Week 8 Lab

Version Control and Distributed Workflows

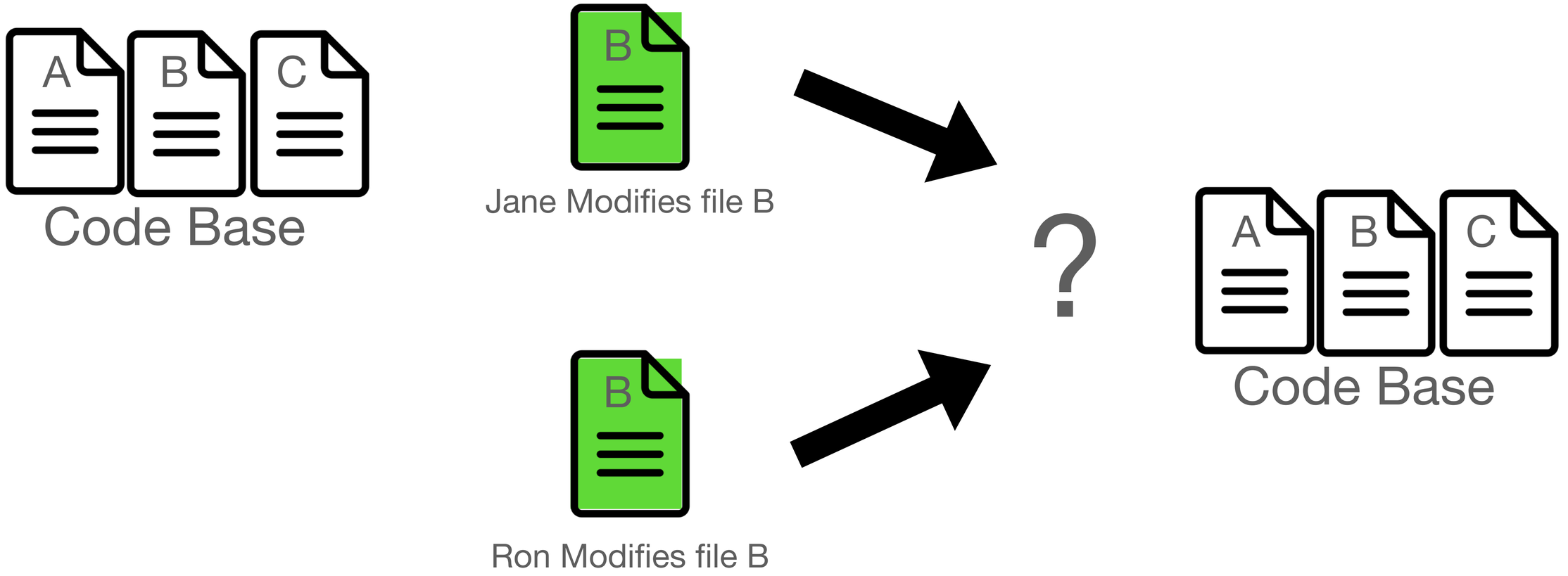
Chaufournier & Wood
CSCI 2541

We have to work in teams??

How do we all work on the same code base without stepping over each other?

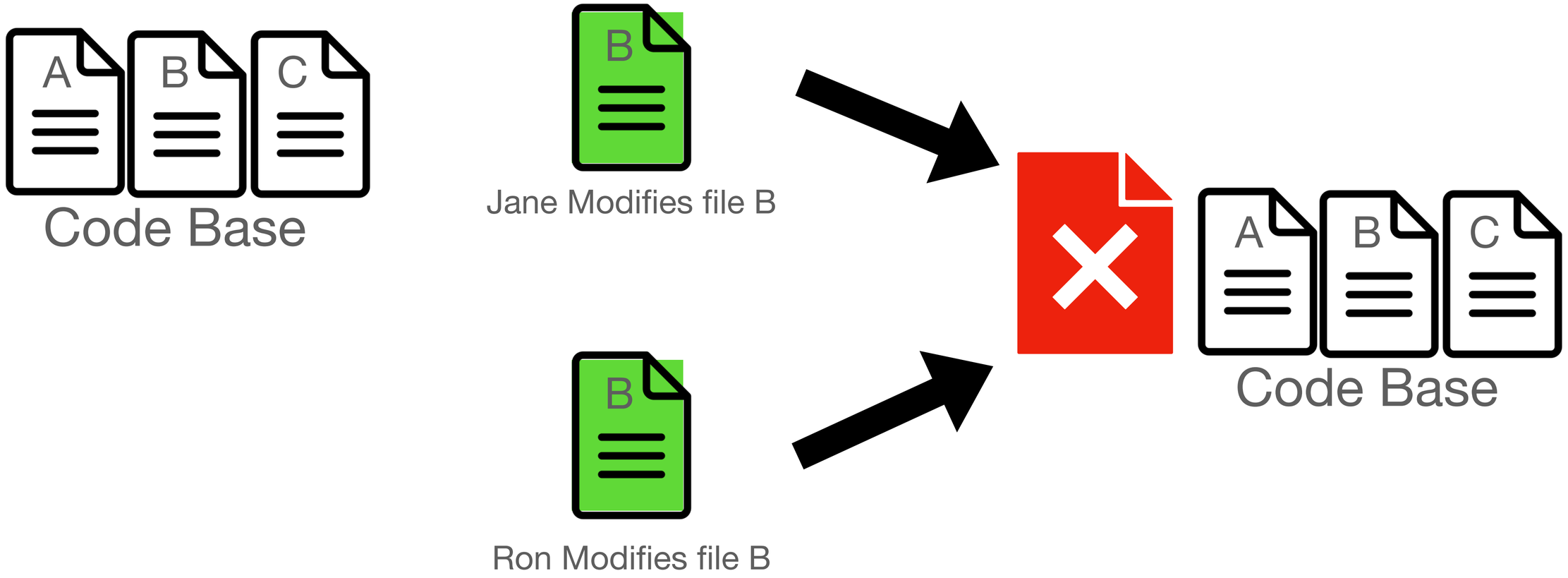
Why do we need version control

How do you decide whose changes to use?

