Git and GitHub Essentials Summary

Fhaheem Tadamarry

Contents

1.1 Overview

Git is a distributed version control system that tracks file changes locally, while GitHub is a cloud-based platform that hosts Git repositories. Together, they allow developers to version, back up, and collaborate on projects seamlessly.

1.2 Key Concepts

- **Git**: Local version control — tracks and manages project changes.
- **GitHub**: Remote hosting service for Git repositories.
- **Repository (Repo)**: A folder containing all project files and Git history.
- **Commit**: A snapshot of your project's state.
- **Push / Pull**: Send or retrieve commits between local and remote copies.

> *Git = your local brain, GitHub = your online memory.*

2.1 Initialize Git in a Folder

To turn a regular folder into a Git repository:

```
cd /path/to/project
git init
```

This creates a hidden `.git/` folder which stores the entire version history.

2.2 Staging and Committing

Add files to be tracked and save your first snapshot:

```
git add .
git commit -m "initial commit"
```

Each commit acts as a restore point in your project's history.

3.1 Create and Link Remote Repository

```
git remote add origin https://github.com/fhaheem/EEL-6764.git
git branch -M main
git push -u origin main
```

This links your local project to the remote GitHub repository.

3.2 Checking Connection

```
git remote -v
```

Displays the URLs used for pushing and pulling.

4.1 Typical Daily Commands

- Pull the latest version from GitHub:

```
    git pull origin main

```

- Stage and commit your local changes:

```
    git add .
    git commit -m "update: homework revisions"

```

- Push changes back to GitHub:

```
    git push

```

4.2 Selective Add Options

- `git add .` → add all new + modified files.
- `git add -u` → add only modified/deleted files.
- `git add file.tex` → add a specific file.

5.1 Purpose

The `.gitignore` file tells Git which files to skip, keeping your repository clean.

5.2 LaTeX Example

```
# macOS
.DS_Store

# LaTeX build files
*.aux
*.bbl
*.blg
*.fdb_latexmk
*.fls
*.log
*.out
*.synctex.gz
*.toc
*.pdf
```

5.3 Removing Already-Tracked Junk

```
git rm -r --cached .
git add .
git commit -m "cleanup: remove ignored build files"
git push
```

6.1 Cloning a Repo from GitHub

To download (clone) a repo onto your computer:

```
git clone https://github.com/fhaheem/EEL-6764.git
cd EEL-6764
```

6.2 Pulling Updates

To update your local copy with the latest online changes:

```
git pull origin main
```

If you made local edits first:

```
git add .
git commit -m "save local work"
git pull origin main
```

7.1 Remove Git Tracking Locally

If you want to keep your files but stop tracking:

```
rm -rf .git
```

7.2 Disconnect from GitHub but Keep Git

```
git remote remove origin
```

7.3 Delete the Entire Folder

```
rm -rf foldername
```

8.1 Daily Workflow Recap

```
# Start your day
git pull origin main

# After editing files
git add .
git commit -m "updated homework and README"

# Upload to GitHub
git push
```

8.2 Key Tips

- Always pull before you start editing.
- Write clear commit messages (what + why).
- Use `.gitignore` to avoid committing junk.
- Never delete your GitHub repo unless intentional.
- Git only uploads differences — not the full project.

> Once you master `git add`, `git commit`, `git push`, and `git pull`, you've learned 90% of daily Git usage.

9.1 Why Branches?

Branches let you create a safe, parallel line of development away from `main`. You can experiment, write HW solutions, or refactor without breaking the stable version. When finished, merge back into `main`.

> Work on `main` for reading and releases; create short-lived feature branches for changes.

9.2 Create a Branch from `main`

```
# Make sure main is up to date
git checkout main
git pull origin main

# Create and switch to a new branch (e.g., HW5 work)
git switch -c hw5-development
# (older syntax) git checkout -b hw5-development
```

9.3 Work, Commit, and Push Your Branch

```
# Edit files locally, then save your work
git add .
git commit -m "feat(hw5): initial draft of solutions"

# Publish the branch to GitHub (first push sets upstream)
git push -u origin hw5-development
```

9.4 Keep Your Branch Fresh

```
# Bring latest main into your branch (rebases your commits on top)
git checkout hw5-development
git fetch origin
git rebase origin/main   # or: git merge origin/main
```

9.5 Merge Back to `main` (CLI)

```
# Switch to main and ensure it's current
git checkout main
git pull origin main

# Merge the branch and push the result
git merge hw5-development
git push
```

9.6 Merge Back to `main` (GitHub UI)

Open a Pull Request (PR) from `hw5-development` into `main` on GitHub, review the diff, then click *Merge*. This keeps a clean, auditable history.

9.7 Clean Up (Optional)

```
# Delete the local branch
git branch -d hw5-development

# Delete the remote branch
git push origin --delete hw5-development
```

9.8 Visual Model

```
main:     A --- B --- C -------- D --- E
                   \
branch:              X --- Y --- Z  (merge -> E)
```

9.9 Quick Reference

- `git switch -c <name>` – create and switch to a new branch.
- `git push -u origin <name>` – publish branch to GitHub and set upstream.
- `git rebase origin/main` – replay your commits on top of latest `main`.
- `git merge <name>` – merge the named branch into the current branch.
- `git branch -d <name>` – delete local branch after merging.