# GIT: Getting Confident

# GIT: Gitting Confident

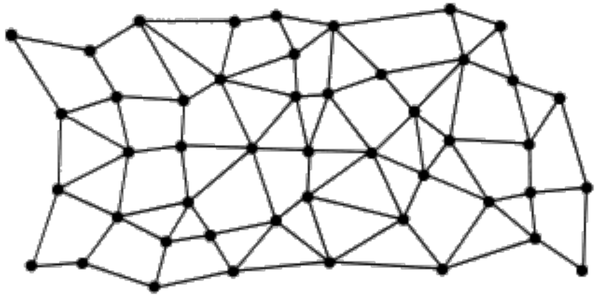
# Assumptions

- **What is a repository**
- **How to:**
  - Create a new repository `git init`
  - Clone `git clone <path.to.git.repository>`
  - Pull `git pull`
  - Commit `git commit`
  - Push `git push`

# You're about to learn about

- DVCS
- Git Config
- Git Terminology
  - Commits
  - Head
  - Workspace & Staging area
- Undoing Changes: git reset
- Feature Branch workflow

# DVCS



# Git is a distributed version control system

# DVCS

- **A Git repository in your machine is a first-class repo in its own right.**
- **In comparison to Centralized version control systems:**
  - Performing actions is extremely fast
    (because the tool only needs to access the hard drive, not a remote server.)
  - Committing can be done locally without anyone else seeing them. Once you have a group of changes ready, you can push all of them at once.
  - Everything (except for pushing and pulling) can be done without an internet connection.

# DVCS

- To be able to collaborate with Git, you need to manage your remote repositories.
- `git remote` allows you to add or remove repositories (other than the one on your local disk) which you can push & pull.

# DVCS

- **What is Github (and similar services)?**
  - A repository hosting service.
  - Usually used as the project's central repository for collaboration (all the developers add as `remote` to push/pull their changes)
  - Provides project management & collaboration tools, such as forking & PRs, issue tracking, wikis etc.

# Configuring git

# Configuring git

- Git is configured through `.gitconfig` text files.
- The `git config` command is a convenience function to set Git configuration values on a global or local project level.

```
git config <level> <configuration> <value>
```

# Configuring git - Levels

| | |
|---|---|
| **Local** | Default option.<br>Local level is applied to the current repository git config gets invoked in.<br>Stored in a file that can be found in the repo's .git directory: `.git/config` |
| **Global** | Applied to an user in the operating system user.<br>Stored at `~/.gitconfig` (on unix systems). |
| **System** | System-level configuration: covers all users on an operating system.<br>Stored at the system root path. `$(prefix)/etc/gitconfig` (on unix). |

Thus the order of priority for configuration levels is: local, global, system. This means when looking for a configuration value, Git will start at the local level and bubble up to the system level.

# Configuring git - Common options

Identity:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Editor:

```
git config --global core.editor "code --wait"
```

# Configuring git - Common options

Colors:

```
git config --global color.ui true
```

Autocorrect:

```
$ git config --global help.autocorrect 1
```

# Configuring git - Aliases

- Custom shortcuts that expand to longer or combined commands.
- Stored in Git configuration files. (you can use the `git config` command to configure aliases)

```
git config --global alias.ci commit
git config --global alias.co checkout
git config --global alias.st status
```

# *Lab*

# Git Terminology

*Commit • Head • Workspace & Staging Area*

# Commits: Git is structured like a “singly” linked list

```
> git commit -m “initial commit”
```

```
> git commit -m “second commit”
```

