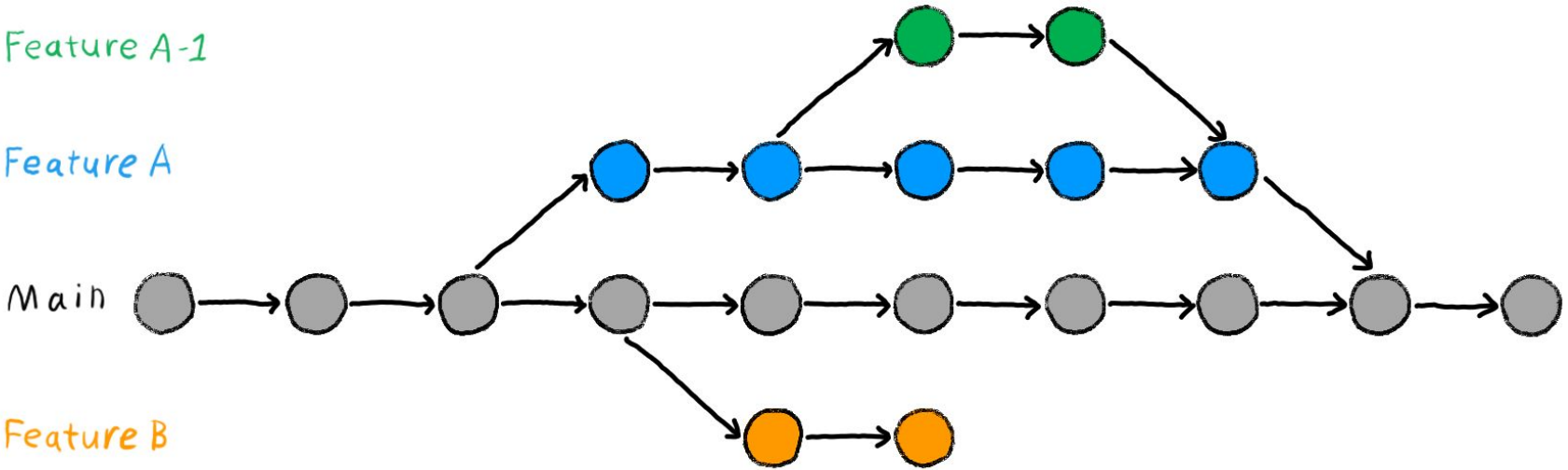
Lesson 1: Working in Parallel

Branches and Pull Requests

Reference:
Research Software Engineering with Python by Damien Irving, Kate Hertweck, Luke Johnston, Joel Ostblom, Charlotte Wickham, and Greg Wilson
https://merely-useful.tech/py-rse/git-advanced.html

Working in Parallel: Branches



Creating new branches

```
$ git branch
```

Check what branches exist in the repository

```
* master
```

The asterisk indicates which branch is currently active

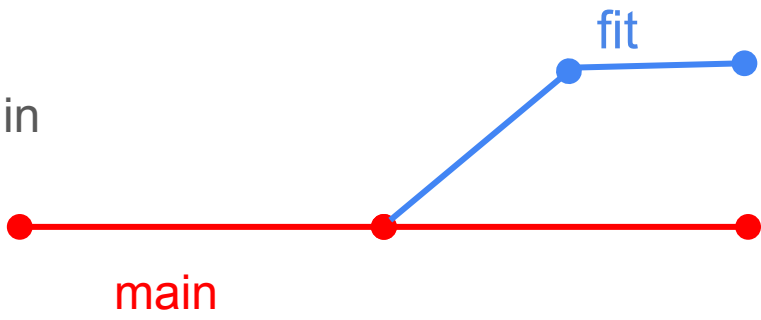
```
$ git branch fit
```

Create a new branch called “fit”

```
$ git branch
```

Check what branches exist in the repo

```
* master
  fit
```



Creating new branches

```
$ git branch
```

Check what branches exist in the repository

```
* master
```

The asterisk indicates which branch is currently active

```
$ git branch fit
```

