COMP 3550

3.1 — GIT & VERSION CONTROL CONCEPTS

Week 3: Version Control & Testing Foundations

WHAT IS VERSION CONTROL?

- Think about your resume:
  - resumev1.docx
  - resumev1.1.docx
  - resumev2.docx
  - resume_final_sept.docx
  - resume_final_v3_april_2025.docx
- 💾 “Save As → Save As → Chaos”

WHAT IS VERSION CONTROL?

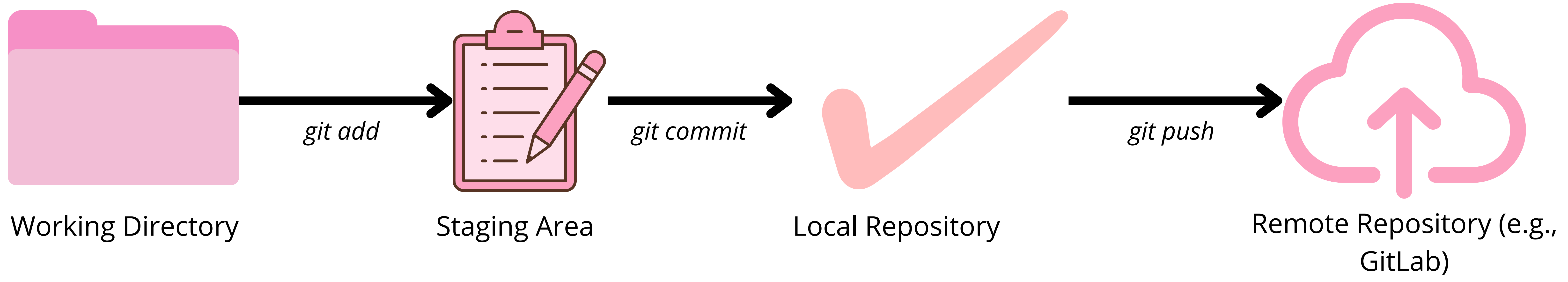
- Think about your resume:
  - resumev1.docx
  - resumev1.1.docx
  - resumev2.docx
  - resume_final_sept.docx
  - resume_final_v3_april_2025.docx
- 💾 “Save As → Save As → Chaos”

Version Control to the Rescue:

- ✅ Track changes over time
- ⏪ Undo mistakes and roll back to earlier versions
- 🤝 Collaborate safely without overwriting each other’s work
- 📜 See who changed what and when

GIT BASICS

- Git Is **Distributed**
  - Every user has a full copy of the project's history
  - You can work offline and still have access to all commits
- Workflow Overview:
  - Local → Staging → Commit → Push



KEY COMMANDS & CONCEPTS

🔄Local vs. Remote:

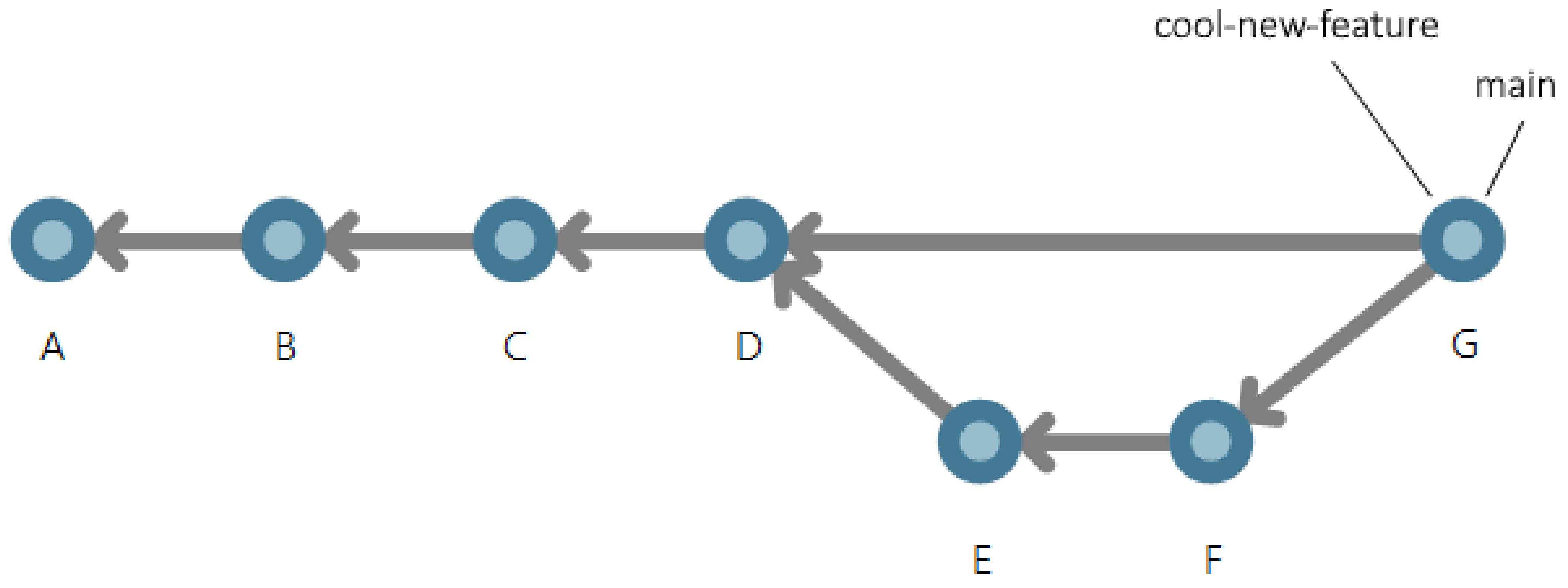
- **Local repo:** where you make and track changes on your own machine
- **Remote repo:** shared version (e.g., on GitHub) where you collaborate with others

KEY COMMANDS & CONCEPTS

Local Commands	Remote Commands
git init – start a new Git repository	**git push** – upload commits to remote repository
git add – stage changes	**git pull** – fetch and merge updates from remote
git commit – save a snapshot	
git status – check what's changed	
git log – view commit history	

