

Teams!

Shopping Cart
due tomorrow!

9. Agile Development

CSCI 2541 Database Systems & Team Projects

Wood

This project is huge, how do we possibly get started working on it?

How do we decide who
should work on what?

How do we make sure
we can deliver on time?

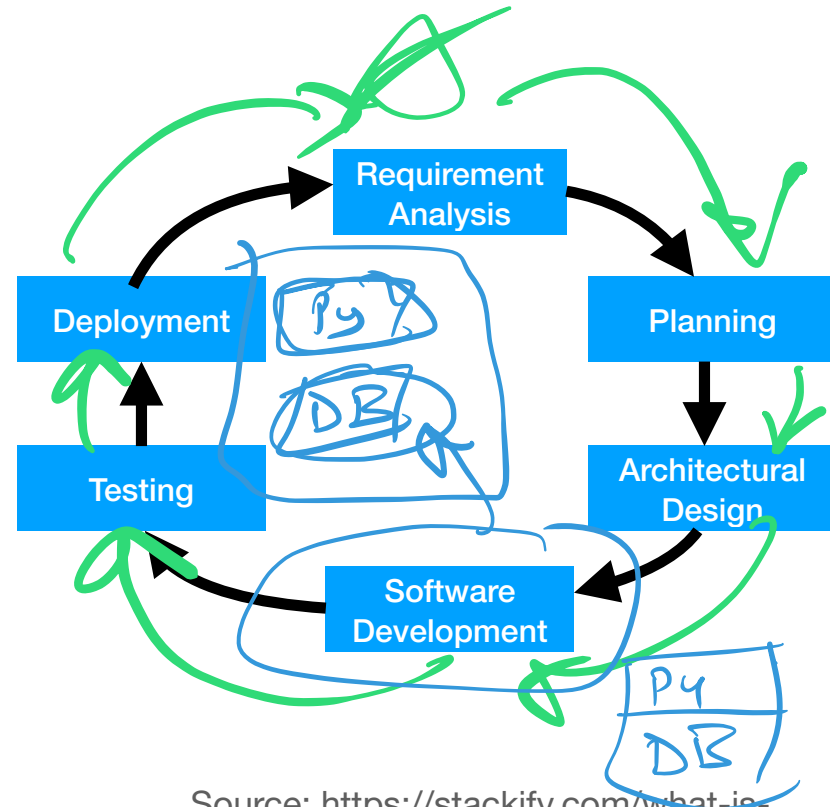
How do we make progress
without being in the same place?

Software Development Lifecycle (SDLC)

“A process that produces software with the highest quality and lowest cost in the shortest time possible.”

Provides a set of phases “to quickly produce high-quality software which is well-tested and ready for production use.”

A strong emphasis on testing, as you need to ensure code quality at every release.



Agile

vs Iterative / Waterfall design

A methodology for delivering software to customers faster and with limited headache.

Instead of having large product launches, product features are released in smaller increments.

Requirements and designs are continuously reevaluated to make teams flexible to change

Key concepts are open communication, collaboration, adaptability, trust.

A value system rather than a framework or defined set of steps.

If Agile is just a set of values how does it help us?

This is where Scrum and Kanban come in.

Scrum

Scrum

Scrum is a framework for getting things done

Scrum encourages teams to learn from their past, work together on problems, and reflect to always improve.

Designed to allow teams to improve and be adaptive to change as they work on projects and reprioritize features as needed.

Work is organized into increments that are completed in sprints.

Releases generally happen at the end of a sprint or a series of sprints.

More Info: <https://www.atlassian.com/agile/scrum>

Scrum Concepts

Scrum board - used to represent stories in progress.

Sprint - A period of time (usually 2 weeks) of which work will be completed.

A User Story - The smallest unit of work, usually written in human readable terms

- i.e. When I click the login button, I am logged into the website.

Product Backlog - The total list of all todo stories relating to the product. Includes new features, bug fixes, and enhancements

Sprint Backlog - The list of all todo stories to be completed by the end of the sprint.

Increment - The goal for the end of the sprint. What should be completed at the end of your sprint period.

The Scrum Ceremonies

Backlog Grooming: Prioritize items and clean stories to keep the backlog up to date

Sprint Planning: Deciding what stories will be undertaken for the next sprint.

Scrum master / lead

Sprint: The time period during which developers actually undertake the work for the increment and make progress. Usually around 2 weeks.

Daily Stand Up: Allows for everyone to update their status and keep everyone on the same page and voice concerns.

Sprint Review: Team gets together to demo what was completed during the sprint.

Sprint Retrospective: Team meets to discuss things that went well and that need work.

Scrum in Review

A incremental process that favors working in sprints and deploying features on a set release schedule.

Key metric is burndown, the number of stories completed in a sprint.

It's the entire teams responsibility to learn from past mistakes and work together to grow.

Can work great, but its structured timeline is hard to adapt to our projects

Kanban



Kanban

Relies on Real-time communication of capacity and full transparency of work

The amount of work in progress is matched to team capacity to make sure things are kept on schedule and no one is overwhelmed.

Work items are represented visually on a kanban board so any team member can view the status of items.

The kanban board is the single source of truth for progress. All impediments and blockers are clearly made visible.

It's the entire team's responsibility to ensure items are moving efficiently through the process.

Releases generally happen as features make its way through the whole process.

More info: <https://www.atlassian.com/agile/kanban>

Kanban Concepts

Kanban Board - A visual board with columns representing the state of work. i.e. Todo, In Progress, Code Review, Done

A **User Story** - The smallest unit of work, usually written in human readable terms.

Product Backlog - The total list of all todo stories relating to the product.

WIP - The work in progress limit. The total number of stories a developer can take on at a given time.

Cycle Time - The amount of time it takes for a unit to travel through the process from In Progress to Done.

A Sample Kanban Board

thelimeburner / demo-aws-app

Unwatch 1 Star

Code Issues 7 Pull requests Actions **Projects 1** Wiki Security Insights Settings

