**============== ============= GIT ============= ==============**

\*\* SHAJ \*\*

**Git work flow:**

**To upload: working directory => stage => local Git repo => GitHub.**

**=> git add (working directory to stage files).**

**=> git commit(stage to local repo with message)**

**=> git push(local to GitHub).**

**To download:**

**=> git clone | git pull (GitHub to working directory)**

**=> git fetch (GitHub to local)**

**=> git check out (local to working directory).**

**=> git diff (to see code difference between working directory and stage).**

**=> git diff HEAD (to see code difference between working directory and local).**

**=> git merge (to combine code between working directory and stage).**

Today we will learn from raghab paul:

**1. How to create github repository**

**2. How to clone repository in eclipse**

**3. How to add eclipse project to github repository**

**4. How to commit, push and pull the changes**

Step 1 : Create GitHub account and SignIn

Step 2 : Start a Project = Create a repository in github.

Step 3 : Start Eclipse

Step 4 : Go to Git Perspective - Git Repositories and click on Add Git Repo

Step 5 : Create a project in Eclipse

Step 6 : Do a right click on Project - Team - Share - Add to git repo

Step 7 : Commit and Push the project to the repo

Step 8 : Commit and Push every change to the repo

======================== GIT Operations =========================

=> Git comes with Eclipse by default. We also can download.

**Temporary Git Token:** ghp\_r4TAuBX1s2QsFwupxjOy2vriskbmAV1lNKZw

**How to Generate a github Token**

**Click on photo on github > Settings > developer settings > personal access token >…**

1. **Create a local git repository:**

=> In Eclipse click **Git perspective** button next to **Quick Access** > add existing repo / clone Git Repository / create new local git repo > id / password > check **store in secure store** > next > select Branch … Mukesh.

=> right click on project > team > share project > create (local repository folder) > browse

> ceate another folder (GIT) > ok > finish > finish.

**Create a local git repo:**

(i) open GIT perspective - create new local git repo - or

(ii) right click on project - team - Share project (if earliar created a repo then it will popup, if not created then click on create)

- check use or create in parent folder > select project > create repository.

**Check the created repository:**

click on window > show view > other > Git > Git Repository > ok.

( we can see the repositoies ).

2. **Create a repository in GitHub:**

=> Login github > click on(+) > new repository > do name > create a repository.. Mukesh

3. **Stage a project:**

=> In Eclipse click **Java** button next to **Quick Access >** right click on project > Team > Share project > select project > add target location > finish.

Now In Eclipse click **GIT** button next to **Quick Access >** select project > Git Staging > select all in **unStaged** > dragAndDrop to **Staged Changes** > do **commit** message > **commit** / **commit** **and** **push** . ….Mukesh.

=> right click on project > team > commit > add to Index(+) > commit message > commit.

click on window > shows view > other > Git > Git repository. ( we can see the Master ).

right click on project > team > show in history. ( we can see the Master here too).

**Commit staged files to local git repo:**

right click on project > Team > commit > commit message > commit.

**Add files to staging:**

right click on project > Team > Add to Index (will be added to staged area)

4. **push a project to gitHub:**

=> Go to Github > click **code** > clone url (copy) > click **Git perspective** button next to **Quick Access** > clone Git Repository > id / password > check **store in secure store** > next > select Branch . Mukesh

=> right click on project > team > Push Branch Master. (we have to give GIT account information).

open Git > create new repo ( + ) > check Read Me > create repository > clone or download > copy URL > paste to Branch Master > enter Git uid + PassWord > next > check Force Overrite > next > finish > enter Git uid + PassWord > ok.

( now check Git to see the repository)

5. **Update a project after any change:**

=> In Eclipse click **GIT** button next to **Quick Access >** select project > Git Staging > select all in **unStaged** > dragAndDrop to **Staged Changes** > do **commit** message > **commit** / **commit** **and** **push**.. Mukesh

**Push changes to Master branch Remote Repo(GitHub):**

Login to GitHub > create new repo(+) > clone url -

in Eclipse > in Git > right click on repo project > remote > push > paste url > git uid/pass > next > add spec > push.

or right click on Eclipse project > remote > push > paste url > git uid/pass > next > add spec > push.

now project available in GitHub.

**Pull Request(pr) to Lead for review the code:**

login to GitHub > click the project > select Branch > click new Pull Request > commit message > select reviewer

Commit message Example: I have commited test2, please review and approve. thaks.

