 Darpan Patel

Connect a GitHub Repo with AWS

```
[ec2-user@ip-172-31-83-140 nextwork-web-project]$ sudo dnf update -y
sudo dnf install git -y
Last metadata expiration check: 0:20:25 ago on Fri May  2 16:36:51 2025.
Dependencies resolved.
Nothing to do.
Complete!
Last metadata expiration check: 0:20:26 ago on Fri May  2 16:36:51 2025.
Dependencies resolved.
================================================
 Package      Arch   Version
                     Repository   Size
================================================
Installing:
 git          x86_64 2.47.1-1.amzn2023.0.2
                     amazonlinux  54 k
Installing dependencies:
 git-core     x86_64 2.47.1-1.amzn2023.0.2
                     amazonlinux 4.7 M
 git-core-doc noarch 2.47.1-1.amzn2023.0.2
                     amazonlinux 2.8 M
 perl-Error   noarch 1:0.17029-5.amzn2023.0.2
                     amazonlinux  41 k
 perl-File-Find
              noarch 1.37-477.amzn2023.0.6
                     amazonlinux  26 k
 perl-Git     noarch 2.47.1-1.amzn2023.0.2
                     amazonlinux  42 k
 perl-TermReadKey
              x86_64 2.38-9.amzn2023.0.2
                     amazonlinux  36 k
 perl-lib     x86_64 0.65-477.amzn2023.0.6
                     amazonlinux  15 k

Transaction Summary
================================================
Install  8 Packages
```

 Darpan Patel

Introducing Today's Project!

Today, I'm setting up version control for my web application project using Git and GitHub. The goal is to connect my EC2-hosted project to a GitHub repository so I can track all changes, collaborate more easily, and prepare the groundwork for future CI/CD automation. I'll be initializing Git in my project folder, linking it to a new remote repo on GitHub, and committing and pushing my existing code. This is a key step in managing my code more professionally and building out my DevOps skills.

Key tools and concepts

Services I used were Amazon EC2, GitHub, Visual Studio Code with Remote-SSH, and Markdown for documentation. Key concepts I learnt include: Remote development on EC2: I learned how to connect and code directly on a cloud-based virtual machine using VS Code's Remote-SSH feature, which simulates real-world DevOps workflows. Version control with Git and GitHub: I practiced staging, committing, and pushing code changes, and understood how GitHub repositories help with collaboration and code management. Creating and formatting README files: Using Markdown, I wrote professional project documentation, which is essential for open-source contributions and job readiness. CI/CD pipeline planning with AWS: Although not yet fully implemented, I prepared for services like CodeArtifact, CodeBuild, CodeDeploy, and CodePipeline—gaining a foundational understanding of how AWS automates software delivery.

Project reflection

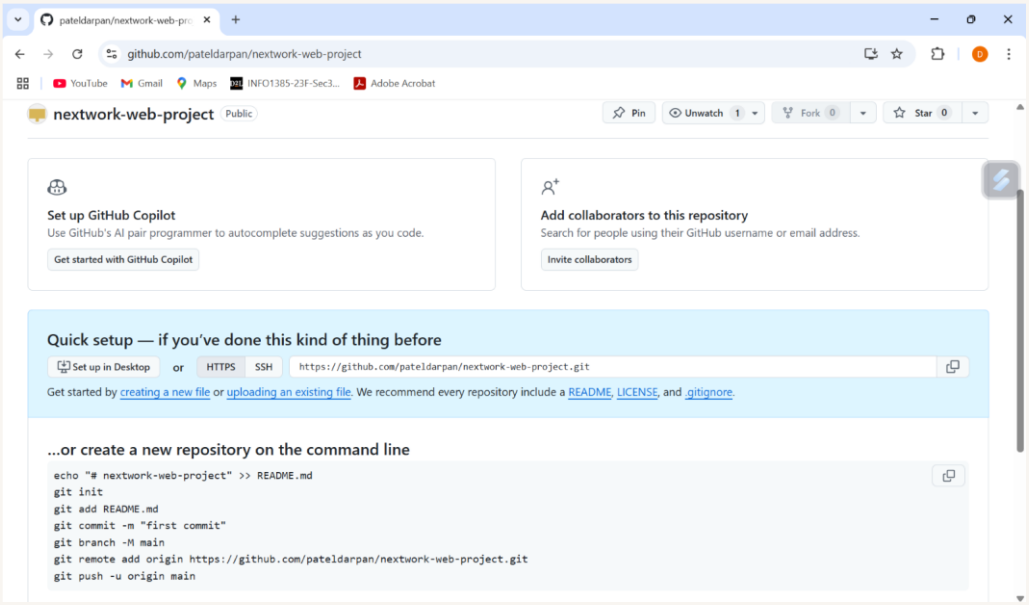
This project took me approximately 2 hours to complete. The most challenging part was configuring the Remote-SSH connection to the EC2 instance and making sure all Git commands were correctly reflected in my GitHub repository.

 Darpan Patel

Git and GitHub

Git is a version control system that helps track changes to your code over time, allowing you to go back to earlier versions if needed and collaborate with others by managing different versions of the same files. It’s like a time machine and collaboration tool for developers. I installed Git using the commands: sudo dnf update -y sudo dnf install git -y These commands updated my EC2 instance’s software and then installed Git, ensuring compatibility and the latest version.

GitHub is a cloud-based platform that hosts and manages code repositories using Git. It allows developers to store, track, and collaborate on code. GitHub provides features like version control, branching, and pull requests, making it easier for teams to collaborate and manage code efficiently. I'm using GitHub in this project to **store my code online**, track changes with version control, and **collaborate with others**. It provides an organized way to manage my web application's code and ensures I can access it from anywhere while keeping it secure and backed up.



 Darpan Patel

My local repository

A Git repository is a storage space where your project's files and the history of their changes are stored. It tracks all the versions of your code, allowing you to keep a record of every change, revert to previous versions if needed, and collaborate with others. The repository can be local (on your own computer) or remote (on platforms like GitHub), making it easy to manage and share code across different environments.

`git init` is a command that initializes a new Git repository in the current directory. It sets up the necessary files and structure to start tracking changes in that folder. When you run `git init`, Git creates a hidden `.git` directory that stores the version history and configuration for the repository. I ran `git init` in my project folder to tell Git that I wanted to start tracking the changes made to my web app code. After running this command, the folder is now a local Git repository, and Git will track any modifications I make to the files inside it.

After running `git init`, the response from the terminal was a message indicating that a new Git repository had been initialized, along with a reminder about the default branch name, which is typically `master` or `main`. A branch in Git is essentially a separate version of your project where you can make changes without affecting the main codebase (typically called the `master` or `main` branch). You can think of branches as parallel universes of your project that allow you to work on features, fixes, or experiments independently. Once your changes are complete and tested, you can merge the branch back into the main branch. This keeps the project organized and makes it easier to manage different features or updates without interfering with the stable version of the code.

 Darpan Patel

```
[ec2-user@ip-172-31-83-140 nextwork-web-project]$ pwd
/home/ec2-user/nextwork-web-project
[ec2-user@ip-172-31-83-140 nextwork-web-project]$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/ec2-user/nextwork-web-project/.git/
[ec2-user@ip-172-31-83-140 nextwork-web-project]$
```

To push local changes to GitHub, I ran three commands

git add

The first command I ran was `git add .`. A staging area is like a preparation zone where you gather all the file changes you want to include in your next saved version (called a commit). By running `git add .`, I told Git to take all the changes in my project folder and place them in the staging area, so I could review and save them in the next step.

git commit

The second command I ran was `git commit -m "Updated index.jsp with new content"`. Using `-m` means I added a message describing what changes I made in this version of the project. This helps keep track of the project's history, so I (or my teammates) can understand what each saved version (commit) includes without needing to look at all the code changes manually.

 Darpan Patel

git push

The third command I ran was `git push -u origin master`. Using `-u` means I set an upstream tracking reference, which tells Git to remember the remote repository (called `origin`) and the branch (`master`) I'm pushing to. This way, in the future, I can simply run `git push` without having to specify the remote and branch every time.

