The steps are:

1. git init
2. git remote add origin (git remote add origin https://github.com/username/project-name.git)
3. git pull origin main
4. git add .
5. git commit (git commit -m "Your message")
6. git push origin main

**git init**

**git init** is like creating a new empty folder where Git can start tracking your files.

Think of it this way:

* You have a regular folder with your project files
* When you run git init, you're telling Git "Hey, please start watching this folder!"
* Git creates a hidden .git folder inside your project
* Now Git is ready to track any changes you make to your files

It's like putting a security camera in a room - the camera is now installed and ready to record, but it hasn't started recording anything yet. You still need to tell it what to record (using other Git commands like git add).

**When do you use it?**

* At the very beginning of a new project
* Only run it once per project
* Run it inside the folder where your project files are

That's it! After git init, your folder becomes a "Git repository" - just a fancy name for a folder that Git is watching.

**git remote add origin**

**git remote add origin** is like telling Git where your project's online backup location is.

**Simple explanation:**

* Your project lives on your computer
* You want to also store it online (like GitHub)
* This command tells Git: "The online home for this project is at this web address"

**What each part means:**

* git remote = working with online locations
* add = adding a new location
* origin = a nickname (like calling your house "home")

**How you actually use it:**

*git remote add origin* [*https://github.com/username/project-name.git*](https://github.com/username/project-name.git)

**Think of it like:**

* You write down your friend's address in your phone
* Instead of remembering "123 Main Street", you just save it as "John's House"
* Now when you want to visit, you just say "go to John's House"
* Same here: instead of typing the long web address, you just say "origin"

**When do you use it?**

* After you create an empty repository on GitHub
* Before you try to upload your code for the first time
* Only once per project

After this command, Git kn**sing the camera example:**

Remember how git init was like installing a security camera in your room? Well, that camera is just recording locally on its own memory card.

**What git remote add origin does:**

* It's like connecting your camera to the internet
* You're telling the camera: "Send all your recordings to this online storage service"
* "origin" is like naming that storage service "My Cloud Backup"

**Think of it this way:**

* Your camera (Git) is recording in your room (your computer)
* But you also want copies stored safely online
* git remote add origin gives the camera the address of your online storage
* Now the camera knows where to upload the recordings when you tell it to

**The full command:**

*git remote add origin* [*https://github.com/username/project.git*](https://github.com/username/project.git)

Is like telling your camera: "Your online backup location is called 'My Cloud Backup' and it's at this internet address"

**When you use it:**

* After installing the camera (git init)
* Before trying to upload any recordings to the cloud
* Only once - the camera remembers the address after that

Now your security camera knows where to send backups of all the changes it's been watching!

**git pull origin main**

**git pull origin main** is like downloading the latest recordings from your online security camera storage.

**Using the camera example:**

Remember:

* Your camera (git init) is installed and recording locally
* You told it where your online backup is (git remote add origin)
* Now other cameras (other people) might have uploaded new recordings to that same online storage

**What git pull origin main does?**

* It's like saying: "Hey camera, check our online storage and download any new recordings I don't have yet"
* Your camera goes to "origin" (your online backup location)
* It looks in the "main" folder (the main recording folder)
* It downloads any new or updated recordings to your local camera

**Think of it this way:**

* Maybe your teammate's camera uploaded new footage while you were away
* Or maybe you worked on another computer and uploaded recordings from there
* git pull makes sure your current camera has ALL the latest recordings
* It syncs your local recordings with what's stored online

**When you use it:**

* Before starting to work on your project
* When you know someone else has uploaded changes
* To make sure you have the most up-to-date version

**What happens:** Your camera downloads the new recordings and adds them to its local storage, so now you have everything that's been recorded by anyone on your team!

It's like syncing your phone photos with the cloud - you get all the latest pictures downloaded to your device.

**Other Example**

**git pull origin main** is like getting the latest version of a shared Google Doc.

**Think of it this way:**

You and your friends are working on the same document together:

* The document lives online (like Google Drive)
* Each person has their own copy on their computer
* People keep making changes and saving them online

**What git pull origin main does?**

* It's like clicking "refresh" to get the newest version
* You're saying: "Download all the latest changes from our shared document"
* Your local copy gets updated with everything new

**Step by step:**

1. Your friend edited the document yesterday and saved it online
2. You open your local copy (but it's still the old version)
3. You run git pull origin main
4. Now your copy downloads all your friend's changes
5. Your document is now up-to-date with the online version

**When you use it:**

* Before you start working (so you don't work on an old version)
* When someone tells you "I just uploaded changes"
* To avoid conflicts when multiple people are working together

**What it prevents:**

* Working on outdated information
* Accidentally overwriting someone else's work
* Having different versions of the same project

It's like making sure everyone is reading from the same page of the same book!

