LAB 4

Introduction to Git

git

- Distributed version control system
- Originally developed by Linus Torvalds for the development of the Linux Kernel in 2005
- Focus on speed and efficiency
- Quite a unique design and therefore sometimes a bit scary and difficult to understand

Installing Git Command Line

Installation

On Windows
Download and install GitHub at *http://windows.github.com*

On Mac
Download and install GitHub at *http://mac.github.com*

OR

If you use Homebrew:

```
$ brew install git
```

Installation

On Linux
You can generally do so through the basic package-management tool that comes with your distribution.

On Fedora:

```
$ sudo yum install git-all
```

On Debian-based distributions:

```
$ sudo apt-get install git-all
```

Check that your installation was successful

Type the following command on the console

```
$ git –version
```

It should print the git version that you have currently installed. Such as:

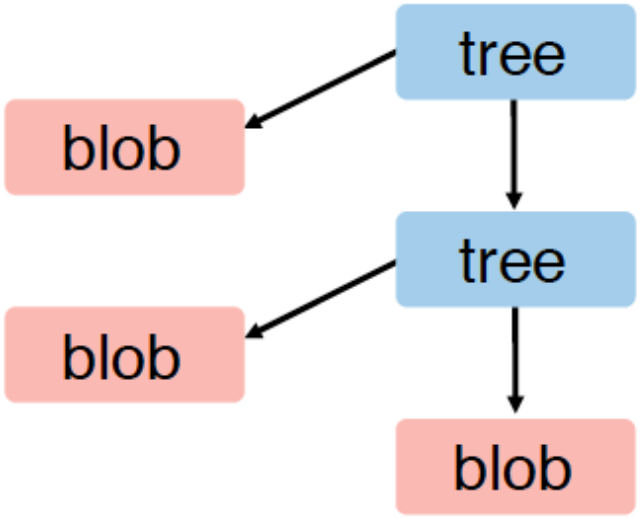
```
git version 2.10.1 (Apple Git-78)
```

Git Object Model

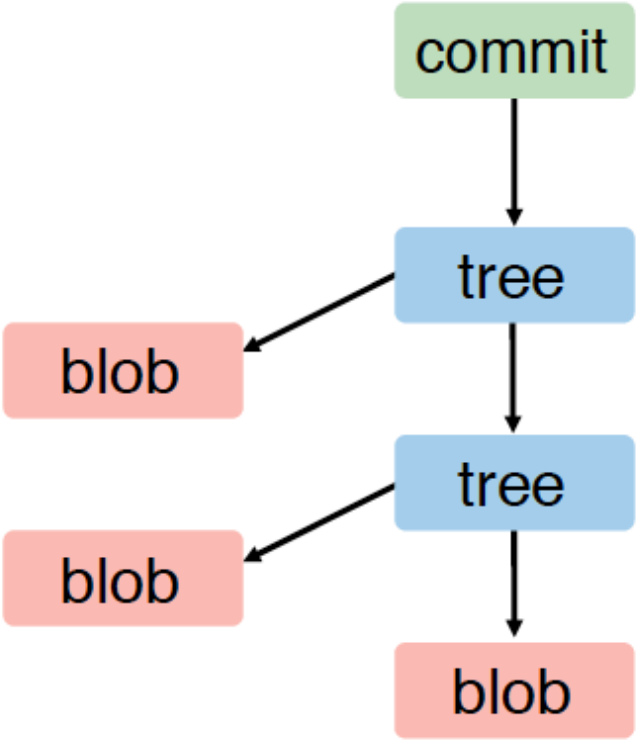
A “blob” is *content* under version control (a file)

blob

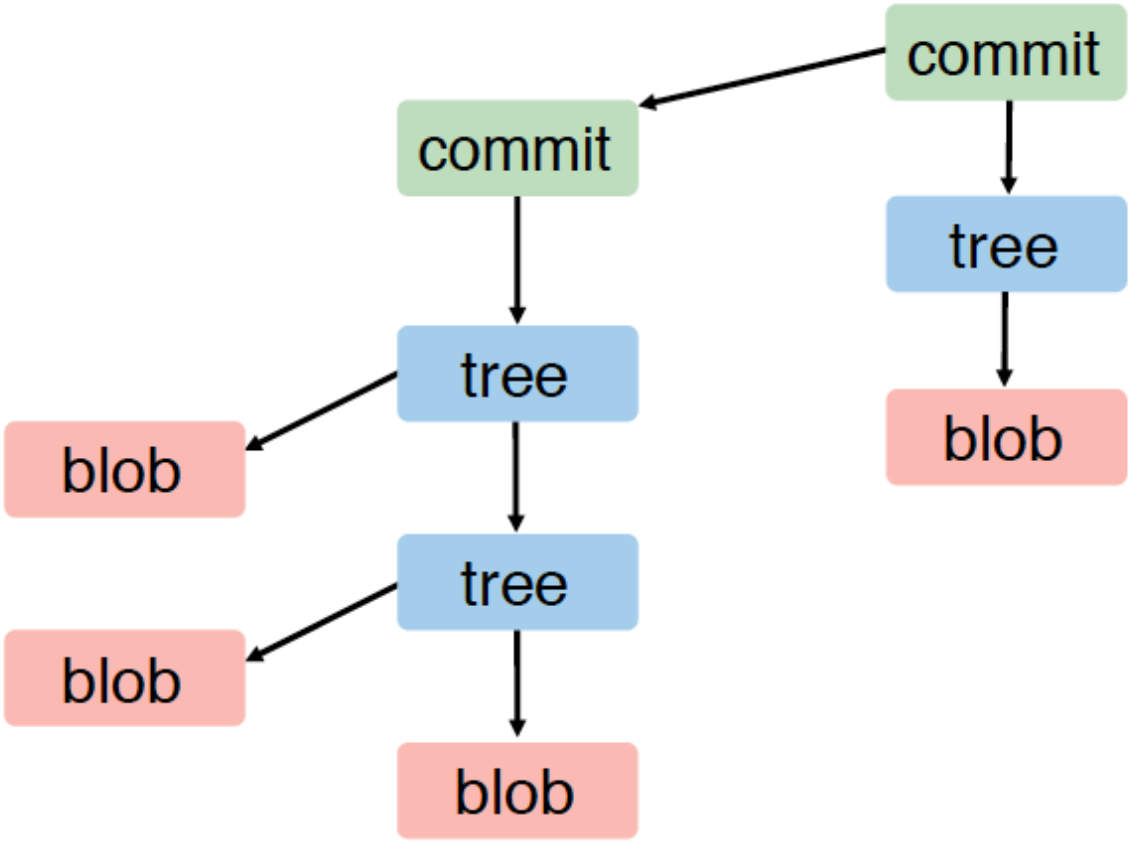
You can have *trees* of blobs
(directories of files)



