SOFTWARE DEVELOPMENT PRACTICES

STAFF GRADED ASSIGNMENT PART-2

LEKHSHREE PAUNIKAR 2022EBCS070

**Assignment Overview:**

* This assignment forms an essential component of your final grades for the course. The assignment will be evaluated and graded by your instructor.
* This assignment is graded out of a total of twenty marks and will constitute 10% of your final module grade. The scoring for each question is provided at the beginning of the question.
* The scope of this assignment spans material covered in pre-recorded lectures, group discussions, and live sessions.
* The tasks in this assignment must be carried out on your **local machine**. This includes setting up your local Git repository, running Docker commands locally, and pushing your repository/image to GitHub or Docker Hub from your local environment.

**Total time estimated for assignment completion:** 3-4 hours.

**Assignment deliverable:**

* **Screenshots and Explanations**: Compile all screenshots (Git commands, Docker build logs, Docker Hub repository, running container, etc.) into a **Single Word or PDF** document.
* Provide short captions or explanations under each screenshot.
* **Source Files**: Include your **HTML pages** for the web application.
* Include the **Dockerfile** for your Docker image.
* (Optional) Include any small Python scripts or other application files if relevant.
* (Optional) **Links**: Provide a **link** to your GitHub repository. Provide a **link** to your Docker Hub repository (where the image is hosted).
* Create a folder containing your source files (or exported diagrams) and the Word/PDF document. Compress the folder into a .zip file.
* Submit the .zip file to the appropriate submission tab in your Coursera assignment.
* Please note this is **NOT a GROUP** exercise. You are required to work on this assignment on your own.
* Do not share solutions or copy verbatim from classmates.

**No. of attempts**: 2

**Assignment Task:**

This assignment introduces you to two foundational DevOps practices: Version Control using Git and GitHub, and Containerization using Docker. By completing these tasks, you will gain practical experience in setting up a project under version control, collaborating through GitHub, and building and deploying a Docker container image. After completing the steps on your local system, capture screenshots and gather any source files (HTML pages, Dockerfile, etc.) to include in your submission. The primary development and version control setup should occur on your local machine.

**Question 1: (10 Marks)**

1. Setup a project repository in your local system, and initialize the repo using git. The project can include a simple Web application (2-3 HTML web pages). [2 Marks]
2. Demonstrate the following git commands on the repo: init, status, add, log, commit, branch and switch. [6 Marks]
3. Create a GitHub account and a GitHub repository. Demonstrate integration between git and GitHub, by pushing the local repo to GitHub. [2 Marks]

**Question 2: (10 Marks)**

1. Create a custom docker image for any application of your choice (Ex: basic python 'hello world' application). [2 Marks]
2. Push the image into Docker Hub. [2 Marks]
3. Mention the detailed steps in creating the docker image and commands to push the image to Docker Hub. [3 Marks]
4. Demonstrate the running of the container in the local environment. [3 Marks]

**Grading Overview**

The assignment will be graded with the following distribution of points:

● Question 1a: 10% of total points

● Question 1b: 30% of total points

● Question 1c: 10% of total points

● Question 2a: 10% of total points

● Question 2b: 10% of total points

● Question 2c: 15% of total points

● Question 2d: 15% of total points

Your submission will be graded primarily based on the following:

**1a. Local Git Repository**. [2 Marks]

* A simple web application (2–3 HTML pages).
* Initialized as a Git repository (git init).

**1b. Demonstration of Git Commands**. [6 Marks]

* Show usage of the following commands:
  + git init
  + git status
  + git add
  + git commit
  + git log
  + git branch
  + git switch
* **Deliverables**:
  + Screenshots or terminal logs of each command in action.
  + Brief explanations of what each command does and the outcome (e.g., “Here’s the screenshot of git commit showing the commit message and changes”).

**1c. GitHub Integration**. [2 Marks]

* Create a GitHub account (if you don’t already have one).
* Create a GitHub repository and **push** your local repo to GitHub.
* **Deliverables**:
  + A link to the **public** (or shared) GitHub repository.
  + Screenshots of the successful push process and the repository as viewed on GitHub.