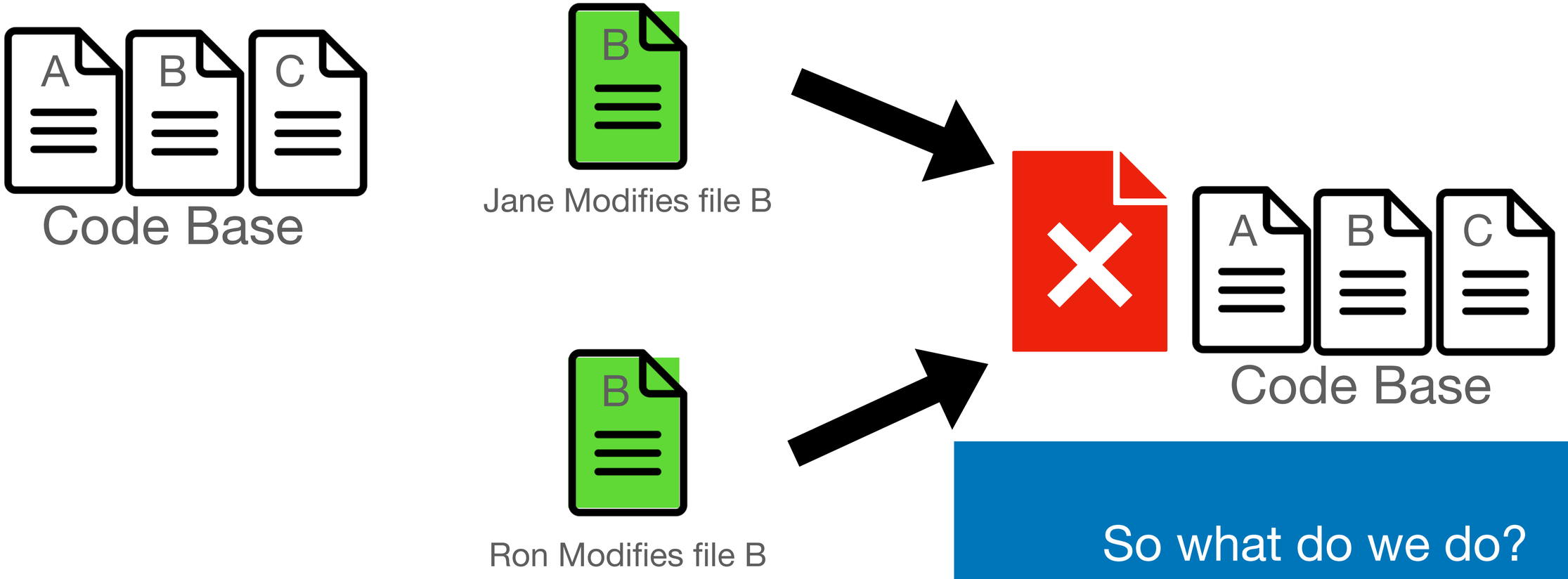


Why do we need version control

The files conflict without a reasonable way to decide.