**Push changes to Remote Repo from new Branch:**

Create(push) new Branch in GitHub repo using Eclipse:

in Eclipse > git > right click on newly made Branch > push Branch > member uid/pass > push > member uid/pass > login > close.

**Commit change to local repo:**

make a local repo for this project into this new Branch.

6. **Pull a project from GitHub after any changes by some one :**

=> In Eclipse Right click on project > Team > pull > select changed items > close.. Mukesh

**Download project from Git to local (Eclipse).**

click on file > import > Git > project from Git > next > clone URL

> paste Git project URL > next > next > finish. ( no need uid/password if public repository).

= now we can see the project in Eclipse.

**import a project to Eclipse.**

> window > show view > other > Git Repositories > open > Clone a Git reposiry > copy/paste project url from GitHub > enter uid/pass of GitHub > select branch > next > finish.

**import project from local git repo to Eclicpse workspace:**

right click on packageExplorer > import > git > project from git > next > existing local repository > next > select project > next > import existing eclipse project > next > finish.

7. **Execute Git Repo (test case) on Jenkins:**

Login to GitHub > clone repo path > login to Jenkins > Source code Management > select **git** > paste repo path > credentials > click add > jenkins (enter git user name, password + do some description) >

8. **Add a team member:**

=> in GitHub > setting > cllaborators > provide user name / email id >

**add team member:**

setting > collaborators > enter memeber name > add collaborator > copy invite link and share project link to the member.

9. **How to trigger notification email from github whenever there is any change/commit in the project.**

Step 1 : Github - Repository - Settings - integration & services - add email

Step 2 : Test and validate by making some change in the project.

**10. how to clone(copy) a project from GitHub to local.**

**Step 1 : Import code from GIT and pull it down - normal project to hard\_disk.**

->in Eclipse.. file -> import -> type git -> project from git -> click **code** in github > clone url (copy from github) -> paste -> next -> select branch -> next -> next -> finish.

**Step 2: Import the normal project from hard\_disk to Eclipse as a Maven Project.**

right click on imported project -> properties -> copy path -> delete Project.

-> file -> import -> type maven -> select existing maven projects -> paste the path -> click on brows -> finish. then run as ' clean and install'.

**Clone Remote repo to local repo:**

Eclipse > Git > clone a git repository > paste project url > member uid/pass > next > master branch > next > finish. (it will be local repo).

**11. How to create a new Branch ?**

Click Master Branch dropdown > do name for new Branch > Create Branch / Enter.

**How to make a new local branch** :

in eclipse > git > git repo > right click on Brances > switch to new Branch > do name > finish.

**12. How to make changes on a Branch ?**

Select Branch > click on file > Edit file > do Edit > Commit change messages > compare & pull request.

**13. How to Merge new branch to main branch ?**

After changing click on new branch > commit message > Create pull request > Merge pull request > confirm merge.

**14.** **How to see the project in GitHub:**

login GitHub > repositories > Branch > select Branch> src > test case

========================== **GIT with PAVAN** ============================

: SDET- QA Automation -

**GIT for Test Lead**…

1) Create a new project in Eclipse

2) Create a local git repo:

(i) open GIT perspective - create new local git repo - or

(ii) right click on project - Team - Share project (if earliar created a repo then it will popup, if not created then click on create)

- check use/create in parent folder - select project - create repository.

3) Add files to staging:

right click on project - Team - Add to Index (will be added to staged area)

4) Commit changed, staged files to local git repo:

right click on project - Team - commit - commit message - commit.

5) Push changes to Master branch Remote Repo(GitHub).

Login to GitHub - create new repo(+) - clone url -

in Eclipse - in Git - right click on repo project - remote - push - paste url - git uid/pass - next - add spec - push.

or right click on project - remote - push - paste url - git uid/pass - next - add spec - push.

now project available in GitHub.

6) add team member:

setting - collaborators - enter memeber name - add collaborator - copy invite link and share project link to the member.

**Git for Team member:**

open the shared link then can see project > copy project url.

1) **Clone Remote repo to local repo:**

Eclipse > Git > clone a git repository > paste project url > member uid/pass > next > master branch > next > finish. (it will be local repo).

2) **import project from local git repo to Eclicpse workspace:**

right click on packageExplorer > import > git > project from git > next > existing local repository > next > select project > next > import existing eclipse project > next > finish.