**2a. Custom Docker Image**. [2 Marks]

* A Dockerfile that builds a simple application (e.g., a Python “Hello World” script or any small web service).
* **Deliverables**:
  + Your Dockerfile content.
  + A short description of how your application works.

**2b.** **Docker Image Creation Steps**. [2 Marks]

* Document the commands you used to build the Docker image (e.g., docker build -t your-image-name .).
* **Deliverables**:
  + Terminal screenshots or logs showing the build process and successful image creation.

**2c. Push to Docker Hub**. [3 Marks]

* Create a Docker Hub account (if you don’t already have one).
* Push your newly built image to Docker Hub (e.g., docker push your-dockerhub-username/your-image-name).
* **Deliverables**:
  + Screenshot of the Docker Hub repository page showing your image.
  + Terminal output from pushing the image.

**2d. Container Deployment Demonstration**. [3 Marks]

* Show that you can run the container locally (e.g., docker run -p 8080:8080 your-dockerhub-username/your-image-name).
* **Deliverables**:
  + Screenshot or terminal log of the container running.
  + If it’s a web application, a screenshot of the application running in your browser is helpful.

**Screenshots and Explanations**: Compile all screenshots (Git commands, Docker build logs, Docker Hub repository, running container, etc.) into a **Single Word or PDF** document.

The given below simple Music Player webpage will be used as our reference project to perform the two foundational DevOps practices: Version Control using Git and GitHub, and Containerization using Docker.



**Description of the Music Player Webpage (HTML, CSS, JavaScript):**

1. **Navigation Bar:** Contains the "Music Player" logo and title, placed horizontally at the top of the page.
2. **Song List:** Displays a list of songs with cover images and song titles.
3. **Sticky Bottom Control Bar:** Contains control buttons (play/pause, next, previous) and a progress bar for the current song.

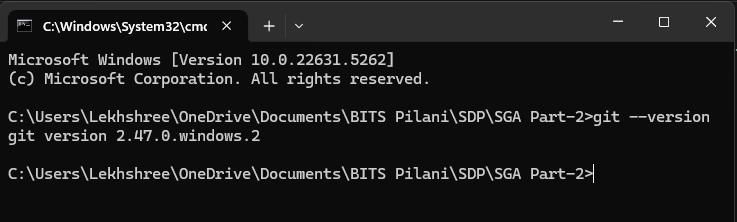
**Functionality**:

* **Play/Pause**: Toggles the audio playback state.
* **Progress Bar**: Shows the progress of the currently playing song and allows the user to seek through the track.
* **Next/Previous Buttons**: Navigate to the next or previous song in the playlist.

1. **GIF Animation:** A GIF is displayed when the song is playing, indicating the song's active state. Adds a visual cue that enhances the feedback of the user’s actions (play/pause).

Checking the version of Git installed in the local system.

git --version



Checking if the current folder is a Git Repository or not.

git status



Initializing a new Git Repository.

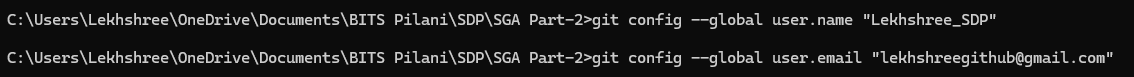
git init



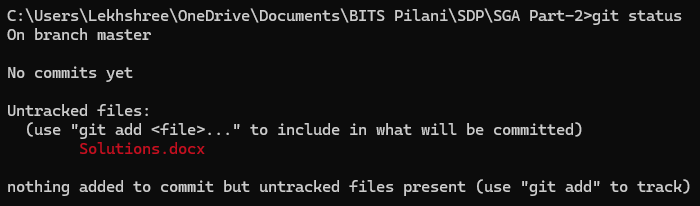
Setting up the Username and Email.

git config --global user.name "Lekhshree\_SDP"

git config --global user.email "lekhshreegithub@gmail.com"



Checking the Git status.