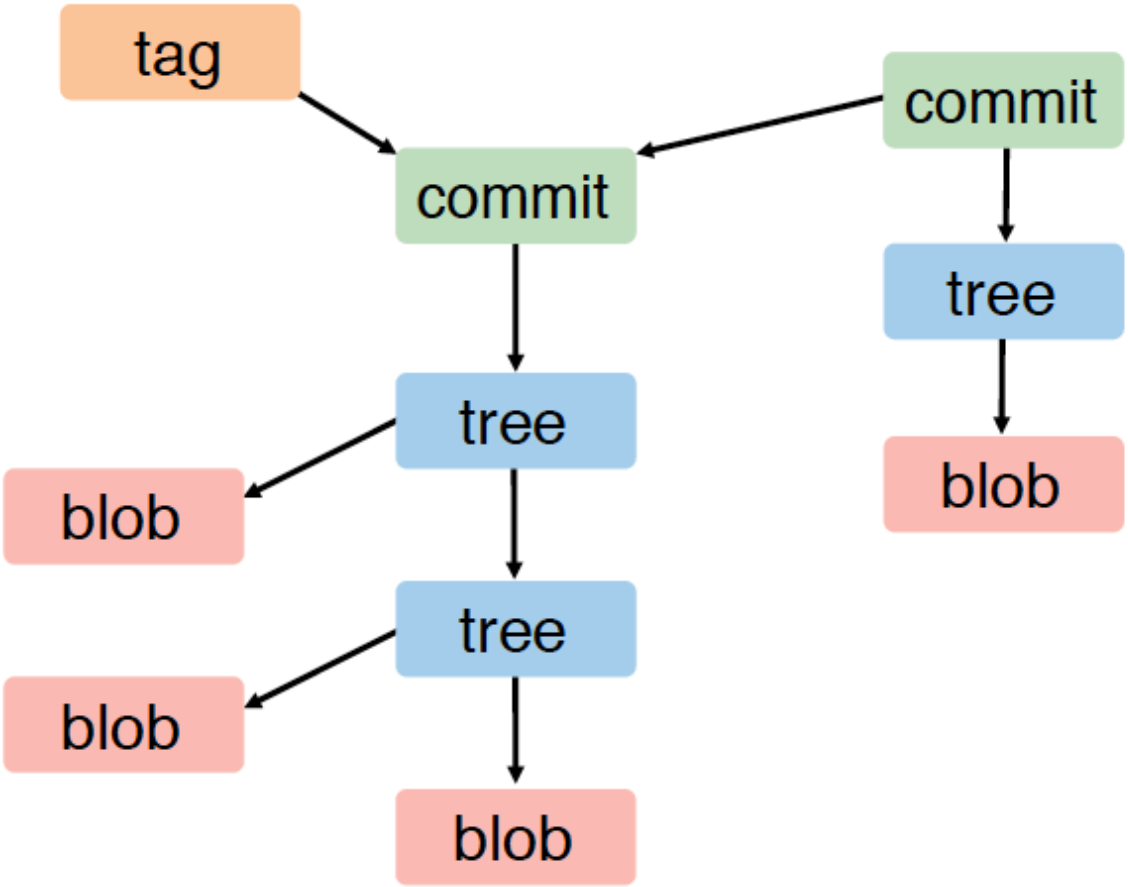
A “commit” is a tree of blobs (a set of changes)



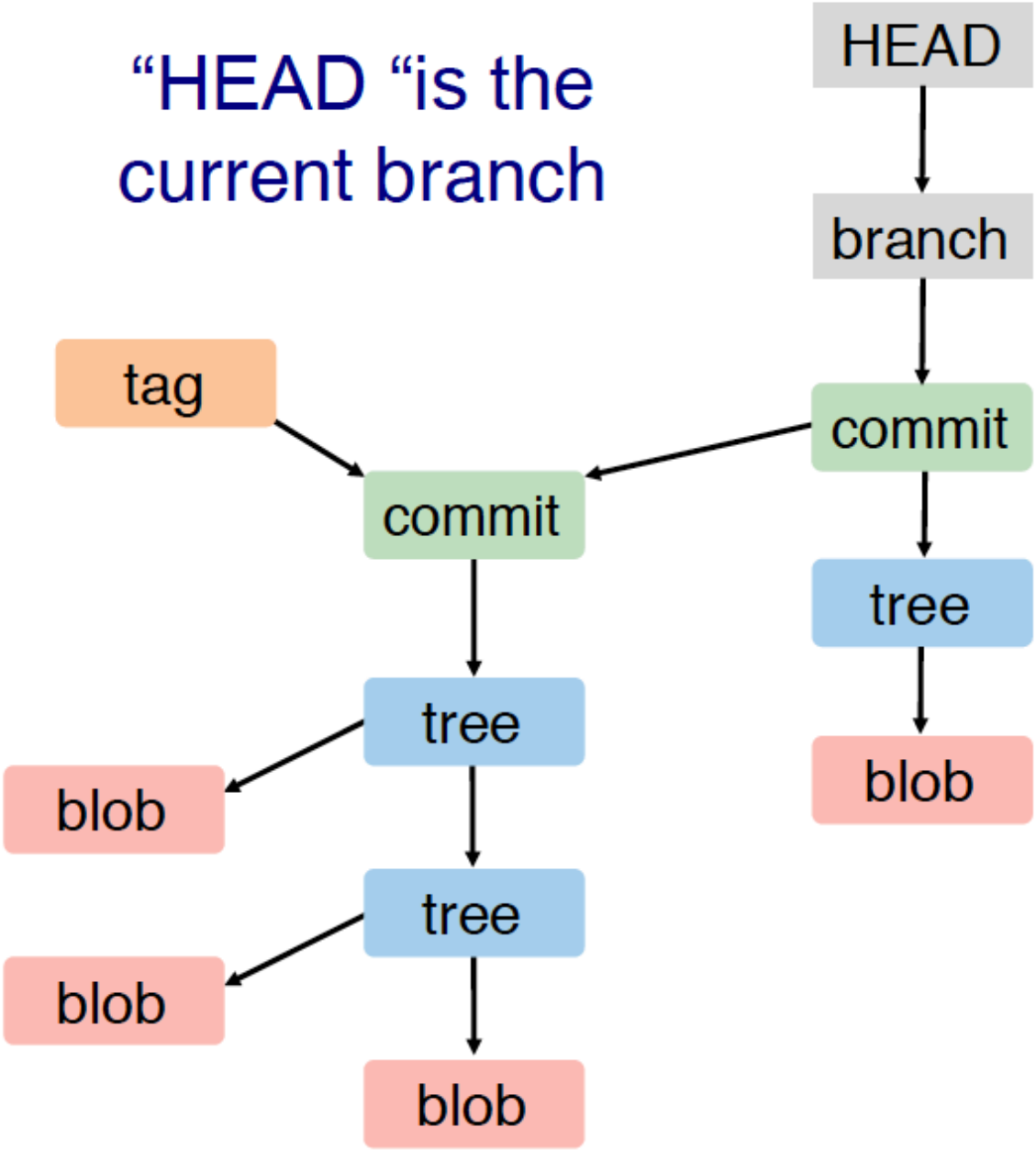
Most commits modify (or merge) earlier commits