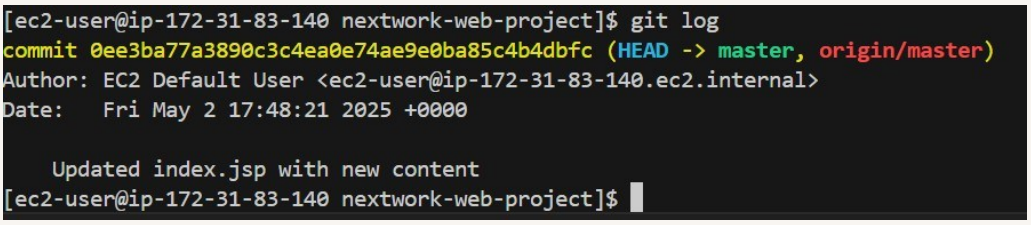
Authentication

When I commit changes to GitHub, Git asks for my credentials because it needs to verify that I have permission to make changes to the remote repository. This authentication step ensures that only authorized users can push updates, preventing unauthorized access. Git used to accept GitHub usernames and passwords for this, but now it requires a Personal Access Token (PAT) instead of a password when using HTTPS, to improve security and prevent password-related vulnerabilities.

Local Git identity

Git needs my name and email because it tracks who makes each change in the project's version history. This information is important for maintaining a record of contributions, especially when collaborating with others. It helps identify which person made specific changes or committed certain updates, making it easier to manage and review code changes over time. If I don't set this information manually, Git will default to using the system's username, which may not accurately reflect the actual contributor's identity.

Running `git log` showed me that it displayed the commit history for my project. It listed all the commits I’ve made, along with details such as the commit hash, the author's name and email, the date of the commit, and the commit message. This is useful for tracking changes over time and reviewing the project's development history.



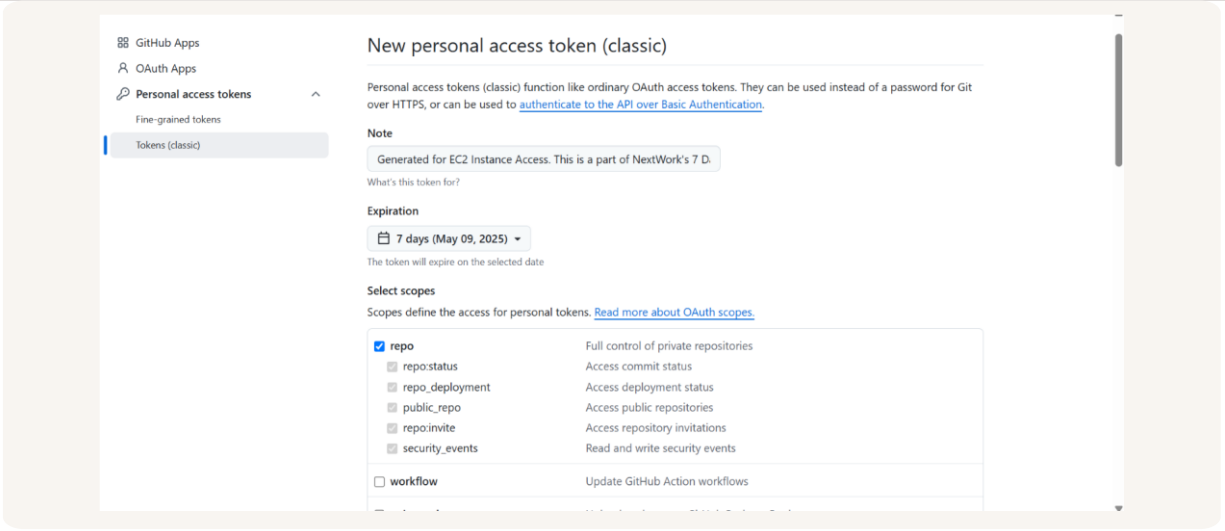
GitHub tokens

GitHub authentication failed when I entered my password because GitHub no longer allows password authentication for Git operations over HTTPS. Instead, it now requires a Personal Access Token (PAT), which is a more secure way to authenticate. This change helps protect accounts from being compromised, as tokens can be scoped with limited permissions and revoked at any time, unlike traditional passwords.

A GitHub token is a unique, randomly generated string of characters used for authenticating and accessing GitHub repositories securely. It's a more secure alternative to using a password for operations like pushing or pulling code over HTTPS. I'm using one in this project because GitHub no longer allows password authentication for these actions due to security concerns. A token provides a safer method of verifying my identity when interacting with my GitHub repository from my local machine or EC2 instance.