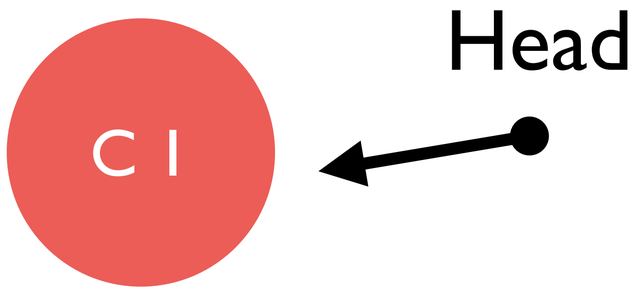
```
> git commit -m “third commit”
```

# Commits

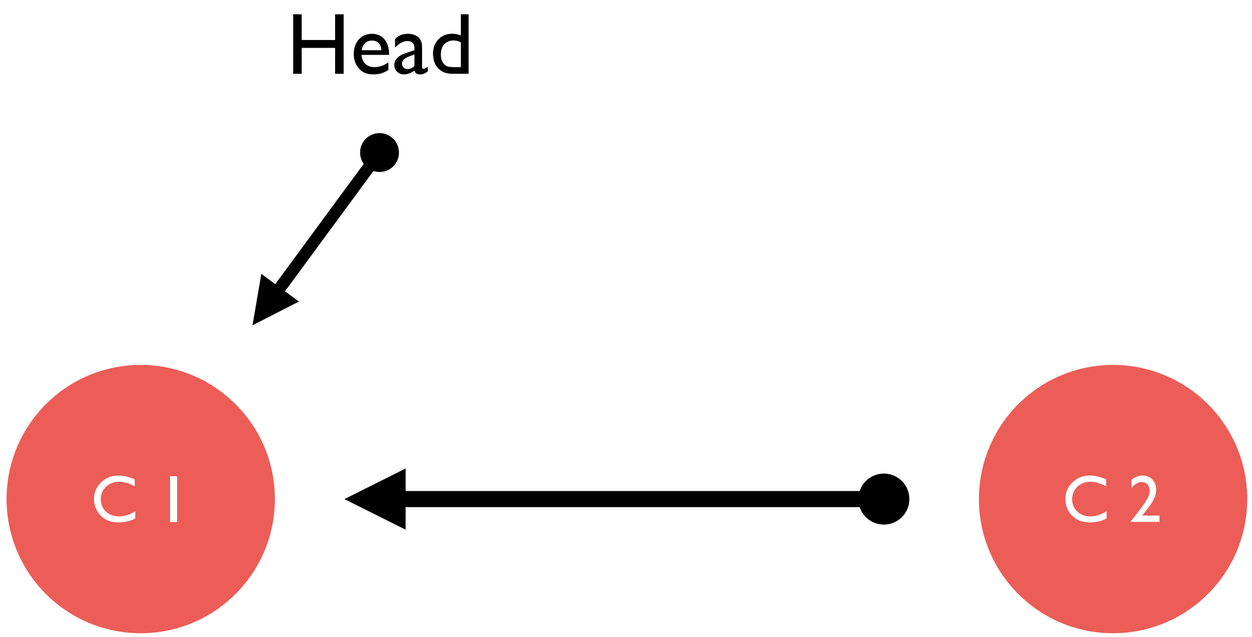
- **Saves the current state of your project at that point in time**
- **Useful because**
  - you can always go back to a previous commit if you mess up
  - documents changes that happen over time
  - organizes changes in such a fashion that makes debugging convenient (i.e. “which commit introduced this bug”?)
- **Commit early and often!**

# Head

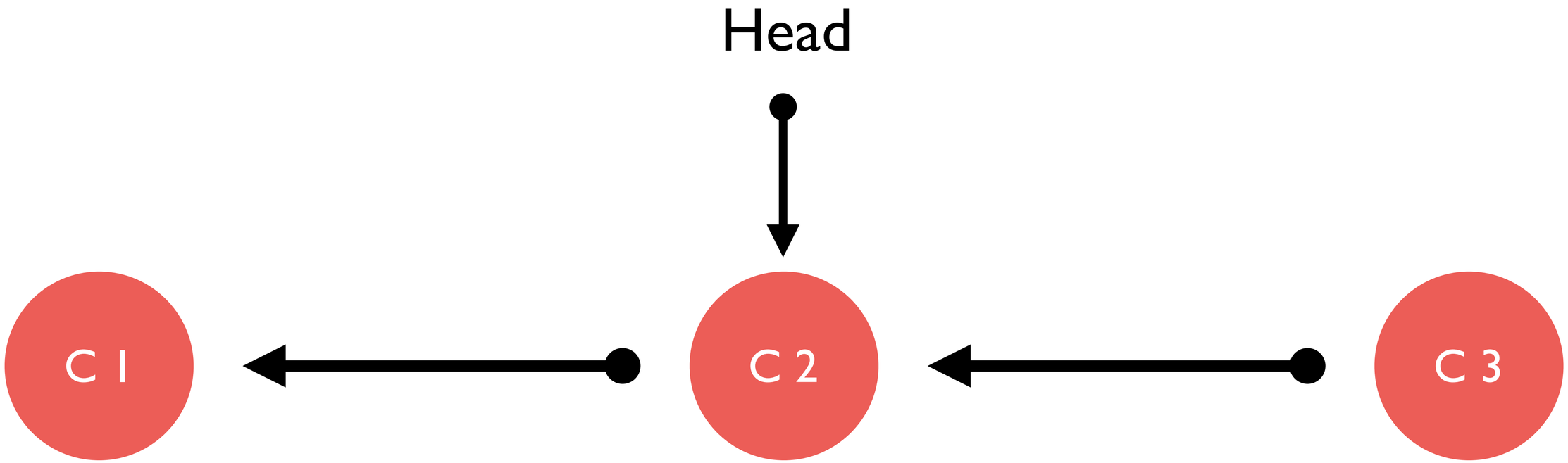
- HEAD is a reference to the last commit in the currently check-out branch.
- We are calling this commit “C1”, but in real life commits are referenced after hashes, for example fed2da64c0efc5293610bdd892f82a58e8cbc5d8. That’s why references like Head are useful.