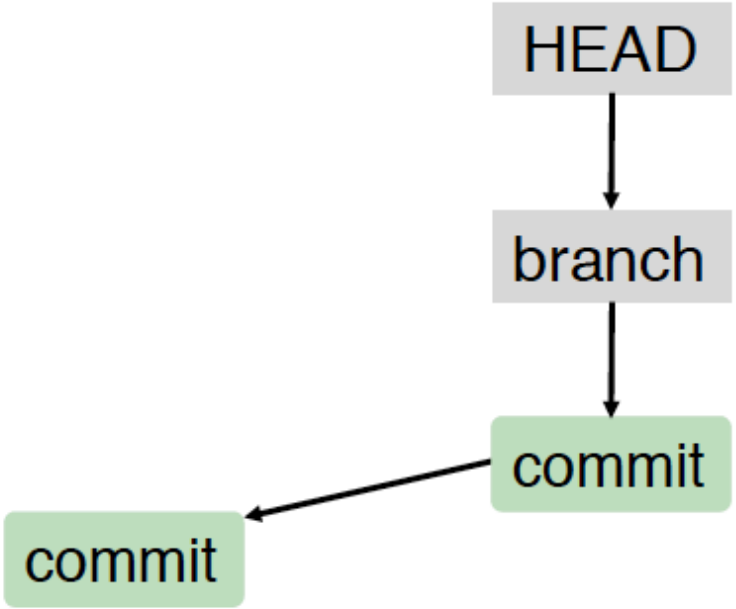


You can “tag” an interesting commit



“HEAD “is the current branch



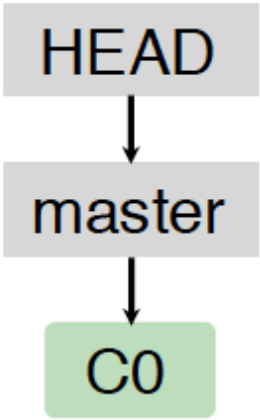


We will focus on commits only for one branch

