Git Lesson 1: Git Basics

Document owner	TENTEK LLC
Course	DevOps Engineering
Topic / Session number	Git / Lesson 1
Objectives / Goals	1. Version control 2. Git Basics

★ Objective 1: Version control

Why Git and GitHub?

Git is a distributed version control system that helps developers track changes in their codebase, collaborate effectively, and manage different versions of a project.

GitHub is a web-based platform that hosts Git repositories and adds collaboration tools like issue tracking, pull requests, and more.

Git vs GitHub

Git and GitHub are complementary tools, but they serve different purposes. Let’s break them down using the essay analogy to make the differences clear and relatable.

Software	Service
Version control	Git repository hosting
Maintained by Linux	Maintained by Microsoft
Open-Source	Free or paid membership
No user management	Built-in user management
Locally installed	Hosted on the web
Minimal external tool configuration	Active marketplace for tool integration
Little to no competition	High competition

What is Git?

Git is a **version control system** that helps you track changes to files (like your essay) and manage different versions of your work. It operates locally on your computer.

Essay Analogy for Git:

- Imagine you're writing an essay on your computer.
- You save versions of your essay at different points in time (e.g., "Essay_v1.docx," "Essay_v2.docx") so you can go back to earlier drafts if needed.
- Git is like the tool you use to organize and track all these versions of your essay, letting you:
  - Save snapshots of your work.
  - Compare drafts to see what changed.
  - Experiment with alternate versions without affecting the main draft.

Example of Using Git:

- You're working on an essay for a class. You write the introduction, save it as a version, then move on to the body paragraphs. If you make a mistake later, you can revert to the earlier saved version of your introduction using Git.

What is GitHub?

GitHub is a **cloud-based platform** that hosts Git repositories and provides additional collaboration features. It's like an online workspace where teams can share, review, and manage their work.

Essay Analogy for GitHub:

- Imagine you want to collaborate on your essay with friends. Instead of emailing drafts back and forth, you upload your essay to Google Drive or Dropbox so everyone can access it.

- GitHub is like Google Drive for essays written using Git - it stores your essay online and lets others contribute while keeping track of who made what changes.

Example of Using GitHub:

- You upload your essay (Git repository) to GitHub so that your group members can add their sections (e.g., one person writes the introduction while another writes the conclusion). Everyone’s changes are tracked, and conflicts are resolved when merging their contributions.

Problems Git Solves

1. **Version Control**: Git allows distributed development, meaning each developer has a complete copy of the project history locally.
2. **Collaboration**: Managing contributions from multiple developers without overwriting each other's work was challenging with older systems. Git's branching and merging capabilities make this seamless
3. **Backup and Recovery**: Git ensures that every version of your project is saved. If something breaks, you can revert to a previous version easily.

Why GitHub?

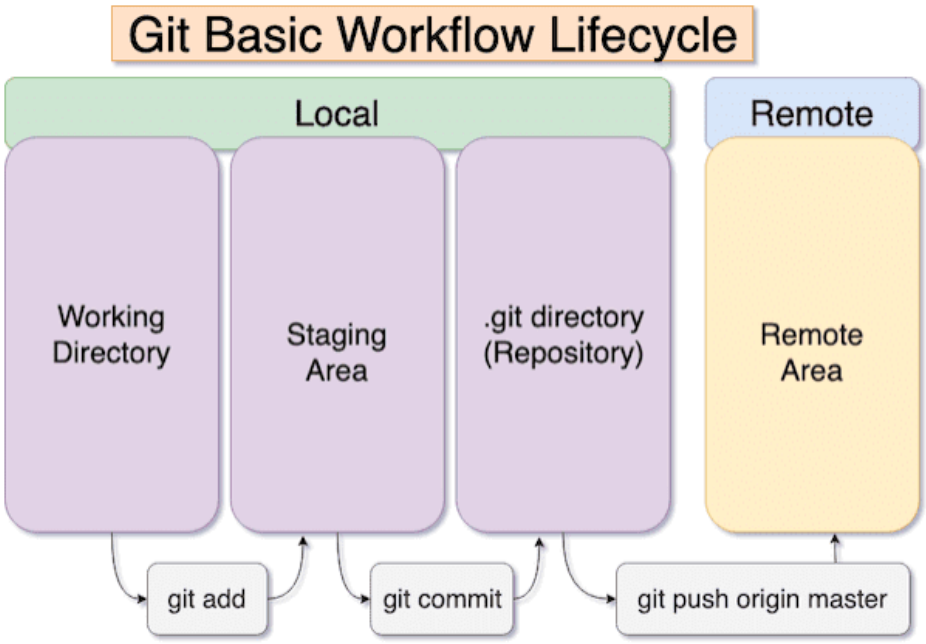
GitHub enhances Git by providing:

- A centralized platform for sharing code.
- Tools for collaboration (e.g., pull requests for code review).
- Project management features like issue tracking.
- A community for open-source contributions

