DOCKER + GITHUB COMANDS

cd D:\repositories\developerFolio-master

Here are some of the most commonly used GitHub-related **Git command-line commands**, organized by category:

**🔧 Setup and Configuration**

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

**📁 Repository Operations**

git init # Initialize a local git repo

git clone <repo\_url> # Clone a remote repo

**📄 Working with Files**

git status # See current changes

git add <file> # Stage a specific file

git add . # Stage all changes

git commit -m "Message" # Commit staged changes

git rm <file> # Delete a file

**🌐 Remote Repositories**

git remote add origin <url> # Add a remote repository

git push -u origin main # Push changes to GitHub

git pull origin main # Pull latest changes

**🌿 Branching**

git branch # List branches

git branch <branch\_name> # Create a new branch

git checkout <branch\_name> # Switch to a branch

git checkout -b <new\_branch> # Create and switch to a branch

git merge <branch\_name> # Merge a branch into current

**🔄 Updating & Syncing**

git fetch # Get changes from remote but don’t merge

git pull # Fetch + merge

git push # Push commits to remote

**🕓 History and Logs**

git log # View commit history

git diff # Show file differences

Would you like a visual cheat sheet or a downloadable PDF version of these commands?

Great! Let's walk through how to verify your remote repository link and make a test commit to ensure everything is working:

**✅ Step 1: Verify Your Remote Repository**

In your terminal, run:

git remote -v

This will show something like:

origin https://github.com/yourusername/developerFolio.git (fetch)

origin https://github.com/yourusername/developerFolio.git (push)

Make sure the URL is correct and points to your GitHub repo.

**✅ Step 2: Make a Test Commit**

You can create a temporary file to test pushing:

echo "Test push" > testfile.txt

git add testfile.txt

git commit -m "Test commit to verify push"

git push

This will create and push a simple file named testfile.txt.

**✅ Step 3: Clean Up (Optional)**

Once you're sure everything works, you can remove the test file:

git rm testfile.txt

git commit -m "Remove test file"

git push

Let me know if you want help checking or editing your remote URL.

Perfect — your remote is correctly set to:

https://github.com/lynx479/developerFolio-master

This means your local repo is linked to your GitHub repository, and you're ready to push and pull as needed.

Now let's finish testing your push:

In your terminal, run the following commands:

echo "Test push" > testfile.txt

git add testfile.txt

git commit -m "Test commit to verify push"

git push

Once you do that, Git will push the commit to your GitHub repository. You can then check your repo on GitHub to confirm that testfile.txt was added.

Want me to guide you through removing the test file afterward too?

Github commands

Here are the **most important Git commands** to **push** and **pull** changes between your PC and GitHub:

**1. Pull Changes from GitHub to Your Local Repo:**

This command fetches changes from the remote repository and merges them into your local branch.

git pull origin master

Replace main with the name of your branch if it's different (e.g., master, dev).

**2. Push Your Local Changes to GitHub:**

This command pushes your local commits to the remote repository.

git push origin main

Replace main with your branch name if necessary.

**3. Add Changes to Git Staging Area (before commit):**

Before committing, use this command to add files to the staging area.

git add .

The . adds all modified files. You can also specify individual files.

**4. Commit Your Changes (Save Changes Locally):**

After staging your changes, commit them with a message.

git commit -m "Your commit message here"

**Summary:**

* **Pull**: git pull origin main
* **Push**: git push origin main
* **Add**: git add .
* **Commit**: git commit -m "message"

These are the essential commands for syncing your local repo with GitHub!

**🧹 Optional: Also clear any saved GitHub credentials**

If Git was remembering your login (e.g., personal access token), you should also remove the credential helper:

bash

نسختحرير

git config --global --unset credential.helper

Then clear saved credentials from your system:

* **Windows:** Open **Credential Manager** → **Windows Credentials** → Delete entries like git:https://github.com

Great observation — that behavior is expected if your browser session was already logged into GitHub. Here's why that happened:

🔄 What's Going On?

When Git asks you to authenticate via browser, it uses GitHub's OAuth or Personal Access Token (PAT) flow. If you're already signed into GitHub in your browser, GitHub automatically approves the request without asking for your email or password again.

🧼 How to Force a Full Logout

To fully log out and ensure Git asks for credentials next time, do this:

✅ Step 1: Sign Out of GitHub in Your Browser

1. Go to <https://github.com/logout>
2. Confirm sign-out

✅ Step 2: Clear Cached GitHub Credentials (Optional but recommended)

If you haven’t already, clear stored credentials in your OS:

* Windows:
  + Open Credential Manager
  + Go to Windows Credentials
  + Remove any entries like git:https://github.com
* macOS:
  + Open Keychain Access
  + Search for github.com
  + Delete any matching credentials

✅ Step 3: Remove Credential Helper (if needed)

If Git is still not prompting, unset the credential helper:

git config --global --unset credential.helper

After doing all this, when you run git pull or git push, it should:

1. Open a browser window
2. Ask for GitHub login (email + password or 2FA)

Would you prefer to avoid this browser login flow and switch to using an SSH key instead?

Conflict avoiding…

When you go back to your home PC, you'll need to reconcile the changes you've made in both locations. Here’s a step-by-step guide on what to do:

1. **Check the status of your local repository:**

On your home PC, open your terminal or Git client and navigate to the repository folder. Run the following command to check the status:

git status

This will show you if there are any uncommitted changes, including the file you added yesterday but forgot to commit.

2. **Add the missing file (if necessary):**

If the file you added yesterday is still uncommitted, add it to the staging area:

git add <filename>

If you have multiple files, you can use:

git add .