3) **Add/modify features/code/test cases:**

do your work...modify or changes on this project.

4) **make a new local branch** :

Right click on project > team > switch to > New Branch > do name > finish.

in eclipse > git > git repo > right click on Brances > switch to new Branch > do name > finish.

5) **Add files to staging:**

right click on project > Team > Add to Index (will be added to staged area)

6) **Commit change to local repo:**

make a local repo for this project into this new Branch.

7) **Push changes to Remote Repo from new Branch:**

Create(push) new Branch in GitHub repo using Eclipse:

in Eclipse > git > right click on newly made Branch > push Branch > member uid/pass > push > member uid/pass > login > close.

8) **Pull Request(pr) to Lead for review the code:**

login to GitHub > click the project > select Branch > click new Pull Request > commit message > select reviewer

Commit message Example: I have commited test2, please review and approve. thaks.

**How to see the project in GitHub:**

login GitHub > repositories > Branch > select Branch> src > test case

How to Merge project:

**====================== Git W3School + techTFQ =======================**

**: Down Load GIT >** go to git website on browser > follow instructions.

**: Git commands -**

**: Git.**

git --version **- to see the Git version.**

git config --global user.name "w3schools-test" - to configure GIT.

git config --global user.email [test@w3schools.com](mailto:test@w3schools.com)

mkdir **myproject** **- mkdir makes a new project directory ( repository folder ).**

cd **myproject** **- enter to the project folder.**

**myproject** mkdir **digitalClock** **- make another folder.**

**myproject** cd **digitalClock** **- enter to the folder.**

**NB: I need to create one repository for each project.**

**digitalClock ls - to see the list of files.**

**digitalClock ls –a - to see the hidden list of files.**

git init **-Initializes empty Git repository in /Users/user/myproject/.git/**

**digitalClock** git init

**digitalClock** git status

**digitalClock touch file\_1.txt - create a file.**

**digitalClock touch file\_2.txt.**

**digitalClock ls - to see the list of files.**

cd desktop - to enter desktop area..

desktop ls - to see repsitory list in desktop.

**digitalClock** git add index.html **- to Stage individual files. Can do multiple files.**

**digitalClock** git add --all **- to Stage all fies instead of individual filenames. or**

**digitalClock** git add .  **- to Stage all fies instead of individual filenames.**

**digitalClock** git ls-files - to see the list of files.

**digitalClock** git ls-files - to see the list of files.

**digitalClock** git rm –cached file\_2.txt - to unstage files.

**git commit –m "message" - to store to local repo.**

**digitalClock** git commit -m "First release of Hello World!"

**digitalClock** git commit -a -m "Updated index.html with a new line" **- to commit without staging.**

**digitalClock** git status **- to check the status**

**digitalClock** git status –short **- to check the status short and compact way.**

M index.html

**Note: Short status flags are:**

**• ?? - Untracked files**

**• A – staged Files**

**• M - Modified files**

**• D - Deleted files**

**git log - To view the history of commits for a repository**

**digitalClock** git log

git log – oneline - to see the history in one line.