Create a new branch called “fit”

```
$ git branch
```

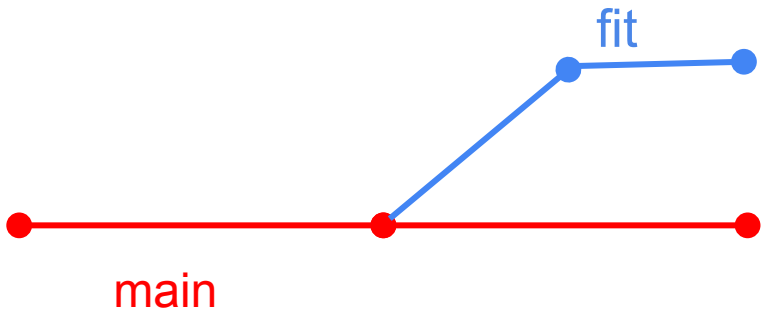
Check what branches exist in the repo

```
* master
  fit
```

In mid-2020, GitHub changed the name of the default branch from “master” to “main”. Owners of repositories can also change the name of the branch.

Checking out a new branch

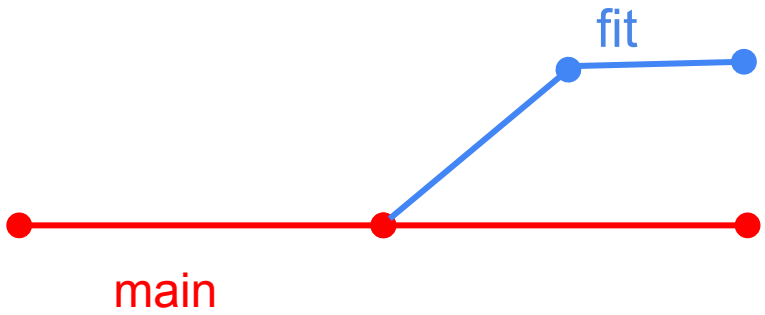
```
$ git checkout fit
$ git branch
```




```
  master
* fit
```

Checking out a new branch

```
$ git checkout fit
$ git branch
```




```
  master
* fit
```

Checking out a new branch: Tutorial

```
$ git branch

$ git branch [name]

$ git branch

$ git checkout name
```

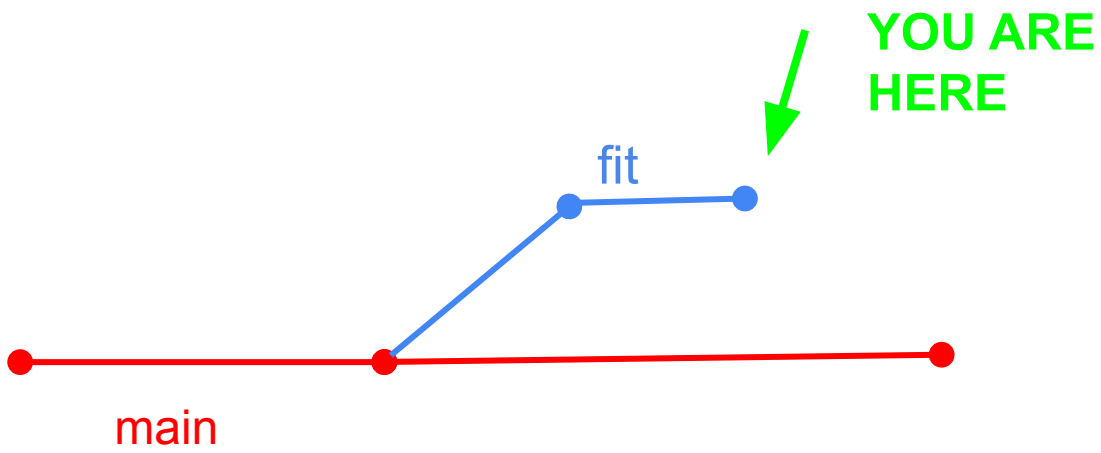
Committing Changes: Tutorial

```
$ git add .     OR        $ git add filename.py

$ git commit -m “your commit message here”

$ git push origin yourbranchname
```