3. **Commit the file:**

Commit the file that you added yesterday to your home PC's local repository:

git commit -m "Add file from home PC"

4. **Pull the latest changes from GitHub:**

Before pushing your local changes, it's important to pull the latest changes from the GitHub repository (since you've already committed changes from your office PC). This will help you avoid any potential merge conflicts.

Run:

git pull origin main

If your default branch is main or master, make sure to replace main with the correct branch name. This will bring in the changes you made from the office PC.

If there are any merge conflicts between your changes (the file you added yesterday) and the changes from your office PC, Git will ask you to resolve them manually. Open the conflicting files, fix the issues, and then mark the conflict as resolved:

git add <conflicting\_files>

After resolving any conflicts, proceed to commit the merge:

git commit -m "Merge changes from office PC"

5. **Push your changes to GitHub:**

Once you've committed all your changes, push them to GitHub:

git push origin main

Summary of Steps:

1. git status to check for uncommitted changes.
2. git add to stage your changes.
3. git commit to commit the changes.
4. git pull to bring in the changes from GitHub (office PC's commits).
5. Resolve any conflicts if they arise.
6. git push to push all your changes back to GitHub.

This process ensures that the changes from both your home and office PCs are properly integrated and pushed to GitHub.

You're seeing Git’s default message prompt for a **merge commit**, likely because you pulled changes from the remote (origin/master) that didn’t match your local branch exactly.

Here's how to handle it:

**✅ Option 1: Complete the Merge (Recommended)**

1. Type a commit message in that prompt (or accept the default one).
2. Save and close the editor (in most cases, this is Vim).

**If you're in Vim (the default):**

* Press i to enter **insert mode** if you want to edit the message.
* Press Esc, then type :wq and press Enter to save and quit.

The merge commit will complete.

**✅ Option 2: Abort the Merge**

If you didn’t want to merge, you can cancel it:

git merge --abort

**Why This Happens:**

This message appears when:

* You run git pull, and your local branch and the remote branch have diverged.
* Git tries to merge them and opens your editor to let you confirm or edit the merge commit message.

Would you like help setting Git to use a different editor (like VS Code or nano) for easier commit messages?

Great! Since you're already building and running Docker images, here’s a **comprehensive list of essential Docker commands**, organized by category:

**🔧 Docker Setup & Version**

docker version # Show Docker version

docker info # System-wide info

**🏗️ Build Images**

docker build -t <name>:<tag> . # Build image from Dockerfile

docker build -f <Dockerfile> -t <name> . # Specify custom Dockerfile

**📦 Images**

docker images # List all local images

docker rmi <image\_id\_or\_name> # Remove image

docker tag <image> <repo>:<tag> # Tag image

docker save -o <file>.tar <image> # Save image to file

docker load -i <file>.tar # Load image from file

**🚀 Run Containers**

docker run -t -p 3000:3000 <image> # Run a container with port mapping

docker run -it <image> /bin/bash # Run with interactive shell

docker run -d <image> # Run in detached mode

docker run --name <name> <image> # Run with specific container name

**📋 Containers**

docker ps # List running containers

docker ps -a # List all containers

docker stop <container\_id\_or\_name> # Stop running container

docker start <container\_id\_or\_name> # Start stopped container

docker restart <container\_id\_or\_name> # Restart container

docker rm <container\_id\_or\_name> # Remove container

docker logs <container\_id\_or\_name> # View logs

docker exec -it <container\_id> bash # Execute command inside container

docker exec -it gracious\_chebyshev /bin/sh

**🗂️ Volumes**

docker volume create <volume\_name> # Create volume

docker volume ls # List volumes

docker volume rm <volume\_name> # Remove volume

docker run -v <volume\_name>:/path/in/container <image>

**🧹 System Cleanup**

docker system prune # Remove unused data (images, containers, etc.)

docker image prune # Remove unused images

docker container prune # Remove stopped containers

**📤 Pushing to Docker Hub (Optional)**

docker login # Login to Docker Hub

docker tag <image> <username>/<repo>:tag

docker push <username>/<repo>:tag

Let me know if you want a Docker Compose version too or a visual cheat sheet!

Great! If you have a **Docker Compose YAML file** (usually named docker-compose.yml), you can manage multi-container applications or even single containers more efficiently using **Docker Compose**.

Here are the **essential Docker Compose commands** you'll want to know:

**▶️ Start Services**

docker-compose up # Start all services defined in the YAML

docker-compose up -d # Run in detached (background) mode

**⛔ Stop Services**

docker-compose down # Stop and remove containers, networks, etc.

**🔄 Rebuild Containers**

docker-compose up --build # Build and start fresh containers

**🧭 Other Useful Commands**

docker-compose ps # List containers managed by Compose

docker-compose logs # Show logs for all services

docker-compose logs <service> # Logs for specific service

docker-compose exec <service> bash # Open bash terminal inside a service

Example:

docker-compose exec web bash # If your service is named "web"

If you’d like, you can paste your docker-compose.yml here and I can help explain or improve it. Would you like help reviewing it?

You're running a container named:

developerfolio-master-react-app-1

To **start this container again** in the future (after a reboot or if it's been stopped), you simply use:

docker start developerfolio-master-react-app-1

**🧭 Other Related Commands:**

* **Stop the container:**
* docker stop developerfolio-master-react-app-1
* **Restart the container:**
* docker restart developerfolio-master-react-app-1
* **Access the container's terminal:**
* docker exec -it developerfolio-master-react-app-1 bash

If bash doesn’t work:

docker exec -it developerfolio-master-react-app-1 sh

Would you like to add a **restart policy** to make it start automatically after system restarts?