Git Architecture: The Three Stages

Git operates on a three-stage architecture, which ensures organized and efficient version control:

1. **Working Directory**:
   This is where you make changes to your files. Think of it as your *drafting table* where you write or edit your work.
2. **Staging Area (Index)**:
   Before saving your changes permanently, you prepare them in the staging area. Imagine putting selected items into a *shopping cart* before checkout. Only the items in the cart will be purchased (committed).
3. **Local Repository**:
   This is where all your saved versions (commits) are stored locally on your computer. It’s like a *photo album* where each photo represents a snapshot of your project at a specific moment.
4. **Remote Repository**:
   If you want to share your work with others, you push it to a remote repository (e.g., GitHub). Think of this as uploading your photo album to a *cloud storage service* so others can view or contribute.



For Example

1. Writing and Editing Your Essay (Working Directory)

- Imagine you're writing an essay on your computer. This is your **working directory**, where you actively make changes to your essay file.
- You might add a paragraph, delete a sentence, or fix some typos. These changes are not yet saved permanently; they're just edits you're working on.

2. Highlighting Parts You Want to Save (Staging Area)

- Once you're happy with certain sections of your essay, you highlight them (e.g., using a highlighter or selecting text). This is like adding changes to the **staging area** using `git add`.
- For example:
  - You might highlight the introduction and conclusion because they're ready.
  - The body paragraphs are still being worked on, so you leave them out for now.

3. Saving a Version of Your Essay (Local Repository)

- After highlighting the parts you want to save, you click "Save As" and create a version of your essay with a name like "Essay_v1.docx". This is like committing your changes to the **local repository** with `git commit`.
- Each time you save, you write a little note about what changed (e.g., "Added introduction and conclusion").

4. Sharing Your Essay Online (Remote Repository)

- Once you've saved your essay locally, you might upload it to Google Drive or email it to a friend so they can see it. This is like using `git push` to send your changes to a **remote repository** like GitHub.
- Now, your essay is available online for others to view or collaborate on.

5. Collaborating with Friends (Branches)

- Imagine this is a group project where everyone contributes to the essay:
  - You create separate versions of the essay for each contributor (branches). For example:

        - One friend works on the introduction.
        - Another works on the conclusion.
        - You work on the body paragraphs.
    - Each person works independently without interfering with others' work.

6. Combining Everyone's Work (Merging Branches)

- Once everyone finishes their part, you combine all the sections into one final version of the essay. This is like merging branches in Git using `git merge`.
- If there are conflicts (e.g., two people wrote different versions of the same paragraph), you resolve them manually before finalizing.

7. Keeping Track of Changes (Version Control)

- Let's say you realize later that the introduction written by your friend was better in an earlier draft. Since every version of the essay has been saved, you can go back and retrieve that specific version. This is like using Git's version history (`git log`) or reverting to an earlier commit.

8. Experimenting Without Risk (Branches for Experimentation)

- Imagine you want to try rewriting the conclusion in a different style but don't want to mess up the current version of the essay. You make a copy of the essay and try out your new ideas there. This is like creating a new branch in Git for experimentation.

9. Submitting Final Essay (Pull Requests)

- When everyone has completed their changes and merged their work into one final version, they submit it as the finished essay for review by the teacher or editor. On GitHub, this process is similar to creating a pull request where changes are reviewed before being finalized.

10. Recovering from Mistakes

- If someone accidentally deletes an important section or messes up formatting, no worries! Since every version has been saved, you can roll back to an earlier version of the essay without losing all your work.

Real-World Examples

1. **Team Collaboration**:
   A team building an app can use branches for different features (e.g., login system, payment service). Once tested and reviewed, these branches are merged into the main branch.
2. **Open Source Contributions**:
   Developers fork an open-source project from GitHub, make improvements in their own branch, and submit a pull request to suggest their changes.
3. **Bug Fixes**:
   If a bug is reported, a developer can create a branch specifically for fixing it without disturbing ongoing feature development.

Key Commands

1. `git init`:
    - *Analogy*: Starting a new essay project.
    - Imagine you open a blank document and decide, "This is where I'll start writing my essay." You're setting up your workspace to track all future changes to this essay.
2. `git add`:
    - *Analogy*: Highlighting parts of your essay that are ready to be saved.

- You've written a few paragraphs, but only some of them are ready to save. You highlight or mark those sections for saving, preparing them for the next step.