**git command -help - See all the available options for the specific command**

**git status -help**

**git help -all - See all possible commands**

**git branch hello-world-images - to create a local branch**

**git checkout -b emergency-fix - to create a new branch then switch to it.**

**git branch - to see the local branches.**

**hello-world-images (out put) [ \* master means control is now in this branch.]**

**\* master**

**git switch hello-world-images - to switch/move to a branch**

**git code - to go to working page to code.**

**git checkout hello-world-images - to switch to a branch or**

**git checkout commit id - to switch to old commit**

**git checkout -b emergency-fix - to create a new branch then switch to it & commit will be stored.**

**or**

**git checkout -c emergency-fix - to create a new branch then switch to it & commit will be stored.**

**git merge emergency-fix - to Merge a branch [emergency-fix will be merged to master]**

**git branch -d emergency-fix - to delete a branch**

**: Github.**

**git remote add origin https://github.com/w3schools-test/hello-world.git**

**( connecting the Github URL as an origin to your local Git repo )**

**git push --set-upstream origin master - to push our master branch to the origin url**

**git remote add origin https://github.com/kaijim/w3schools-test.github.io.git**

**Now we have 2 remotes:**

**• origin - our own fork, where we have read and write access**

**• upstream - the original, where we have read-only access**

**git pull origin - pull is a combination of fetch and merge.**

**It is used to pull all changes from a remote repository into the branch you are working on.**

**git fetch origin - fetch gets all the change history of a tracked branch/repo**

**git diff origin/master - verify by showing the differences between our local master and origin/master:**

**git merge origin/master - merge combines the current branch, with a specified branch.**

**example:**

**git commit -a -m "Updated index.html. Resized image"**

**git status**

**git push origin**

**git branch - to see all local branches.**

**git branch -r to see remote branches only.**

**git branch –a - -a option to see all local and remote branches.**

**git push gh-page master - to push to master of gh-page repo.**

**Git Revert HEAD - to revert back to previous commit.**

**git reset 9a9add8 - reset our repository back to the specific commit using commit id.**

**git reset – hard 9a9add8 - reset our repository back to the specific commit using commit id**

**git commit --amend -m "Added lines to README.md" - to amend the most recent commit.**

🡺 to see the Git version.

git –version

git version 2.30.2.windows.1

🡺 Now let Git know who you are. This is important for version control systems, as each Git commit uses this information:

git config --global user.name "w3schools-test"

git config --global user.email [test@w3schools.com](mailto:test@w3schools.com)

🡺 Now, let's create a new Git folder for our project:

mkdir myproject : mkdir **make**s a **new directory**.

cd myproject : cd **changes** the **current working directory**.

🡺 Once you have navigated to the correct folder, you can initialize Git on that folder:

git init

Initialized empty Git repository in /Users/user/myproject/.git/

🡺 You just created your first local Git repo. But it is empty. So let's add some files, or create a new file.

Example: file- index.html.

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
  
</body>  
</html>

🡺 As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

**Staged** files are files that are ready to be **committed** to the repository you are working on.

git add index.html

🡺 The file should be **Staged**. Let's check the status:

git status

🡺 You can also stage more than one file at a time. Let's add 2 more files to our working folder.

A README.md file that describes the repository (recommended for all repositories):

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
  
</body>  
</html>

Now add all files in the current directory to the Staging Environment:

git add –all

Using --all instead of individual filenames will stage all changes (new, modified, and deleted) files.

🡺 Since we have finished our work, we are ready move from stage to commit for our local repo.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

git commit –m "message"

git commit -m "First release of Hello World!"

🡺 Git commit without stage - some time when you make small changes.

The -a option will automatically stage every changed, already tracked file.

git commit -a -m "Updated index.html with a new line"

🡺 check the status of our repository. But this time, we will use the --short option to see the changes in a more compact way:

git status --short

M index.html

**Note:** Short status flags are:

* ?? - Untracked files
* A - Files added to stage
* M - Modified files
* D - Deleted files

🡺 To view the history of commits for a repository, you can use the log command:

git log

🡺 If you are having trouble remembering commands or options for commands, you can use Git help.

git *command* -help -  See all the available options for the specific command

git *status* -help

git help -all -  See all possible commands

**Note:** You can also use --help instead of -help to open the relevant Git manual page

🡺 In Git, a branch is a new/separate version of the main repository.

Let's say you have a large project, and you need to update the design on it.

Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects.

When we are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new branch:

git branch hello-world-images

Now we created a new branch called "hello-world-images"

Let's confirm that we have created a new branch:

git branch

hello-world-images

\* master

We can see the new branch with the name "hello-world-images", but the \* beside master specifies that we are currently on that branch.

Now let’s switch/move to hello-world-images branch.

checkout is used to swithc/move to…

git checkout hello-world-images

Now we have moved our current workspace from the master branch, to the new branch. Now let’s make some changes on file.

After made changes to a file we need to add both files to the Staging Environment for this branch:

git add –all

We are happy with our changes. So we will commit them to the branch:

git commit -m "Added image to Hello World"

Now we have a new branch, that is different from the master branch.

**Note:** Using the -b option on checkout will create a new branch, and move to it, if it does not exist

We are currently on the branch hello-world-images. We added an image to this branch, so let's list the files in the current directory:

ls

We can see the new file img\_hello\_world.jpg, and if we open the html file, we can see the code has been altered. All is as it should be.

Now, let's see what happens when we change branch to master, switch/move to

Master branch.

git checkout master

Switched to branch 'master'

🡺 **Emergency Branch**

Now imagine that we are not yet done with hello-world-images, but we need to fix an error on master.

