🎯 GIT COMPLETE WALKTHROUGH

👥 Two Developers Working Together (Full Story Walkthrough)

👨‍💻 Dev 1 = **Vatsal**
👨‍💻 Dev 2 = **Jay**
->Project = **Simple Website**
Files:

```
index.html
style.css
```

🏁 PHASE 1 – Project Starts (Vatsal creates first feature)

Vatsal clones the repo:

```
git clone https://github.com/team/sample-project.git
cd sample-project
```

Check project state:

```
git status
```

🧠 Git says: working tree clean.

Vatsal edits:

```
index.html
```

Adds homepage structure.

Before saving, he checks what exactly changed:

Copy

```
git diff
```

👉 Shows added HTML lines.

Now stage changes:

Copy

```
git add .
```

Save snapshot:

Copy

```
git commit -m "Add homepage structure"
```

Upload to GitHub:

Copy

```
git push
```

☁️ Code is now safely in cloud.

🔄 PHASE 2 – Jay pulls and works on styling

Jay opens project:

Copy

```
git pull
```

He checks history:

Copy

```
git log --oneline
```

🧠 Jay sees Vatsal's commit.

Jay creates his own branch:

```
git switch -c improve-style
```

Jay edits:

```
style.css
```

Adds colors and layout.

Checks changes:

```
git diff
```

Stages:

```
git add .
```

Commits:

```
git commit -m "Improve page styling"
```

Pushes:

```
git push
```

Jay creates Pull Request and merges.

🔁 PHASE 3 – Vatsal syncs Jay's changes

Vatsal updates his code:

```
git pull
```

Now Vatsal sees Jay's styling changes.

⚡ PHASE 4 – Parallel safe work

Vatsal edits:

```
index.html
```

Jay edits:

```
style.css
```

They both:

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

No conflicts because different files 👍

⚔️ PHASE 5 – Conflict Scenario (Deep Dive)

🚨 Situation:
Both Vatsal and Jay accidentally edit the SAME file:

```
index.html
```

Same lines.

💥 Step 1 – Jay pushes first

Jay commits and pushes successfully.

GitHub now has Jay's version.

😳 Step 2 – Vatsal tries to pull

Vatsal runs:

Copy

```
git pull
```

Git says:

❌ CONFLICT detected!

Because:

- Git sees two different edits on same lines.
- Git cannot decide whose version is correct.

🔍 Step 3 – Vatsal opens conflicted file

Inside `index.html`, Git shows:

Copy

```
<<<<<<< HEAD
<h1>Welcome from Vatsal</h1>
=======
<h1>Welcome from Jay</h1>
>>>>>>> main
```

This means:

Symbol	Meaning
HEAD	- Vatsal's version

Symbol	Meaning
Middle	- Separator
Bottom	- Jay's version

🧠 Step 4 – Vatsal decides what to keep

Options:

- Keep Vatsal version
- Keep Jay version
- Combine both

Example fix:

Copy

```
<h1>Welcome from Vatsal & Jay</h1>
```

Delete conflict markers manually.

Save file.

✅ Step 5 – Finalize conflict resolution

Tell Git conflict is resolved:

Copy

```
git add .
git commit -m "Resolve merge conflict in homepage"
git push
```

🎉 Conflict resolved safely.

😂 Analogy:

Two people editing same sentence in a Google Doc simultaneously.
Teacher asks: "Which version is correct?"
You decide and finalize ✍️

🧳 PHASE 6 – Stash Scenario (Deep Dive)

🚨 Situation:
Jay is working on a new feature in:

```
style.css
```

But suddenly Vatsal says:

> "Urgent! Pull latest changes NOW."

Jay's code is unfinished and cannot be committed.

🛡️ Step 1 – Hide unfinished work safely

Jay runs:

```
git stash
```

What happens:

- Git temporarily hides Jay's local changes.
- Working folder becomes clean.
- No data lost.

🧠 It's like:
Putting half-cooked food in fridge 🧊🍕

🔄 Step 2 – Pull latest changes safely

```
git pull
```

Now Jay has Vatsal's latest code.

🎁 Step 3 – Restore hidden work

```
```

```
git stash pop
```

Jay's unfinished changes return.

Now Jay continues working.

😂 Analogy:

You pause your video game 🎮
Finish homework 📚
Resume game exactly where you left off.

🕵️ PHASE 7 – History & Difference Investigation (Deep Dive)

This phase helps teammates understand:
✓ What changed?
✓ Who changed?
✓ When changed?

🔍 Scenario A – Jay checks what Vatsal changed

Jay runs:

Copy

```
git log --oneline
```

Sees:

Copy

```
a21f Fix navbar alignment
b32d Add homepage content
```

Now Jay knows project history.

🔎 Scenario B – Jay checks exact code difference

Copy

```
git diff
```

Shows added / removed lines.

Useful before committing or reviewing teammate changes.

🧠 Scenario C – Compare before pushing

Jay checks:

- Did I accidentally break something?
- Are there unnecessary changes?

Git diff helps verify.

😂 Analogy:

Watching CCTV replay 📹
Seeing exactly who entered which room and when.

⭐ TIER 1 COMMANDS USED

Copy

```
git status
git add .
git commit -m "message"
git pull
git push
git branch
git switch
```

⭐ TIER 2 COMMANDS USED

Copy

```
git log --oneline
git diff
git stash
```

```
git merge
git fetch
```

🎯 MASTER FLOW TO MEMORIZE

Copy

```
git pull
→ work
→ git status
→ git diff
→ git add .
→ git commit
→ git push
```