SZA Technologies

Introduction to Version Control and GIT

Introduction to GIT

Presented by SZA Technologies

WORKING IN TECHNICAL AND E-COMMERCE FIELD

Contents

GIT

1.1 Version Control

- Track the file system changing
- Also called revision control system
- Tightly coupled with software development
- It has snapshot of each step
- Use as database
- It has complete timeline
- **Independent**
  - It has independence of all technical of soft projects
  - Not bother which software or thing you are using
- You don't need to take backups
- When you submit your work to VCS (version control system) it will take it for whole time. This is storing version of VCS.

1.1.1 Why

Restoring data or projects.

It will give information character by character or line by line.

Backup

It is not a backup. But can use as backup in GIT.

It will create a central service.

2.1.1 Types

VSS, SVN, GIT (Linux Contributor)

VSS

Micro visual source safe (for short projects).

SVN

Apache Subversion (for big project)

SVN is centralized version control system (CVCS)

GIT

It is distributed version control system (DVCS).

Every user has complete copy of project files.

Every update or execute on central main server.

3.1.1 Drawbacks of (CVCS)

- Submissions of updates is necessary (Internet required).
- If server crashed (data lost)

Push

Mean to send or submit on central server.

Pull

You want to check the changes any user perform.

4.1.1 Advantages of (DVCS)

- Fast process because no need of internet. We need internet for push and pull only.
- Each user is locally dealing that's why no effect on main repository.
- Each user can share the copy or course.
- If system crashed then any user's copy can use.

5.1.1 Why work on VCS:

- To use changes snaps
- Team work
- Easily know which user changed the code
- Collaborate with team members
- Recalling of each user
- Each user should update each step
- If two users make changes on the same line then at the time of submission conflict come.

2.1 Basic Operations of GIT

Initialize

ADD

Commit

Pull

Push

3.1 Advance Operations of GIT

Branching

Merging

Rebasing

Repository

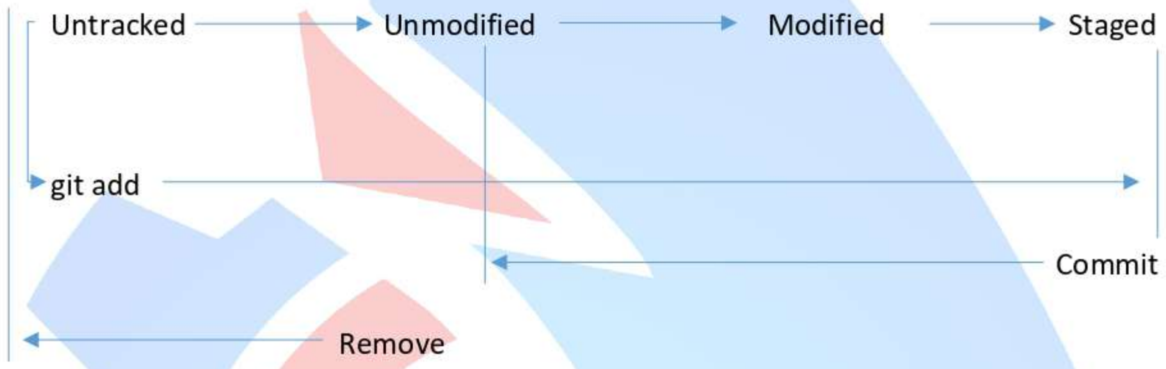
It's a hidden folder (.git)

Your each data is in repository.

Working Directory:

In which directory you are currently working.

4.1 File Status



The above figure is showing the status of file process.

First the will untracked, second stage of file will be unmodified, third stage will be modified.

When you put command "`git add`" the file will become staged.

When you commit the file it will again unmodified. Then you have to modify it and then staged it.

Staging Area

Commit will not perform until you are not in staging area.

5.1 Basic Work Flow of GIT

1. Create directory through terminal. e.g C:\Repo\Myproject
2. Initialize repository in this folder. "`git init`" this will create hidden folder (.git).
3. Create files like "first.txt & second.txt".
4. Check status of your project. "`git status`" it will show two untracked files.
5. Add these files to staging area.
   (i) "`git add first.txt second.txt`".

(ii) OR "`git add .`" it will add all files of this folder.

(iii) OR "`git add *.txt`"

6. Commit file

   ```
   git commit -m "new feature added"
   ```

   - The text inside quotation mark is a comment and writing comment is very important. Because you will recognize by this text or comment which was made.

7. To check the history or to check what and when changes happen. Write this command "`git log`".

Head

Current branch is called head or offered as head.

- Next commit will become head.

Hash

Git will create a unique hash for each commit you will recognize by this hash.

Reset

"`git reset`" it will un-stage the staged files.

"`git reset --hard`" it will un-stage also reset the changes. It will also reset the changes of un-staged.

In **UI** use "discard" button.

Ignore files

- Make ".gitignore" file
- Inside this file write which you want to ignore (like files and folders).
- These ignore files will not be the part of commit history.

In **UI**

- Right click on the file and click on 'ignore'. It will automatically create a '.gitignore' file.

6.1 Branch

Divide the context in branches.

Context in GIT is a branch.

Head refers the current branch.