# Head



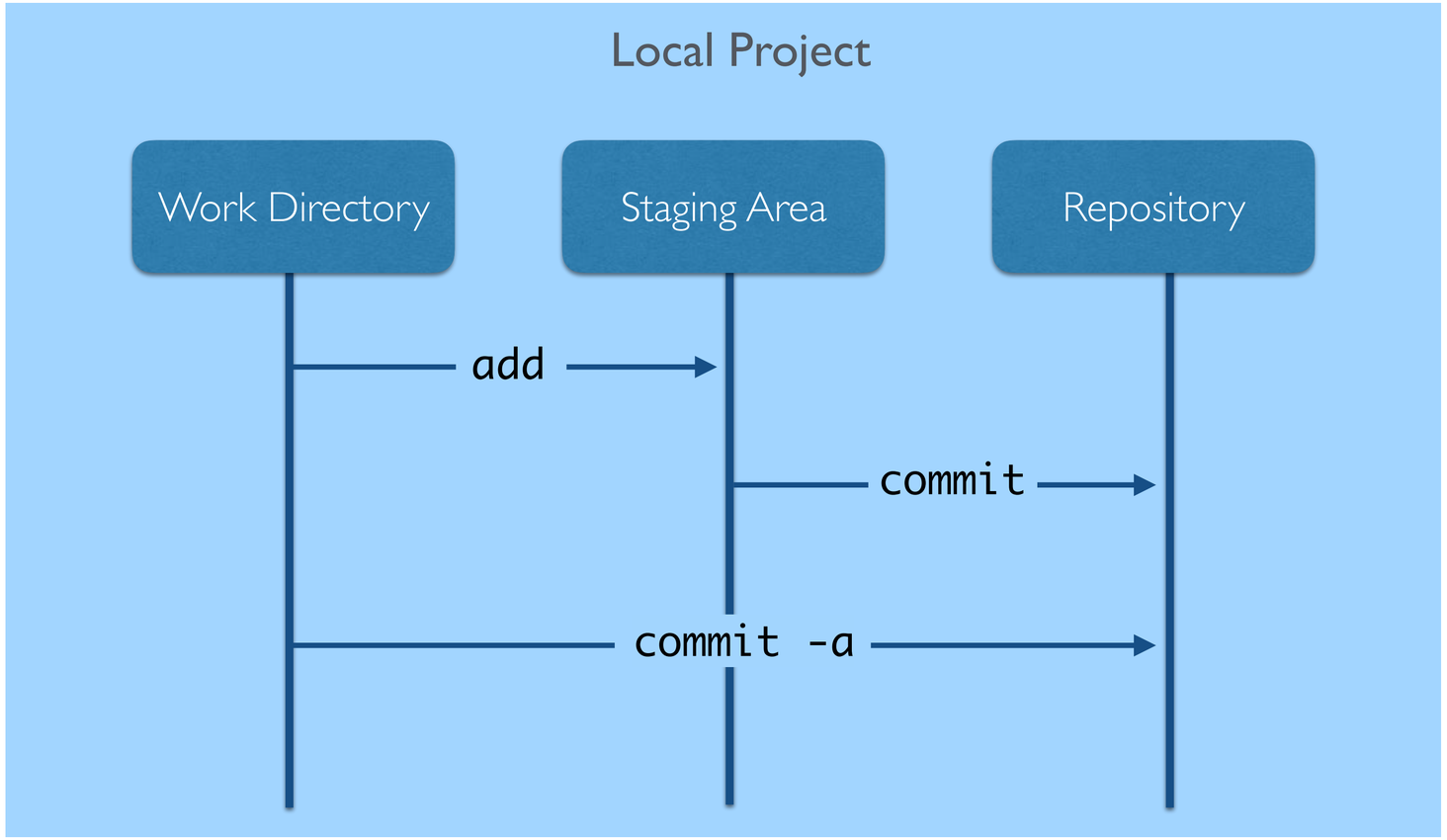
# Head



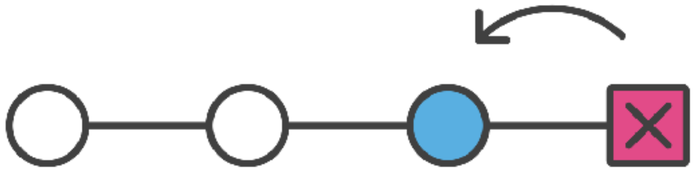
# Workspace & Staging area

- **Workspace:** Your local working directory (where you do your actual work). It contains tracked files, untracked files and a special directory ".git".
- **Staging area:** Used for preparing commits. You can add files to the next commit.
- The **Repository** itself is the virtual storage of your project. It allows you to save versions of your code, which you can access when needed.