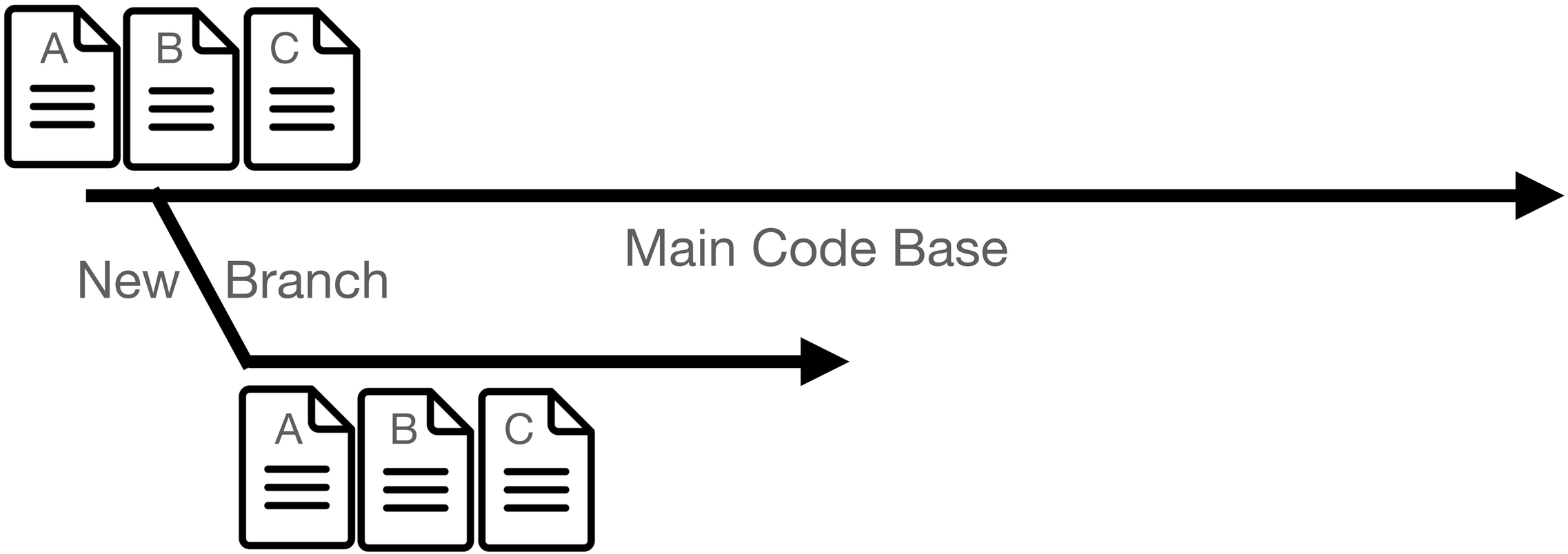
Why do we need version control



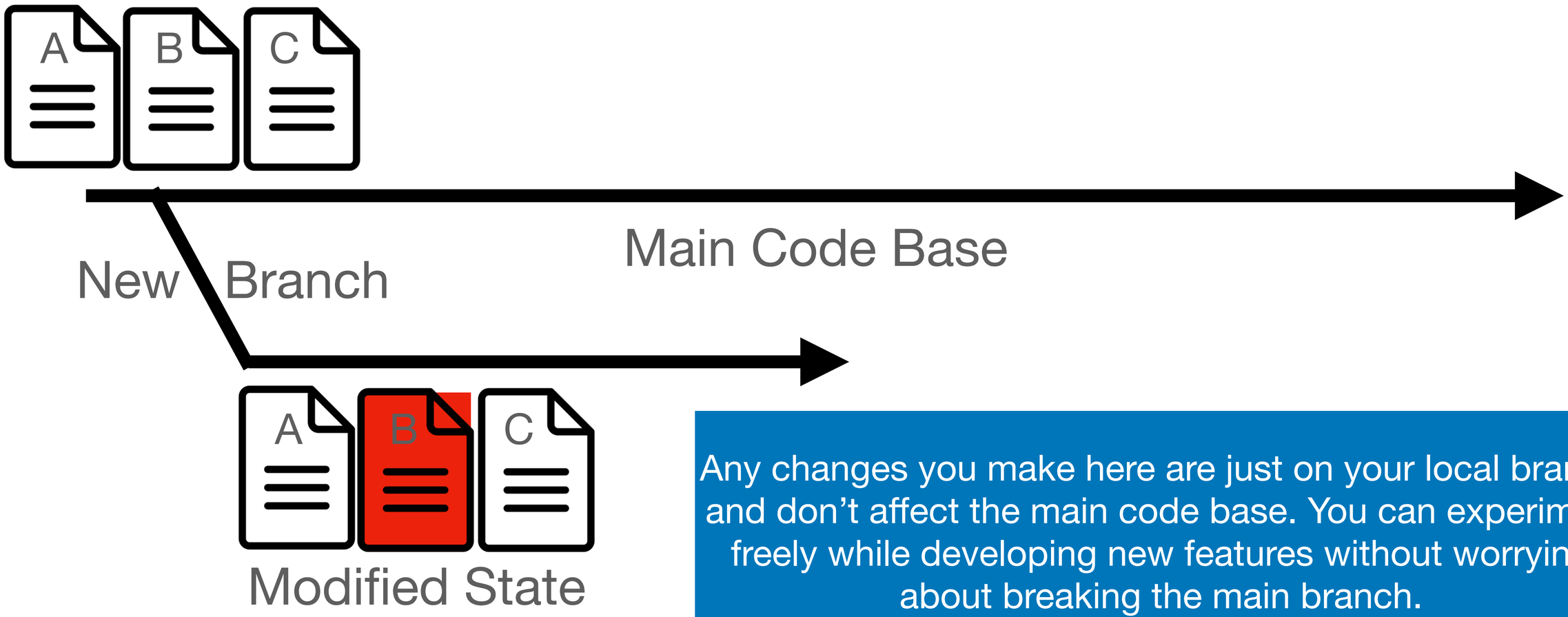
Distributed Version Control

Think of your codebase like the main timeline

To make changes you first create a new branch



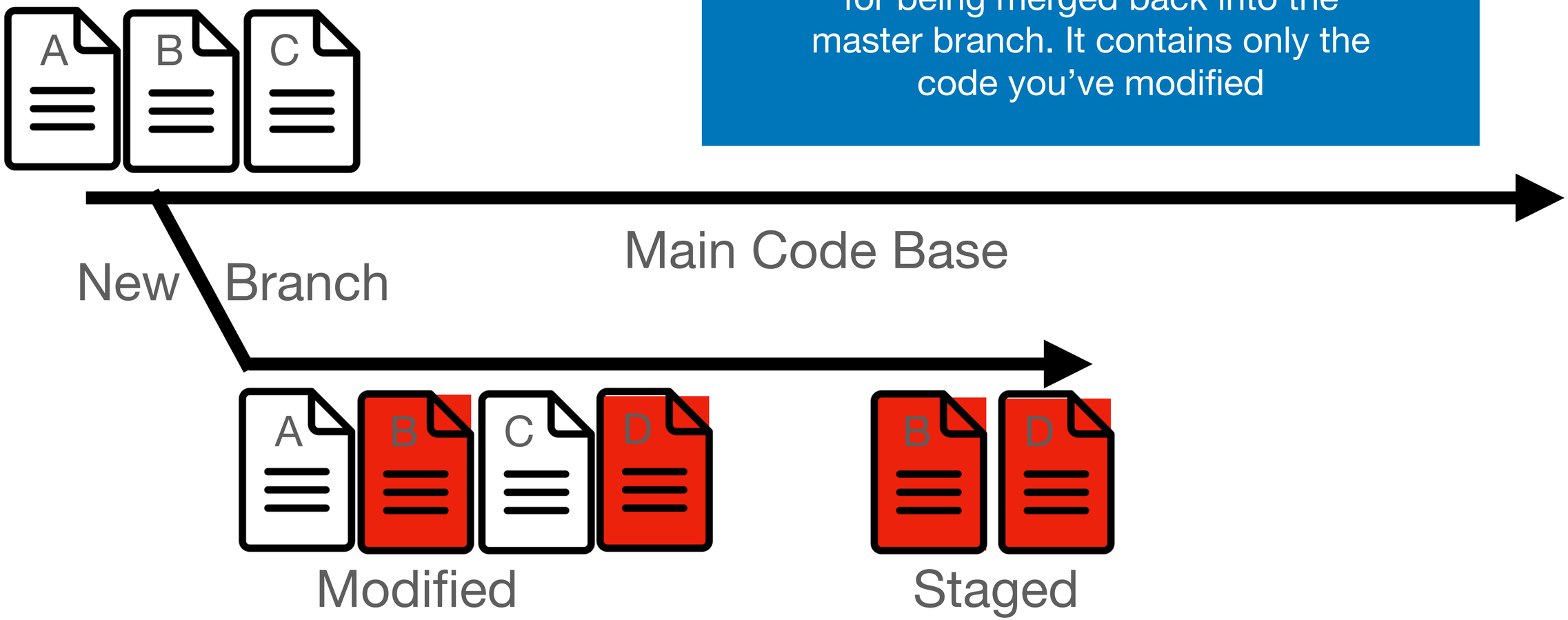
Entering the Modified State



Any changes you make here are just on your local branch and don’t affect the main code base. You can experiment freely while developing new features without worrying about breaking the main branch.

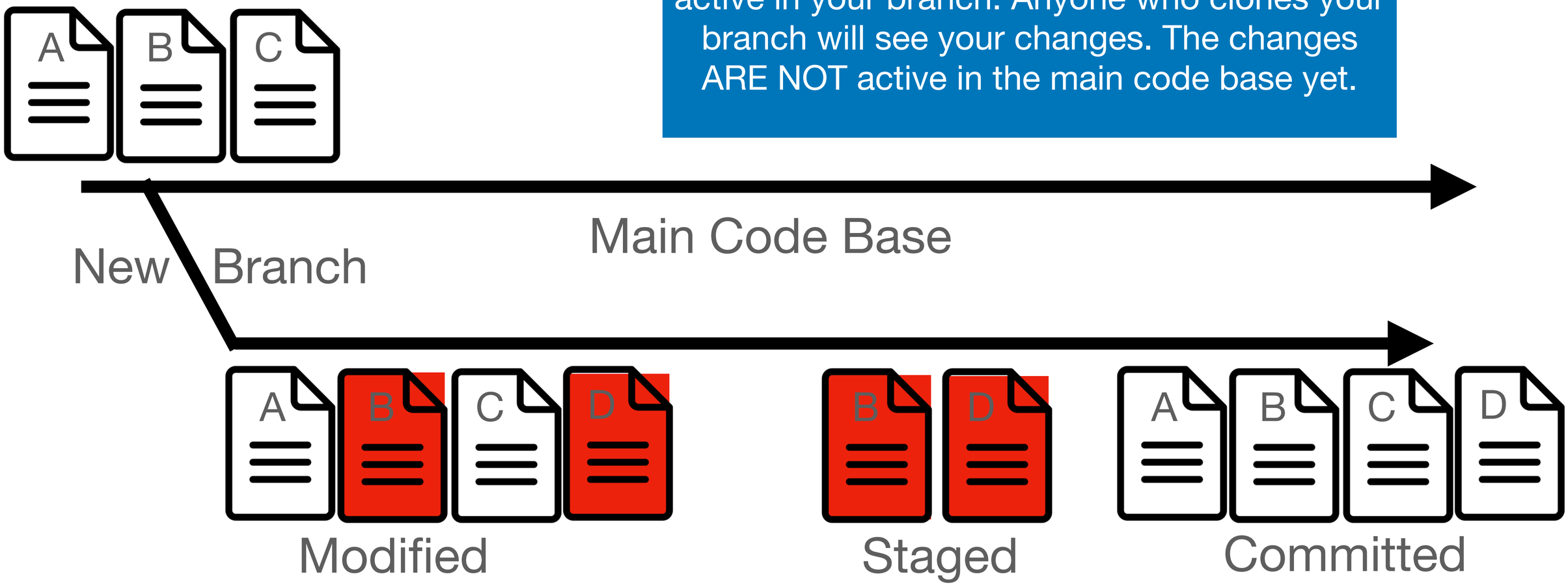
Entering the Staged State

The Staged State prepares your changes for being merged back into the master branch. It contains only the code you've modified