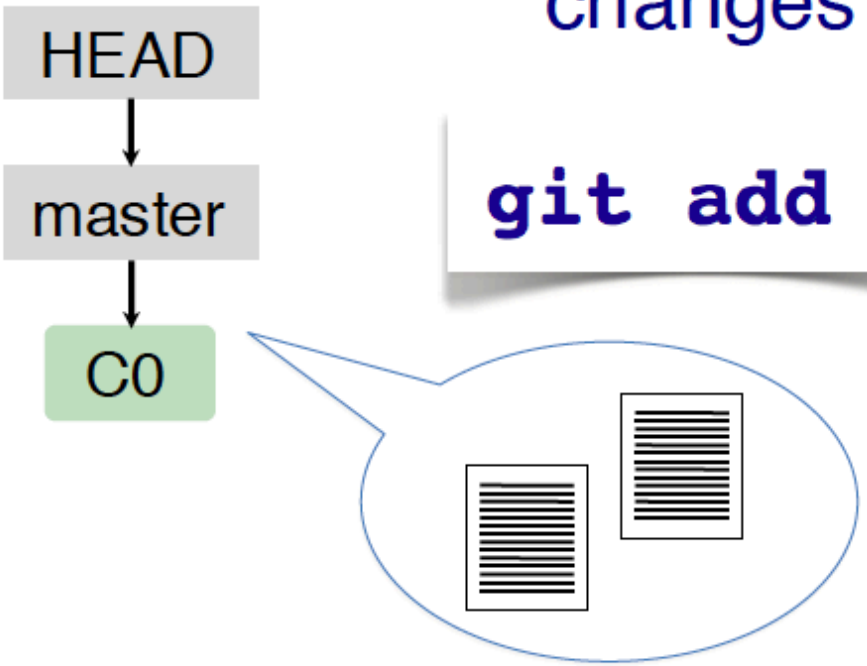
Git Basic Operations

Create a git repo

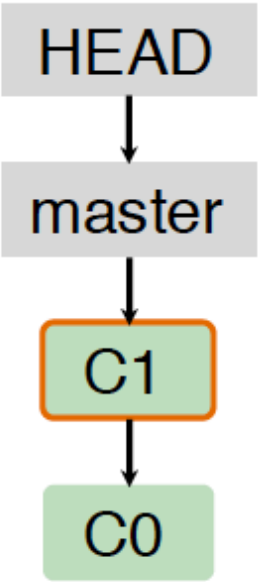
```
mkdir repo
cd repo
git init
```



Tell git to “stage” changes



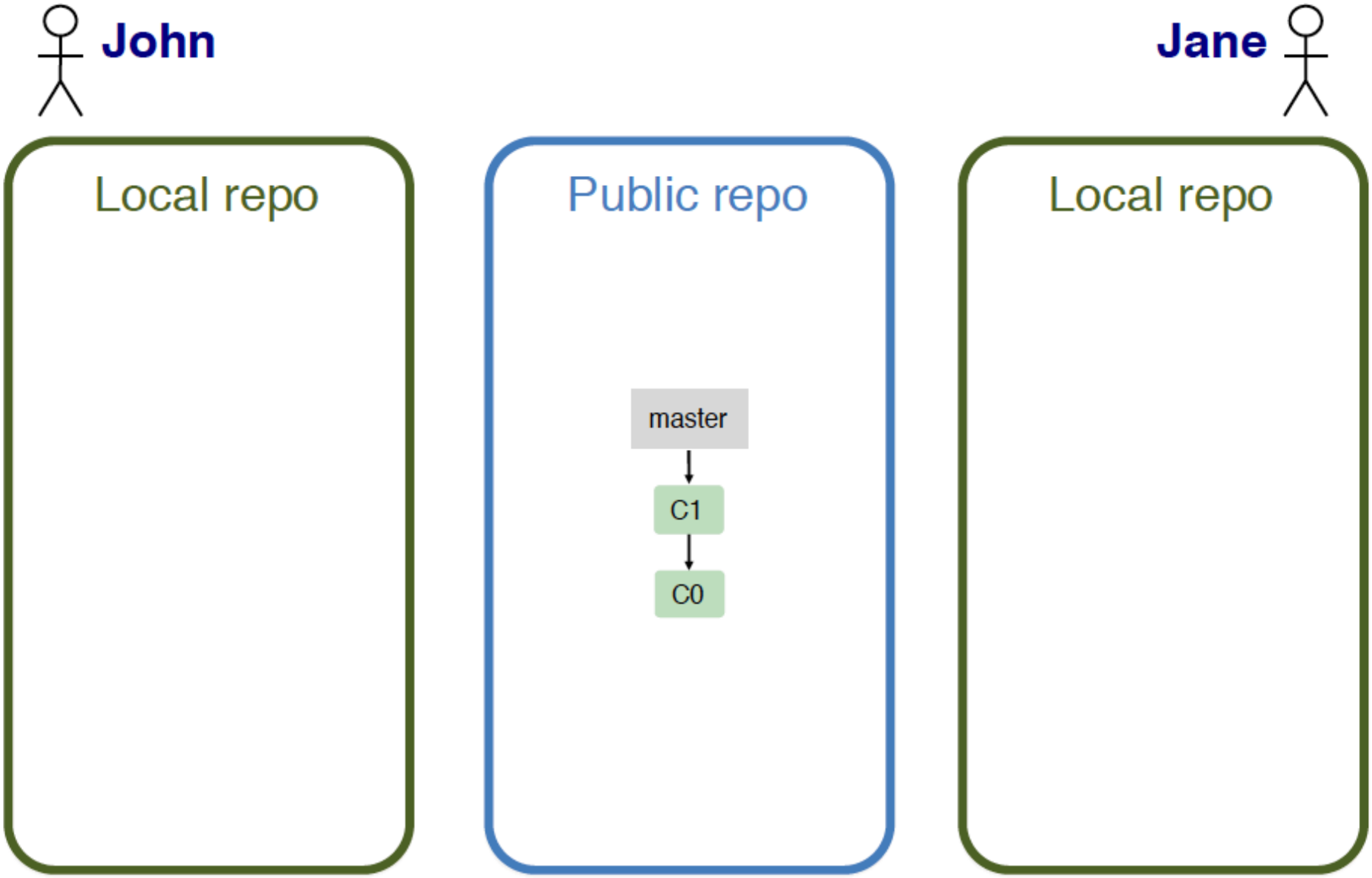