# Workspace & Staging area



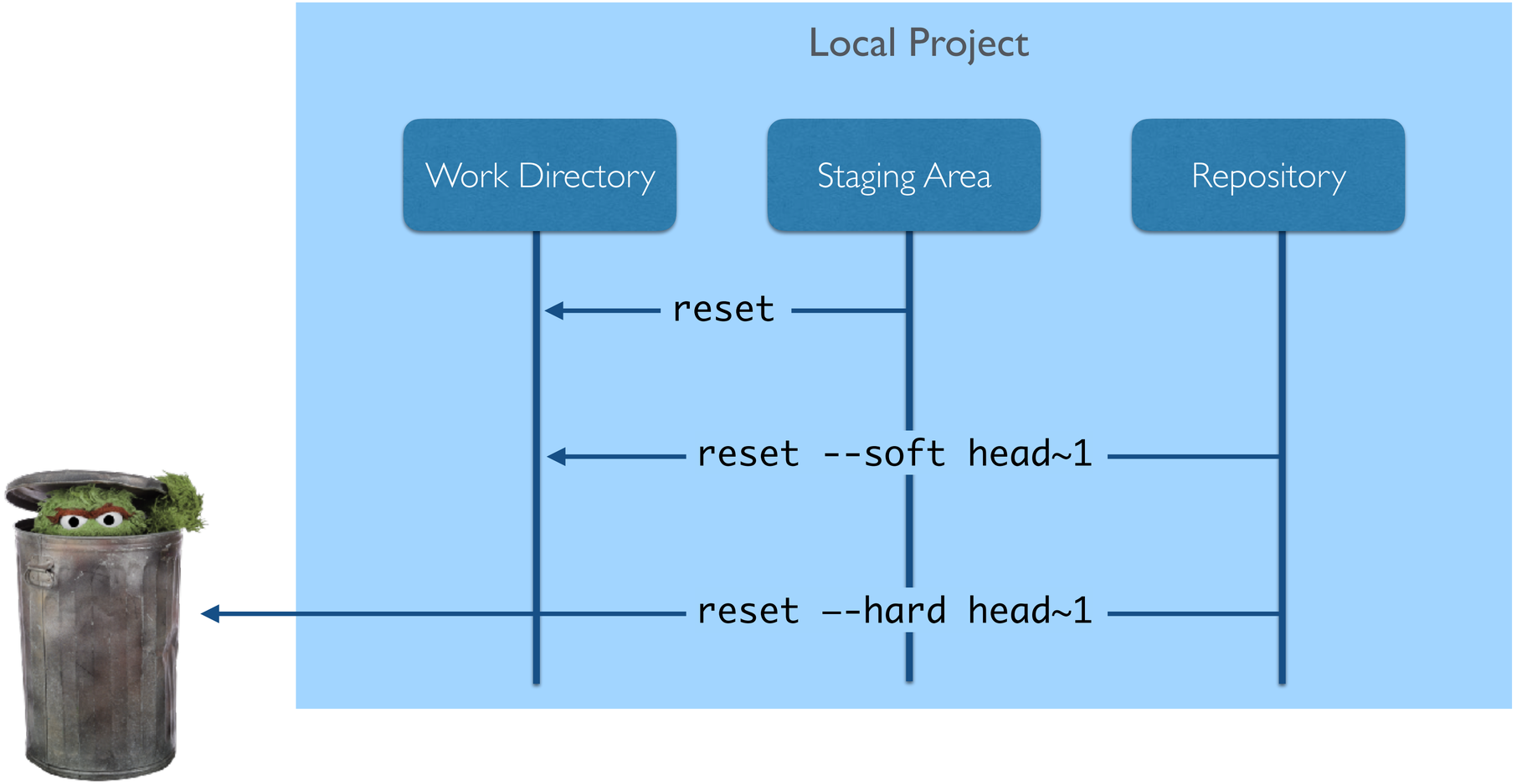
# Undoing Changes: `git reset`



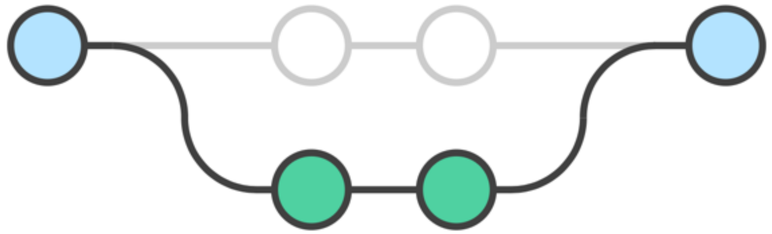
# git reset

- **A complex and versatile tool for undoing changes:**
  - Undo Staging: `git reset`
  - Undo Commit (or Commits): `git reset <commit>`
    - **soft**: Keep changed files
    - **hard**: Delete changes files

# Undoing Changes: git reset



# Branches and Merging

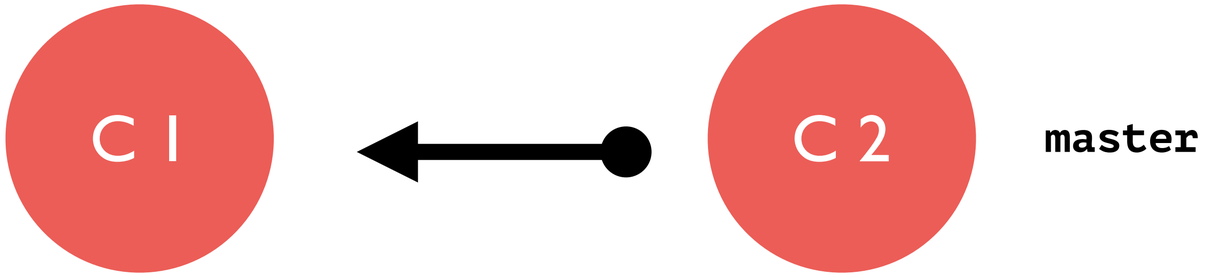


# Scenario: two people are working on a project

# Problems

- How can I show what I've done in an efficient manner?
- If we don't like my work, how can I easily get back to where I was?
- If we do like my work, how can I integrate it together with your work?