Committing your changes

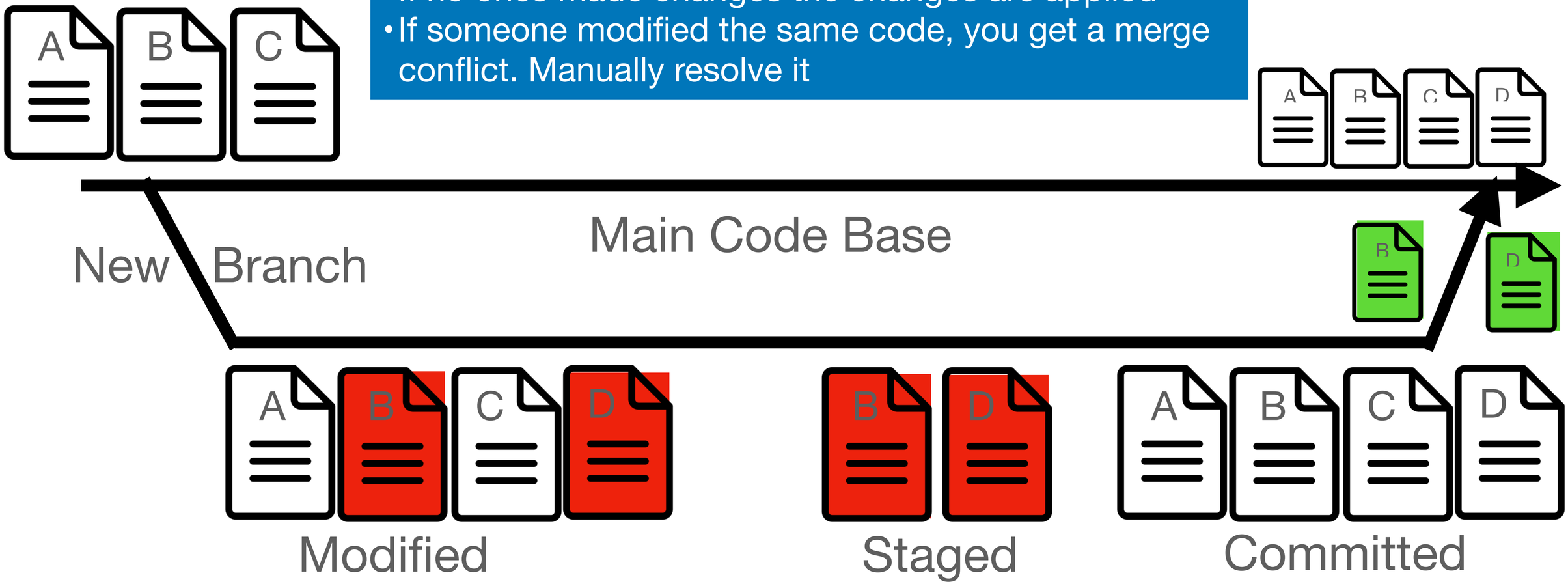
Committing your code, makes your changes active in your branch. Anyone who clones your branch will see your changes. The changes ARE NOT active in the main code base yet.



Merging your changes to main

By merging to Main you:

- Take all files that have modifications
- Compare them against the main branch.
- If no ones made changes the changes are applied
- If someone modified the same code, you get a merge conflict. Manually resolve it

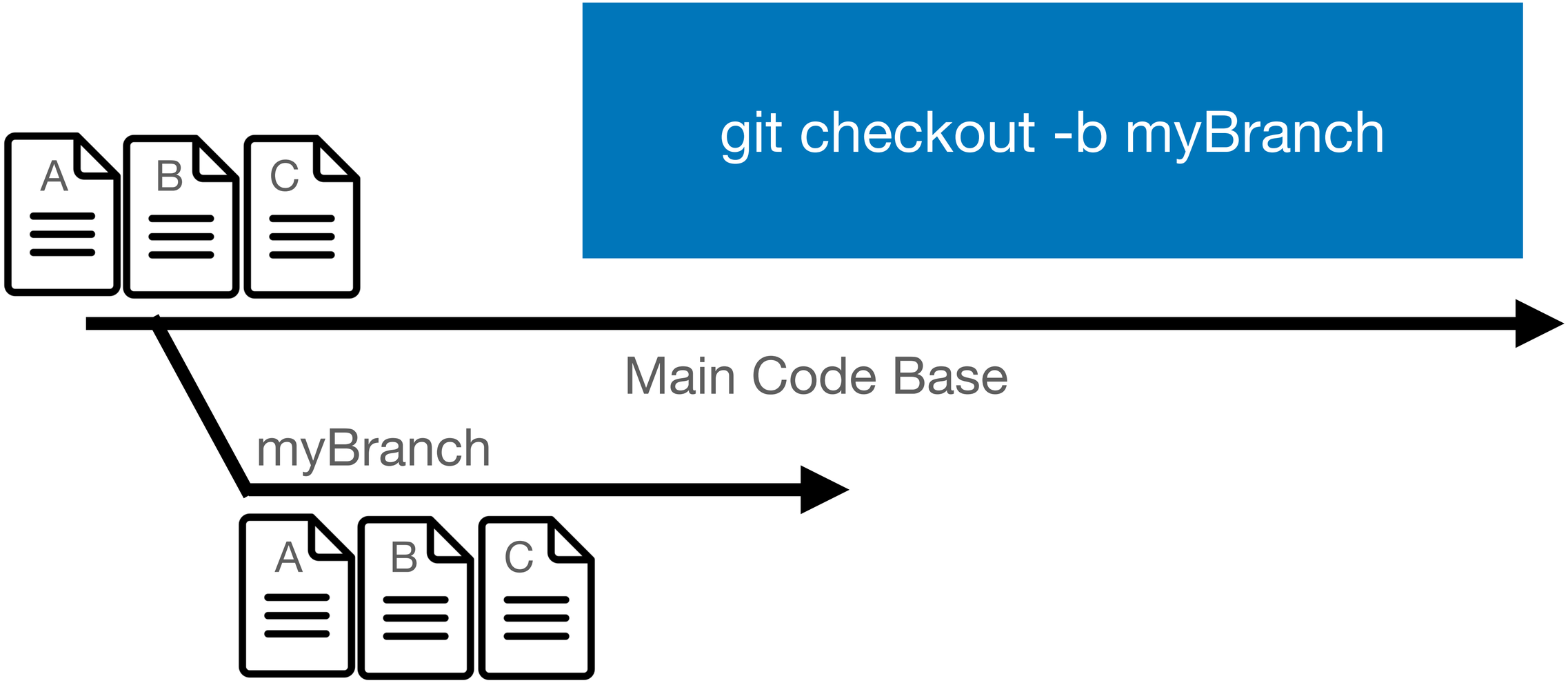


How does this tie into Git?