```
git add …
```

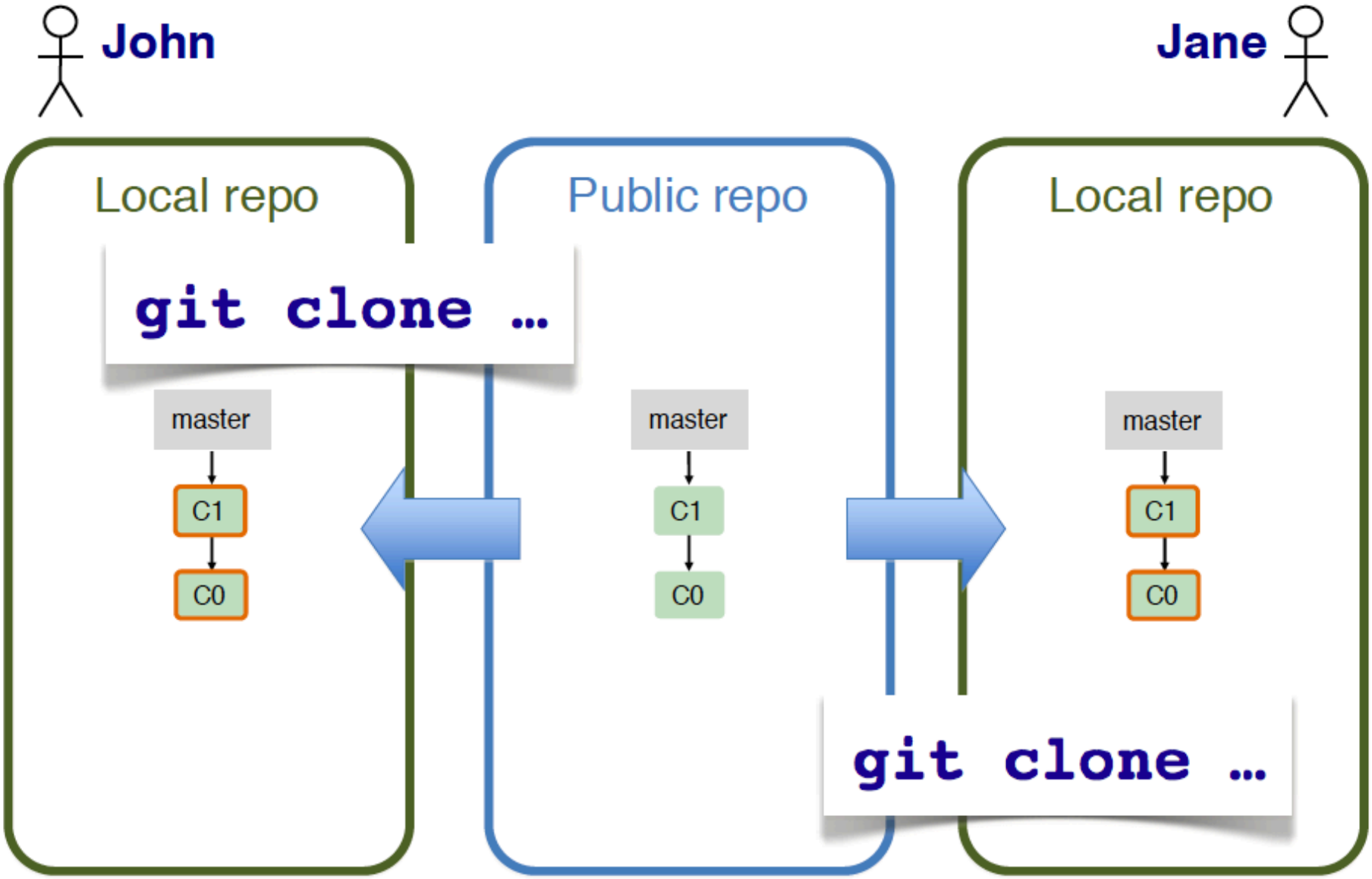


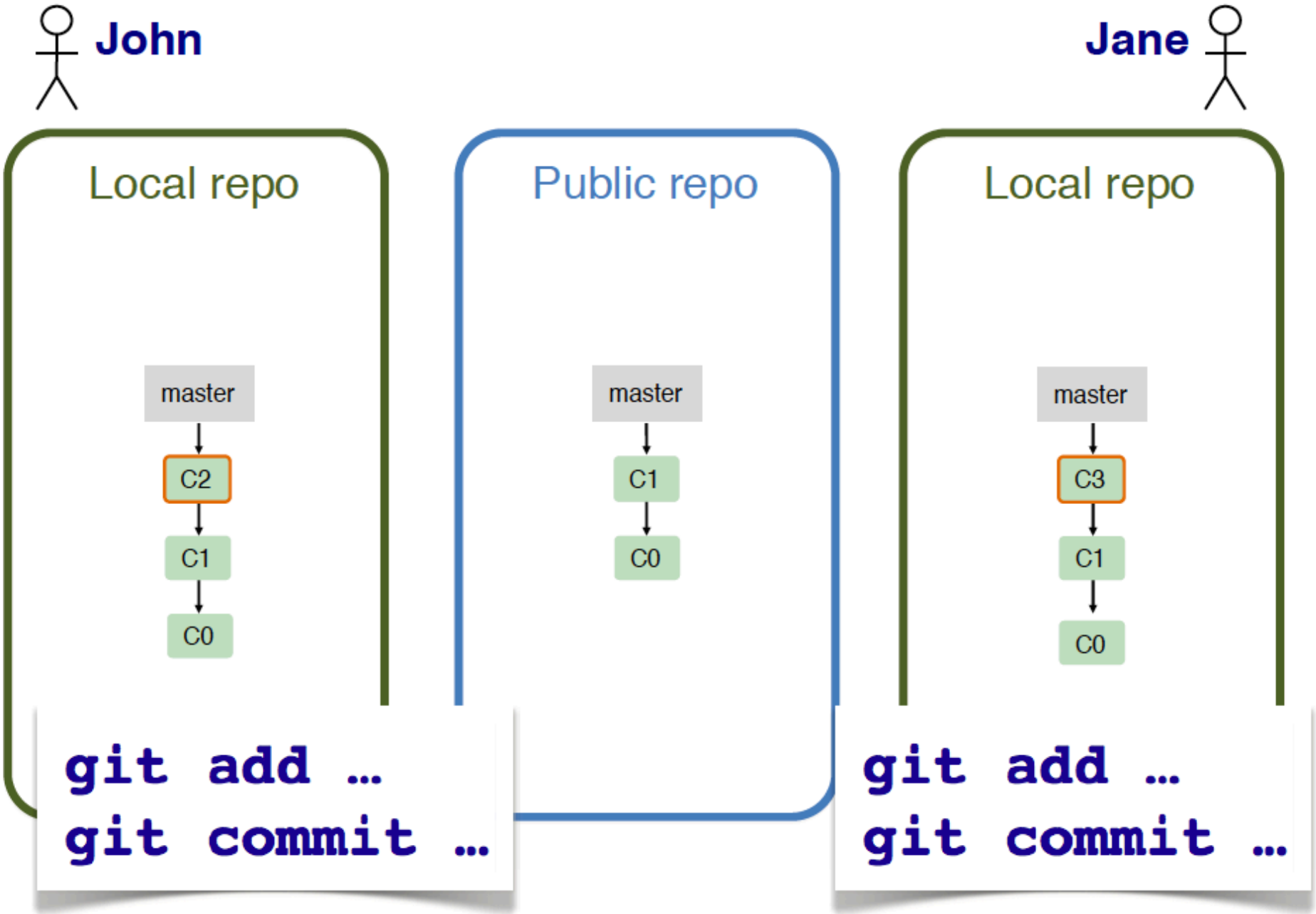
Commit your changes

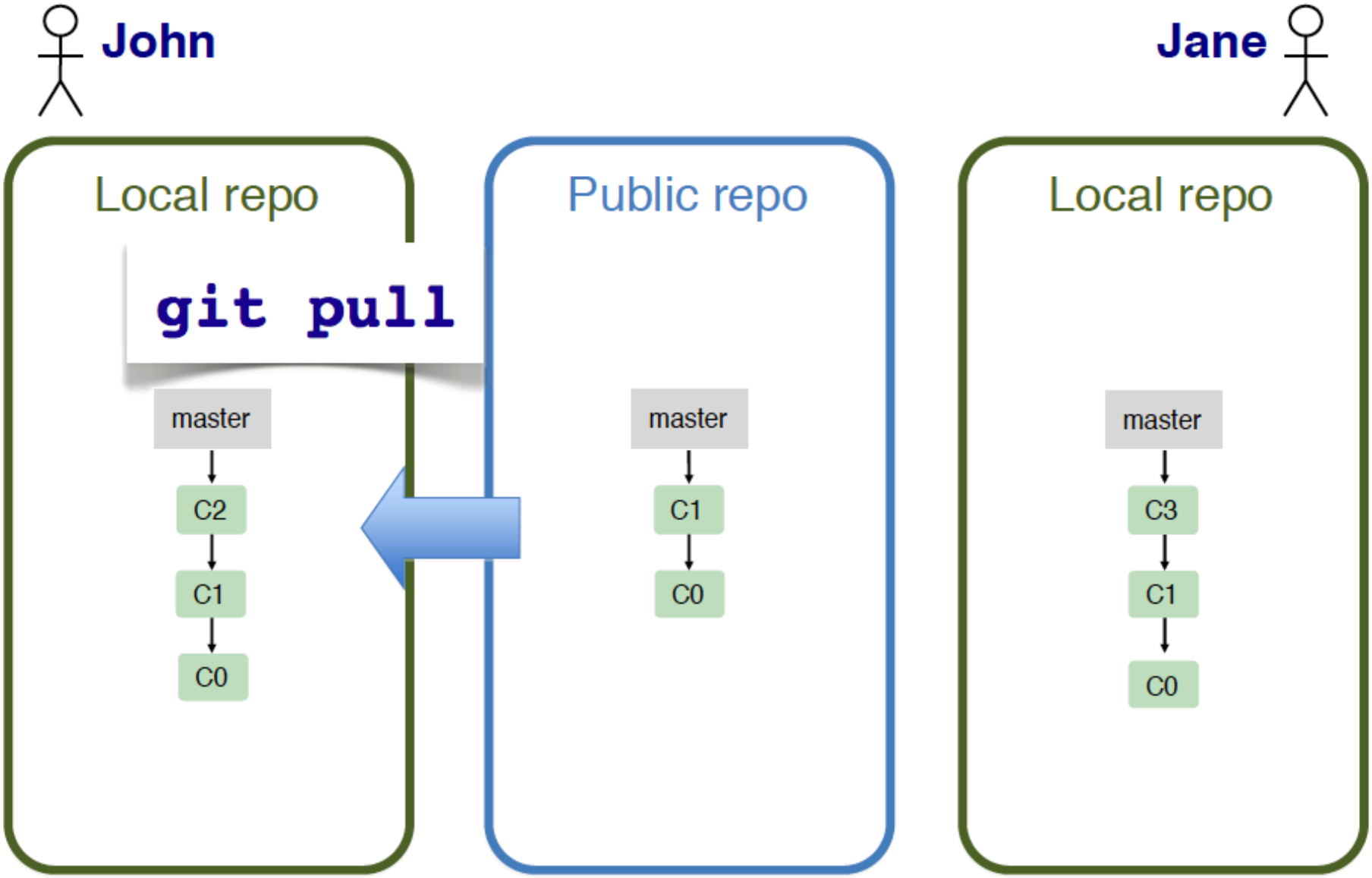
```
git commit …
```

Collaborating