So we create a new branch to deal with the emergency:

git checkout -b emergency-fix

Switched to a new branch 'emergency-fix'

Now we have created a new branch from master, and changed to it. We can safely fix the error without disturbing the other branches.

We have made changes in this file, and we need to get those changes to the master branch.

stage the file, and commit:

git add index.html

git commit -m "updated index.html with emergency fix"

**🡺** Merge Branches.

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to switch/move to the master branch:

git checkout master

Switched to branch 'master'

Now we need to merge the current branch (master) with emergency-fix:

git merge emergency-fix

🡺 Delete Branch.

As master and emergency-fix are essentially the same now, we can delete emergency-fix, as it is no longer needed:

git branch -d emergency-fix

🡺 Merge conflict.

Now we can move over to hello-world-images and keep working. Add another image file (img\_hello\_git.jpg) and change index.html, so it shows it:

git checkout hello-world-images

Now, we are done with our work here and can stage and commit for this branch:

git add --all

git commit -m "added new image"

We see that index.html has been changed in both branches. Now we are ready to merge hello-world-images into master. But what will happen to the changes we recently made in master?

git checkout master

git merge hello-world-images

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

The merge failed, as there is conflict between the versions for index.html. Let us check the status:

git status

On branch master

You have unmerged paths.

(fix conflicts and run "git commit")

(use "git merge --abort" to abort the merge)

Changes to be committed:

new file: img\_hello\_git.jpg

new file: img\_hello\_world.jpg

Unmerged paths:

(use "git add ..." to mark resolution) both modified: index.html

This confirms there is a conflict in index.html, but the image files are ready and staged to be committed.

So we need to fix that conflict.

Now we can stage index.html and check the status:

git add index.html

git status

The conflict has been fixed, and we can use commit to conclude the merge:

git commit -m "merged with hello-world-images after fixing conflicts"

And delete the hello-world-images branch:

git branch -d hello-world-images

🡺 **Github account**:

Go to [GitHub](https://www.github.com/) and sign up for an account and log in.

**Note:** Remember to use the same e-mail address you used in the Git config.

: create a repository- click (+) icon for new repository.

: Push Local Repository to GitHub

Copy the project URL, or click the clipboard icon on github page.

Now paste it the following command:

git remote add origin https://github.com/w3schools-test/hello-world.git

git remote add origin URL specifies that you are adding a remote repository with the specified URL as an origin to your local Git repo.

Now we are going to push our master branch to the origin url, and set it as the default remote branch:

git push --set-upstream origin master

Now, go back into GitHub and see that the repository has been updated:

🡺 Edit Code in GitHub

Let's try to edit the README.md file in GitHub. Just click the edit button:

Add some changes to the code, and then commit the changes. For now, we will "Commit directly to the master branch".

: click on commit changes icon.

🡺 Pulling to Keep up-to-date with Changes

pull is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on.

When working as a team on a project, it is important that everyone stays up to date.

Any time you start working on a project, you should get the most recent changes to your local copy.

With Git, you can do that with pull.

git pull origin

pull is a combination of 2 different commands: fetch and merge.

: fetch gets all the change history of a tracked branch/repo.

So, on your local Git, fetch updates to see what has changed on GitHub:

git fetch origin

: we can also verify by showing the differences between our local master and origin/master:

git diff origin/master

🡺 **Git Merge**

: merge combines the current branch, with a specified branch.

git merge origin/master

🡺 **Git Push to GitHub | Push Changes to GitHub**

Let's try making some changes to our local git and pushing them to GitHub.

git commit -a -m "Updated index.html. Resized image"

git status

git push origin

🡺 **Create a New Branch on GitHub**

On GitHub, access your repository and click the "master" branch button.

There you can create a new Branch. Type in a descriptive name, and click Create branch:

The branch should now be created and active. You can confirm which branch you are working on by looking at the branch button. See that it now says "html-skeleton" instead of "main"?

Start working on an existing file in this branch. Click the "index.html" file and start editing:

After you have finished editing the file, you can click the "Preview changes" tab to see the changes you made highlighted:

If you are happy with the change, add a comment that explains what you did, and click Commit changes.

You now have a new branch on GitHub, updated with some changes!

🡺 **Git Pull Branch from GitHub**

Now continue working on our new branch in our local Git.