It shows that a file “Solutions.docx” is untracked file. This is an empty Word Document file meant to store the solutions of the SDP SGA-2 Assignment. Though it exists in the directory, but it has not been added to Git for version control yet.

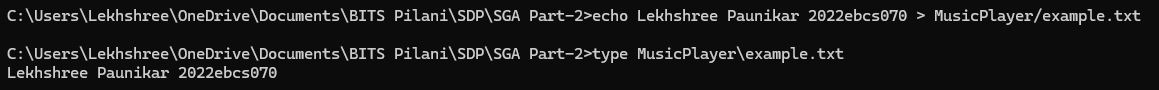
Creating a new project folder named “MusicPlayer” inside which we will write all of our code (HTML, CSS and Javascript).



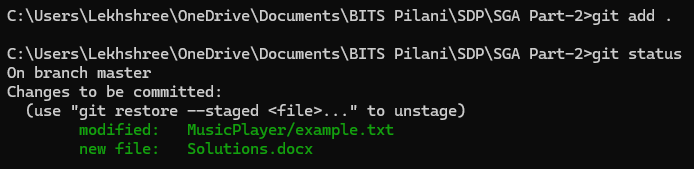
Adding the newly created folder to Git.



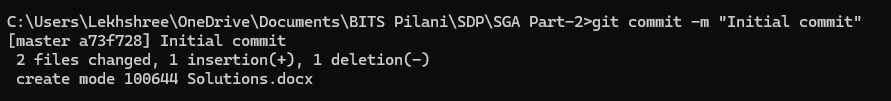
Since Git does not track empty directories, so we must make sure that there is at least one file inside the folder. So, we will make a text file named “example.txt” containing my student ID.



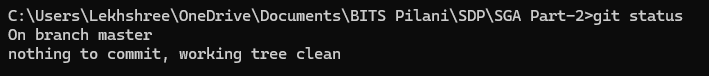
Adding all files in the current directory and subdirectories to the Git staging area.



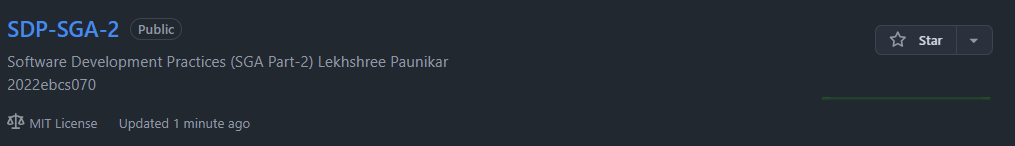
The green colored files are the ones that are Staged. After staging the files, we need to commit them. We will save the changes with a message "Initial commit."



Checking the committed changes again using git status



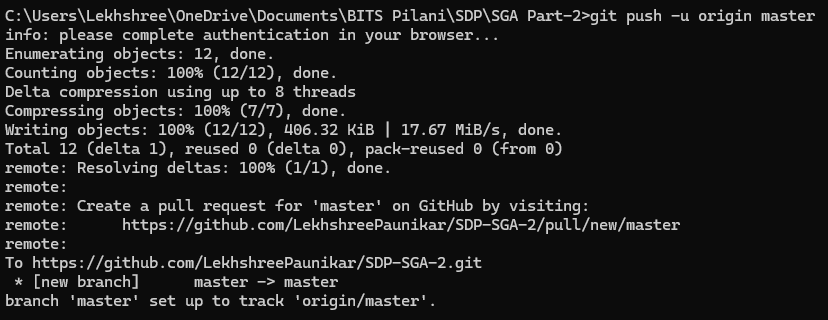
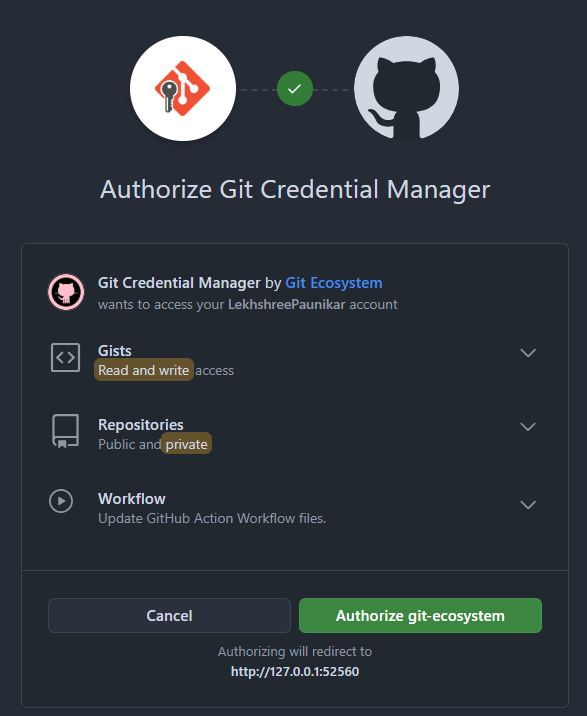
Now we have connected this local repository to a remote GitHub repository, for which we will create a new repository in our GitHub Account. I prefer not to initialize a README.md file and using the MIT license for my Repository.



