**it**

* **What it is:** Git is a free and open-source version control system (VCS) that allows you to track changes in computer files and code. It's particularly useful for collaborative projects, where multiple people can work on the same files without conflicts. Git excels at:
  + **Version tracking:** It keeps a record of every change made to your files, allowing you to revert to previous versions if needed.
  + **Collaboration:** Enables teams to work on the same project simultaneously without overwriting each other's work.
  + **Branching and merging:** Facilitates creating isolated development environments (branches) and merging them back into the main project seamlessly.

**Installing Git on Windows**

1. Visit the official Git downloads page:<https://git-scm.com/download/win>
2. Download the latest 64-bit Git for Windows installer (unless you're using a 32-bit system).
3. Double-click the installer and follow the prompts.
   * During installation, choose the "Use Git Bash as my default shell" option for a more convenient command-line experience.

**GitHub**

* **What it is:** GitHub is a popular web-based hosting service specifically designed for Git repositories. It offers additional features like:
  + **Collaboration tools:** Makes it easier for teams to work together on projects.
  + **Version control visualization:** Provides a visual representation of your project's history.
  + **Issue tracking:** Helps manage bugs, tasks, and feature requests.
  + **Social coding:** Enables developers to discover and contribute to open-source projects.

**Key Git Commands**

* **Basic commands:**
  + git init: Initializes a new Git repository in your current directory.
  + git clone <url>: Clones an existing Git repository from a remote server (like GitHub).
  + git status: Shows the current status of your working directory (untracked, modified, staged files).
  + git add <file>: Adds a file or directory to the staging area for the next commit.
  + git commit -m "message": Creates a snapshot of your changes with a descriptive message.
  + git push: Uploads your local commits to a remote repository (like GitHub).
  + git pull: Downloads changes from a remote repository and merges them into your local branch.
* **Branching and merging:**
  + git branch <branch\_name>: Creates a new branch.
  + git checkout <branch\_name>: Switches to a different branch.
  + git merge <branch\_name>: Merges changes from one branch into another (may require resolving conflicts).

**Git vs. GitHub**

* Git is the version control system itself, a tool for tracking changes.
* GitHub is a web-based platform that provides a user interface and additional features on top of Git.

**GitLab**

* **What it is:** GitLab is another web-based platform similar to GitHub, offering Git repository hosting and version control functionalities along with other project management features.

**Git Clone Commands**

* git clone <url>: Clones an existing Git repository from a remote server (e.g., GitHub, GitLab).
  + Example: git clone https://github.com/username/repository.git clones the repository from username's GitHub account.

**Git Push Commands**

* git push origin <branch\_name>: Pushes your local commits from the current branch to the remote branch named <branch\_name> on the default remote server (often called origin).
  + Example: git push origin main pushes your local commits on the main branch to the remote main branch.

**Git Pull Commands**

* git pull origin <branch\_name>: Downloads changes from the remote branch named <branch\_name> on the default remote server (origin) and merges them into your local branch.
  + Example: git pull origin main downloads changes from the remote main branch and merges them into your local branch.

**Git History**

* git log: Shows the history of commits made to the repository, including commit messages, author, and date.
* git reflog: Shows a more detailed history of Git operations, including branch changes and checkout commands.

**Branching and Merging**

* Branches allow you to create isolated development environments to work on features independently without affecting the main project.
* Merging integrates changes from one branch into another, potentially requiring you to resolve conflicts if both branches modified the same files.

\*\*Resolve Merge Conflicts

When merging branches in Git, conflicts can arise if both branches modified the same lines in a file. Here's how to resolve them:

**1. Identify Conflicted Files:**

* Use the git status command to see which files have conflicts. Git will mark them as "unmerged paths."

**2. Open Conflicted Files:**

* Open each conflicted file in a text editor. Git typically uses markers like <<<<<<<, =======, and >>>>>>> to indicate the different versions from your branch, the common ancestor (the commit before the branches diverged), and the other branch, respectively.

**3. Resolve Conflicts Manually:**

* Carefully examine the conflict markers and decide which changes to keep from each branch. You can:
  + Keep changes from one branch entirely.
  + Combine changes from both branches (e.g., if they modify different parts of the same line).
  + Discard changes from one branch if they're no longer relevant.
* Edit the file content to reflect your chosen version(s), removing the conflict markers.

**4. Stage Resolved Files:**

* Once a conflicted file is resolved, use git add <file> to stage it for the next commit. This tells Git that you've addressed the conflict.

**5. Create a Merge Commit:**

* Use git commit -m "message" to create a new commit that captures the merged changes and your conflict resolution.

**Additional Tips:**

* Use a merge tool provided by your Git client or editor for a visual way to resolve conflicts.
* Consider the context and purpose of each branch's changes when deciding which ones to keep.
* If you're unsure about a conflict, consult with other developers who worked on the branches.
* There are advanced strategies like git checkout --theirs or git checkout --ours to accept changes entirely from one branch, but use them with caution, especially in collaborative environments.

By following these steps, you can effectively resolve merge conflicts and ensure a smooth merging process in your Git workflow.

In Git, "**origin**" is a shorthand name for the remote repository that a project was originally cloned from. More precisely, it is used instead of that original repository's URL - and thereby makes referencing much easier. Note that origin is by no means a "magical" name, but just a standard convention.

What practical problems does got solves ?