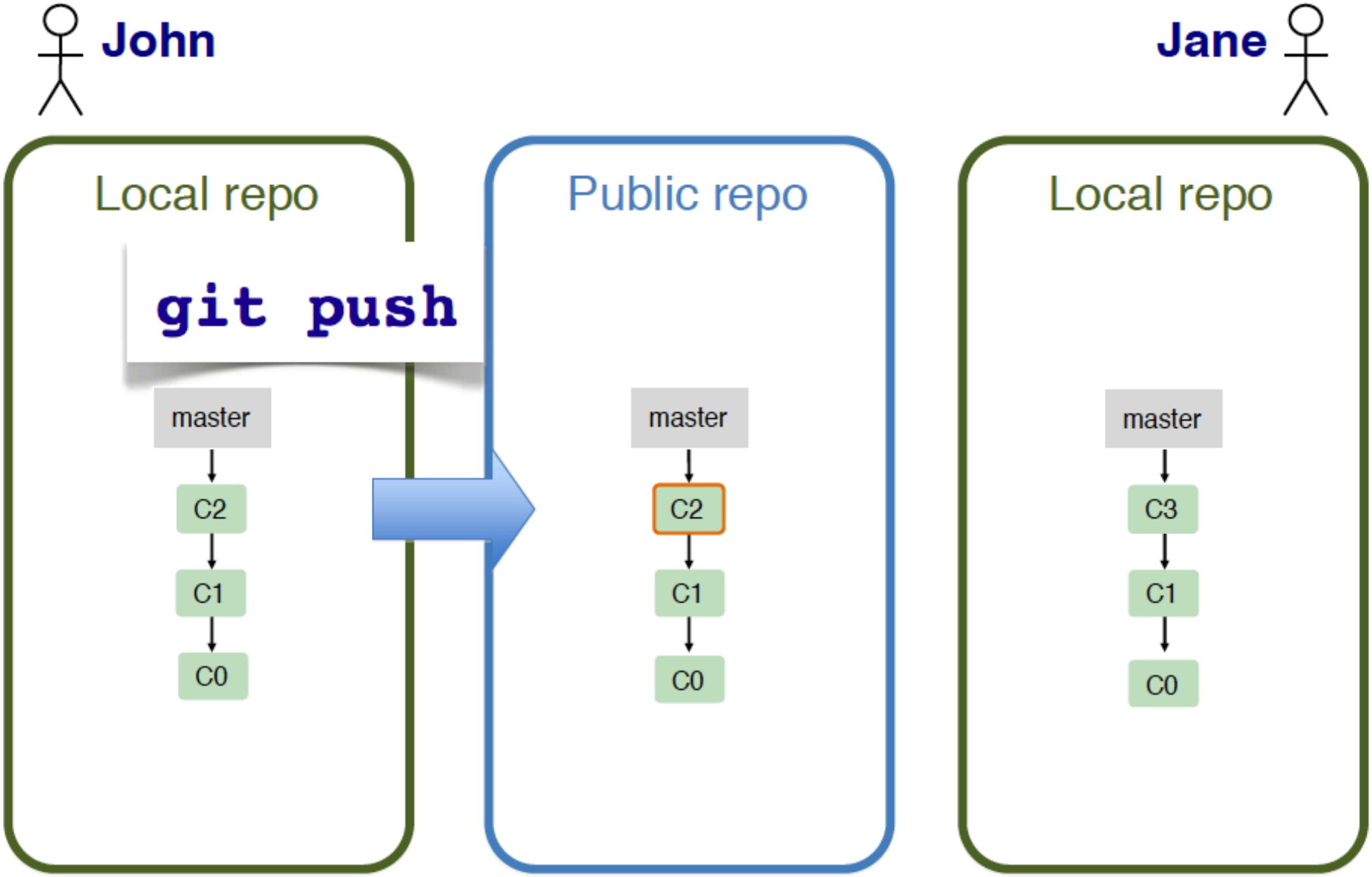

John
Jane
Local repo
Public repo
Local repo
master
C1
C0

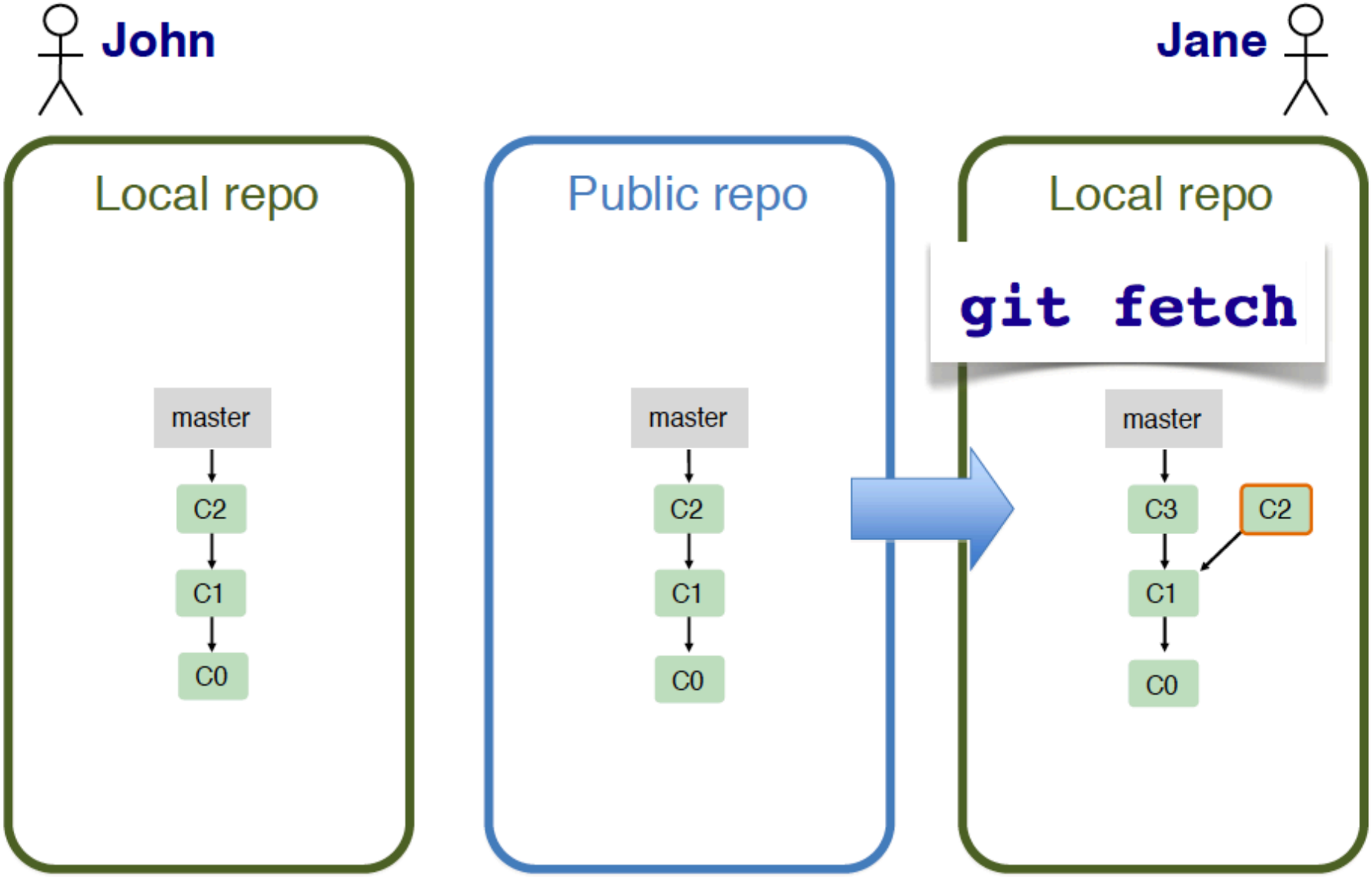

John
Jane
Local repo
Public repo
Local repo
git clone …
master
C1
C0
master