Committing Changes: Tutorial



Merging

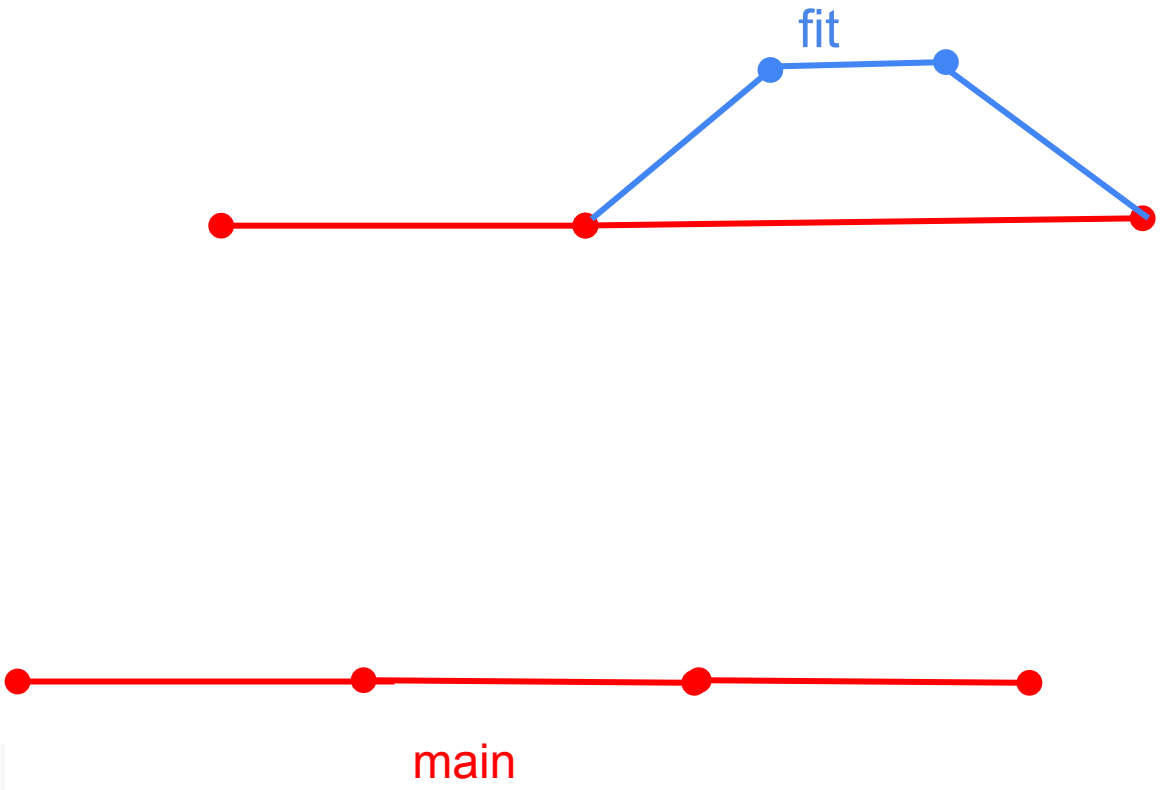
```
$ git checkout master
$ git branch
```

```
  fit
* master
```

```
$ git merge fit
```

```
$ git branch -d fit
```

```
Deleted branch fit (was 1577404).
```



Handling Conflicts: Merge or Rebase

```
$ git status
[Resolve the conflict]
$ git add <filename>

        $ git merge -- continue

        OR

        $ git rebase -- continue


$ git commit -m “resolved merge conflicts”
$ git push origin yourbranchname
```