Sample Kanban Board

Updated 6 minutes ago

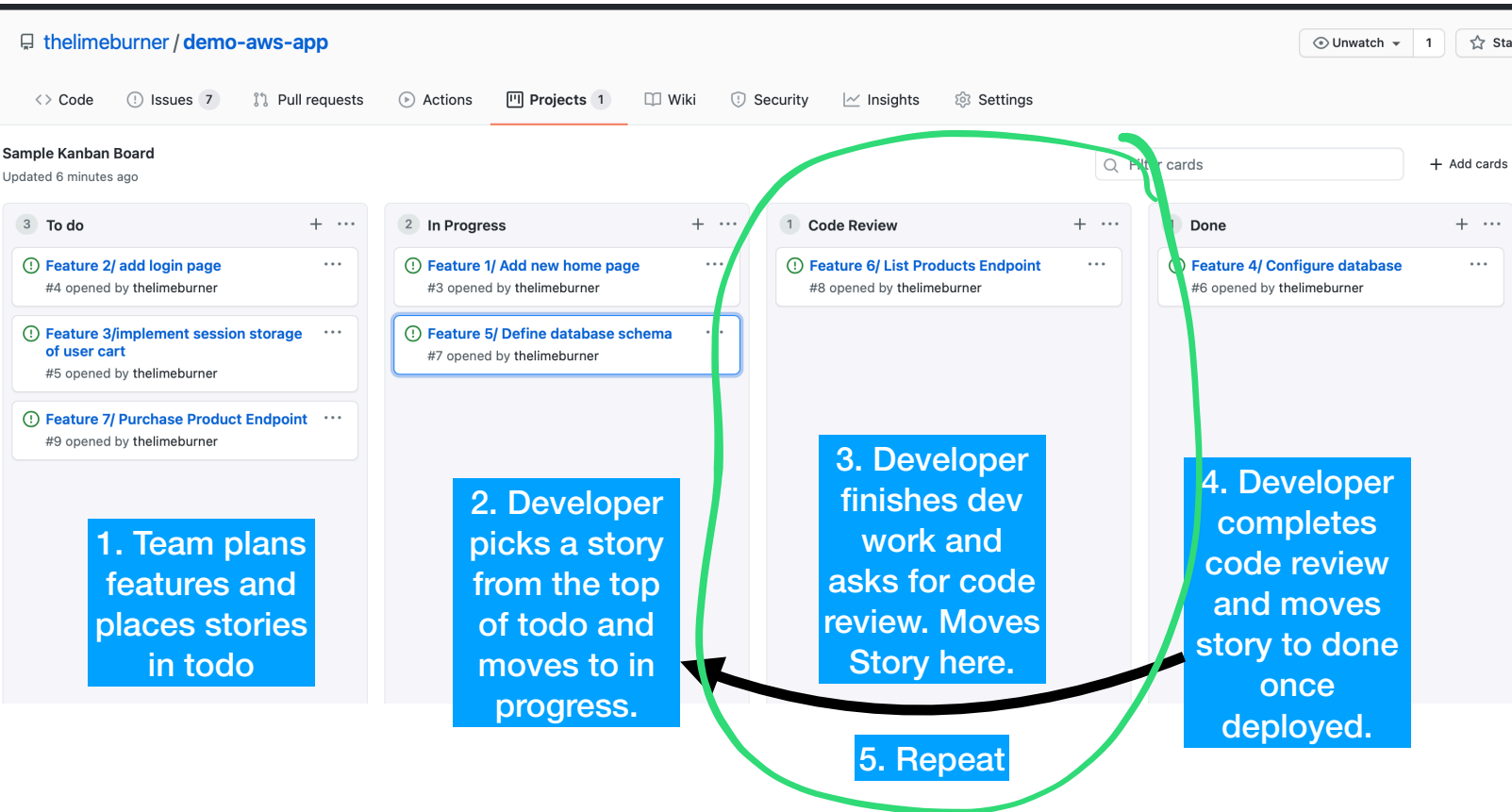
Filter cards + Add cards

The Kanban board is organized into four columns:

- To do (3 items):**
 - Feature 2/ add login page (#4 opened by thelimeburner)
 - Feature 3/implement session storage of user cart (#5 opened by thelimeburner)
 - Feature 7/ Purchase Product Endpoint (#9 opened by thelimeburner)
- In Progress (2 items):**
 - Feature 1/ Add new home page (#3 opened by thelimeburner)
 - Feature 5/ Define database schema (#7 opened by thelimeburner)
- Code Review (1 item):**
 - Feature 6/ List Products Endpoint (#8 opened by thelimeburner)
- Done (1 item):**
 - Feature 4/ Configure database (#6 opened by thelimeburner)

A green arrow points from the 'To do' column to the 'In Progress' column, indicating a transition of a task.

The Kanban Process



Kanban in Review

A continuous process that favors limiting work in progress and deploying features as they finish.

Key metric is minimizing the cycle time for a story

It's the entire teams responsibility to keep stories moving through the process.

Low overheard framework and very flexible to change.

Course Project

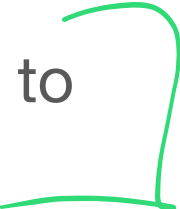
You must follow the Kanban Agile Dev Methodology

We will use Trello to host project boards

- Each team must create its own board (free account)
- Invite mentor and instructors

Break the project down into user stories 

Use Trello to track who is working on what

We will review your board and commit history to track how work was divided up! 

User Story

Describes a user interaction with the system with a well defined output/result

① *Clicking “login” lets a user log into the website*

- User should enter username and password *in a form*
- Check form data against DB
- Store authenticated user info in session or return error

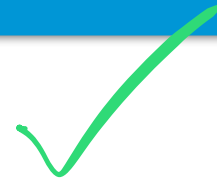
② *DB can store course offering information*

- Define tables for storing a course (department, number, title?) and a specific offering (course id, instructor, term, year?)
- Pre-load sample courses and offerings into DB and provide SQL syntax to insert and search for courses

Tips:

Try to keep user stories small

- Something you can do in a day



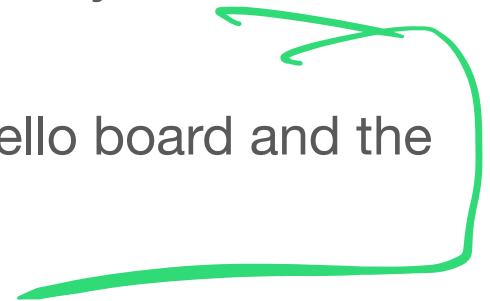
Use Trello's commenting feature to discuss issues / make suggestions

- Or ask for help from your mentor!