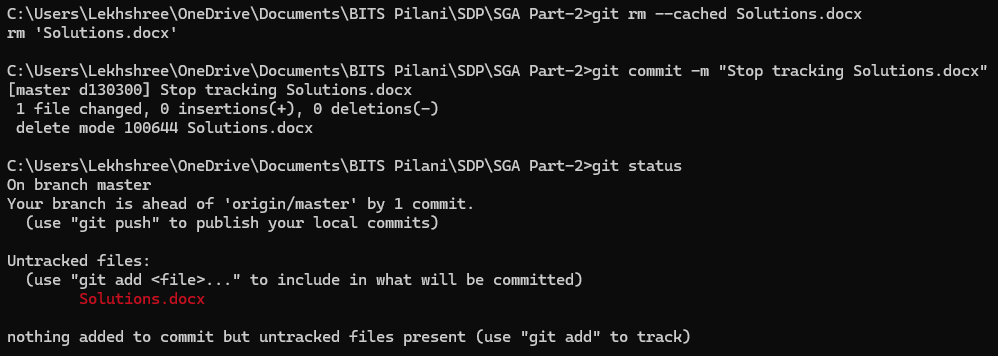
Adding the Remote Repository URL.



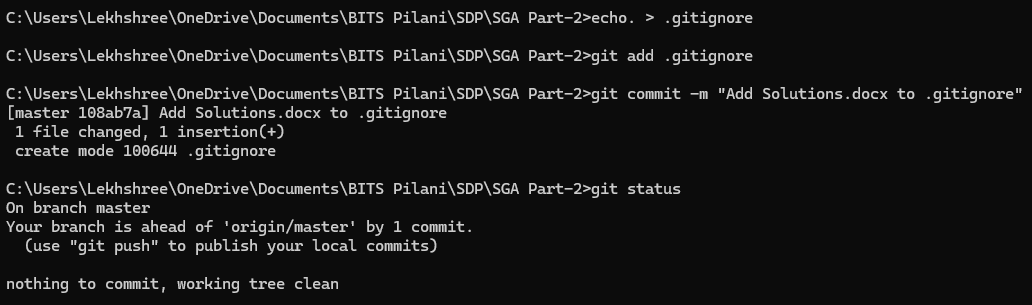
Pushing changes to GitHub after successfully linking the local repository to GitHub and Authenticating the account.