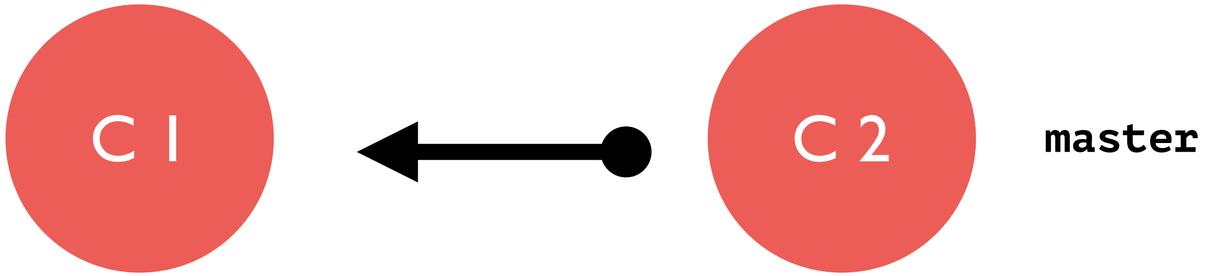

BRANCHES




HEAD: master C2

```
>
```

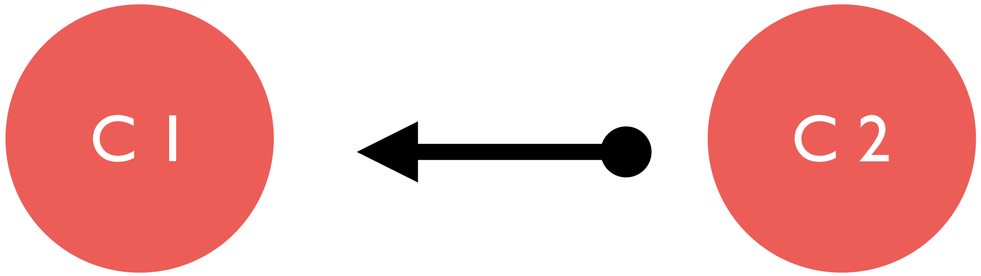
my-feature




HEAD: master C2

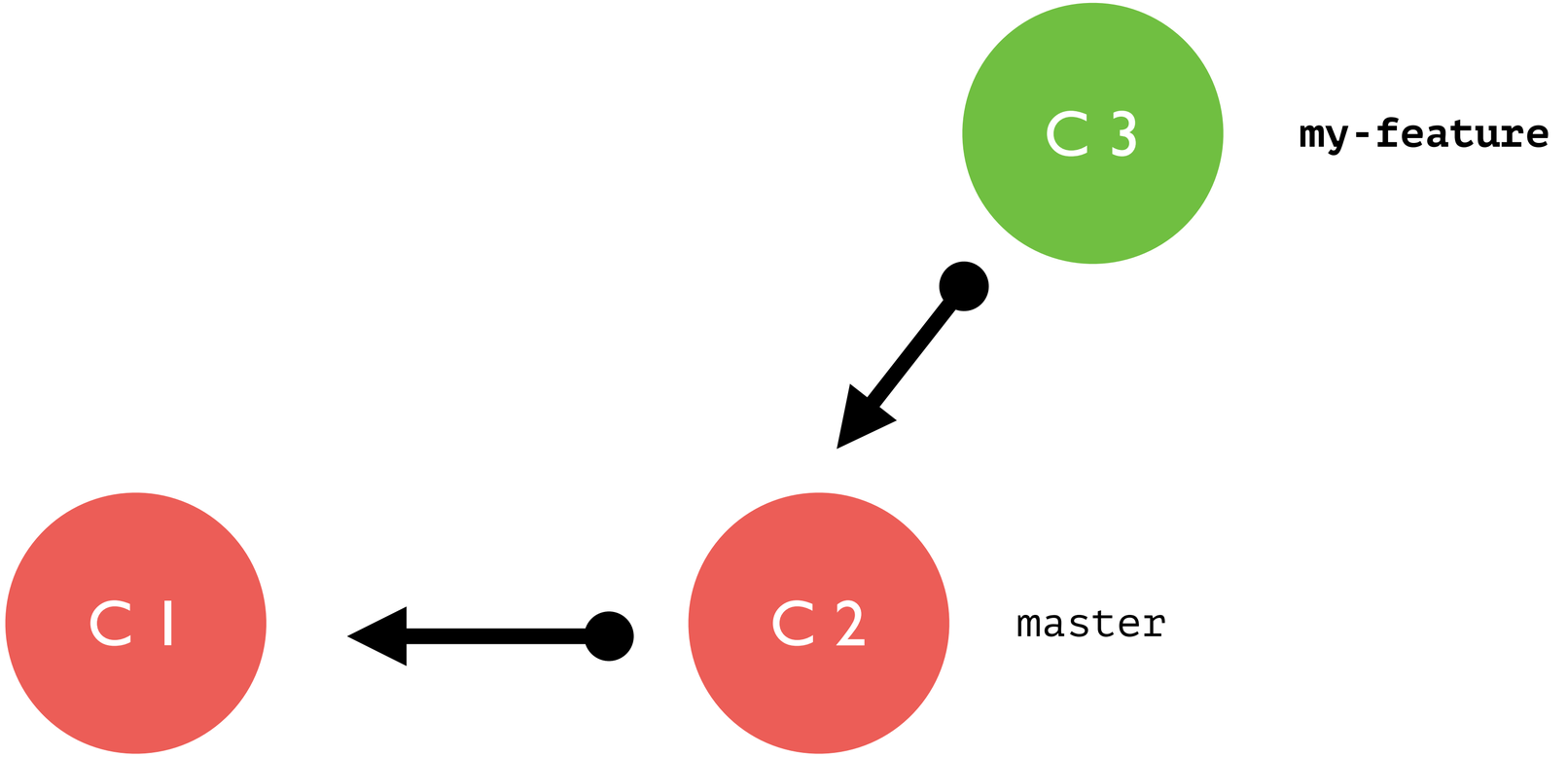
```
> git branch my-feature
```

**my-feature**



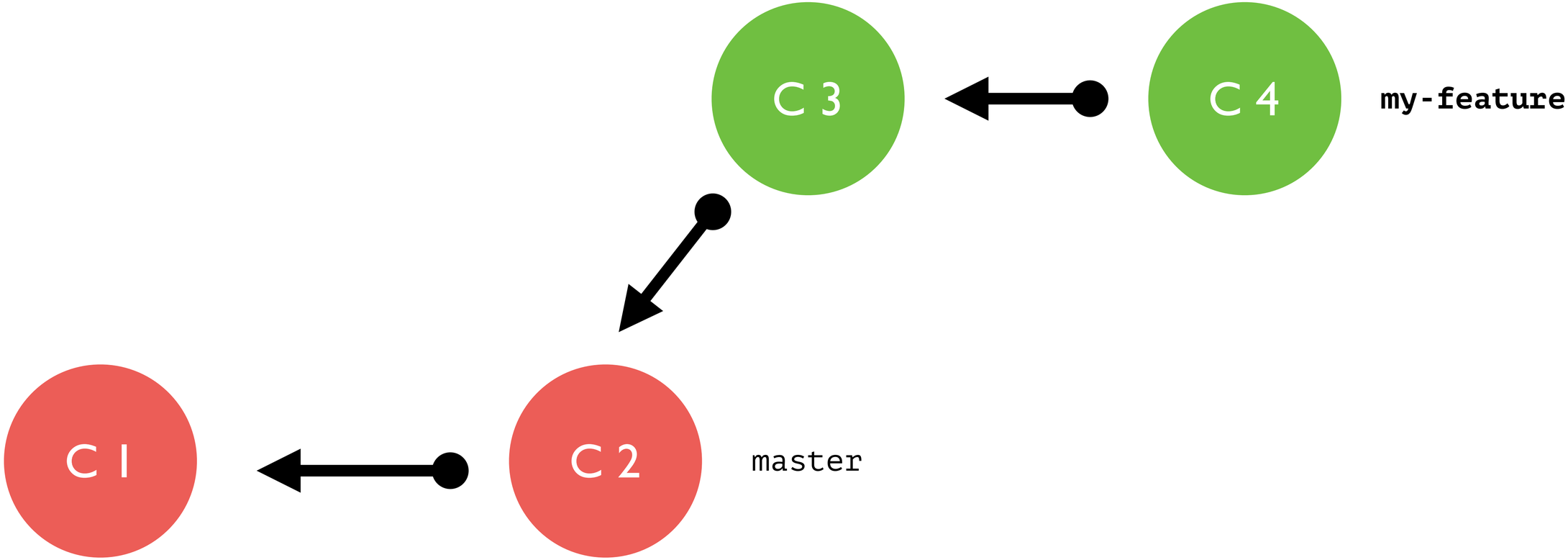

HEAD: my-feature C2

```
> git checkout my-feature
```




HEAD: my-feature C3

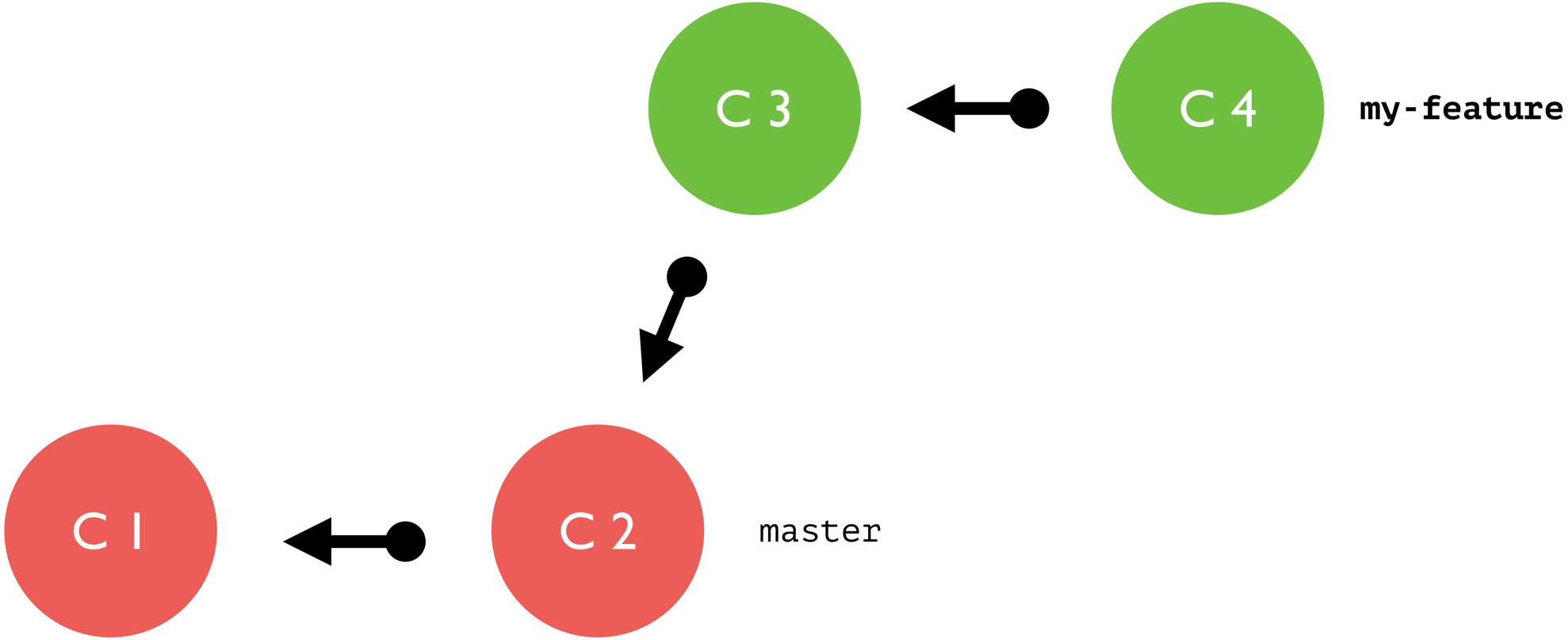
```
> git commit -m “add stuff”
```




HEAD: my-feature C4

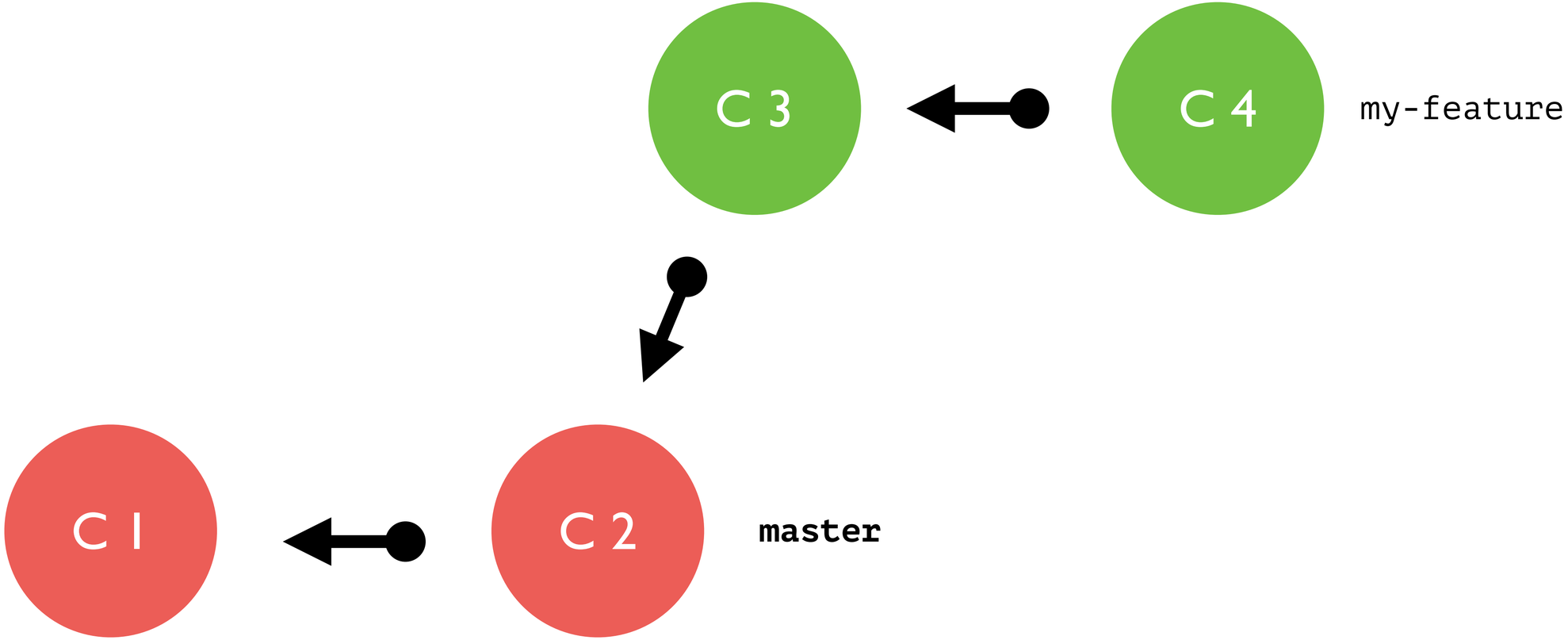
```
> git commit -m “moar stuff”
```


MERGING



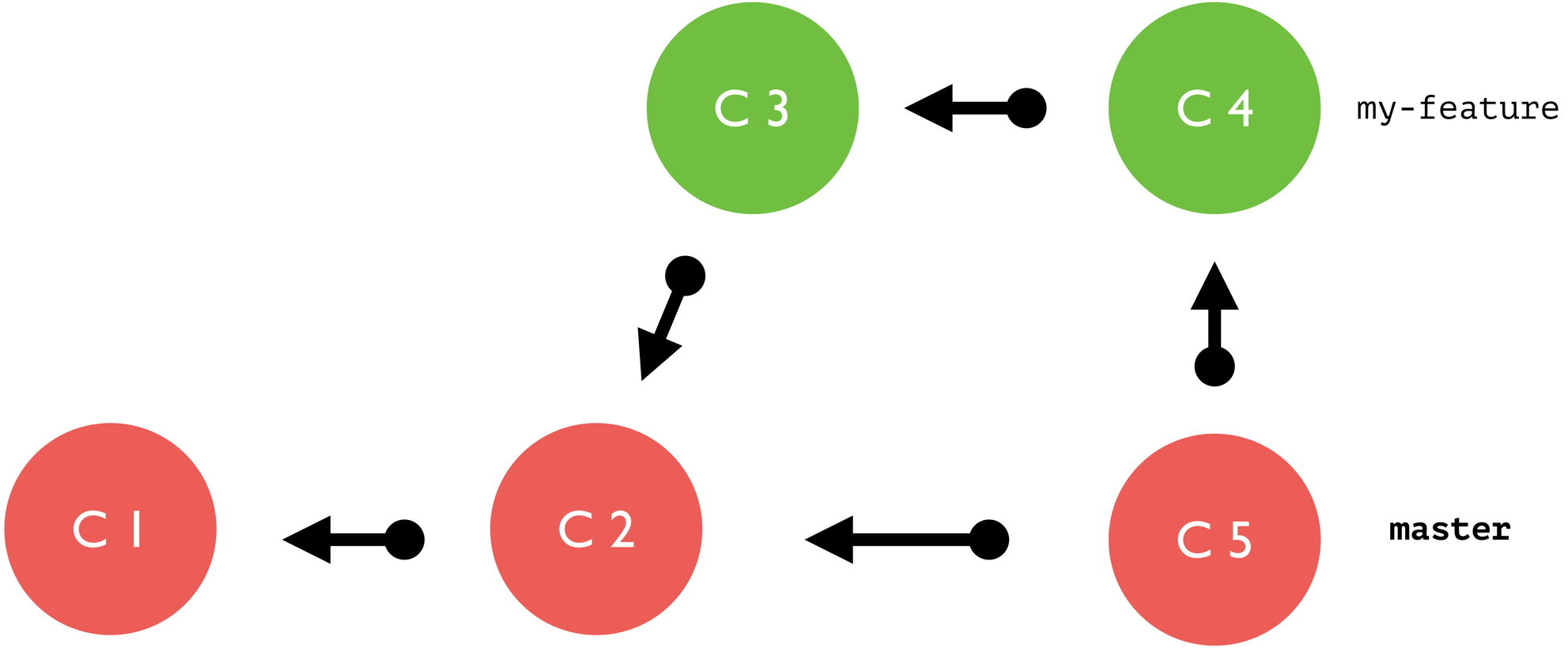

HEAD: my-feature C4

```
>
```




HEAD: master C2

```
> git checkout master
```




HEAD: master C5

```
> git merge my-feature
```


PULL REQUESTS

# Pull Requests

- Merging a branch on the remote, plus some ceremony (ex. code review by another team member)

- Feature of Github, not explicitly part of Git

```
> git push origin cool-branch
