Git and Git hub

23 December 2024 05:08 AM

Git :

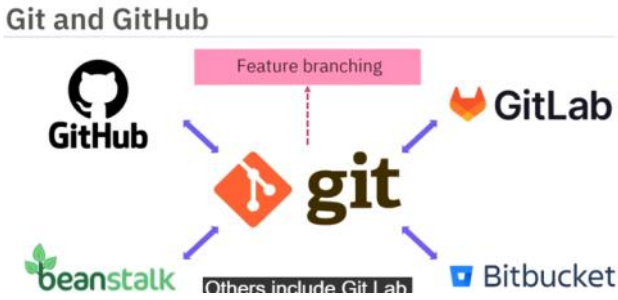
- Used to version control
- Collaborative software development
- Opensource software under GNU(General public
- se)

What can be done in git:

- Distributive version control
- Create repositories
- Share and integrate changes

Version control :

- Tracks the changes to source code
- Recovers older versions
- Facilitates collaboration



SSH protocol : Secures remote login from one computer to another
Repository : Contains project folders
Fork : Copy of repository
Pull request : To request for review and approval
Working directory : contains files and subdirectories
Commit : Snapshot of the project's current state.
Branch : separate line of development
Merging : Combines changes from one branch to another
Clone : Local copy of the remote git repository

Git Hub:

Git Repository Model

- What is special about the Git Repository model?
  - Distributed version-control system
  - Tracks source code
  - Coordinates among programmers
    - Tracks changes
    - Supports non-linear workflows
  - Created in 2005 by Linus Torvalds

What is Git?

- Git is a distributed version-control system
  - Tracks changes to content
  - Provides a central point for collaboration
- Git allows for centralized administration
  - Teams have controlled access scope
  - The main branch should always correspond to deployable code
- IBM Cloud is built around open-source tools including Git repositories

What is GitHub?

- GitHub is an online hosting service for Git repositories
  - Hosted by a subsidiary of Microsoft
  - Offers free, professional and enterprise accounts
  - As of August 2019, GitHub had over 100M repositories
- What is a Repository?
  - A data structure for storing documents including application source code
  - A repository can track and maintain version-control

What is GitLab?

GitLab is:
- A DevOps platform, delivered as a single application
- Provides access to Git Repositories
- Provides source code management

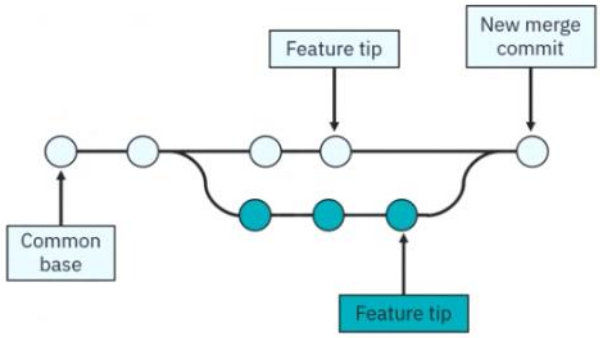
GitLab enables developers to:
- Collaborate
- Work from a local copy
- Branch and merge code
- Streamline testing and delivery with CI/CD

Git hub branches:

- Branches stores all files in github
- Main branch stores the deployable code created automatically
- New branch is a exact copy of the original branch with the changed code
- Common base - Initial source of the project
- Code is branched when new features developed

Merging branches

- Start with a **common base**
- Code is branched while new features are developed
- Both branches are undergoing changes
- When two streams of work are ready to merge, each branch's code is identified as a **tip**
- Two tips are merged into a third, combined branch



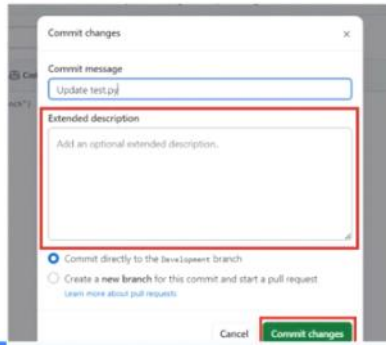
\

Make a commit

- To change the contents of a file
  - Select file
  - Click pencil icon
  - Make changes
  - Scroll down to find **Commit changes**
- Saved changes are called commits

Make a commit

- In **Commit changes** box, add a comment describing the changes
- Choose to commit directly to the current branch or to create a new branch
- Click **Commit changes**
- Best practices:
  - Don't end with a period
  - Less than 50 characters
  - Active voice



Best practices are to write comments in the comment spaces.
Must be in active voice

What is a pull request?

Makes the proposed (committed) changes available for others to review and use

Can follow any commits, even if code is unfinished

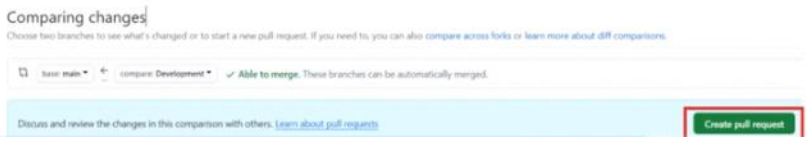
Can target specific users

GitHub automatically makes a pull request if you make a change on a branch you do not own

Log files record the approval of the merge

Open a pull request

- Click **Pull requests** and select **New pull request**
- Select the new branch from the compare box
- Confirm that changes are what you want to assess
- Add a title and description to the request
- Click **Create pull request**



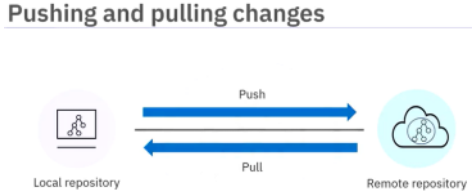
To Merge pull request:

- To merge a committed code change into the main code
  - Click **Merge pull request**
  - Click **Confirm merge**
  - Delete the obsolete branch

If you created the repository means you are the administrator and have the rights.

Git Workflows:

1. Cloning a repository



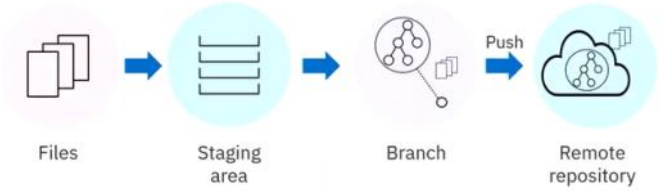
2.

3. Its good to work with a new branch
4. after the development files moved to staging area(temporary

storage)

5.

6. Pushing the branch into remote repository



7. Then the code is reviewed by a maintainer
8. Merged to the main branch

Git Commands:

1. **Mkdir -** creates a directory
2. **Cd -** changes to the current directory
3. **Git init** - set ups necessary files and data structures for project's version control.
4. **Git Add** - It adds the new files to the git directory and changes to staging area
   - → **git add <filename.txt>** (to add a specific file)
   - → **git add .** (to add all the files that are new or changed in the current directory)
   - → **git add -A** (to add all changes in the entire working tree, from the root of the repository, regardless of where you are in the directory structure)

5. **Git commit** - Saves the changes with a descriptive message
6. **Git log** - to review preview changes to a project
7. **Git branch** - Lists, created, renames and deletes branches within a git repository
   - → To delete the branch, first check out the branch using git checkout and then run the command to delete the branch

locally. - **git branch -d <branch-name>** (to delete a branch)
   - → **git branch <new-branch>** (to create a new branch)
   - → **git branch** (to list branches)
8. To create git branch called cod_list- use **Git branch cod_lst**
9. To switch to the newly created branch (switches to the existing branches)- **git checkout cod_lst**
10. To view all the changes in the git working directory - **git status**
11. Then use- **git merge cod_lst**
12. **git reset -** When used with –hard HEAD, this command discards all changes made to the working directory and staging area and resets the repository to the last commit (HEAD).
    - → **git reset**
    - → **git reset –hard HEAD**
13. **git clone <repository URL>** - It copies a repository from a remote source to your local machine.
14. git pull origin main - First, switch to the branch to merge changes by running the **git checkout** command. Then, run the **git pull** command fetch the changes from the main branch of the origin remote repository and merge them into your current branch.
15. **Git push origin <branch-name>** - uploads local repository content to a remote repository. Make sure you are on the branch that you want to push by running the git checkout command first, then push the branch to the remote repository.
16. **git version** - displays the current Git version
17. shows changes between commits, commit and working tree. It also compares the branches.
    - → **git diff** (shows the difference between the working directory and the last commit)
    - → **git diff HEAD~1 HEAD** (shows the difference between the last and second-last commits)
    - → **git diff <branch-1> <branch-2>** (compares the specified branches)
18. **git revert HEAD** - This will create a new commit that undoes the changes made by the last commit.
19. **git config –global user.email <Your GitHub Email>** - It sets a global email configuration for Git. This needs to be executed before doing a commit to authenticate the user's email ID.
20. **git config –global user.name <Your GitHub Username>** - It sets a global username configuration for Git. This needs to be executed before doing a commit to authenticate users' username.
21. **git remote** - It lists the names of all remote repositories associated with your local repository.
    - → **git remote -v** - It lists all remote repositories that your local Git repository is connected to, along with the URLs associated with those remote repositories.