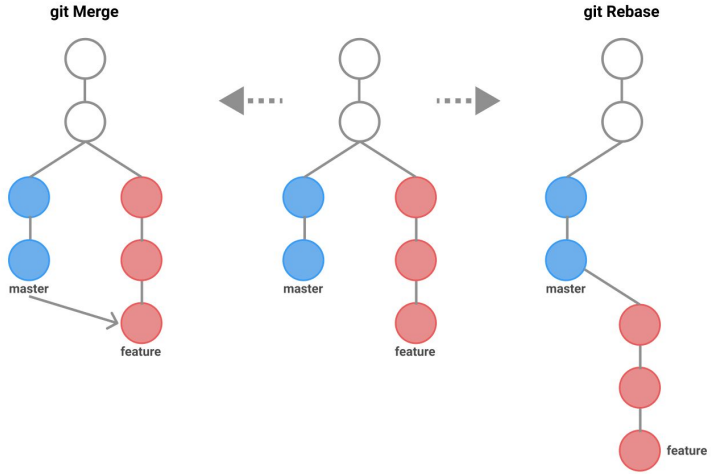
Handling Conflicts: Merge or Rebase

```
$ git status
[Resolve the conflict]
$ git add <filename>
```

```
$ git merge -- continue
```

OR

```
$ git rebase -- continue
```




```
$ git commit -m "resolved merge conflicts"
$ git push origin yourbranchname
```

Handling Conflicts: Merge or Rebase

```
$ git merge -- continue
```

- Integrates changes from one branch to another
- Result is a non-linear, bifurcating history
- Ideal for cases where you want to main a complete, chronological history

```
$ git rebase -- continue
```

- Rewrites commit history to apply changes from one branch to the tip of another
- Result is linear. All changes looks like they were made in a single series of steps, even though they were made in parallel
- Ideal for cleaning up commit history before merging into a main branch. Streamlines a sequence of commits before sharing with team

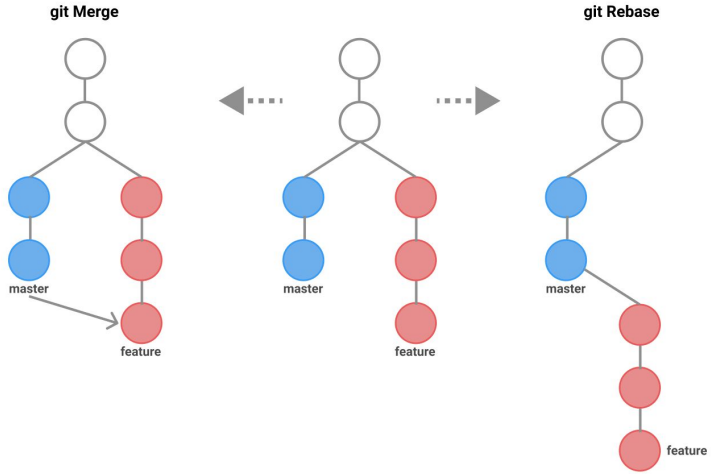
Handling Conflicts: Merge or Rebase

```
$ git status
[Resolve the conflict]
$ git add <filename>
```

```
$ git merge -- continue
```