```

HEAD: cool-branch

collin / example

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

*No description, website, or topics provided.*

Edit

Add topics

1 commit | 1 branch | 0 releases | 1 contributor

Your recently pushed branches:

**cool-branch** (less than a minute ago) — Compare & pull request

# Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across fork

base: master ← compare: cool-branch ✓ Able to merge. These branches can be automatically merged

adds a cool file

Write Preview

A very detailed description.

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Create pull request

# adds a cool file #1

Open **collin** wants to merge 1 commit into `master` from `cool-branch`

Conversation 0 | Commits 1 | Files changed 1

**collin** commented just now — Owner

A very detailed description.

 `adds a cool file` Verified 4c66333

Add more commits by pushing to the **cool-branch** branch on **collin/example**.

**This branch has no conflicts with the base branch**

Merging can be performed automatically.

Merge pull request ▾ You can also open this in GitHub Desktop or view command line instructions.

```
> git checkout master
```

HEAD: master

```
> git pull origin master
```

HEAD: master


MERGE CONFLICT

# Merge conflicts

- Fairly common: not the end of the world
- Happens when Git can't automatically resolve two commits into one - needs a human to decide what version to keep
- Makes sure someone else's work doesn't overwrite another's unintentionally

## script.js - master

```
console.log(‘hello world')
```

script.js - f/howdy

```
console.log('howdy world')
```

**script.js - master**

```
console.log('hello world')
```

script.js - f/goodbye

```
console.log('goodbye world')
```

script.js - f/howdy

```
console.log('howdy world')
```

script.js - master

```
console.log('hello world')
```

**script.js - master**

```
console.log('goodbye world')
```

script.js - f/goodbye

```
console.log('goodbye world')
```

Uh oh - no history of changing “goodbye” to “howdy”

script.js - f/howdy

```
console.log(‘howdy world')
```

script.js - master

```
console.log(‘hello world')
```

**script.js - master**

```
console.log(‘goodbye world')
```

script.js - f/goodbye

```
console.log(‘goodbye world')
```

```
<<<<<<< HEAD (current version)
console.log('goodbye world')
=======
console.log('howdy world')
>>>>>>> howdy (incoming change)
```

# Takeaways

- Commits - places in our project we can always travel back in time to
- HEAD - where you are now
- Branches - alternate realities where you can make all the changes (commits) you want
- Merging - what happens when you want your alternate reality to become…reality
- Pull Requests - a great feature of Github that allows others to easily review your code
- Merge conflicts - the current branch is on top, the incoming branch is on the bottom