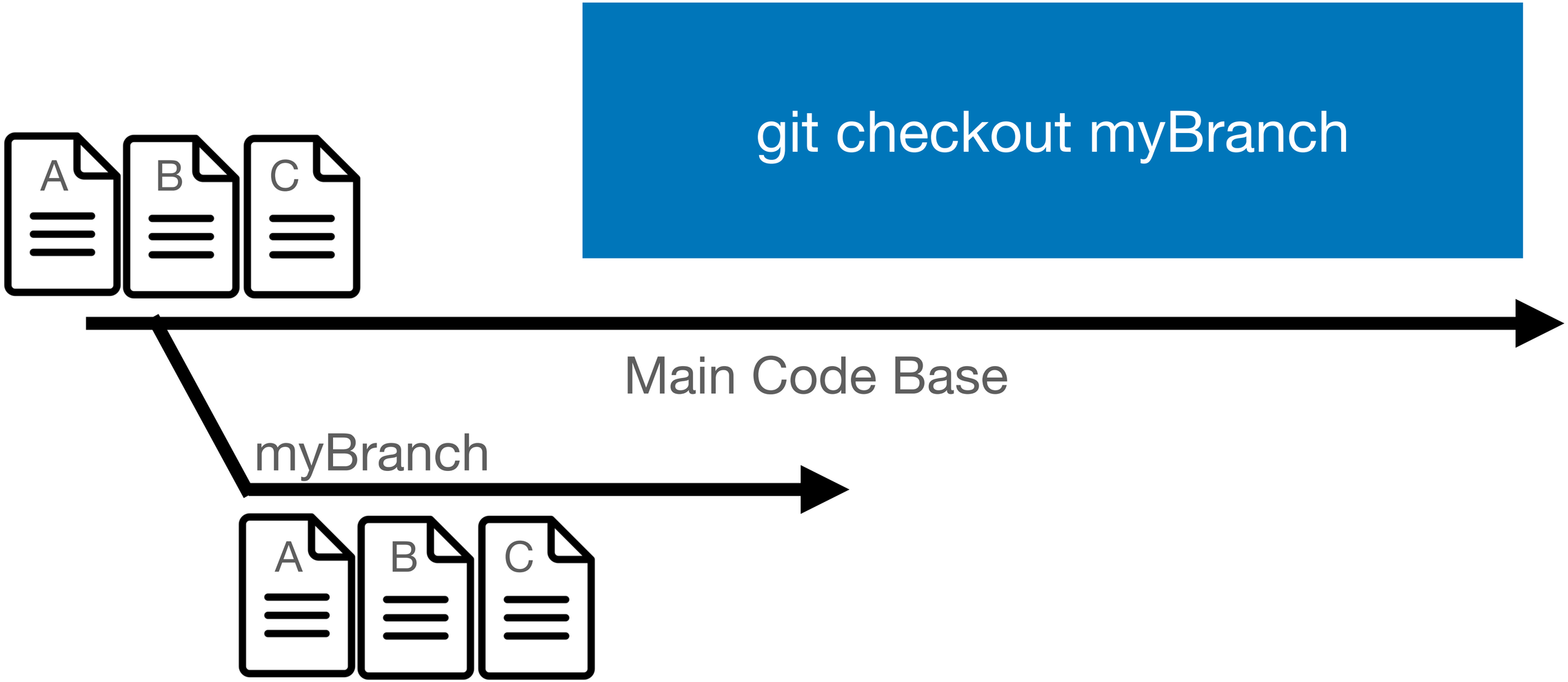
Creating a git repo


```
git init
 git remote add origin git@github.com:thelimeburner/testRepo.git
```