→ **git remote add origin <URL>** - adds a remote repository named "origin"

→ **git remote rename origin upstream** -
The git remote rename command is followed by the name of the remote repository (origin) you want to rename and the new name (upstream) you want to give it. This will rename the "origin" remote repository to "upstream."

→ **git remote rm origin** - It removes a remote repository with the specified name(origin).

22. **git format-patch HEAD~3** (creates patches for the last three commits) - It generates patches for email submission. These patches can be used for submitting changes via email or for sharing them with others.
23. **git request-pull origin/main <myfork or branch_name>** - It generates a summary of pending changes for an email request. It helps communicate the changes made in a branch or fork to the upstream repository maintainer.
24. **git send-email *.patch** - sends a collection of patches as emails. make sure to set the email address and name using the **git config** command to client know the sender's information when sending the emails.
25. **git am <patchfile.patch>** - It applies patches to the repository. It takes a patch file as input and applies the changes specified in the patch file to the repository.
26. **git daemon –base-path=/path/to/repositories** - It exposes repositories via the Git:// protocol. The Git protocol is a lightweight protocol designed for efficient communication between Git clients and servers.
27. **git instaweb –httpd=webrick** - It instantly launches a web server to browse repositories. It provides a simplified way to view repository contents through a web interface without the need for configuring a full web server.
28. **git rerere** - It reuses recorded resolution of previously resolved merge conflicts. Please note that rerere.enabled configuration option needs to be set to "true" (**git config –global rerere.enabled true**) for git rerere to work. Additionally, note that git rerere only applies to conflicts that have been resolved using the same branch and commit.

Steps :

1. Create a new directory and branch called newbranch
2. Make newbranch the active branch - git checkout -b newbrchname [It creates new branch and points to the created branch]
3. Create an empty file called newbranchfile - echo "this is a new file" > newfile.txt
4. Add the newly created file to your branch
5. Commit the changes in your newbranch
6. Revert the last committed changes
7. Create a new file called newgoodfile

8. Add the latest file to newbranch
9. Commit the changes
10. Merge the changes in newbranch into master

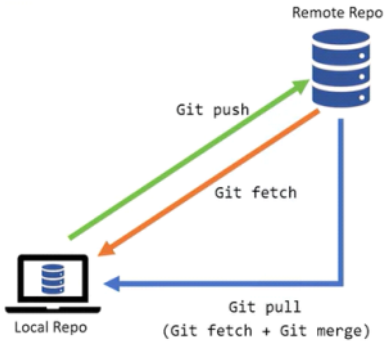
Staging area - While doing git add the files goes into staging area. It is an area where commits can be formed and reviewed before completing the commit.

Syncing Local Changes

- To sync code back to GitHub
  - Run the `git add <files>` command
  - Changed files move to staging area
  - Next, run `git commit -m <message>` to commit changes in the staging area
- To move changes fully into the GitHub repository
  - Use the `git push` command

Remote Repositories

- Remote repos are stored elsewhere
- Push, pull, and fetch data to share work
- Origin refers to your fork
- Upstream refers to the original work



-

Forking a Project

- Forking
  - Takes a copy of a GitHub repository to use it as the base for a new project
  - Submit changes back to the original repository
- Independently make changes to a project
  - Submit a **pull** request to the original project owner
  - Owner decides whether to accept updates
- Keep a copy of the license file
  - Often a legal requirement

Syncing a Fork of a Project

To keep a fork in sync with the original work from a local clone:

- Create a local clone of the project
- Configure Git to sync the fork
  - Open a Terminal and change to the directory containing the clone
  - To access the remote repository, type `git remote -v`
  - Type `git remote add upstream <clone directory>`
  - To see the change, type `git remote -v`