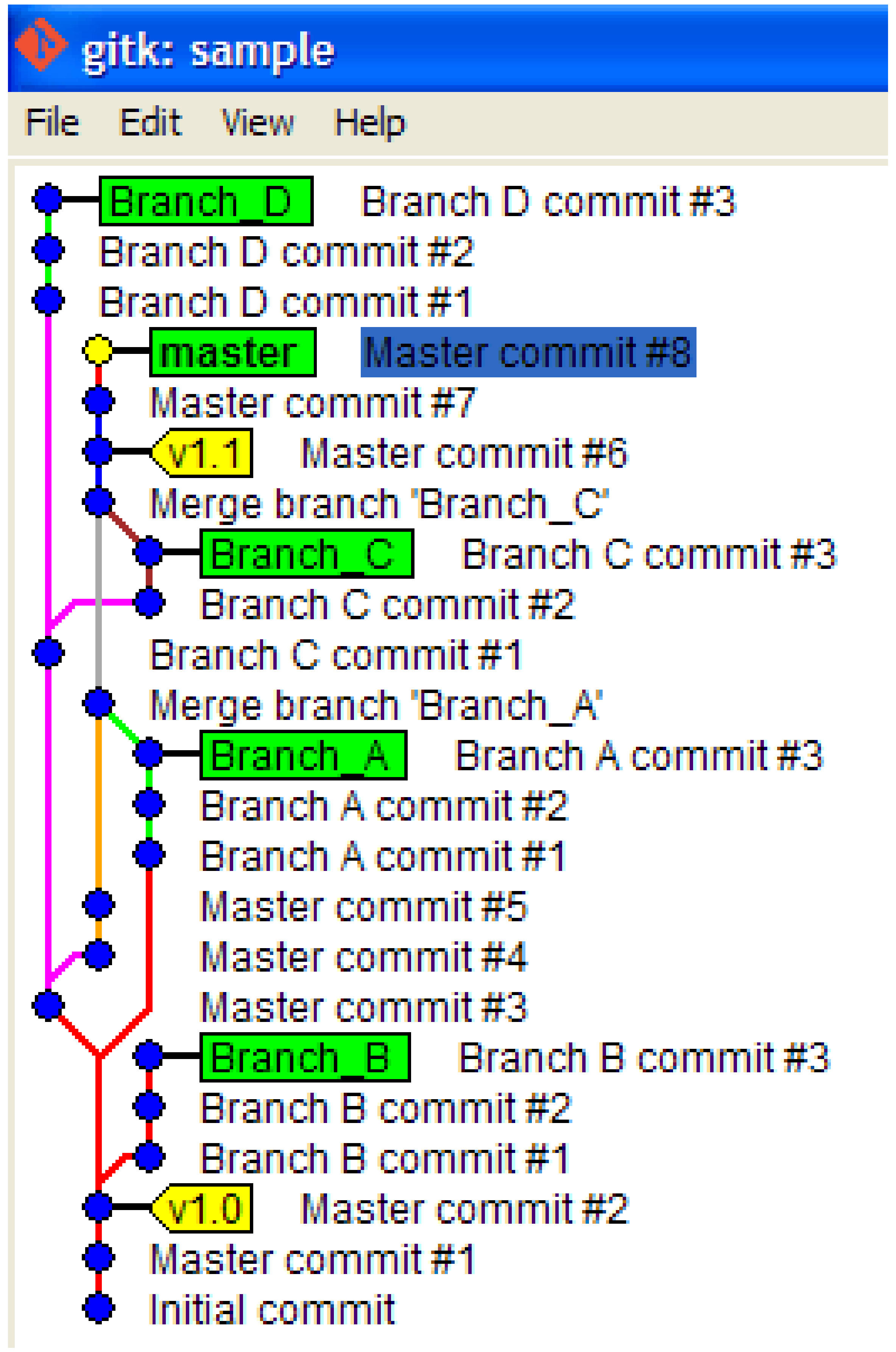
BRANCHES & MERGES

- Branch = Parallel Line of Work all off the main line
  - Use branches to try new features, fix bugs, or experiment
  - Keeps the main project stable
- Merging = Bringing It Back Together
  - Combine changes from a branch into the main codebase (usually main)
  - Git tries to merge automatically — but sometimes conflicts need resolving



BRANCHES & MERGES

- but what if we are ALL working on code?
- things can get messier



Branch_D Branch D commit #3
Branch D commit #2
Branch D commit #1
master Master commit #8
Master commit #7
v1.1 Master commit #6
Merge branch 'Branch_C'
Branch_C Branch C commit #3
Branch C commit #2
Branch C commit #1
Merge branch 'Branch_A'
Branch_A Branch A commit #3
Branch A commit #2
Branch A commit #1
Master commit #5
Master commit #4
Master commit #3
Branch_B Branch B commit #3
Branch B commit #2
Branch B commit #1
v1.0 Master commit #2
Master commit #1
Initial commit