During Phase 2, you **cannot** talk with your Phase 1 teammates

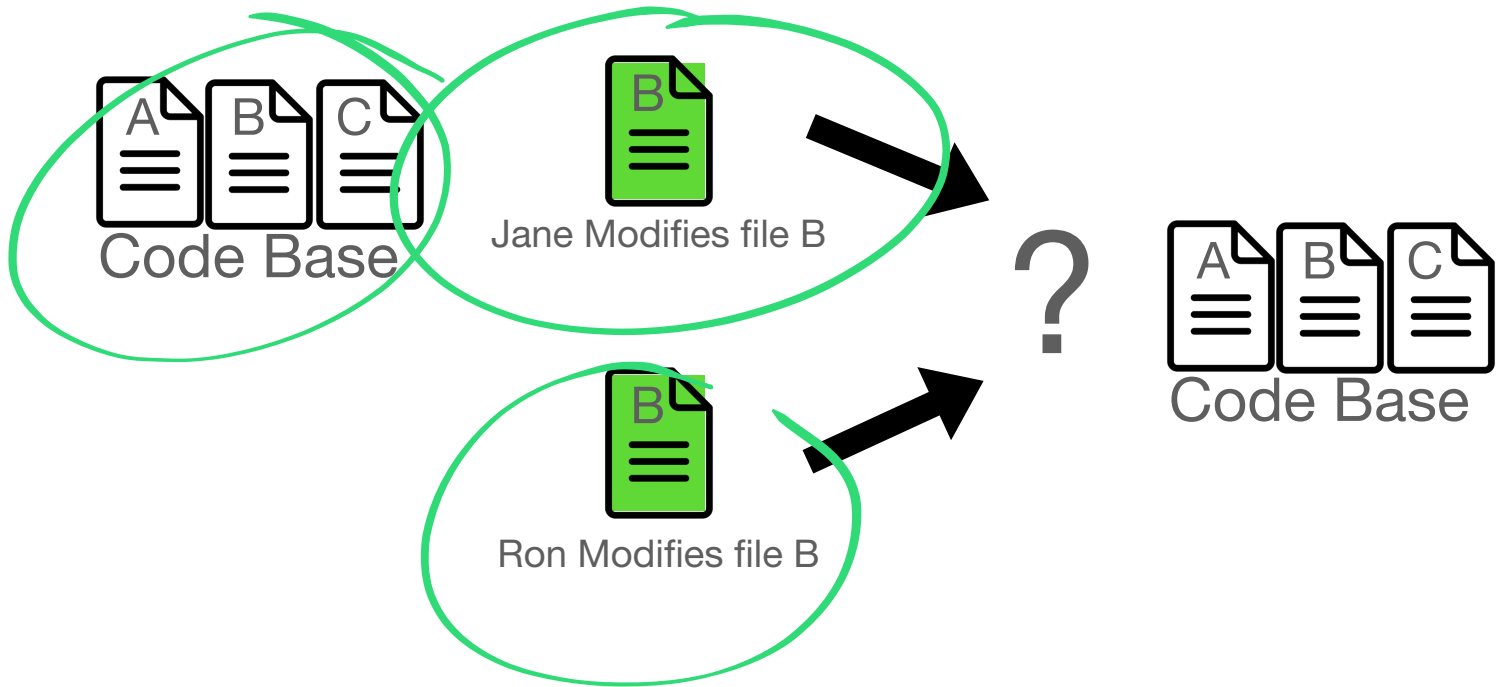
- But you can look back at your old Trello board and the documentation you write!



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

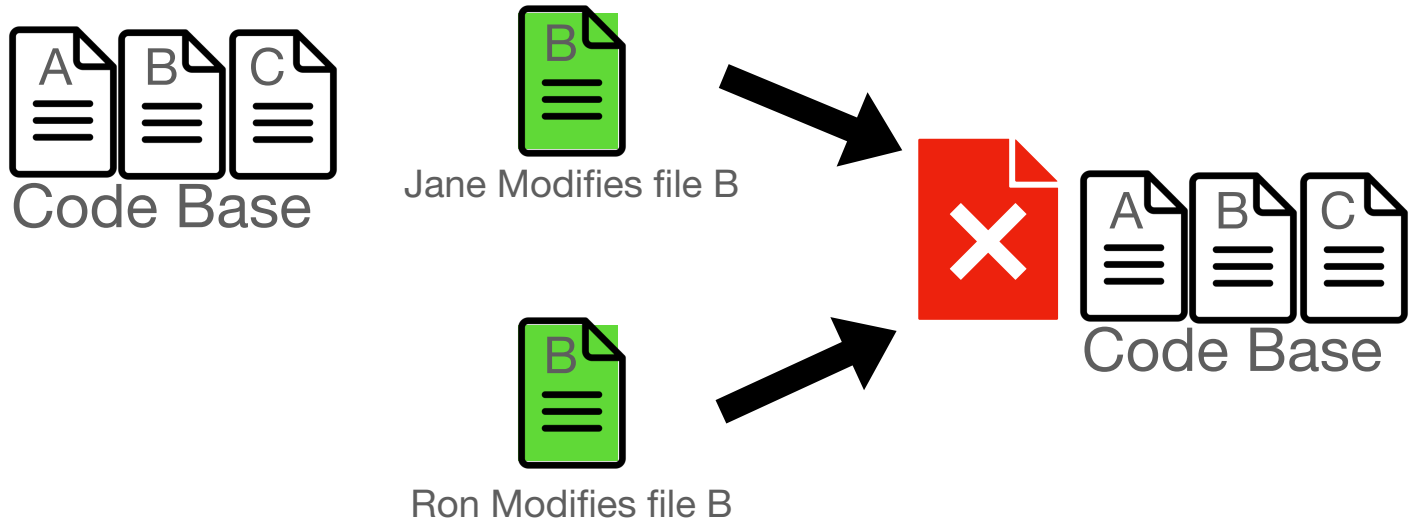
Why do we need version control

How do you decide whose changes to use?

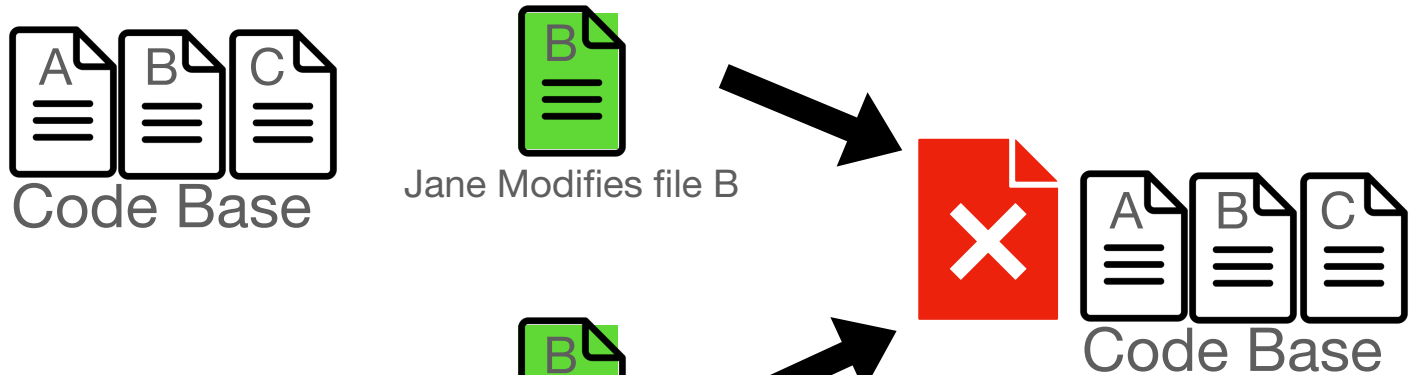


Why do we need version control

The files conflict without a reasonable way to decide.



Why do we need version control



So what do we do?
There's got to be a
better way...

