Git

Git Basics for Newbie Developers

Building Your First App

Scenario: You're a new software engineer tasked with building a simple Todo App

Meet Alex: Our Newbie Developer

- **Alex's Mission**: Build a Todo App from scratch
  - Never used version control before
  - Wants to track changes safely
  - Plans to add features incrementally
  - Worried about losing work

Git - The Tool

- **Taking snapshots** of your project at different points
- **A history book** of all changes you've made
- **A safety net** that lets you undo mistakes
- **A way to work on different features** simultaneously

Setting Up: Alex's First Day

Alex creates a new project folder and initializes Git:

```
# Create project folder
mkdir todo-app
cd todo-app

# Initialize Git repository
git init

# Check status
git status
```

Output:

```
On branch main
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

We have two keywords: **branch** and **commit**.

Branch

Think of it like parallel universes for your code.

- You start on the "main" branch (your primary timeline), but you can create new branches to experiment with features or fixes without affecting the main code.
- Later, you can merge successful changes back to main.

Commit

A commit is like taking a photograph of your project at a specific moment.

- It captures exactly what all your files look like right then.
- Each commit has a message describing what changed, creating a history you can look back through or return to if needed.

Creating and Adding the First File

Alex starts with a simple HTML file (**index.html**):

```html
<!DOCTYPE html>
<html>
<head>
    <title>My Todo App</title>
</head>
<body>
    <h1>Todo List</h1>
    <p>Coming soon...</p>
</body>
</html>
```

Let's check what Git sees:

```
> git status
On branch main
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html
```

- Git finds a new file (index.html), and it wants to know what to do with it.

Git's Three States

```
Working Directory  →  Staging Area  →  Repository
     (Modified)            (Staged)          (Committed)
```

Alex's file right now:

- **Working Directory (WD)**: `index.html` exists but Git isn't tracking it
- **Staging Area**: Empty
- **Repository**: Empty

Why Three, not Two?

Working Directory (WD): Walking around the store, putting items in your basket

- You're trying things, changing things, experimenting

Staging Area: Your shopping cart at checkout

- You review what you're about to buy
- You can remove items you changed your mind about
- You can add forgotten items
- Everything here will be "purchased" together
- Staging Area is called "index"

Repository = Your receipt/purchase history

- Once you hit "Pay," it's permanent and recorded.

Adding Files: The `git add` Command

- Alex needs to tell Git to track the file:

```
# Add specific file
git add index.html # git add . to add all the files

# Check status
git status
```

Output:

```
On branch main
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
  New file:   index.html
```

Now: File is in the **staging area**

- You can change anything in the staging area (index) before commit.

Committing Changes: Taking the Snapshot

Alex saves (commits) the first version to the repository:

```
git commit -m "Add initial HTML structure for todo app"
```

Output:

```
[main (root-commit) a1b2c3d] Add initial HTML structure for todo app
 1 file changed, 10 insertions(+)
 create mode 100644 index.html
```

First commit created! The file is now in the repository.

- git log --oneline command shows there is a commit with ID 282a841.
- It shows the HEAD is in the main branch

```
> git log --oneline
282a941 (HEAD -> main) Add initial HTML structure for todo app
```

HEAD - You are here

```
Commit A  →  Commit B  →  Commit C
                              ↑
                             HEAD
```

HEAD = "Where am I right now in my project's history?"

- Which commit are you currently "looking at"
- Where new commits will be added

Adding More Features

Alex adds a CSS file to make it look better:

```css
body {
    font-family: Arial, sans-serif;
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;
}

h1 {
    color: #333;
    text-align: center;
}
```

The Workflow: Add → Commit

`git add` and `git commit -m` can be shortened as `git commit -am`.

```
# Add the CSS file
git add style.css

# Commit the change
git commit -m "Add basic styling with CSS"
```

```
# Let's see our history
git log --oneline
```

Output:

```
a1822d9 (HEAD -> main) Add basic styling with CSS
282a941 Add initial HTML structure for todo app
```

- We have two commits.
- The HEAD is pointing to the newest commit on the main branch.

Modifying Existing Files

- Alex updates the HTML to include the CSS:

```html
<!DOCTYPE html>
<html>
<head>
    <title>My Todo App</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Todo List</h1>
    <ul id="todoList">
        <li>Learn Git basics</li>
        <li>Build todo app</li>
    </ul>
</body>
</html>
```

git diff

`git diff` command shows the difference between the index and the working directory.

```
> git diff
diff --git a/index.html b/index.html
index 59033b7..255b9c1 100644
--- a/index.html
+++ b/index.html
@@ -2,9 +2,13 @@
 <html>
 <head>
     <title>My Todo App</title>
+    <link rel="stylesheet" href="style.css">
 </head>
 <body>
     <h1>Todo List</h1>
-    <p>Coming soon...</p>
...
```

Understanding the Changes

@@ -2,9 +2,13 @@ shows what lines are updated.

- You added four lines.
- Previously, it was nine at line 2, but now (+) it is 13 at line 2
- 13 - 9 = 4

```
@@ −2,9 +2,13 @@
    ↑  ↑  ↑  ↑
    │  │  │  └── NEW: showing 13 lines
    │  │  └────── NEW: starting at line 2
    │  └────────── OLD: showing nine lines
    └───────────── OLD: starting at line 2
```

Tracking Modified Files

```
git status
```

Output:

```
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
modified:   index.html
```

The file is **modified** but not yet staged.

- `git add` and `git commit -m` to make a new commit.