**Example clone and branching —**

C:\Users\ANKIT\Desktop\LocalRepository>**git clone https://github.com/AshuMItter/fun.git**

Cloning into 'fun'...

remote: Enumerating objects: 6, done.

remote: Counting objects: 100% (6/6), done.

remote: Compressing objects: 100% (4/4), done.

remote: Total 6 (delta 1), reused 0 (delta 0), pack-reused 0

Receiving objects: 100% (6/6), done.

Resolving deltas: 100% (1/1), done.

C:\Users\ANKIT\Desktop\LocalRepository>dir/w

Volume in drive C is Windows

Volume Serial Number is 02CE-9C12

Directory of C:\Users\ANKIT\Desktop\LocalRepository

[.] [..] [fun]

0 File(s) 0 bytes

3 Dir(s) 14,393,204,736 bytes free

C:\Users\ANKIT\Desktop\LocalRepository>cd fun

C:\Users\ANKIT\Desktop\LocalRepository\fun>**git branch**

\* main

C:\Users\ANKIT\Desktop\LocalRepository\fun>**git push --set-upstream origin main**

branch 'main' set up to track 'origin/main'.

Everything up-to-date

C:\Users\ANKIT\Desktop\LocalRepository\fun>**git status**

On branch main

Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

C:\Users\ANKIT\Desktop\LocalRepository\fun>git branch

\* main

C:\Users\ANKIT\Desktop\LocalRepository\fun>**git branch -a**

\* main

remotes/origin/FourthBranch

remotes/origin/HEAD -> origin/main

remotes/origin/main

remotes/origin/secondBranch

remotes/origin/thirdBranch

C:\Users\ANKIT\Desktop\LocalRepository\fun>**git remote add origin** https://github.com/AshuMItter/fun.git

error: remote origin already exists.

C:\Users\ANKIT\Desktop\LocalRepository\fun>git status

On branch main

Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

C:\Users\ANKIT\Desktop\LocalRepository\fun>git checkput -b demo-branch

git: 'checkput' is not a git command. See 'git --help'.

The most similar command is

checkout

C:\Users\ANKIT\Desktop\LocalRepository\fun>**git checkout -b demo-branch**

Switched to a new branch 'demo-branch'

C:\Users\ANKIT\Desktop\LocalRepository\fun>

**Git local repo ,push and branch**

C:\Users\ANKIT\Desktop>mkdir LocalRepo

C:\Users\ANKIT\Desktop>cd localrepo

C:\Users\ANKIT\Desktop\LocalRepo>**git init**

Initialized empty Git repository in C:/Users/ANKIT/Desktop/LocalRepo/.git/

C:\Users\ANKIT\Desktop\LocalRepo>git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

readme.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\ANKIT\Desktop\LocalRepo>git commit -a -m "added readmefile"

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

readme.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\ANKIT\Desktop\LocalRepo>**git status**

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

readme.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\ANKIT\Desktop\LocalRepo>git add

Nothing specified, nothing added.

hint: Maybe you wanted to say 'git add .'?

hint: Disable this message with "git config advice.addEmptyPathspec false"

C:\Users\ANKIT\Desktop\LocalRepo>**git add .**

C:\Users\ANKIT\Desktop\LocalRepo>**git commit -a -m "added readmefile"**

[master (root-commit) a71b559] added readmefile

1 file changed, 1 insertion(+)

create mode 100644 readme.txt

C:\Users\ANKIT\Desktop\LocalRepo>git status

On branch master

nothing to commit, working tree clean

C:\Users\ANKIT\Desktop\LocalRepo>**git remote add demorepo https://github.com/AshuMItter/demo.git**

C:\Users\ANKIT\Desktop\LocalRepo>git push --set-upstream demorepo main

error: src refspec main does not match any

error: failed to push some refs to 'https://github.com/AshuMItter/demo.git'

C:\Users\ANKIT\Desktop\LocalRepo>**git push --set-upstream demorepo master**

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 237 bytes | 237.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)

To https://github.com/AshuMItter/demo.git

\* [new branch] master -> master

branch 'master' set up to track 'demorepo/master'.

C:\Users\ANKIT\Desktop\LocalRepo>git status

On branch master

Your branch is up to date with 'demorepo/master'.

nothing to commit, working tree clean

C:\Users\ANKIT\Desktop\LocalRepo>git branch

\* master

C:\Users\ANKIT\Desktop\LocalRepo>**git branch -a**

\* master

remotes/demorepo/master

C:\Users\ANKIT\Desktop\LocalRepo>git remote add origin https://github.com/AshuMItter/demo.git

C:\Users\ANKIT\Desktop\LocalRepo>git status

On branch master

Your branch is up to date with 'demorepo/master'.

nothing to commit, working tree clean

C:\Users\ANKIT\Desktop\LocalRepo>**git push demorepo**

Everything up-to-date

C:\Users\ANKIT\Desktop\LocalRepo>git status

On branch master

Your branch is up to date with 'demorepo/master'.

nothing to commit, working tree clean

C:\Users\ANKIT\Desktop\LocalRepo>  
  
C:\Users\ANKIT\Desktop\LocalRepo>**git checkout -b demo-branch**

Switched to a new branch 'demo-branch'

C:\Users\ANKIT\Desktop\LocalRepo>**git push demorepo demo-branch**

Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)

remote:

remote: Create a pull request for 'demo-branch' on GitHub by visiting:

remote: https://github.com/AshuMItter/demo/pull/new/demo-branch

remote:

To https://github.com/AshuMItter/demo.git

\* [new branch] demo-branch -> demo-branch

C:\Users\ANKIT\Desktop\LocalRepo>

Link - <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/incorporating-changes-from-a-pull-request/reverting-a-pull-request>

Link - <https://www.w3schools.com/git/>