

# Building Your Code Time Machine

## A Discovery Exercise

### The Problem

You're working on a data analysis project. You have a file called `analysis.py`. Over the course of a week, you make many changes. Sometimes you break things. Sometimes you fix them. Sometimes you want to remember what you did yesterday.

This exercise will help you discover a system for tracking your work.

## 1 Part 1: Saving Your Work

### 1.1 Your First Day

You start with this code in `analysis.py`:

```
def calculate_mean(data):
    total = sum(data)
    return total / len(data)
```

1. You realize this code crashes when `data` is empty. You fix it:

```
def calculate_mean(data):
    if len(data) == 0:
        return 0
    total = sum(data)
    return total / len(data)
```

Write down exactly what changed (use `+ prefix` for lines added, `- prefix` for lines removed):

2. Later that day, you add a new function:

```

def calculate_mean(data):
    if len(data) == 0:
        return 0
    total = sum(data)
    return total / len(data)

def calculate_median(data):
    sorted_data = sorted(data)
    return sorted_data[len(sorted_data) // 2]

```

Write down what changed this time (use + prefix for lines added, - prefix for lines removed):

## 1.2 Designing a Tracking System

3. You want to remember what your code looked like at different points in time. For each snapshot you save, what three pieces of information would be essential to include? Consider: How will you know when this snapshot was created? How will you remember what changes you made? How will you know what the code looked like at this point?
  
  
  
  
  
  
  
  
4. You decide to save snapshots in boxes. Here's what you have so far:

### Snapshot 1

**When:** Monday 9:00 AM  
**Message:** "Started project"  
**Files:** analysis.py (original version)

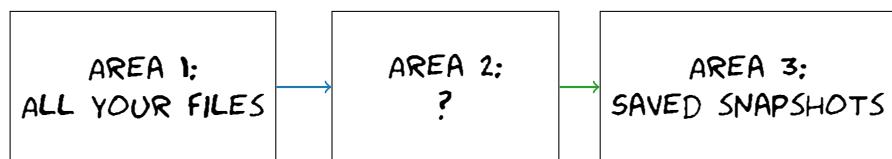
## 1.3 Choosing What to Save

You spend all morning working. You now have:

- analysis.py (finished, working perfectly)
- test.py (finished, working perfectly)
- scratch.py (messy experimental code, not ready)
- temp\_output.txt (temporary file, you don't need it)

5. You want to create a snapshot. Should you include all four files?

6. You realize you need a way to choose which files go into a snapshot. Design a system with three areas:



What should Area 2 be? What purpose does it serve? (Hint: Think about the problem you encountered with the four files earlier)

7. You have a file in Area 1 that you want to include in your next snapshot. What would you call the action of moving it to Area 2?

---

You have files in Area 2 and you want to create a snapshot in Area 3 containing those files. What would you call this action?

---

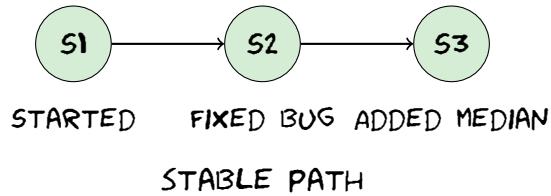
8. Why is it useful to have Area 2 instead of going directly from Area 1 to Area 3?

## 2 Part 2: Parallel Work

### 2.1 The Experiment Dilemma

You want to try two completely different approaches to analyzing your data. You don't know which will work better. You want to try both without losing either one.

Your current snapshot history represents your stable, working code. This is the path that always works:



9. You want to try Method A and Method B starting from S3. Both will modify analysis.py. You want to keep the stable path stable and working. If you just keep making snapshots in a straight line on the stable path, what problem will you run into?
10. Starting from S3 on the stable path, you want to create snapshots for both Method A and Method B without losing either one or breaking the stable path. Draw what your snapshot history should look like. How can you arrange the snapshots so you can experiment with both methods while keeping the stable path stable?
11. You've created separate paths for experimentation while keeping the stable path stable. What would you call these experimental paths? Give them descriptive names based on what they're testing.

### 3 Part 3: Hands-On with Git

Now let's use actual Git commands to implement the system you designed. You'll create a repository, make commits, work with branches, and collaborate.

#### 3.1 Setup

Installing Git:

- macOS: Open Terminal and type `git --version`. If not installed, it will prompt you to install.
- Windows: Download from <https://git-scm.com/download/win> and install. Use Git Bash for commands.

#### 3.2 Exercise: Your First Repository

Open your terminal (macOS) or Git Bash (Windows) and follow these steps:

##### 12. Create a project folder and initialize Git:

```
mkdir my-project  
cd my-project  
git init
```

What did `git init` do? Check by running `ls -a` (macOS) or `dir /a` (Windows). You should see a `.git` folder.

##### 13. Create your first file and make a snapshot (commit):

```
echo "print('Hello, Git!')" > hello.py  
git add hello.py  
git commit -m "Add hello script"
```

Which part of your three-area system does `git add` implement? Which part does `git commit` implement?

**14. Create a branch for experiments:**

```
git branch experiment  
git checkout experiment  
echo "print('Experimental feature')" >> hello.py  
git add hello.py  
git commit -m "Add experimental feature"
```

What does git branch do? What does git checkout do? How does this relate to the parallel paths you designed?

**15. Switch back to the main branch and check your file:**

```
git checkout main  
cat hello.py
```

(Use type hello.py instead of cat on Windows)

What do you notice about the contents of hello.py? Why does it look different from when you were on the experiment branch?

**16. Merge your experiment into main:**

```
git merge experiment  
cat hello.py
```

What happened to hello.py? How does this relate to combining work from different paths?

## 17. Setup remote repository (GitHub):

Create a free account at <https://github.com> if you don't have one. Then create a new repository called my-project (empty, no README).

GitHub will show you commands. They'll look like this:

```
git remote add origin https://github.com/YOUR-USERNAME/my-project.git  
git push -u origin main
```

What does git remote add do? What does git push do? How does this relate to the backup server concept?

## 18. Make a change on GitHub and pull it:

Go to your repository on GitHub. Click on hello.py and click the pencil icon to edit. Add a new line:

```
print("Edited on GitHub")
```

Click "Commit changes" at the bottom. Then in your terminal:

```
git pull  
cat hello.py
```

What does git pull do? How is this different from git push?

## 4 Git

The system you just designed and used is called Git. Here are the real names for the concepts you discovered:

- Snapshot = Commit
- Area 2 (choosing what to save) = Staging Area
- Parallel paths = Branches
- Combining work = Merge

19. Now that you know the real names, write a short description of what Git is and why it's useful: