Introduction to GitHub

Kirpajeet Singh
B.E CSE Student, Chitkara University

10/05/2023

CML

What is GitHub?

- GitHub is a collaboration platform built on top of a distributed version control system called Git. One does not have to worry about losing data on his hard drive or managing a project across multiple computers - one can sync from anywhere. You can track issues, build & test the things and finally deploy.

Why Use of GitHub for Projects?

- **Version Control** (Allows experiments and mistakes without messing up in final product)
- Keep your **Code in One Place**
- Great **Collaboration** Platform
- Git, GitHub, GitHub.com, GitHub Desktop,
- Hello World - https://guides.github.com/activities/hello-world/
- GitHub Guide - https://guides.github.com/

Setting Up With GitHub

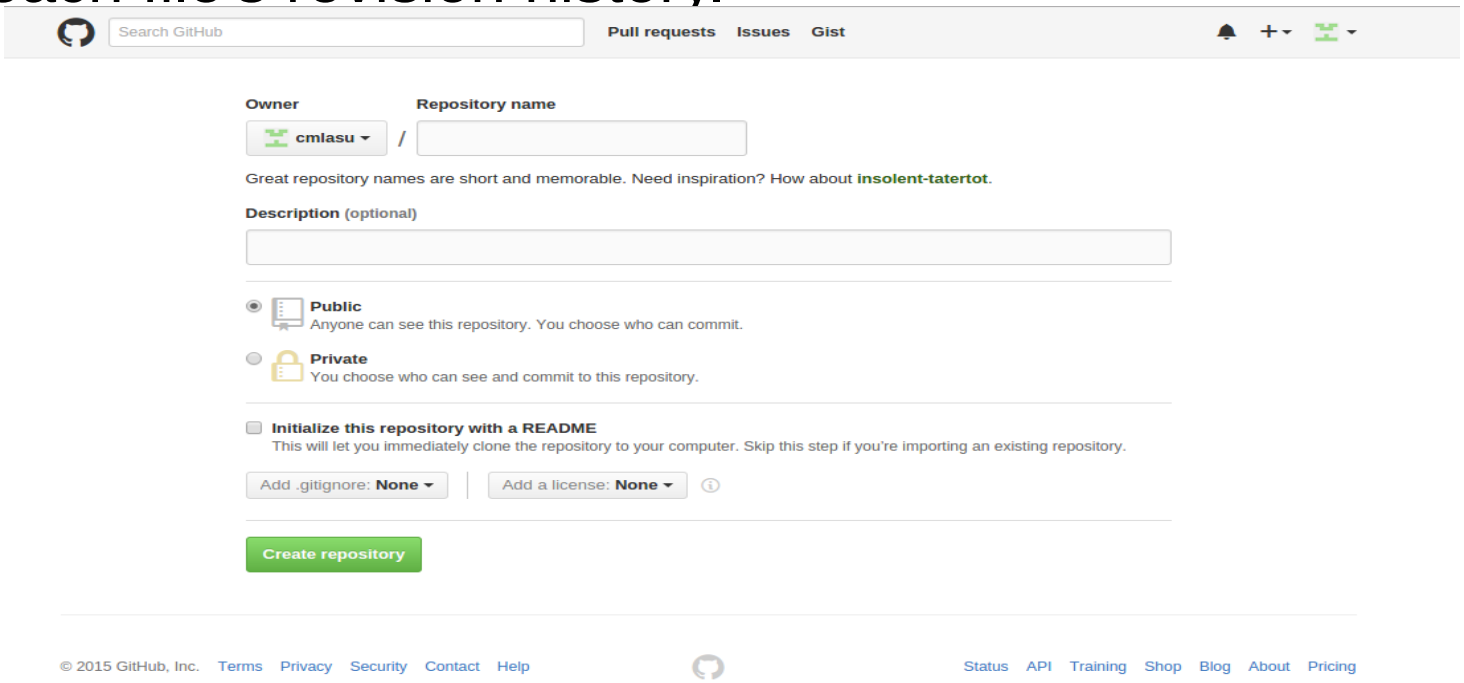
- Connecting over HTTPS/SSH
- Using SSH Key – check else create
- Add your key to SSH agent
- Add your key to GitHub Account

  https://help.github.com/articles/generating-ssh-keys/

- Next step is to create and use repository…

GitHub Repositories

- Basic unit of GitHub, most commonly a single project. Repositories can contain folders and files, including images – anything your project needs. **ReadME** – Project Description
- Contains all of the project files (including documentation), and stores each file's revision history.



Clone Repository

- Go to the directory where you want to have your repository folder (usually, your home) and then type the clone command for your repository. For example, to get cml-cgra type-

  git clone git@github.com:cmlasu/cml-cgra.git

- To update your repository with existing version, type

  git pull origin master.

  If it it other branch than your master then type –

  git pull origin your_branch_name

- **Public Repository (Any Number, Free)** - Anyone can see a public repository, but you choose who can commit to it.
- **Private Repository (Paid Subscription)** - By default, only you can see a private repository. You choose who can see and commit to this repository by adding collaborators.

Commit And Push

- git status [-s] ? hello.c
- git add hello.c A hello.c

  (You have to add before commit, You can't skip)
- git commit –m 'my changes'
- OR git commit –a 'my changes'
- After commit, checking status should show

  # On branch master
  nothing to commit (working directory clean)
- OR git commit –m 'my changes'
- git push cml-cgra master|cgra_shail
- Possible Issue: Outdated commit or simultaneous commit by different developers. So, your commit can be rejected automatically. Solution - Merge/Fetch then commit.