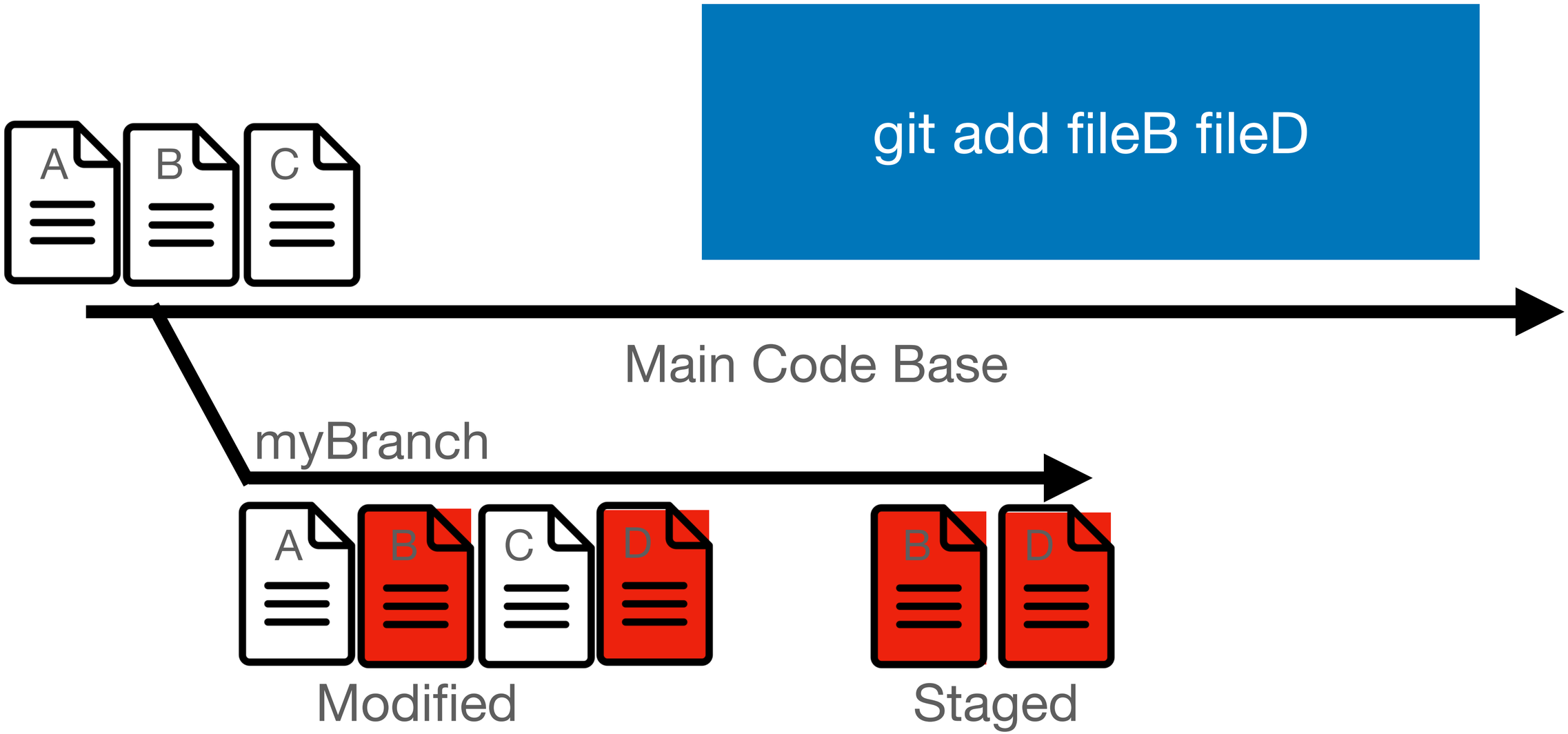
Creating a new branch



Switching to an existing branch



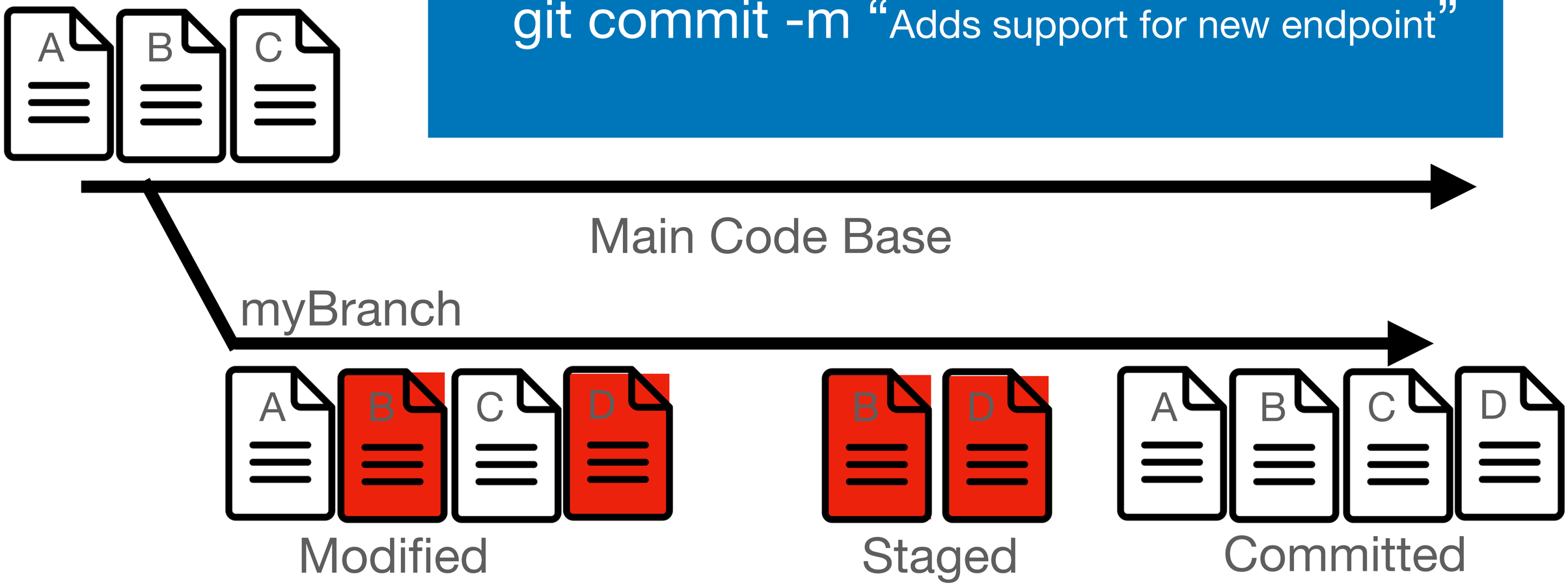
Entering the Staged State



Committing your changes

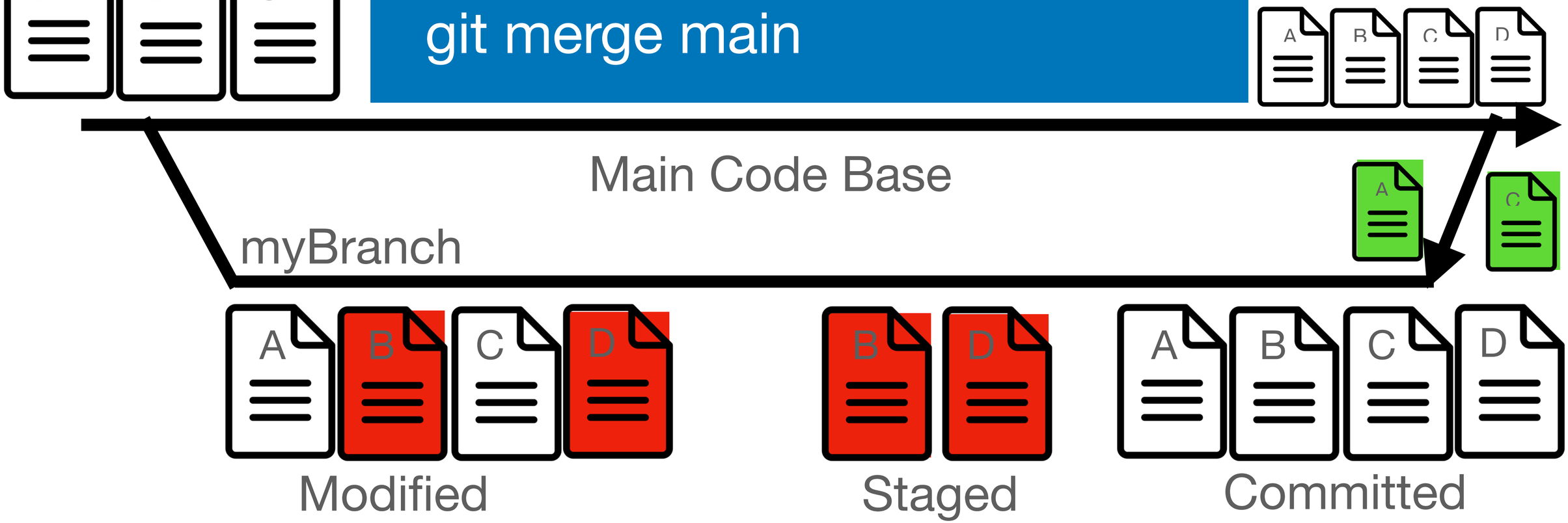
All commits need a human readable commit message

```
git commit -m “Adds support for new endpoint”
```