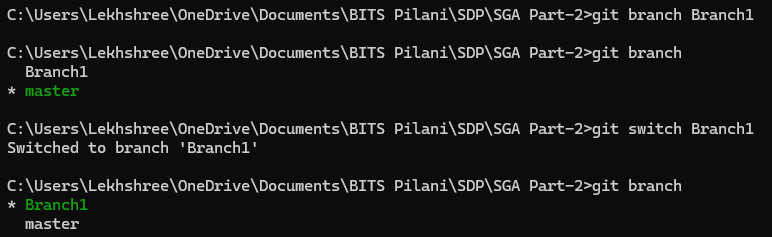
Since I am continuously making changes in the Solutions.docx file. I don’t want Git to ignore it and instead track other important files. For this I have to **remove the file from the Git index** while keeping it in my working directory as it is already once ‘added’ and ‘committed’.



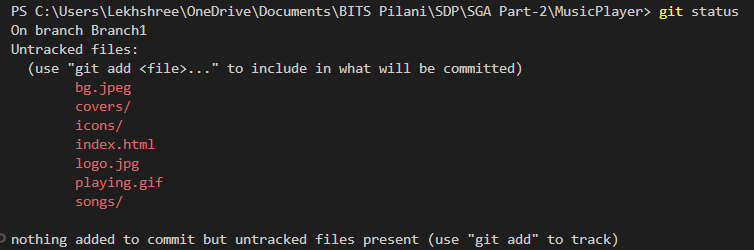
Now to make the process even more easier I will be using .gitignore file and add Solutions.docx file in it so that it no longer shows up as an untracked file.

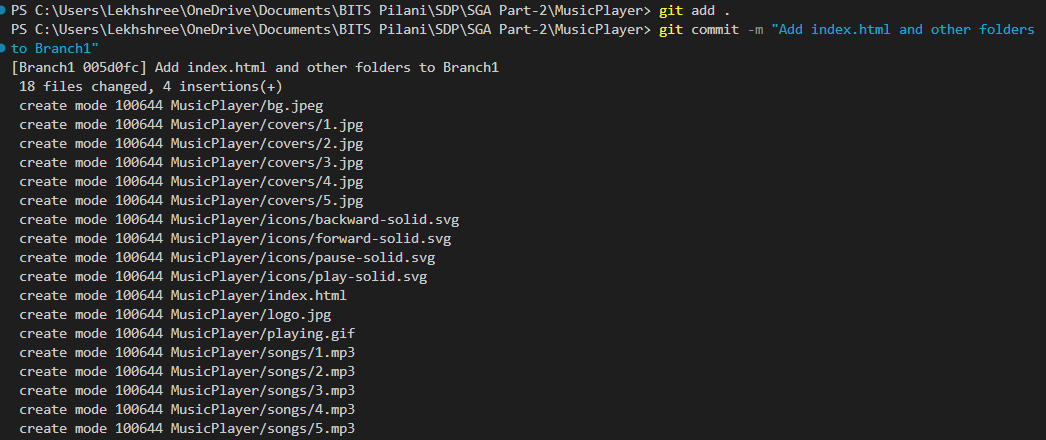


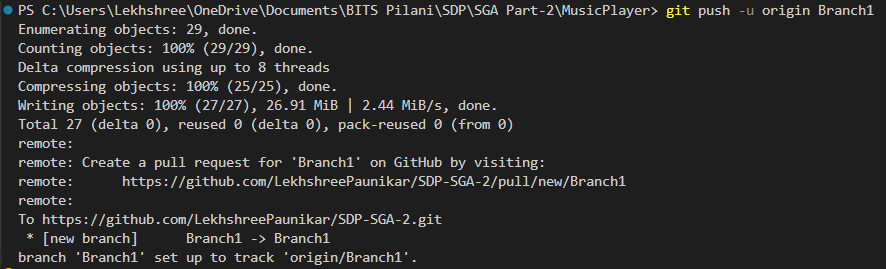
Now I will create a new branch named “Branch1” where I will write my basic HTML code along with other files and folders. After creating this new branch I need to switch from \*master >> \*Branch1 before making any further changes



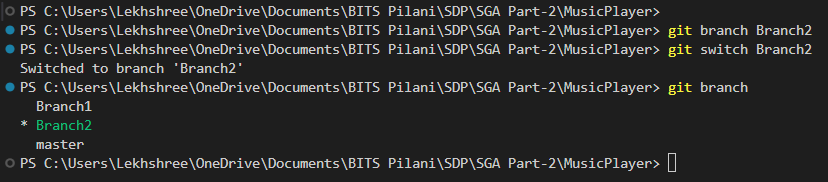
Now I will type my HTML code using VSCode and add files. Once all done I will add-commit-push the changes using the terminal present in VSCode itself.