**git add .**

**Camera Example:**

Your security camera has been watching you work, but it hasn't taken any official photos yet.

**What git add . does?**

* It's like telling your camera: "Take a snapshot of EVERYTHING you can see right now!"
* The dot (.) means "photograph the entire room"
* Your camera takes a picture of all your files in their current state
* This photo goes into a "ready to save" pile, but isn't permanently stored yet

**Think of it:**

* You've been moving furniture, adding decorations, cleaning up
* The camera watched it all happen
* git add . = camera takes one big group photo of how everything looks NOW
* The photo is staged and ready, but not yet saved to the permanent album

**Shopping Example:**

**It's like putting everything in your shopping cart at once:**

* You've been walking around the store picking things up
* git add . = you grab EVERYTHING you've been carrying and put it all in your cart
* The dot (.) means "everything I'm holding"
* Now all your items are in the cart, ready for checkout
* But you still haven't paid yet (that's git commit)

**When you use it:**

* After you've made changes to your files
* When you want to prepare ALL your changes to be saved
* Right before you do git commit

It's the "select all" button for your changes!

**git commit**

**git commit**

**Camera Example:**

Remember your camera took that snapshot with git add .? Now it's time to save it permanently!

**What git commit does:**

* It's like taking that photo from the "ready to save" pile and putting it in your permanent photo album
* You write a little note on the back saying "What I did today" (the commit message)
* Now this moment is saved forever in your camera's memory
* You can always look back and see exactly how your room looked at this moment

**Think of it:**

* git add . = camera takes the photo
* git commit = photo goes into the permanent album with a description
* The photo is now part of your project's history forever

**Shopping Example:**

**It's like going to the checkout and paying:**

* Your cart is full of items (git add . put them there)
* git commit = you go to the cashier and actually buy everything
* You get a receipt that says what you bought and when (the commit message)
* Now those items officially belong to you forever
* You can always look at your receipt to remember what you bought

**The full command usually looks like:**

*git commit -m "Your message"*

The -m part lets you write a note explaining what you did.

**When you use it:**

* Right after git add .
* When you want to permanently save your changes
* Think of it as "saving your progress" in a video game

Now your changes are officially recorded in your project's history!

**git commit -m "Your message"**

Here **git commit -m "Git Demo"**

**Camera Example:**

Your camera has that snapshot ready, and now you're putting it in the permanent photo album WITH a label!

**What this does:**

* It's like taking that photo and writing "Git Demo" on the back before putting it in your album
* The -m means "message" - like writing a caption
* "Git Demo" is your description of what this photo shows
* Now when you flip through your photo album later, you'll see this picture labeled "Git Demo"

**Think of it:**

* git add . = camera takes the photo
* git commit -m "Git Demo" = put photo in album with the label "Git Demo"
* Forever after, this moment in your project's history will be called "Git Demo"

**Shopping Example:**

**It's like completing your purchase with a specific receipt description:**

* Your cart is full (git add .)
* You go to checkout and pay (git commit)
* The -m "Git Demo" is like writing "Git Demo" in the memo field of your payment
* Your receipt will show you bought items for "Git Demo" purpose
* Later when you look at your purchase history, you'll see "Git Demo" and remember what this was for

**Why the message matters:**

* Future you will thank present you for the clear description
* Other people working on the project will understand what you did
* It's like leaving breadcrumbs so you can retrace your steps

**The result:** Your changes are now permanently saved in your project's history with the label "Git Demo"!

**git push origin main**

**git push origin main**

**Camera Example:**

Your camera has saved the photo in its local album, but now it's time to upload it to your online cloud storage!

**What this does:**

* It's like taking your photo album and uploading all the new pictures to your online backup
* "origin" is the nickname for your cloud storage service (remember from git remote add origin)
* "main" is like the "main folder" where you store your primary photos
* Your camera uploads the "Git Demo" photo (and any others) to the cloud

**Think of it:**

* git commit = photo saved in your camera's local album
* git push origin main = upload that photo to your online cloud storage
* Now other people can see your "Git Demo" photo online
* Your backup is safe in the cloud!

**Restaurant Delivery Example:**

**It's like sending your completed order to the kitchen:**

* You've been working on a recipe at your table (your computer)
* You wrote down all the final instructions (git commit -m "Git Demo")
* git push origin main = you send that recipe to the main kitchen (online repository)
* Now the head chef (other team members) can see your "Git Demo" recipe
* Everyone working in the restaurant has access to your latest recipe

**What happens:**

* Your local work gets uploaded to the shared online space
* Other people can now download your changes with git pull
* Your work is safely backed up online
* The team can see what you've been working on

**When you use it:**

* After you've committed your changes
* When you want to share your work with others
* To backup your work online

Now your "Git Demo" is live and available to everyone!