Lets pull from our GitHub repository again so that our code is up-to-date:

git pull

Now our main branch is up todate. And we can see that there is a new branch available on GitHub.

Do a quick status check:

git status

And confirm which branches we have, and where we are working at the moment:

git branch

\* master

So, we do not have the new branch on our local Git. But we know it is available on GitHub. So we can use the -a option to see all local and remote branches:

git branch –a

\* master

**Note:** branch -r is for remote branches only.

We see that the branch html-skeleton is available remotely, but not on our local git. Lets check it out:

git checkout html-skeleton

And check if it is all up to date:

git pull

Already up to date.

🡺 **Push a Branch to GitHub**

Let's try to create a new local branch, and push that to GitHub.

git checkout -b update-readme

Switched to a new branch 'update-readme'

And we make some changes to the README.md file. Just add a new line.

add to the Staging Environment:

So now we check the status of the current branch.

git status

make change/modify as needed then add to stage area, then commit them to the branch:

git add README.md

Check the status of the branch:

git status

We are happy with our changes. So we will commit them to the branch:

git commit -m "Updated readme for GitHub Branches"

Now push the branch from our local Git repository, to GitHub, where everyone can see the changes:

git push origin update-readme

🡺 **pull request**

A pull request is how you propose changes. You can ask some to review your changes or pull your contribution and merge it into their branch.

Go to project in Github > code > do some message > create pull request.

Pull request can also be merged.

🡺 **Git GitHub Flow**

The GitHub flow works like this:

: Create a new Branch

: Make changes and add Commits

: Open a Pull Request

: Review

: Deploy

: Merge

🡺 **Git GitHub Pages**

With GitHub pages, GitHub allows you to host a webpage from your repository. Let's try to use GitHub Pages to host our repository.

**Create a New Repository**

Start by signing in to GitHub. GitHub pages need a special name and setup to work, so we start by creating a new repository:

**Push Local Repository to GitHub Pages**

We add this new repository as a remote for our local repository, we are calling it gh-page (for GitHub Pages).

Copy the URL from here And add it as a new remote:

git remote add gh-page <https://github.com/w3schools-test/w3schools-test.github.io.git>

Make sure you are on the master branch, then push the master branch to the new remote:

git push gh-page master

🡺 **Fork a Repository**

A fork is a copy of a repository. This is useful when you want to contribute to someone else's project or start your own project based on theirs.

**Clone a Fork from GitHub**

Now we have our own fork, but only on GitHub. We also want a clone on our local Git to keep working on it.

A clone is a full copy of a repository, including all logging and versions of files.

Move back to the original repository, and click the green "Code" button to get the URL to clone:

Open your Git bash and clone the repository:

git clone <https://github.com/w3schools-test/w3schools-test.github.io.git>

Take a look in your file system, and you will see a new directory named after the cloned project:

**Note:** To specify a specific folder to clone to, add the name of the folder after the repository URL, like this: git clone https://github.com/w3schools-test/w3schools-test.github.io.git myfolder

Navigate to the new directory, and check the status:

cd w3schools-test.github.io

git status

And check the log to confirm that we have the full repository data:

git log

Now we have a full copy of the original repository.

🡺 **Configuring Remotes**

Basically, we have a full copy of a repository, whose origin we are not allowed to make changes to.

Let's see how the remotes of this Git is set up:

git remote –v

We see that origin is set up to the original "w3schools-test" repository, we also want to add our own fork.

First, we rename the original origin remote:

git remote rename origin upstream

git remote –v

Then fetch the URL of our own fork:

And add that as origin:

git remote add origin https://github.com/kaijim/w3schools-test.github.io.git

git remote –v

**Note:** According to Git naming conventions, it is recommended to name your own repository origin, and the one you forked for upstream

Now we have 2 remotes:

* origin - our own fork, where we have read and write access
* upstream - the original, where we have read-only access

Now we are going to make some changes to the code. In the next chapter, we will cover how we suggest those changes to the original repository.

🡺 **Push Changes to Our GitHub Fork**

We have made a lot of changes to our local Git.

Now we push them to our GitHub fork:

commit the changes:

git push origin

create a pull request:

🡺 Git Ignore

When sharing your code with others, there are often files or parts of your project, you do not want to share.

Examples

: log files

: temporary files

: hidden files

: personal files etc.

Git can specify which files or parts of your project should be ignored by Git using a .gitignore file.