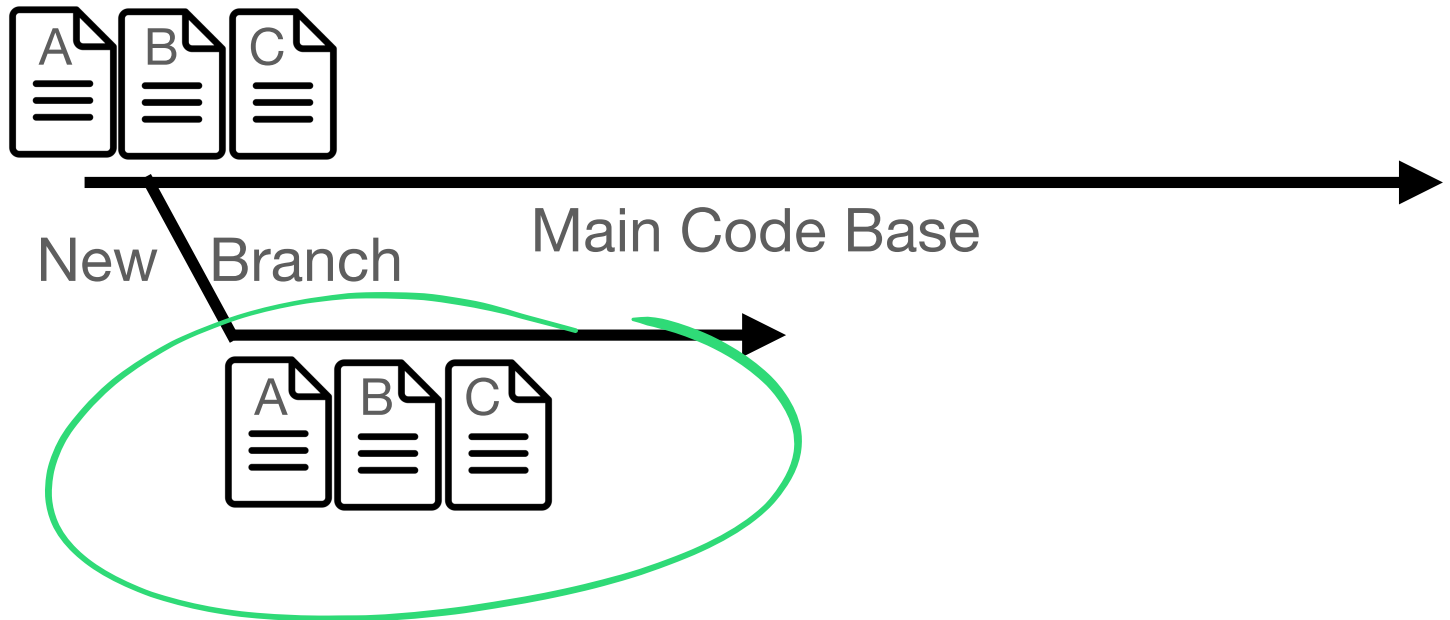
Distributed Version Control

Think of your codebase like the main timeline

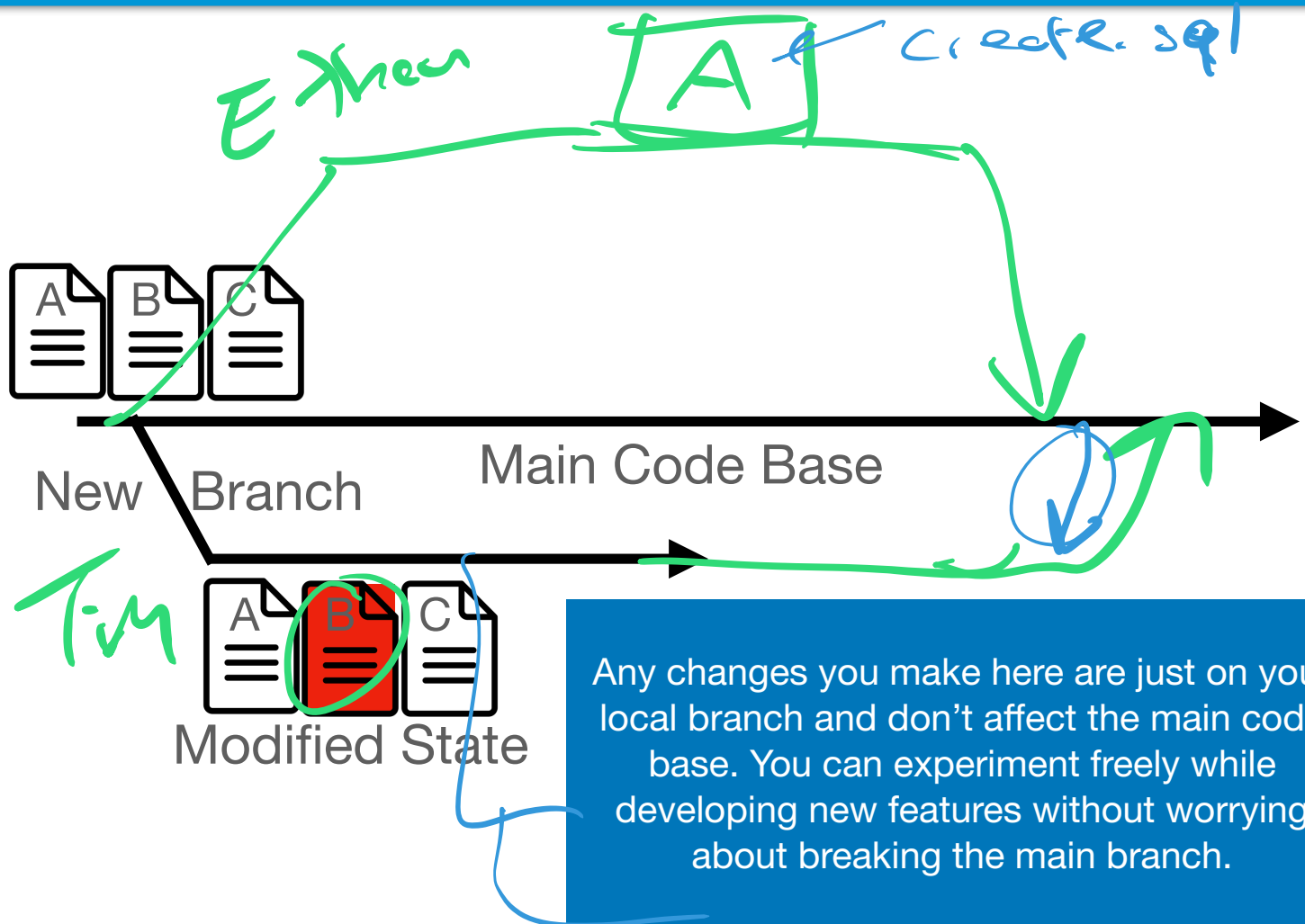


Main Code Base

To make changes you first create a new branch

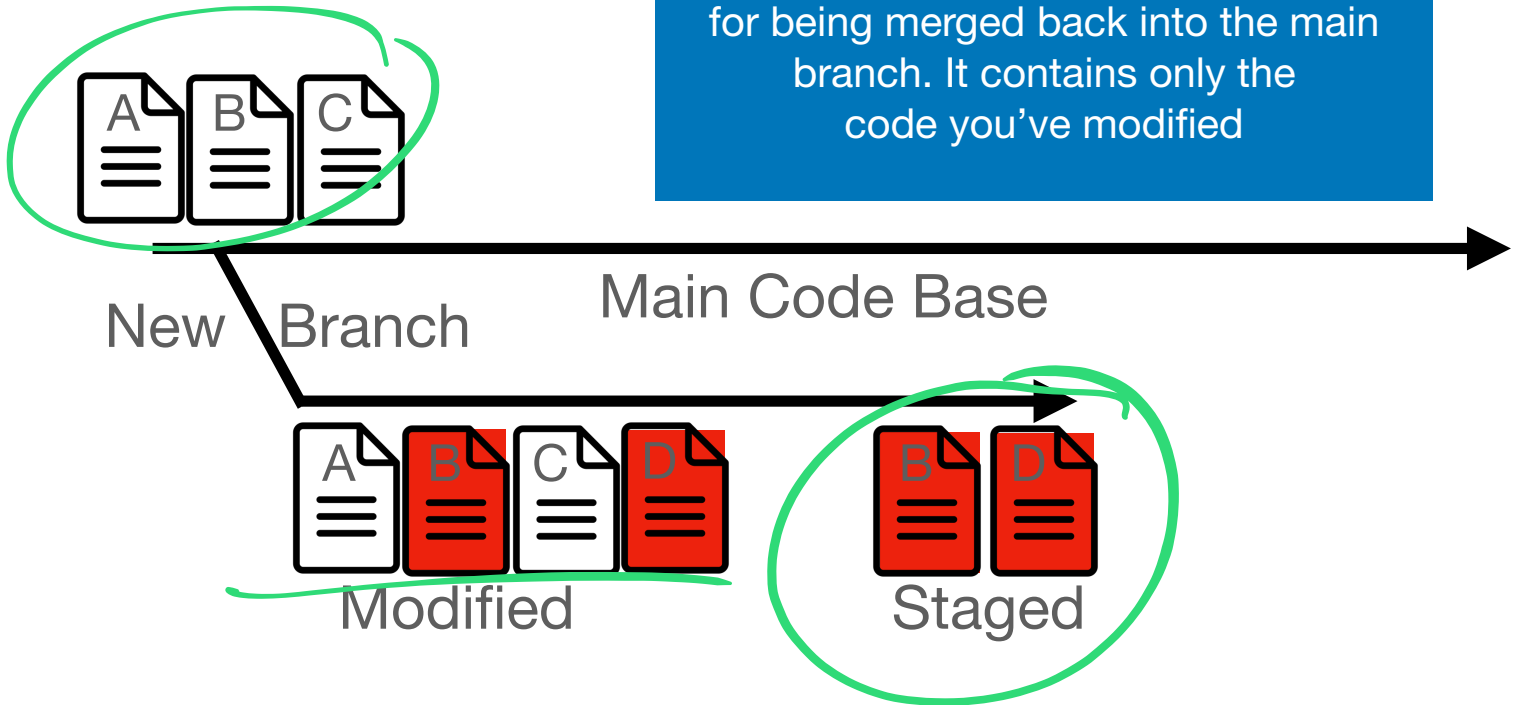


Entering the Modified State



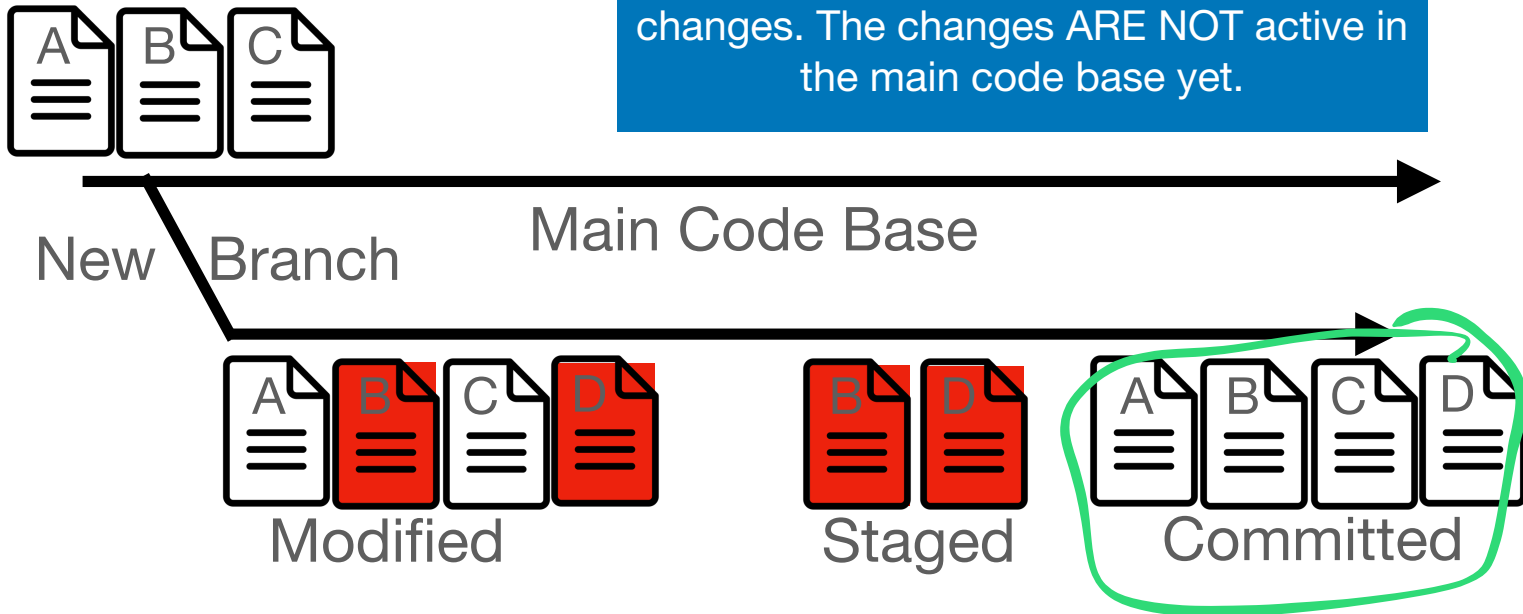