I could set up a GitHub token by navigating to my GitHub account's settings, selecting Developer settings, and then choosing Personal access tokens. From there, I clicked Generate new token, selected the necessary scopes (like repo for full access to private repositories), and then clicked Generate token. After the token was generated, I

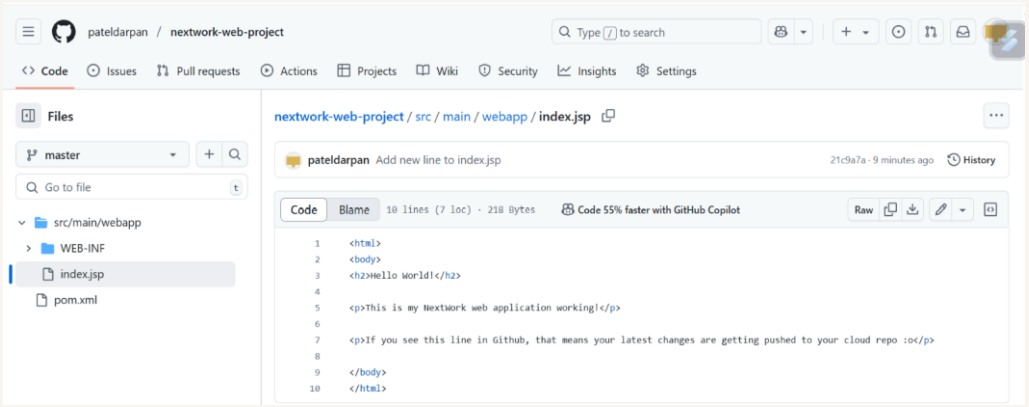
copied it and stored it safely, as I wouldn't be able to view it again once the tab was closed.



Making changes again

I wanted to see Git working in action, so I updated the `index.jsp` file in the `nextworkweb-project` by adding a new line of HTML. I couldn't see the changes in my GitHub repo initially because saving the file in VSCode only updates the local version. To reflect the changes in GitHub, I had to stage the file with `git add .`, commit it using `git commit -m "Add new line to index.jsp"`, and push it with `git push`.

I finally saw the changes in my GitHub repo after staging the modified file with `git add .`, committing the change using `git commit -m "Add new line to index.jsp"`, and then pushing it to the remote repository with `git push`. Once those steps were complete, I refreshed my GitHub repo page, and the updates appeared.



Setting up a README file

As a finishing touch to my GitHub repository, I added a README file, which is a markdown document that explains what the project is about, how it works, and how others can set it up or contribute. It acts as the first point of reference for anyone visiting the repository. I added a README file by running the command `touch README.md` in my VS Code terminal, which created a new blank file where I can now document the details of my project.

My README is written in Markdown because it's a lightweight markup language that's perfect for formatting plain text into well-structured documents—especially on platforms like GitHub. Special characters can help you format text in Markdown, such as: * `#` for headers (e.g., `#` = H1, `##` = H2) * `**bold**` for **bold** text * `*italic*` for *italic* text * `-` or `*` for bullet points * Backticks (`` ` ``) for inline code * Triple backticks (` ``` `) for code blocks * `[text](URL)` for hyperlinks Using Markdown in a README helps clearly communicate what the project is, how to use it, and how

to contribute—making it easy for others (and future you) to understand and navigate your work. Would you like a quick reference Markdown cheat sheet to keep handy?

placeholder

My README file has 6 sections that outline the purpose, tools, setup, and contact information for my project.

> Here's a breakdown: Introduction Describes the goal of the project: deploying a Java web app using AWS CI/CD. Includes personal motivations and how the project supports my career in cybersecurity and IT infrastructure. Technologies Lists tools used: AWS EC2, VS Code, GitHub, and AWS CI/CD services. Highlights challenges faced (like GitHub authentication and remote EC2 access) and how I solved them. Setup Provides step-by-step instructions to clone the repository and install dependencies using Maven. Includes troubleshooting tips for common setup issues. Contact Shares my name and email address. Includes a link to my LinkedIn profile and a placeholder for a professional photo. Conclusion Expresses thanks to readers and credits NextWork for guidance. Encourages others to start the DevOps learning project. Table of Contents Offers quick navigation to each section