OR

```
$ git rebase -- continue
```

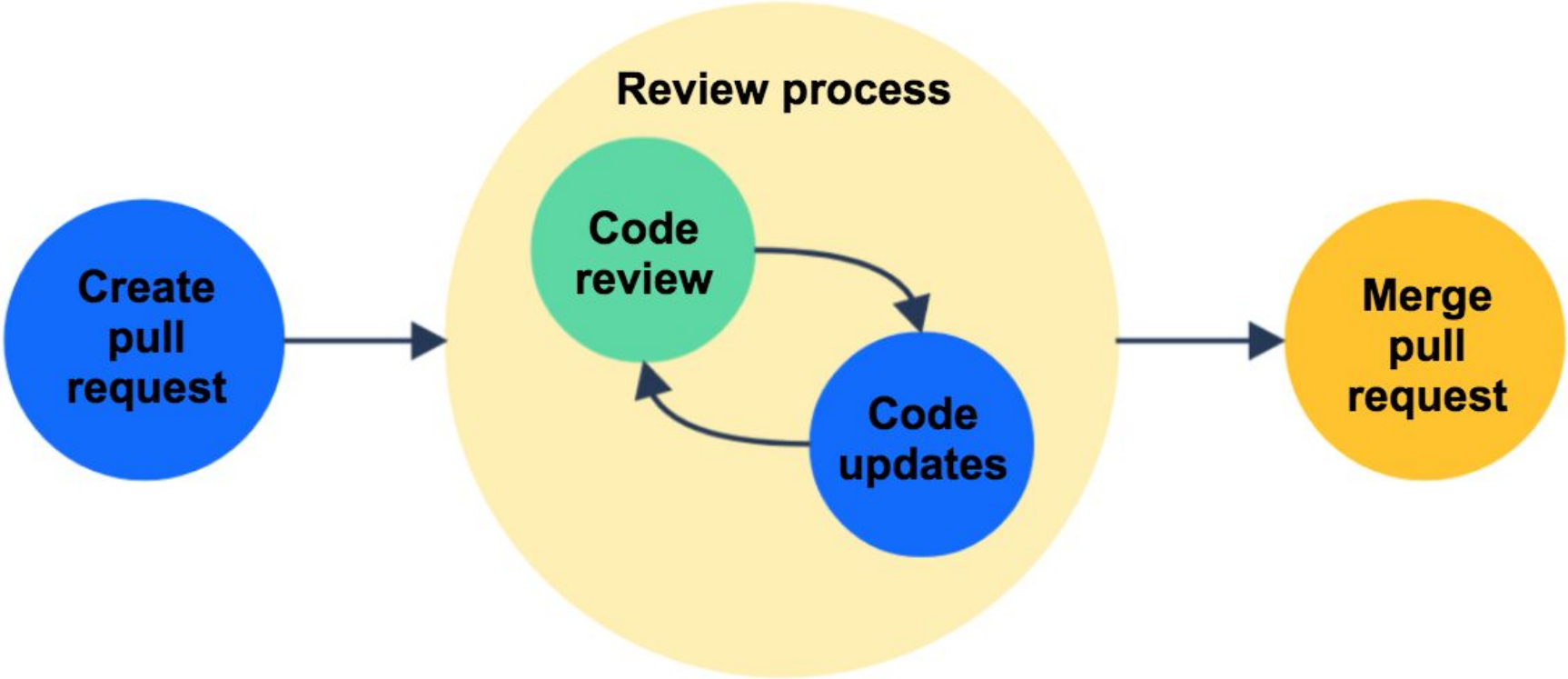



```
$ git commit -m "resolved merge conflicts"
$ git push origin yourbranchname
```

Branch Based Workflow Best Practices

- `git checkout main`
  - Make sure we're in the main branch
- `git checkout fit`
  - Make a new branch from the main branch
  - Always create a branch when making changes
- Make changes
- `git merge main fit`
  - Test that the merge fits WITHIN THE BRANCH
  - Resolve any conflicts WITHIN THE BRANCH
- `git checkout master`
- `git merge fit main`
  - We should not have any conflicts, tests should pass

Working in Parallel: Pull Requests



Making a branch:

- Collaborating within a repository
  - Branches are used when multiple people are working within the same repo. Every person or feature gets its own branch
- Fixing Bugs
  - Making a branch to fix bugs or develop features keeps the main branch free from unstable code until you're finished
- Isolating Work
  - Branches are great for isolating different parts of a project and staying organized

Forking:

- Working on a Separate Project
  - Fork when you want to create a separate project that you can take in a completely new direction
- Contributing to Open Source Projects
  - In open source projects, you typically don't have write access. Fork, make changes, and then submit a pull request from your fork.
- Maintaining a Personal Copy
  - if you want a version of a project for personal use or experimentation, forking is the way to go

Pull Requests: Forking

```
$ git clone [repository URL]
$ cd [repository name]
$ ls

$ git remote add upstream
[original repo url]
$ git remote -v

$ git pull upstream master
```

Pull Requests: Tutorial

```
$ git clone [repository URL]
$ cd [repository name]
$ ls

$ git remote add upstream
[original repo url]
$ git remote -v

$ git pull upstream master
```