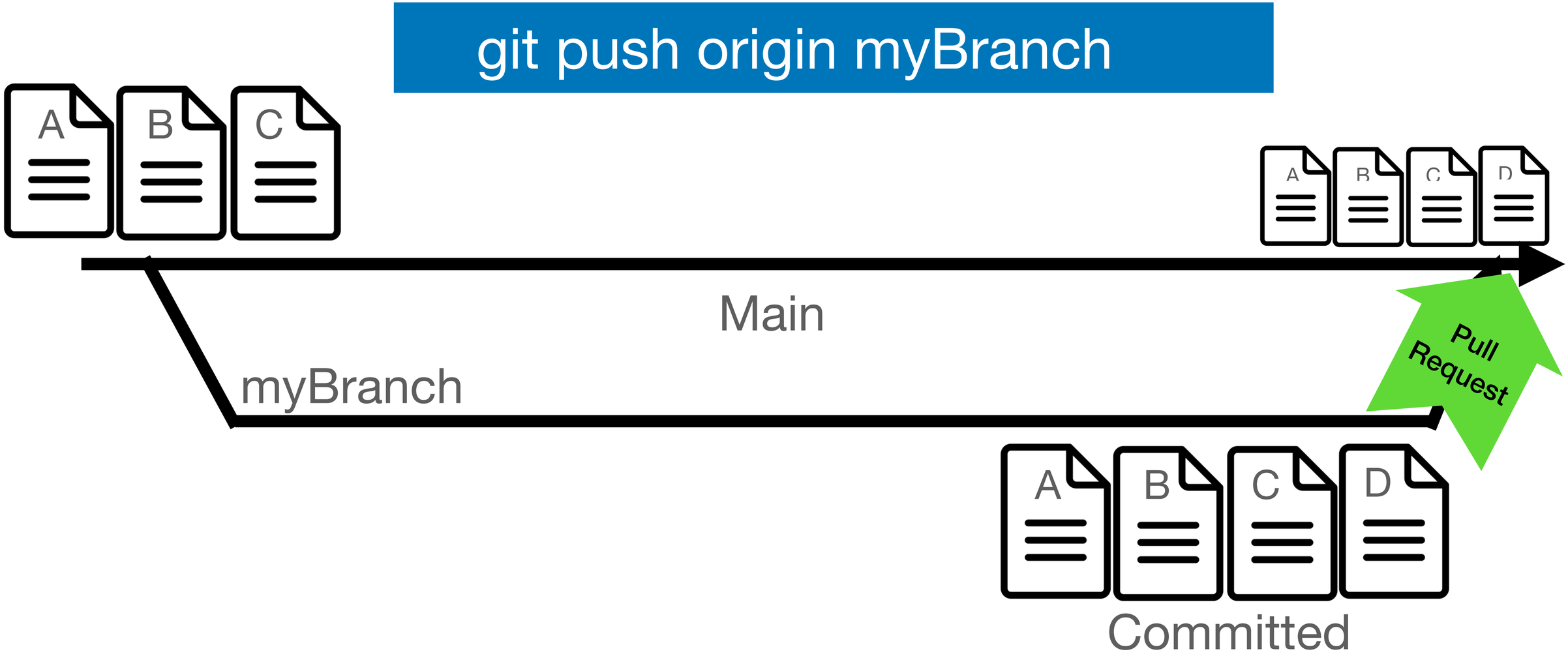


Bringing Mains changes back into your branch

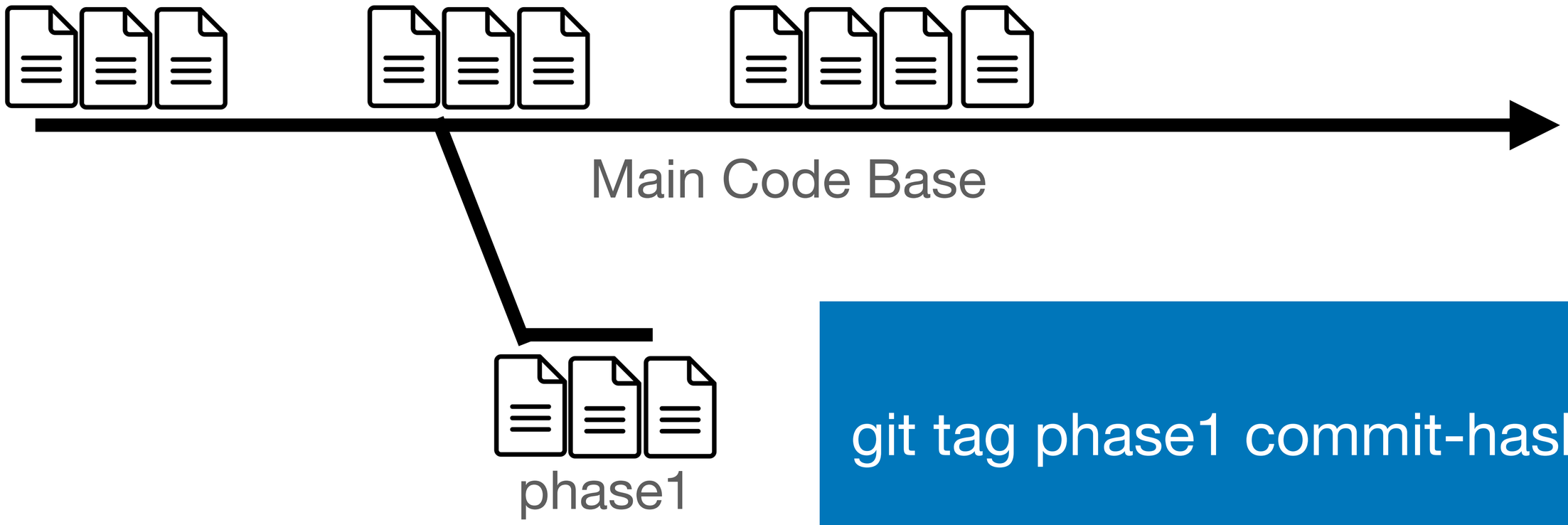
```
git checkout main
git pull main
git checkout myBranch
git merge main
```