```
> git add index.html
> git commit -m "Link CSS and add initial todo items"
mcho@mac todo-app> git commit -m "Link CSS and add initial todo items"
[main f4c1698] Link CSS and add initial todo items
 1 file changed, 5 insertions(+), 1 deletion(-)

> git log --oneline
f4c1698 (HEAD -> main) Link CSS and add initial todo items
a1822d9 Add basic styling with CSS
282a941 Add initial HTML structure for todo app
```

Deleting Files: When Alex Changes Mind

Alex realizes they don't need a separate CSS file and wants to use inline styles:

- You can delete the file, and `git add .` to get the same results.

```
# Remove file from both working directory and Git
git rm style.css

# Check status
git status
```

Output:

```
On branch main
Changes to be committed:
 deleted:    style.css
```

```
> git commit -m "Remove external CSS file"
[main 7f614ac] Remove external CSS file
 1 file changed, 11 deletions(-)
 delete mode 100644 style.css
> git log --oneline
7f614ac (HEAD -> main) Remove external CSS file
f4c1698 Link CSS and add initial todo items
a1822d9 Add basic styling with CSS
282a941 Add initial HTML structure for todo app
```

What if I need the deleted file?

In Git, nothing is deleted, allowing us to revert to any previous version.

- `git checkout` is riding a time machine.
- We know that f4c1... is the commit before the deletion.

```
> git checkout f4c1
Note: switching to 'f4c1'.

You are in a 'detached HEAD' state.
```

Detached Head

When we check out to a previous commit, the HEAD is moved.

- Now, the HEAD is not pointing to the main branch anymore.
- We call this `detached head.`
- We should not make any changes in this state.

```
> git log --oneline
f4c1698 (HEAD) Link CSS and add initial todo items <--
a1822d9 Add basic styling with CSS
282a941 Add initial HTML structure for todo app
```

Go back to the branch

`git checkout main` moves the HEAD to the tip of the main branch.

```
Before: A → B → C → D ← main branch tip
                ↑
              HEAD (you were exploring here)

After:  A → B → C → D ← HEAD (back at the tip!)
                      ↑
                 main branch
```

Reset --hard (Careful!)

- We cannot delete a commit, but we can make it as if the commit did not happen using `git reset-- hard`.

```
> git reset --hard f4c1
HEAD is now at f4c1698 Link CSS and add initial todo items
> git log --oneline
f4c1698 (HEAD -> main) Link CSS and add initial todo items
a1822d9 Add basic styling with CSS
282a941 Add initial HTML structure for todo app
```

Branching: Working on New Features

Alex wants to add an "Add Todo" feature, but doesn't want to break the main version:

- We can make a new branch that is separated from the main branch using the `git checkout -b`.

- Remember that `Detached HEAD` when we `git checkout`.
- We can add the `-b` option to create a new branch and move the HEAD to the tip of the new branch.

```
# Create and switch to a new branch
> git checkout -b add-todo-feature
Switched to a new branch 'add-todo-feature'
# Check current branch
```

We can use the `git switch -c` to get the same results.

- `git branch` shows all the branches we have.
- `*` shows the branch that the HEAD is in.

```
# Or using newer syntax
git switch -c add-todo-feature

> git branch
* add-todo-feature
  main
```

Developing on the Branch

- Alex adds JavaScript functionality:

```
function addTodo() {
    const input = document.getElementById('todoInput');
    const list = document.getElementById('todoList');

    if (input.value.trim() !== '') {
        const li = document.createElement('li');
        li.textContent = input.value;
        list.appendChild(li);
        input.value = '';
    }
}
```

Updating HTML for the Feature

```html
<!DOCTYPE html>
<html>
<head>
    <title>My Todo App</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Todo List</h1>
    <input type="text" id="todoInput" placeholder="Enter new todo">
    <button onclick="addTodo()">Add Todo</button>
    <ul id="todoList">
        <li>Learn Git basics</li>
        <li>Build todo app</li>
    </ul>
    <script src="script.js"></script>
</body>
</html>
```

Committing Branch Changes

- We create a new commit in the `add-todo-feature` branch.

```
# Add all changes
> git add .

# Commit the feature
> git commit -m "Add JavaScript functionality to add new todos"

# See commit history on this branch

> git log --oneline --graph --all
* ecaa600 (HEAD -> add-todo-feature) Add JavaScript functionality to add new todos
* f4c1698 (main) Link CSS and add initial todo items
* a1822d9 Add basic styling with CSS
* 282a941 Add initial HTML structure for todo app
```

Switching Between Branches

Alex can switch back to the main to see the original version:

- **On main branch**: `index.html`
- **On add-todo-feature branch**: `index.html` and `script.js` (with new features)

```
# Switch to main branch
git checkout main # or git switch main

# Check the files
ls
```

Merging: Bringing Features Together

Alex is happy with the feature and wants to merge it into main:

- Merging is pulling another branch into the branch with the HEAD.

```
# Make sure we're on the main branch
git checkout main
```

When we merge,

```
> git merge add-todo-feature
Updating f4c1698..ecaa600
Fast-forward
 .DS_Store | Bin 0 -> 6148 bytes
 script.js |  11 +++++++++++
 2 files changed, 11 insertions(+)
 create mode 100644 script.js
```

After Merging

All commits from the feature branch are now in main!

```
> git log --oneline --graph --all
* ecaa600 (HEAD -> main, add-todo-feature) ...
* f4c1698 Link CSS and add initial todo items
* a1822d9 Add basic styling with CSS
* 282a941 Add initial HTML structure for todo app
```

We can delete the branch when we don't need it.

```
# Clean up: delete the feature branch
git branch -d add-todo-feature
```