3. `git commit`:
   - *Analogy*: Taking a snapshot of your essay at this point in time.
   - Once you've highlighted the sections you want to save, you press "Save As" and name it something meaningful (e.g., "Essay_v1.docx"). This saves a version of your essay along with a note about what changed (e.g., "Added introduction and conclusion").
4. `git status`:
   - *Analogy*: Checking what's ready to save and what still needs work.
   - You take a moment to review your essay and see which parts are highlighted (staged for saving) and which parts are still being worked on (unstaged).
5. `git branch`:
   - *Analogy*: Creating alternate versions of your essay for experimentation.
   - You decide to try rewriting the conclusion in a different style, so you make a copy of your essay file (e.g., "Essay_v2.docx") and work on that version without affecting the original.
6. `git merge`:
   - *Analogy*: Combining different versions of your essay into one final draft.
   - After experimenting with different conclusions, you decide which one works best and merge it back into the main essay document.
7. `git push` / `git pull`:
   - *Analogy*: Uploading or downloading your essay to/from Google Drive (or another cloud service).
   - If you want to share your essay with friends, you upload it (push) to Google Drive so they can access it. If they've made edits, you download their updated version (pull) to see their changes.

Big Picture Takeaways

- Git helps organize and track every step in your project's development.
- It enables collaboration by allowing multiple people to work on different parts simultaneously.
- Mistakes are not final because Git allows you to roll back changes or experiment safely using branches.
- Platforms like GitHub make sharing and reviewing code easier.

★ Objective 2: Git Basics

Git, the word, originated from a British English slang that means a stupid or unpleasant person. Well, this ain't the kind of Git we're going to talk about today…

If you're completely new to Git, this will help you get started with Git, understanding what Git is meant for and some fundamental concepts you need to understand about Git.

1. `.gitignore`: Ignoring Files in Git

The `.gitignore` file tells Git to ignore certain files or folders that you don't want to track. In the essay project, this could include drafts, temporary files, or system-generated files.

Essay Analogy for `.gitignore`

- Imagine you're writing an essay on your computer, and text editor creates temporary files like `~Essay.docx` or backup files like `Essay_backup.docx`. These files are unnecessary for version control.
- You can create a `.gitignore` file to tell Git: "Don't track these files."

Example for Essay Project

Let’s say you have the following files in your essay folder:

```
Essay.docx
~Essay.docx
Essay_backup.docx
Notes.txt

```

You only want to track `Essay.docx` and `Notes.txt`. Your `.gitignore` file would look like this:

```
# Ignore temporary and backup files
~*.docx
*_backup.docx

```

Practical Use

- Add the `.gitignore` file to your repository so Git knows which files to ignore.
- If you’ve already committed a file you want to ignore, untrack it using:

  ```
  git rm --cached FILENAME
  ```

2. Commit Naming Conventions

Commit messages are like labels for saved versions of your essay. A good commit message explains what changes were made and why.

Essay Analogy for Commit Messages

- Imagine you’re saving versions of your essay. Instead of naming them vaguely like “Version 1” or “Final Final Draft,” you use descriptive names like:
  - “Added introduction”
  - “Fixed typos in paragraph 3”
  - “Revised conclusion for clarity”

Best Practices

1. **Be Clear**: Describe what changed.
   - Example: `Add introduction section about climate change`
2. **Use Present Tense**: Write as if giving a command.
   - Example: `Fix typos in conclusion`
3. **Keep It Short**: The first line should be under 50 characters.

Examples of Good Commit Messages

- `Add opening paragraph about global warming`
- `Fix grammar errors in body paragraphs`
- `Update references section with new sources`

3. When and How to Commit

When to Commit

1. **Commit Often**: Save progress after completing meaningful steps.
   - Example: After finishing the introduction or revising a paragraph.
2. **Avoid Committing Half-Done Work**: Only commit when a section is complete.
   - Example: Don’t commit a half-written sentence; finish the paragraph first.
3. **Group Related Changes**: Keep related updates together in one commit.
   - Example: If you fixed typos and added new references, make separate commits for each.

How to Commit

1. Stage changes using `git add`:
   - Add specific files (e.g., just the essay):

     ```
     git add Essay.docx
     ```

   - Add all changes (e.g., essay + notes):

     ```
     git add .
     ```

2. Commit with a meaningful message:

```
git commit -m "Add conclusion about renewable energy"
```

4. What’s Inside the `.git` Directory?

The `.git` directory is where Git stores all the information about your essay project, including its history, metadata, and configuration.

Essay Analogy for `.git`

- Think of the `.git` folder as your personal assistant who keeps track of everything:
  - What versions of your essay exist.
  - Who made changes (if working with others).
  - Which branch (draft) you’re currently working on.

Key Components of `.git`

1. **HEAD**: Points to the current draft (branch) you’re working on.
   - Example: If you’re working on “Final Draft,” HEAD points there.
2. **objects/**: Stores every version of your essay as compressed data.
   - Example: Each saved version (commit) is stored here.
3. **refs/**: Keeps track of branches (different drafts) and tags (specific snapshots).
4. **config**: Contains settings for your essay project repository.
5. **logs/**: Tracks every change made (like a diary).

Practical Use

- If you accidentally delete your working directory but still have the `.git` folder, you can recover your project using:

```
git checkout .
```

Download GIT cheat sheet below for a quick reference in case you forget git commands.

SWTM-2088_... eet.pdf
17 Sep 2023, 05:11 PM