C1
C0
master
C1
C0
git clone …

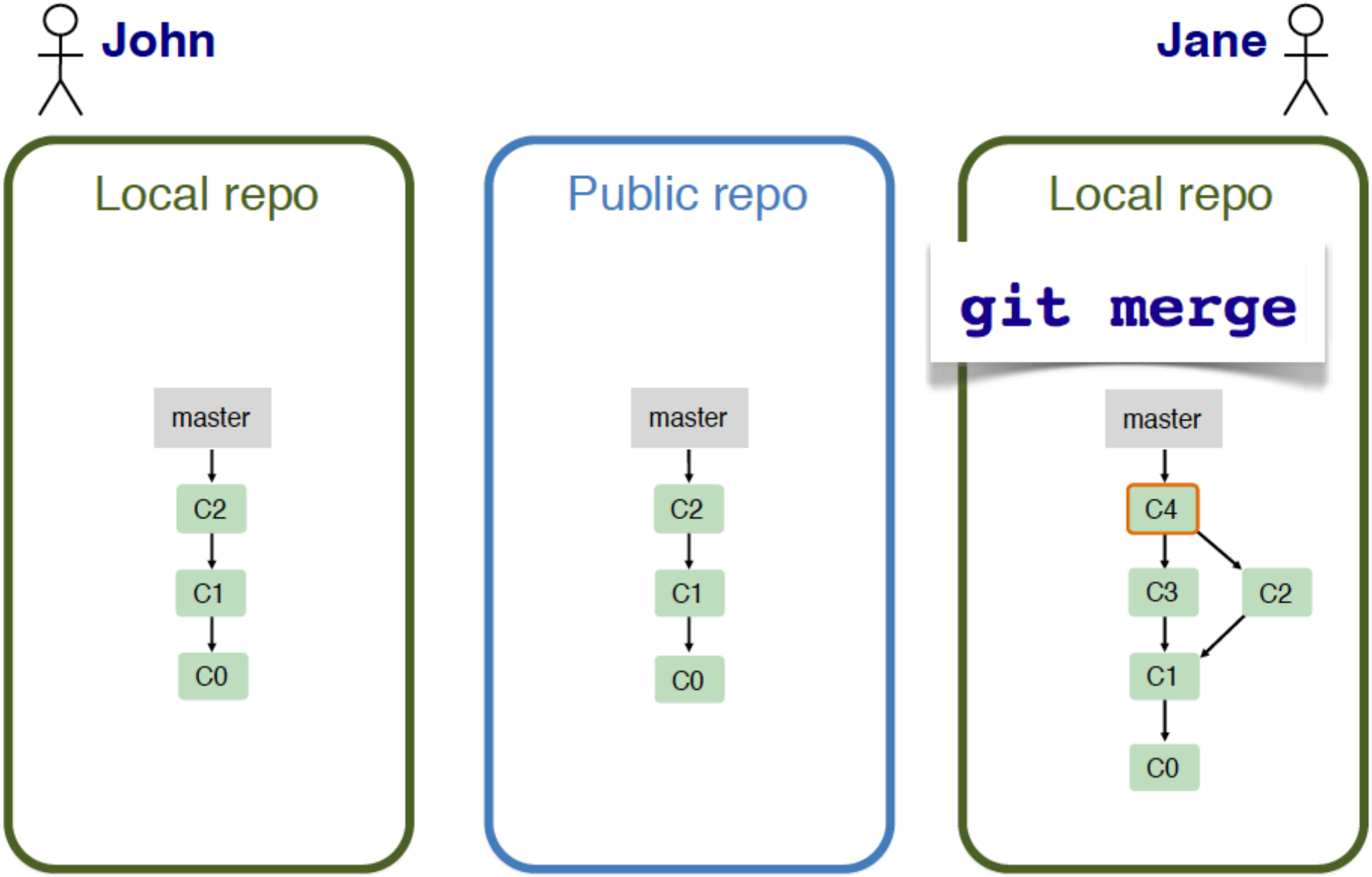

John
Jane
Local repo
Public repo
Local repo
master
C2
C1
C0
master
C1
C0
master
C3
C1
C0
git add …
git commit …
git add …
git commit …