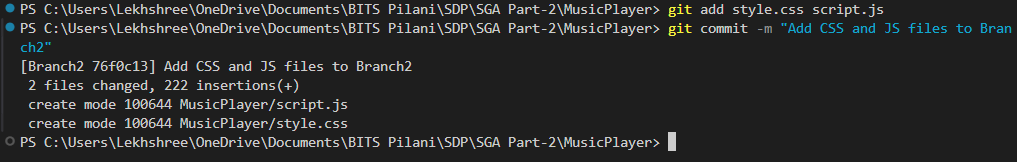


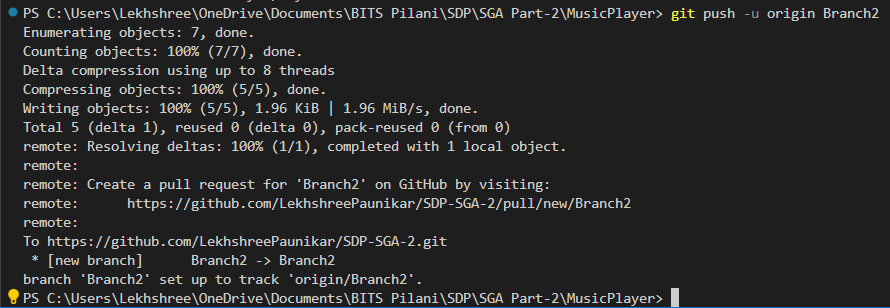




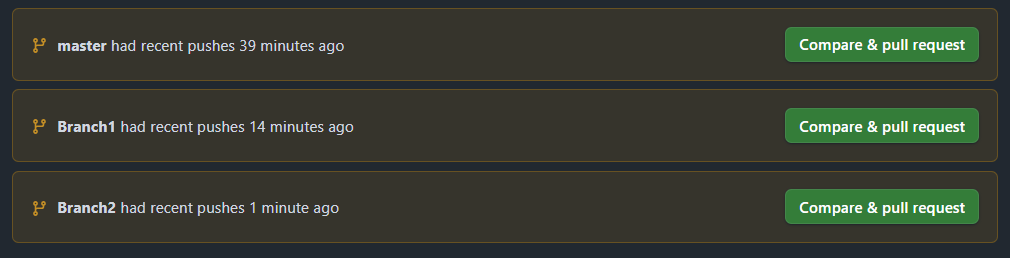
After successfully doing all the required changes in Branch1, I will now create a new branch “Branch2” where I will write my CSS and JavaScript code.

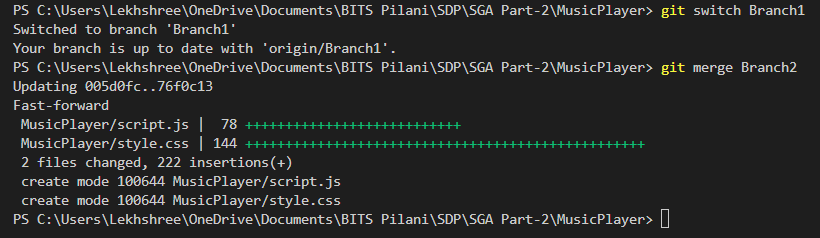






Now that both branches exist, it’s time to merge them. We will merge Branch2 into Branch1 for which we will first switch back to Branch1.





**Explanation**: A fast-forward merge means that there were no conflicts, and the changes from Branch2 were directly applied to Branch1 as if Branch1 had simply moved forward. **No conflicts** are shown in the output. This means that the merge was successful without requiring any manual intervention to resolve file conflicts.