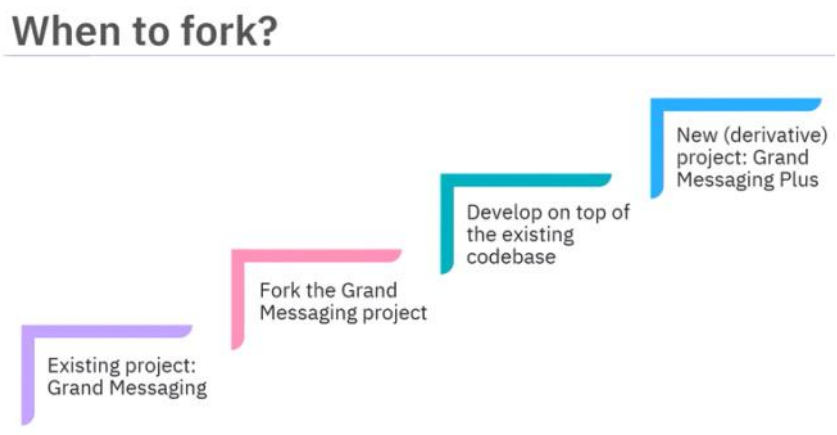
To grab upstream branches:
-git fetch upstream

To merge changes into the master branch
Git merge upstream/master -

Git pull upstream - To fetch and merge the remote branch in the same step. It reduces the steps to sync with a remote branch .

Automatic merges are not always desired

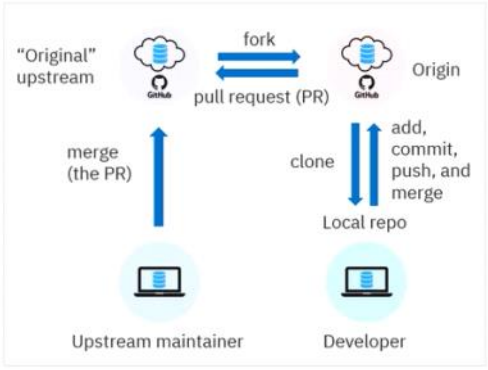
Cloning vs Forking:
When to fork: To work on code base without having to change the original one you can fork . The changes does not applied to original repo features



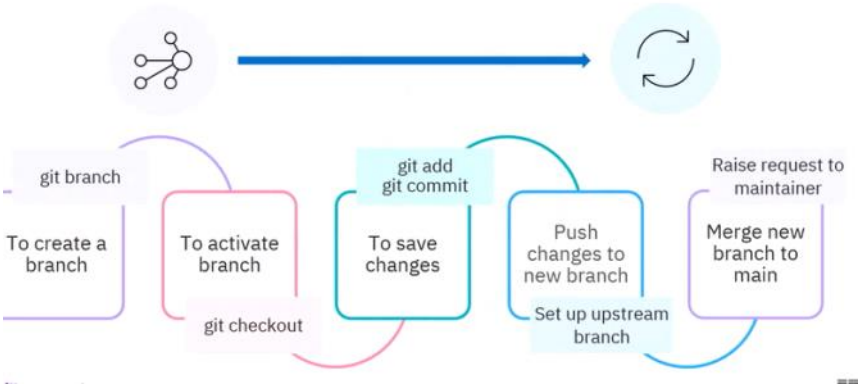
Pull request

Maintainers can:

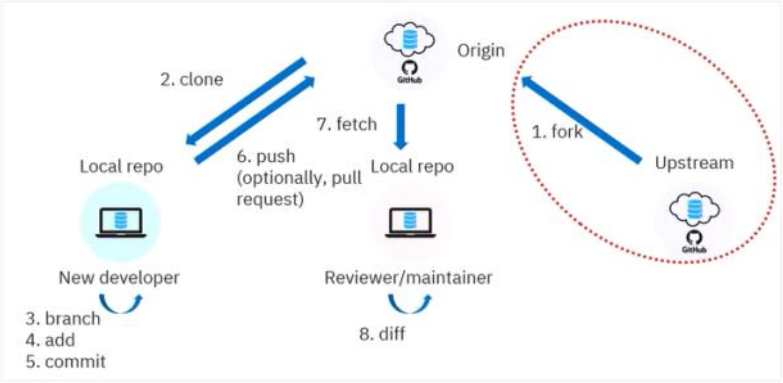
- Review, provide feedback, and merge
- Ask to perform conflict resolution



Create branches and synchronize changes



Clone workflow



GitHub developer

A developer communicates with others using these commands:

Command	Description
git clone	To create a copy from the upstream that the developer wants to work on
git pull and git fetch	To 'pull' or 'fetch' from 'origin' to keep up-to-date with the upstream
git push	To 'push' commits to the shared repository
git request-pull	To create a summary of changes for upstream to pull

GitHub integrator

Integrators use the following commands

Command	Description
git pull	To merge from trusted lieutenants
git revert	To undo botched commits
git push	To publish the project

GitHub repository administrator