Git will not track files and folders specified in .gitignore. However, the .gitignore file itself IS tracked by Git.

Create .gitignore

To create a .gitignore file, go to the root of your local Git, and create it:

touch .gitignore

We are just going to add two simple rules:

Ignore any files with the .log extension

Ignore everything in any directory named temp

Example

# ignore ALL .log files

\*.log

# ignore ALL files in ANY directory named temp

temp/

Now all .log files and anything in temp folders will be ignored by Git.

Note: In this case, we use a single .gitignore which applies to the entire repository.

It is also possible to have additional .gitignore files in subdirectories. These only apply to files or folders within that directory.

🡺 **Git Revert**

revert is the command we use when we want to take a previous commit and add it as a new commit, keeping the log intact.

Step 1: Find the previous commit:

Step 2: Use it to make a new commit:

Let's make a new commit, where we have "accidentally" deleted a file:

git commit -m "Just a regular update, definitely no accidents here..."

Now we have a part in our commit history we want to go back to. Let's try and do that with revert.

Git Revert Find Commit in Log

First thing, we need to find the point we want to return to. To do that, we need to go through the log.

To avoid the very long log list, we are going to use the --oneline option, which gives just one line per commit showing:

The first seven characters of the commit hash

the commit message

So let's find the point we want to revert:

git log –oneline

We want to revert to the previous commit: 52418f7 (HEAD -> master) Just a regular update, definitely no accidents here..., and we see that it is the latest commit.

Git Revert HEAD

We revert the latest commit using git revert HEAD (revert the latest change, and then commit), adding the option --no-edit to skip the commit message editor (getting the default revert message):

git revert HEAD --no-edit

Now let's check the log again:

git log –oneline

**Note:** To revert to earlier commits, use git revert HEAD~x (*x* being a number. 1 going back one more, 2 going back two more, etc.)

🡺 **Git Reset**

reset is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that commit.

Step 1: Find the previous commit:

Step 2: Move the repository back to that step:

After the previous chapter, we have a part in our commit history we could go back to. Let's try and do that with reset.

**Git Reset Find Commit in Log**

First thing, we need to find the point we want to return to. To do that, we need to go through the log.

To avoid the very long log list, we are going to use the --oneline option, which gives just one line per commit showing:

The first seven characters of the commit hash - this is what we need to refer to in our reset command.

the commit message

So let's find the point we want to reset to:

git log –oneline

We want to return to the commit: 9a9add8 (origin/master) Added .gitignore, the last one before we started to mess with things.

**Git Reset**

We reset our repository back to the specific commit using git reset commithash (commithash being the first 7 characters of the commit hash we found in the log):

git reset 9a9add8

Now let's check the log again:

git log –oneline

**Git Undo Reset**

Even though the commits are no longer showing up in the log, it is not removed from Git.

If you know the commit hash you can reset to it:

git reset e56ba1f

Now let's check the log again:

git log –oneline

🡺 **Git Amend**

Git commit –amend

commit --amend is used to modify the most recent commit.

It combines changes in the staging environment with the latest commit, and creates a new commit.

This new commit replaces the latest commit entirely.

**Git Amend Commit Message**

One of the simplest things you can do with --amend is to change a commit message.

Let's update the README.md and commit:

git commit -m "Adding plines to reddme"

Now let's check the log:

git log –oneline

Oh no! the commit message is full of spelling errors. Embarrassing. Let's amend that:

git commit --amend -m "Added lines to README.md"

And re-check the log:

git log –oneline

We see the previous commit is replaced with our amended one!

**Git Amend Files**

Adding files with --amend works the same way as above. Just add them to the staging environment before committing.

https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Git basic commands \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//git user name and password**

git config --global user.name "Sam Smith" (set global user name | also work as local username)

git config --global user.email sam@example.com (set global user email | also local user email)

git config --global user.name (check global user name)

git config --global user.email (check global user email)

git config user.name "Sam Smith" (set local username)

git config user.email sam@example.com (set user email)

git config user.name (check local user name)

git config user.email (check local user email)

git init (will initialize emply repo(.git) with a branch(master) in the specified folder.

git add file-path or [git add . (. means all files)] (will add files to stage area)

git commit -m "commit message" (will save the staged files into local repo)

git status (display the state of add + commit commands)

git branch branch-name (create new branch: will create a copy of full master branch)

git branch -m new-branch-name (rename branch) 1st checkout to a branch then use this command.