Some commands for branching

"`git branch`" it will show the list of branches.

"`git branch -v`" it will show the commit of branches.

"`git branch new-dev`" switch to 'new-dev' branch.

"`git log new-dev .. branch2`" it will show the commit difference in 'new-dev' and branch2.

"git merge new-dev" It will merge new-dev branch into current branch.

In **UI**

- Click on "branch + add branch".
- For merge click on merge.

7.1 Conflict

Changes on same branch and on same line is called conflict.

In **UI**

- Head represents the current branch.
- Double click on file and make changes in tree section.

- Then stage and commit.
- Now your current branch is updated.
- Move to other branch and merge to update it.

8.1 Stash

Save temporarily your updates is called stash.

You can stash for long time.

Some commands for stash

"`git stash`" it will save the local changes.

"`git stash save <name>`" save local changes and give name.

"`git stash list`" it will show the list of stashes.

"`git stash pop`" it will apply the last change and remove from history.

"`git stash apply <name>`" it will apply named stash and will not remove from history.

"`git stash clear`" it will clear the list.

In **UI**

- Click on "Save Stash" or press "ctrl + s".
- Then click on "Apply Stash" to apply.
- If you choose apply and drop it will remove from list.

9.1 Push and Pull from github

Pulling from github.com

"`git clone <URL>`" it will pull data from your github account. Copy URL from your account and paste it in the command.

Some commands for pushing, pulling and fetching

- "`git push origin master`" in this command origin is 'repository (standard name)' and master is 'branch'.
- "`git push`" push your changes to remote repository.
- "`git fetch`" it will get changes from remote repository but can't view by fetcher.
- "`git merge`" after fetching you have to merge to view.
- "`git pull`" it will do both works fetch and merge.
- "`git remote -v`" it will show your local repository connected to which remote repository.
- "`git remote show origin`" it will how the complete summary of your remote repository.

Connect your local repository to remote repository

- "`git remote add <nameRepository URL>`" it will create or connect your local repository to the remote repository.

In **UI**

- Click on new repository and
- Then click on 'clone'.
- Create repository on github.
- Open terminal in local repository.

"`git remote add <name of repository> URL`" “git push –u <remote repository name> master”

- This use after connection of local to remote.
- ‘-u’ create tracking connections between local and remote.

“git log <remoteRepoName/branch>” for log of remote repository.

In **UI**

- Go to ‘Remote’ (available at top bar) then click ‘add’.
- Add URL and name of repository.

You can create branch on ‘github’.

Production Branch: for customers.

Development Branch: for host and approval.

Hot fixes branch: if bug in production.

10.1 Pull request by github

6.1.1 Fork:

By github:

- You can clone public repository
- You can make changes and then create pull request for original user.
- You have to connect both remote repository either with your and user’s from which you pull.

11.1 Delete Branch

"`git branch -d <branchName>`" this deletion is for local branch.

"`git branch -dr <origin/branchName>`"

- To delete the remote branch
- It will not delete from 'github.com'

"`git push origin --delete <branchName>`" to delete from github.com.

12.1 Undo changes (local)

"`git checkout Head <filename>`" it will undo changes of this file.

"`git reset --hard Head`" it will undo all changes of this particular time.

13.1 Commit undoing

"`git revert <commitHash>`" it will not delete commit but it will undo changes and create another commit.

In **UI**

- Click on 'revert' button.

"`git reset --hard <commitHash>`" it will delete the commit also from the history.

In **UI**

- Click on 'reset' button.

"`git reset --keep <commitHash>`" it will delete commits. But these commits will show locally.

14.1 Force Push

"`git push -f origin master`" here '-f' is for force.

In **UI**

- Go to 'Edit + Preference + Push (tick: allow modifying push commit).

Merge commits

If additional commits in both branches then git will create additional commits.

Rebase

"`git rebase <branchName>`" it will show multiple lines of branch and commit detail in single line.

15.1 Readme file in Github UI

Heading:

- Use '#'.
- You can add six headings by using six '#'.
- For bigger heading use single '#'.

Bold Text:

- Use double steric (**).
- ** This text is bold**

Italic:

- Use single steric (*).

- *This text will be in italic form*.

For Codes:

- Use back ticks "```" for showing any coding.

-
  ```
  git push
  git pull
  ```

Links:

- For showing your any link use [Link Name] (URL).
- You will see just 'Link Name' but not the URL. When you click on Link Name it will send you to the URL address.
- E.g. [GitHub Page] (https://page.github.com/)

For listing:

- Use dash or numbering for listing (- or 1.).
- - James
  -George
  -Fame
  Output: **o** James
  o George
  o Fame
- 1. James
  2. George
  3. Fame
  Output: 1 James
  2 George
  3 Fame

Nested Lists:

- Use numbering and dash for nested listing (1. Or -).
- 1. Pakistan
  -Multan
  Output: 1 Pakistan
  o Multan

Task List:

- Use dash and square brackets (- [ ]).
- -[X] Finish work

  ☑ Finish work.

  - [ ] Pending task 1

  [] Pending Task 1

  -[ ] Pending task 2

Quotation

- “> quotation is here”

Cross on line:

- ~~ This line is cross line~~

  Output: ~~This line is cross line~~