Fetch/Pull

- git fetch cml-cgra
- Run merge, diff etc locally
- git pull cml-cgra = git fetch + merge

Diff - Another Snapshotting Command!

- git diff (show diff of unstaged changes i.e. after past commit)
- git diff --cached (show diff of staged changes)
- git diff HEAD (show diff of all staged or unstaged changes i.e. difference between working directory and last commit, ignoring the staging)
- git diff –stat (show summary of changes instead of diff)

Reset

- git reset HEAD -- file (unstage files from index and reset pointer to head)
- git reset -> undo last commit and put files back onto stage
- --soft specifies where it stops.
- git reset --soft HEAD~ (parent of HEAD)

  Last commit will be undone and files touched will be back on the stage again
- git reset --hard (discards staged changes and changes in working directory. It un-stages files AND undo any changes in the working directory since last commit.)
- git rm file will remove the file from the staging area entirely and also off your disk (the working directory). To leave the file in the working directory, you can use git rm --cached

Stash

- git stash -> add current changes to the stack
- git stash list -> View stashes currently on the stack
- git stash apply -> grab the item from the stash list and apply to current working directory
- git stash drop -> remove an item from the stash list
- The last item added onto the stash will be referenced by stash@{0} and increment those already there by one.

Branching

- git branch -> list your available branches. Current branch will have a star next to it and in green color.
- When you run git init it will automatically create a 'master' branch for you by default.
- git branch branch_name -> Create a new branch at your last commit. So if you record some commits at this point and then switch to 'testing‘. It will revert your working directory context back to when you created the branch in the first place - you can think of it like a bookmark for where you currently are!
- git checkout branch_name to switch branch we're currently on.
- Shortcut: git checkout –b branch_name
- git branch –v (see last commit on each branch)
- git branch –d branch_name (delete a branch)
- git push remote_name : branch_name (delete remote branch)

Merging

- git merge – Merge any branch into current branch
- Also performs more complex merges from various branches like modification in code rather than just addition/deletion
- Merge Conflicts! – Occurs when same block of code is edited in different branches there is no way for computer to figure out.
- Much like subversion, Git inserts standard merge conflict markers. We have to resolve them (manually, mostly) and then to commit changes.
- git mergetool -> graphical mergetool from git
- git diff -> Git inserts standard merge conflict markers
- Lastly do git add - to tell Git the file has been resolved you have to stage it.

Log and Tag

- git log – Show commit history of branch
- To see a chronological list of the parents of any branch, you can run git log when you are in that branch.
- To see more compact version of log, add --oneline option.
- Can also see when the history was branched and merged with the very helpful --graph option. Here is the same command but with the topology graph turned on: git log --oneline --graph
- git tag -> to mark a commit or point in your repo as important.
- -a means make an annotated tag.
- git tag -a -> Git will open your editor and have you write a tag message, just like you would write a commit message.
- When you run git log --decorate, you can see our tag there.

Remote

- git remote –> list your remote alias(es).
- With –v option, you can see actual URLs also.
- git remote add - add a new remote repository of your project
- git remote rm -> removing an existing remote alias
- git remote rename [old-alias] [new-alias] -> rename remote aliases
- git remote rename [old-alias] [new-alias] rename remote aliases
- You can set a different push URL when you include the --push flag. This allows you to fetch from one repo while pushing to another and yet both use the same remote alias.
- Internally, the git remote set-url command calls git config remote, but has the added benefit of reporting back any errors.

Issues

- Great way to keep track of bugs, tasks and enhancements during development. Easily pull right people into conversation.
- Title, Description, Label, Milestone, Assignee, Comments
- Notifications: @mentions, references;
- watching, participating
- Issue Dashboard; Overview and Reports
- Pulse – Underneath every repository. Pulse is a snapshot of everything that's happened in the repository in the past week (or day, or past 3 months, etc).
- Advantages: Bug Tracker, Requests for tasks, History-Keeper of development issues

Forking Projects

- Fork the repository
- Clone your fork
- Making and pushing changes
- Making a pull request

User Accounts

- Your identity on GitHub. Can be member of any number of organizations. User account includes –
  - Unlimited public repositories and collaborators on all plans
  - Personal plans for private repositories
  - Ability to add unlimited repository collaborators
- Two permission levels – repository owner and collaborator(s)
- No need to have multiple accounts for different purposes like for business and for personal usage.
- User accounts are intended for humans, but you can give one to a robot, such as a continuous integration bot, if necessary.

Organization Accounts

- Organizations are great for businesses and large open-source projects that need multiple owners and admins. They include:
  - Business plans for private repositories
  - Team-based access permissions
  - Unlimited owners, administrators & collaborators using teams
  - Billing receipts that can be sent to a second email address
  - Owners team access to organization members' two-factor authentication (2FA) status
- Organizations manage membership with teams. 4 level of accesses for teams: Owners, admin access teams, write access teams, and read access teams. Except for Owners, team members only have access to the team they are in and the repositories assigned to that team.

Organization Teams

- Adding people into teams into organization-

https://help.github.com/articles/adding-people-to-teams-in-an-organization/

- Adding Users and teams

https://help.github.com/enterprise/11.10.340/admin/articles/adding-users-and-teams/

- Permission levels for organization Teams

https://help.github.com/articles/permission-levels-for-an-organization-repository/

Additional Useful Links

- https://help.github.com/articles/what-are-the-differences-between-svn-and-git/
- https://training.github.com/kit/courses/github-for-developers.html
- Git Reference - http://gitref.org/index.html
- Finally - https://help.github.com/

Thank You