Entering the Staged State

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



Committing your changes

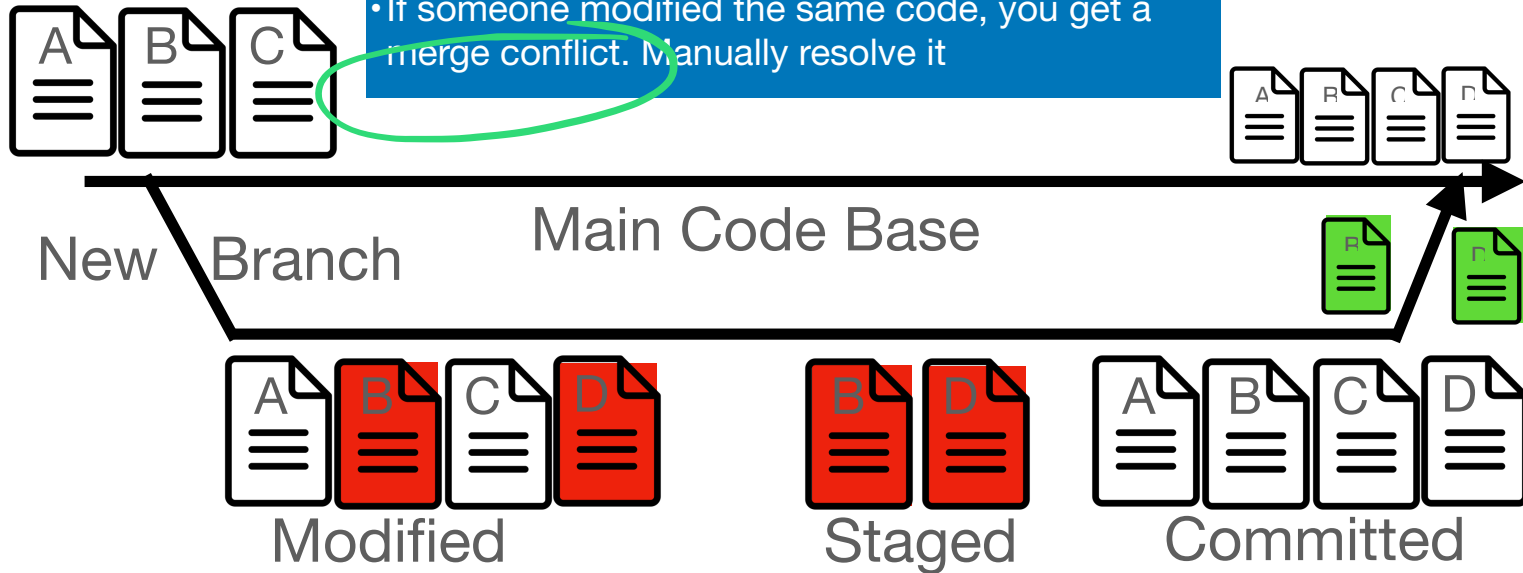
Committing your code, makes your changes active in your branch. Anyone who clones your branch will see your changes. The changes ARE NOT active in the main code base yet.



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



Main Code Base

```
git clone SOME_GIT_REPO
```

Creating a new branch

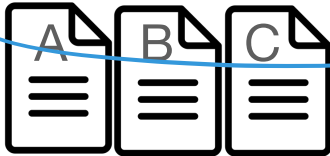
```
git checkout -b myBranch
```



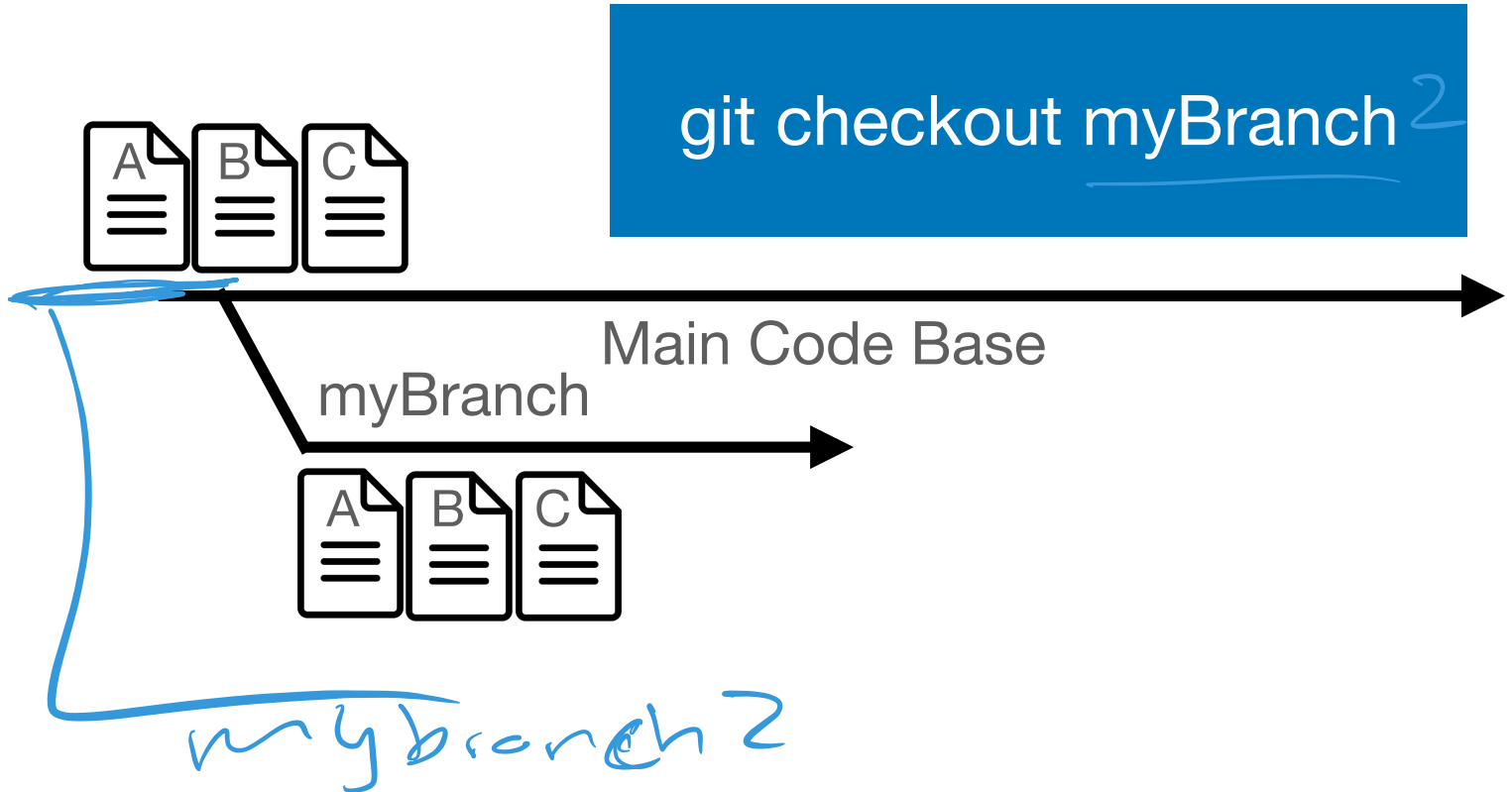
main

Main Code Base

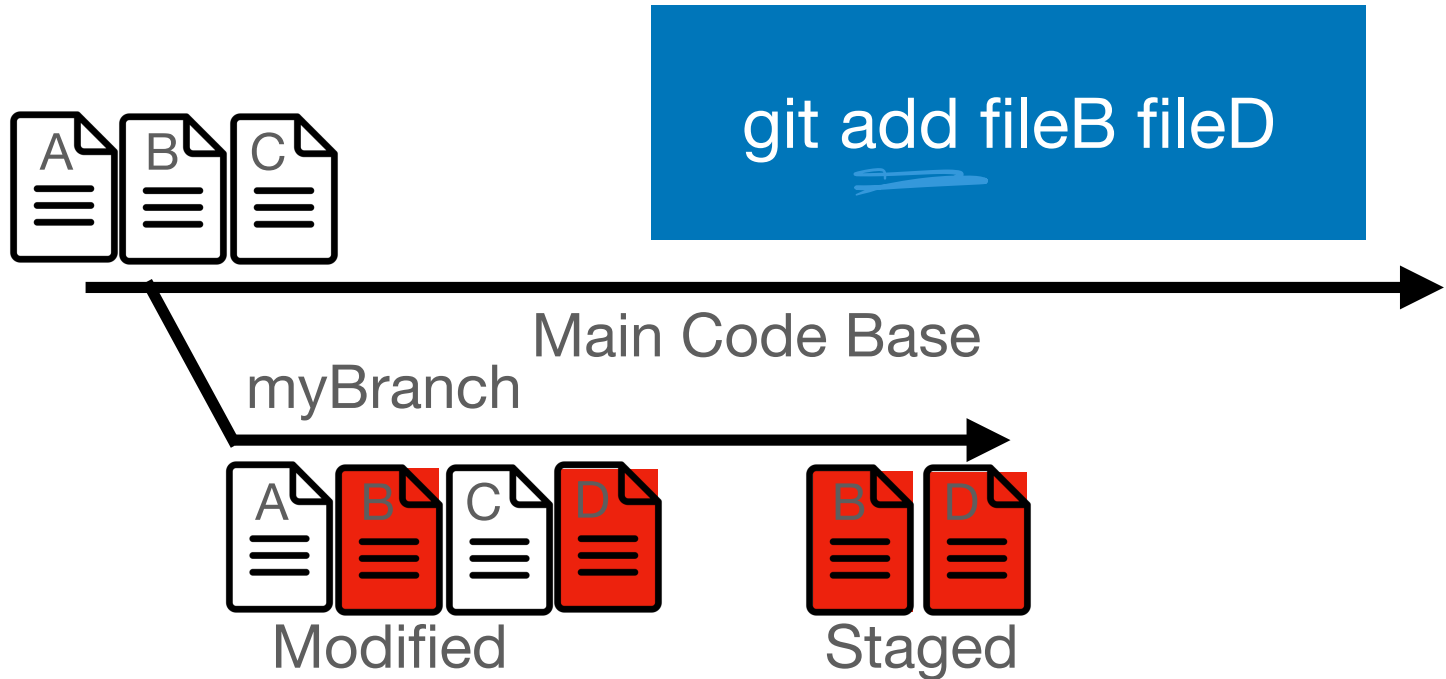
myBranch



Switching to an existing branch

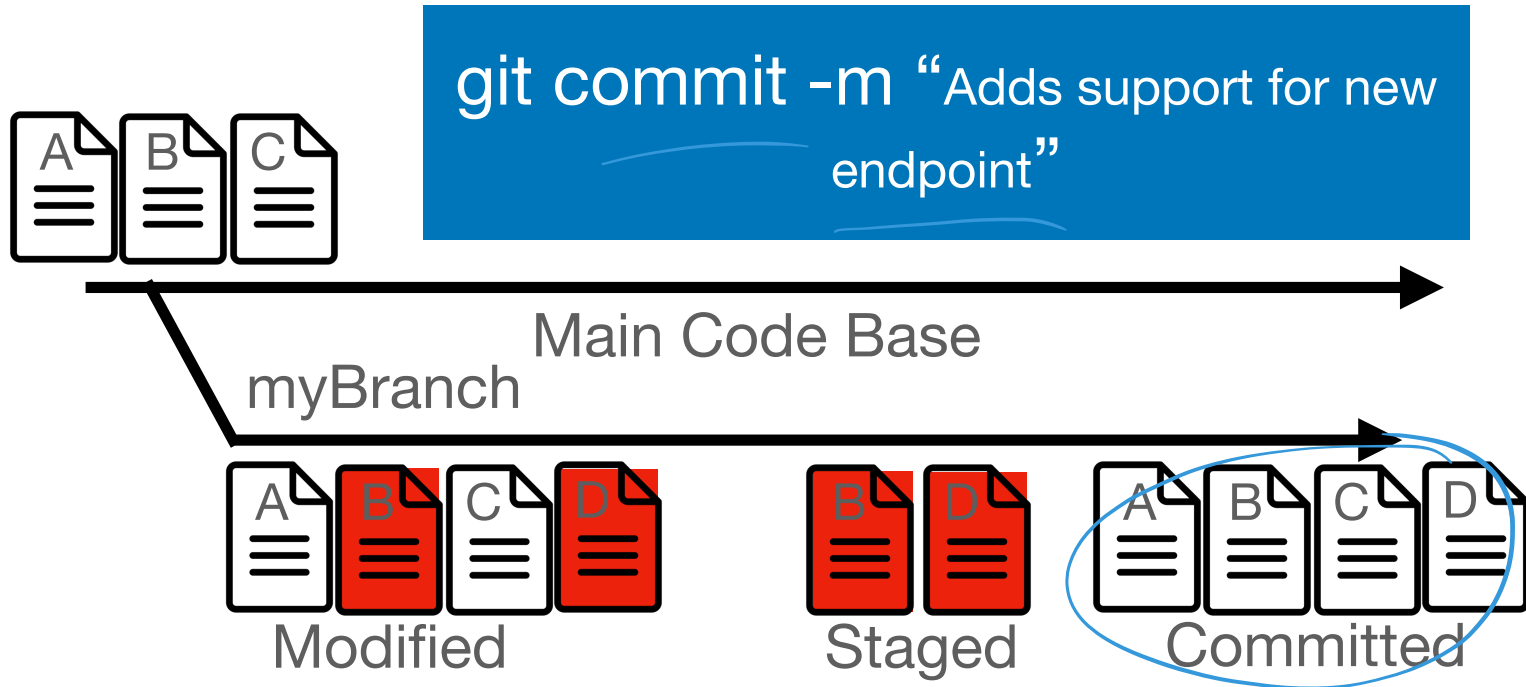


Entering the Staged State

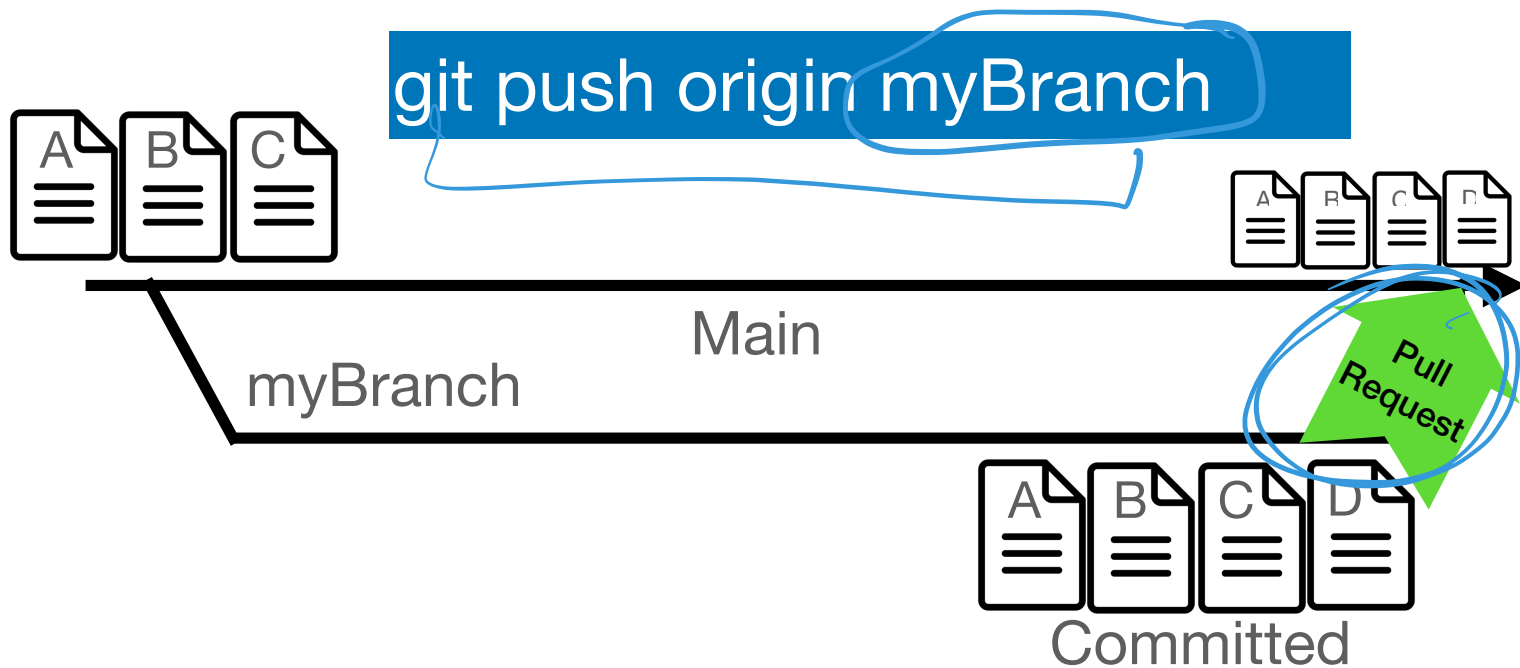


Committing your changes

All commits need a human readable commit message



Getting your changes into Main with Pull Requests



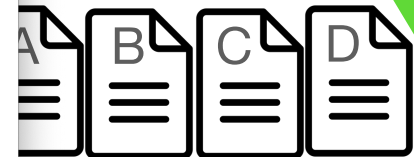
Getting your changes into Main with Pull Requests

The screenshot shows a GitHub Pull Request (PR) page for the repository 'sdnfv / openNetVM'. The PR title is 'Skeleton NF Functionality Improvements #313'. The PR is open and shows 15 commits being merged from the 'jettjacobs:develop' branch into the 'sdnfv:develop' branch. The PR description includes a summary of changes resolved from issue #312 and a list of additional functionality: 'Additional Command-Line Arguments' and 'Timer-based or Packet-based print delays'. The PR also includes a table with the following content:

This PR includes	
Resolves issues	
Breaking API changes	
Internal API changes	

The PR page also shows a comment from 'jettjacobs' dated Oct 29, 2021, and a list of reviewers, including 'dennisafa'. The PR is currently in progress and requires at least 2 approving reviews to be merged.

Branch



Committed

Pull Requests

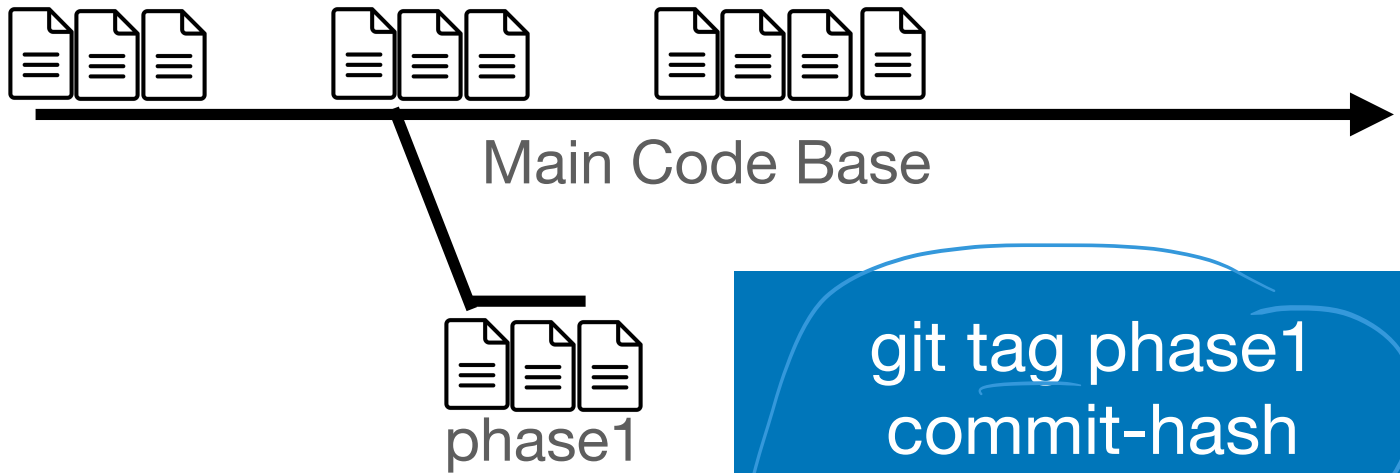
Pull Requests allow you to inform others on your team about a new features or code being added to the codebase

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 a part of the codebase.

Marking an official release of your code



Merge Conflicts

```
You, seconds ago | 1 author (You)
1  from flask import Flask
2  app = Flask('app')
3
4  @app.route('/')
5  def hello_world():
6      return """
7          <html>
8              <body>
9                  <h1>Todays Date is March 2nd</h1>
10 <<<<<< HEAD (Current Change)
11 <p>Welcome to our website.</p>
12 =====
13 <p>Welcome to our new website.</p>
14 >>>>> feature (incoming change)
15 </body>
16 </html> You, a minute ago • first commit
17 """
18
19 app.run(host='0.0.0.0', port=8080)
20
```

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 main 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)
```

```
@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)
```

Variable naming style is not consistent,
Date is incorrect

Use a descriptive variable name

Remove Dead code

```
@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)
```

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... Settings

0 / 1 files viewed Review changes

```
5 index.js
@@ -8,7 +8,8 @@ const color = "#"+((1<<24)*Math.random())|0).toString(16);
8
9 app.get('/', (req, res) => {
10   const ipaddr = ip.address();
11 -   let html = "<html><body bgcolor=\""+color+"\"><h1 align=\"center\">Hello from "+ipaddr+"</h1>
    </body></html>";
12   res.send(html);
13   });
14
...
@@ -47,4 +48,4 @@ app.get('/aws', async (req, res) => {
47
48
49
50 - app.listen(port, () => console.log('Example app listening at http://localhost:${port}'))
51 + app.listen(port, () => console.log('Example app listening at http://localhost:${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 main.

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.