitted item {safe delete}

git branch -D <branch\_name> : this will **NOT** notify if there is any unmerged/uncommitted item

1. Stash : temp space to store the current changes

Git stash save “message”

1. To see all the items in the stash  
   git stash list : displays the items in the stash in form of stash@{id (0……..n)}
2. To copy the stash item to a branch  
   Switch to the branch: git checkout <branch\_name>  
   copy the Stash item from the stash to the branch  
   git stash apply <stash@{id}>
3. Clear the stash, deletes all elements from the stash.  
   git stash clear
4. To Cut the item from the stash and move to the current branch, it pops the latest stashed item (last in first out)  
   git stash pop
5. Regular merge (3-Way Commit): multiple branches working / committing on the branch.
6. Fast Forward Merge: When there is no dev/change done on master branch, so the merge happens called fast forward merge.
7. Rebase merge: Linear git , dosent look like branches were created
8. Git reset : 3 types of git resets are available : soft, mixed {default}, hard  
   git reset –soft <Commit Hashcode>  
   git reset –mixed <Commit Hashcode>  
   git reset --hard <Commit Hashcode>
9. Reset works with Staging Area, Checkout works with Working Directory
10. What is head pointing to, which commit  
    git rev-parse HEAD
11. Tag vs Branch  
    git tag -a <version name> -m “message”
12. Git tag : will display the tag name
13. Git push origin <tag name>
14. Merging

git update-git-for-windows (i.e. the auto updater of Git for Windows) now [reports correctly when it failed to access the GitHub API](https://github.com/git-for-windows/build-extra/pull/239).