git branch (List all of the branches in your repository (local + remote))

git branch -a (List all remote branches.)

git branch -d branch-name (delete the branch)

git checkout branch-name (switch to branch-name)

git checkout –b branch-name (create & switch to branch-name)

git fetch branch-name (download content from remote to local repo)

git clone remote\_repo\_url (when first time you download new repo from remote to local git)

git clone remote\_repo\_url (download a copy of the repo into same/different directory/place)

git pull remote\_repo\_url (fetch & merege new commits from remote to existing local repo)

git push origin master

//create a variable for remote repo

git remote add <do variable name for repo> github-repo-url (make connection to remote repo)

git remote add origin github-repo-url (make connection initiated repo to remote repo)

//create a variable for remote repo with Git access token

git remote set-url origin https://USERNAME:TOKEN@github.com/USERNAME/REPO.git

git remote add origin/git clone https://<username>:<personal\_token>@gitlab.com/gitlab-org/gitlab.git

// check connected remote repo

git remote (u can see the connected github repo variable name)

git remote –v (u can see the connected github repo url)

//rename remote repo variable name

git remote rename oldName newName

git remote rename origin newName

//remove remote repo variable

git remote remove remoteRepoVariableName

git remote remove origin

//push repo to github

git push -u origin master (upload the initiated content to remote repo origin master branch)

//push repo to github with user name and password

git push https://username:password@myrepository.biz/file.git master OR

git push –u https://username:password@myrepository.biz/file.git - -all

git merge branch-name (will merge unactive specified branch to active branch.

git fetch origin (if u make mess in your branch then use this command

git reset --hard origin/master to get original code from master branch again)

\*\*\* Git Crendential manager \*\*\*

If git ask for user name and password then go to Git Credential Manager and install.

**Note:** after updating on a branch first we need to merge master branch to updated branch.

**Merge conflict example:** I created a new branch. I did some work on it. But in the mean time someone/owner deleted some code from master branch.

After work when I tried to merge master branch to my updated branch then it shows error conflict.

**What should I do now !** I need to communicate with the woner of master branch, if he says that we don’t need the code which is deleted.

Then I will delete that code from my branch and then stage, commit and merge. There will be no conflict any more.

\*\* push code to Github using SSH key \*\*

// generate SSH key

1. cmd > ssh-keygen -t ed25519 -C "your\_email@example.com" > enter > enter >

2. Login to GitHub and go to settings

3. Go to SSH and GPG keys setting

4. Add new SSH-Key

On this page you will see all of your SSH-Keys. To add new one, click the New SSH key button.

Add new ssh key

Fill the title and key. Remember to copy the public key on your local computer then paste it, Not the private key.

5. push code to Github

git remote add origin2 git@github.com:Arjun4testLead/E-Commerce-Project.git

git branch -M main

git push -u origin2 master

\*\* key location in this PC: C:\Users\jawad\.ssh

-----BEGIN OPENSSH PRIVATE KEY-----

b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW

QyNTUxOQAAACBSfZW6JTYtexXybXpPc8FuD+tsroeUocjhnqh7JBjpUAAAAJi1TGOltUxj

pQAAAAtzc2gtZWQyNTUxOQAAACBSfZW6JTYtexXybXpPc8FuD+tsroeUocjhnqh7JBjpUA

AAAEARm/PTvtY4Vk/7HSgDG92Y9cMRLBnv2dX8N9j8iBe9yVJ9lbolNi17FfJtek9zwW4P

62yuh5ShyOGeqHskGOlQAAAAFWFyanVuZGV2bmF0QHlhaG9vLmNvbQ==

-----END OPENSSH PRIVATE KEY-----

--- begin public key –

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIFJ9lbolNi17FfJtek9zwW4P62yuh5ShyOGeqHskGOlQ arjundevnat@yahoo.com

--- end public key --

\*\*\*\*\*\*\*\*\*\*\*\*\* commands from github repo \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

or create a new repository on the command line

echo "# Test-Repo" >> README.md

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/Jawad-Karim/Test-Repo.git

git push -u origin main

push an existing repository from the command line

git remote add origin https://github.com/Jawad-Karim/Test-Repo.git

git branch -M main

git push -u origin main

Rahul shetty Github: rahulshettyacademy (github.com)