Getting your changes into Main with Pull Requests



Marking an official release of your code



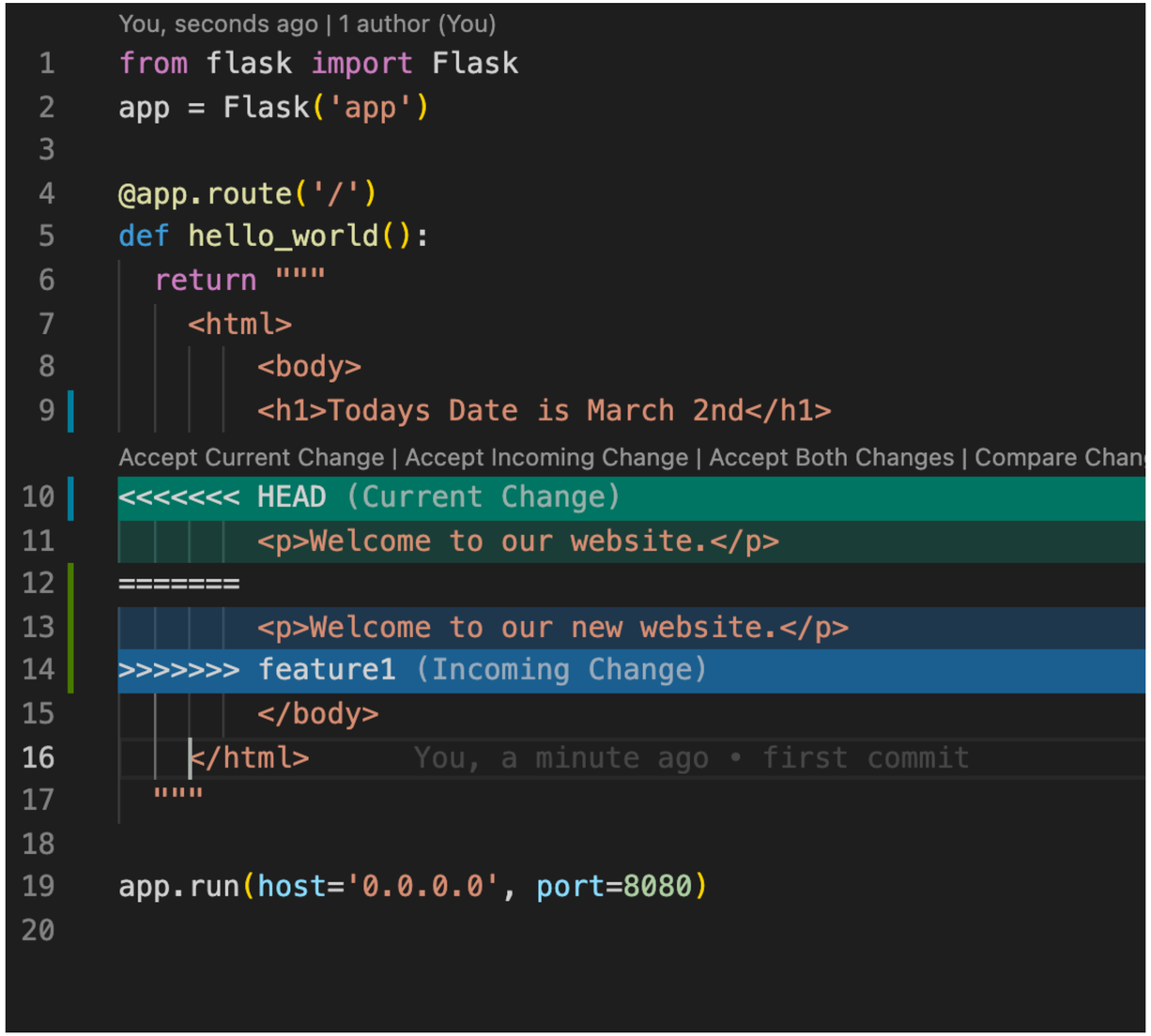

```
git tag phase1 commit-hash
```

Pull Requests

- Pull Requests allow you to inform others on your team about a new features or code being added to the codebase
- They provide a way for teams to discuss changes being made and enable an easy way to do code review
- Changes in a pull request display whats been modified and is to be merged into main if approved.
- Once a pull request is approved by reviewers. The code is merged into main and becomes apart of the codebase.

Merge Conflicts

Example of Merge Conflict



- Arise when two people edited the same line in a file.
- Require manual intervention
- You need to go into the file and decide which change should be persisted.
- Delete the line you don’t want along with the added lines from git.
- Commit changes and git merge again

Code Reviews

- Code Review is the process by which team members review each others code for things like
  - Bugs
  - Style choices
  - Dead code
  - Security issues
  - Design Decisions
  - and much more
- A good place to ask clarifying questions or act as knowledge transfer
- Code Reviews normally take place right before merging a branch into master and is usually an iterative process.
- Teams typically have rules that say a code change needs at least 1 review before merging

Sample Code

```
@app.route('/')
def hello_world():
  visitDate = "March 2rd 2020"
  x = """
    <html>
        <body>
        <h1>Todays Date is {0}</h1>
        <p>Welcome to our website.</p>
        </body>
    </html>
  """
  #print("DebuggCode")
  # x = 0
  return outputText.format(x)

@app.route('/endpoint2')
def endpoint2():
  visit_date = "March 3rd"
  print("DEBUGG, VISITED")
  output_text = """<html>
        "<body>"
        <h1>Todays Date is {1}</h1>
        <p>Welcome to our website.</p>
        </body>
    </html>
  """
  return output_text.format(visit_date)

app.run(host='0.0.0.0', port=8080)
```

```python
@app.route('/')
def hello_world():
  visitDate = "March 2rd 2020"
  x = """
    <html>
        <body>
        <h1>Todays Date is {0}</h1>
        <p>Welcome to our website.</p>
        </body>
    </html>
  """
  #print("DebuggCode")
  # x = 0
  return outputText.format(x)

@app.route('/endpoint2')
def endpoint2():
  visit_date = "March 3rd"
  print("DEBUGG, VISITED")
  output_text = """<html>
        "<body>"
        <h1>Todays Date is {1}</h1>
        Welcome to our website.
        </body>
    </html>
  """
  return output_text.format(visit_date)

app.run(host='0.0.0.0', port=8080)
```

Variable naming style is not consistent, Date is incorrect

Use a descriptive variable name

Remove Dead code

Remove debug statements to keep code clean

This should be a zero instead of a 1

Should we wrap these in a paragraph tag?

A sample Code Review on Github

thelimeburner / demo-aws-app

Unwatch 1 | Star 0 | Fork 0

Code | Issues | Pull requests 1 | Actions | Projects | Wiki | Security | Insights | Settings

Feature Add New Paragraph #1

Edit | Open with

Open **thelimeburner** wants to merge 1 commit into `master` from `feature/new-paragraph`

Conversation 0 | Commits 1 | Checks 0 | Files changed 1 +3 −2

Changes from **all commits** | File filter... | Jump to... | 0 / 1 files viewed | Review changes

5 index.js Viewed

```
@@ -8,7 +8,8 @@ const color = "#"+((1<<24)*Math.random()|0).toString(16);

 app.get('/', (req, res) => {
     const ipaddr = ip.address();
-    let html = "<html><body bgcolor=\""+color+"\"><h1 align=\"center\">Hello from "+ipaddr+"</h1></body></html>";
+    let html = "<html><body bgcolor=\""+color+"\"><h1 align=\"center\">Hello from "+ipaddr+"</h1>
+    <p>This is a sample demo app that demonstrates how a load balancer works.</p></body></html>";
     res.send(html);
 });

@@ -47,4 +48,4 @@ app.get('/aws',async (req,res)=>{

-app.listen(port, () => console.log(`Example app listening at http://localhost:${port}`))
+app.listen(port, () => console.log(`Example app listening at http://localhost:${port}`))
```

ProTip! Use `n` and `p` to navigate between commits in a pull request.

Tips for working with Distributed Version Control

- Use branch protection rules to protect your main branch from being changed without code review
- Always pull the latest changes before trying to merge to master.
- Try to keep pull requests to small changes that are atomic. This simplifies code review.
- Name new branches **feature/new-feature** or **bugfix/fixing-bad-logic** to make it easy to understand what a branch does.
- Use git tags to mark official releases that never change.
- Incorporate Peer Review into your git workflow.

Assignment 1

https://classroom.github.com/g/lhuM3Li7

- Together with your team setup a repo from the link above
  - Name it based on your team tag
- Open the README.md in the assignment and complete the assigned tasks
  - Create a new endpoint and practice the git workflow
  - Record your team norms and code review standards