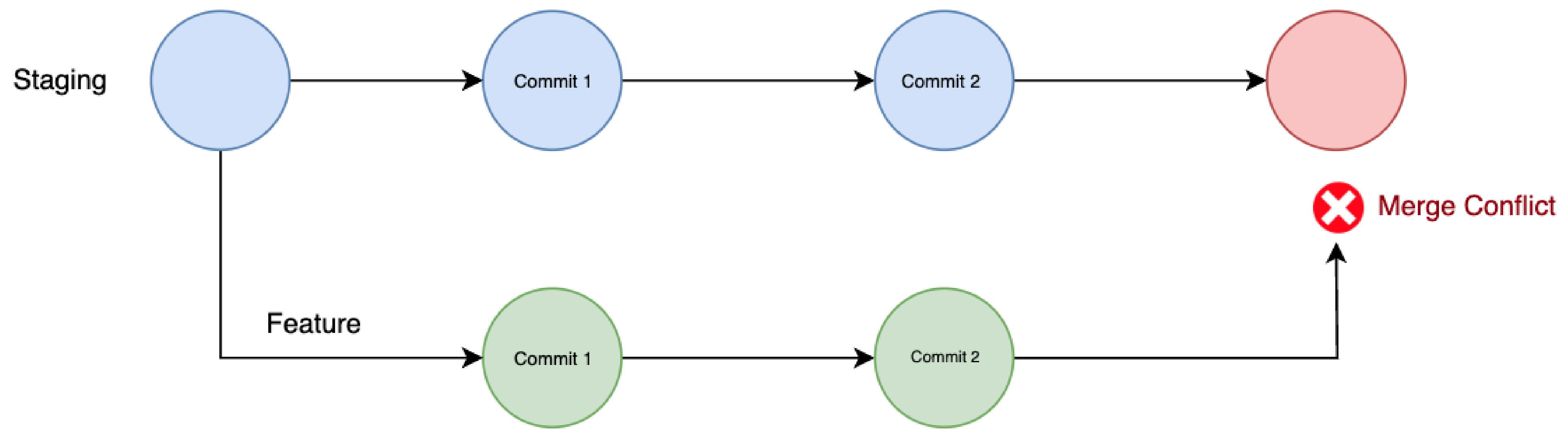
MERGE CONFLICTS

What They Are:

- A merge conflict happens when two branches edit the same part of a file, and Git doesn't know which change to keep
- Git stops and asks you to resolve it manually

How They Happen:

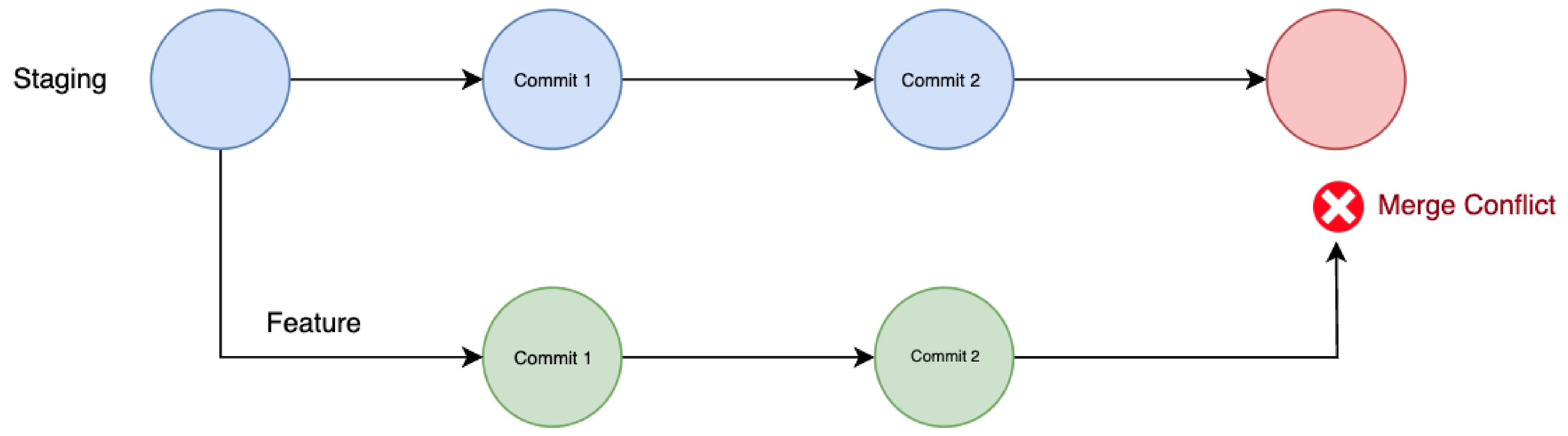
- Two people edit the same line of code in different branches
- You made changes locally, but the remote branch changed too
- You forgot to pull before pushing



MERGE CONFLICTS

Tips to Avoid Merge Conflicts

- Communicate with teammates about what files you're editing
- Pull before you push – always get the latest updates first
- Make small, frequent commits instead of huge changes
- Use feature branches to isolate work



GIT PHILOSOPHY & BEST PRACTICES

- *Commit early, commit often*
  - Don’t wait — save your work in logical chunks
  - Frequent commits make it easier to track changes, fix bugs, and collaborate
- *Atomic Commits = One Change Per Commit*
  - Each commit should do one thing (e.g., fix a bug, add a feature, update docs)
  - Makes history clean, readable, and easy to roll back if needed
- Write meaningful commit messages like:
  - *Fix login redirect bug*
  - **NOT** *stuff, update, final, etc.*

PAUSE & REFLECT

Looking to learn more or practice? Check out some references here! Do **not** wait to learn git. This is crucial for your success in the project.

GeeksForGeeks Resource and Tutorial

GitHub Resource and Tutorial