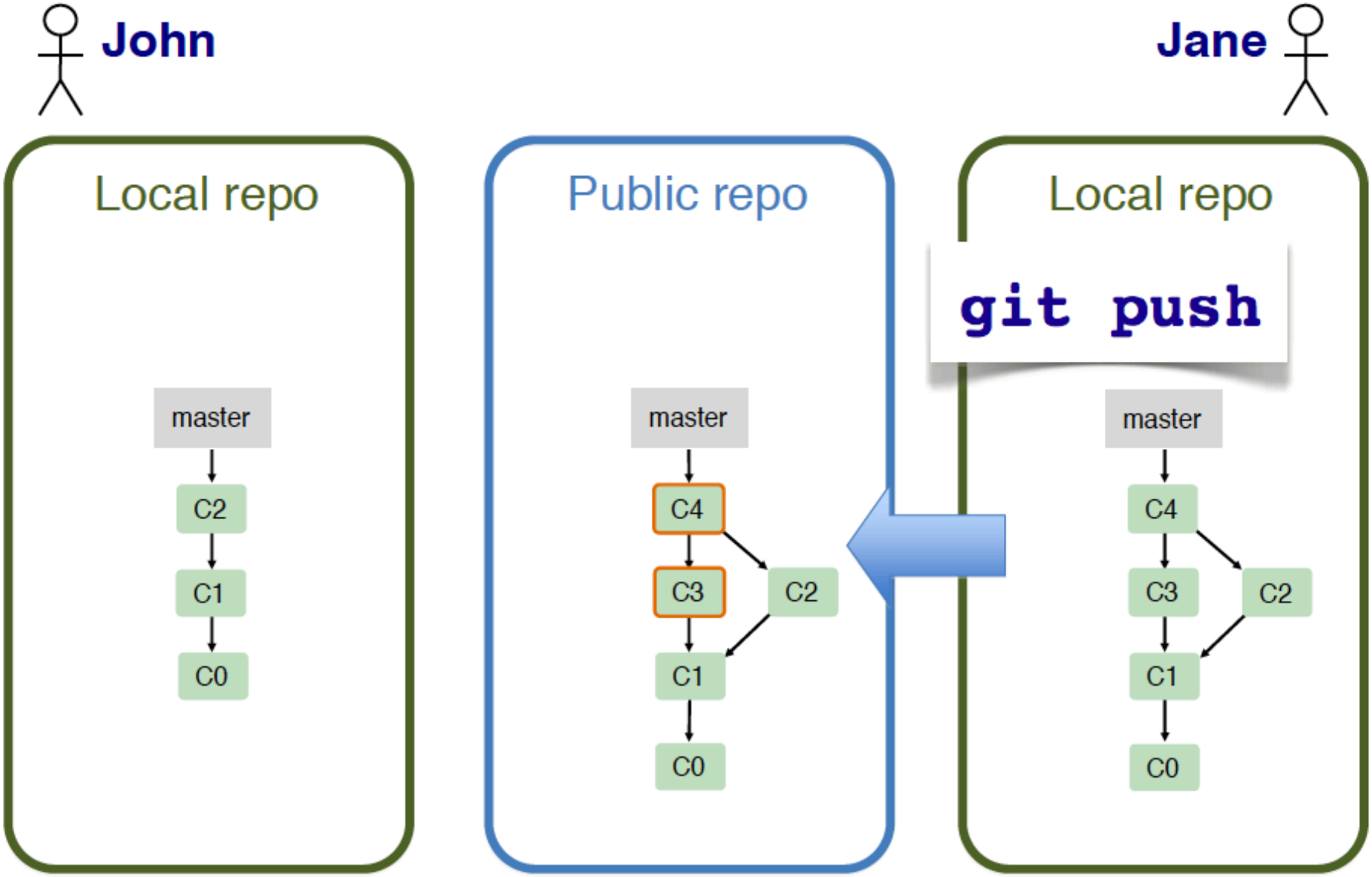
(nothing new to pull)

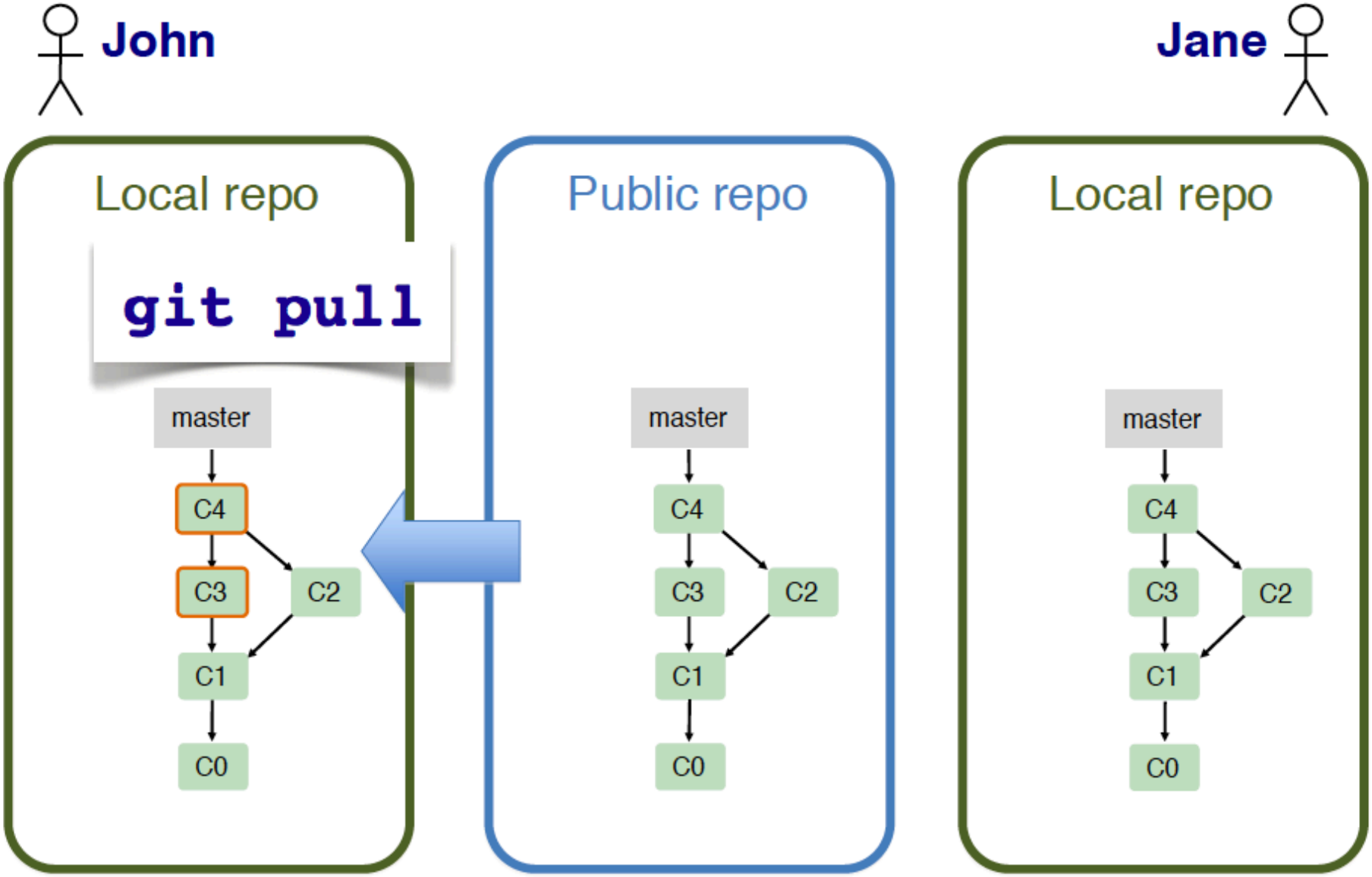

John
Jane
Local repo
git push
master
C2
C1
C0
Public repo
master
C2
C1
C0
Local repo
master
C3
C1
C0


John
Jane
Local repo
master
C2
C1
C0
Public repo
master
C2
C1
C0
Local repo
git fetch
master
C3
C2
C1
C0



NB: **git pull** = fetch + merge

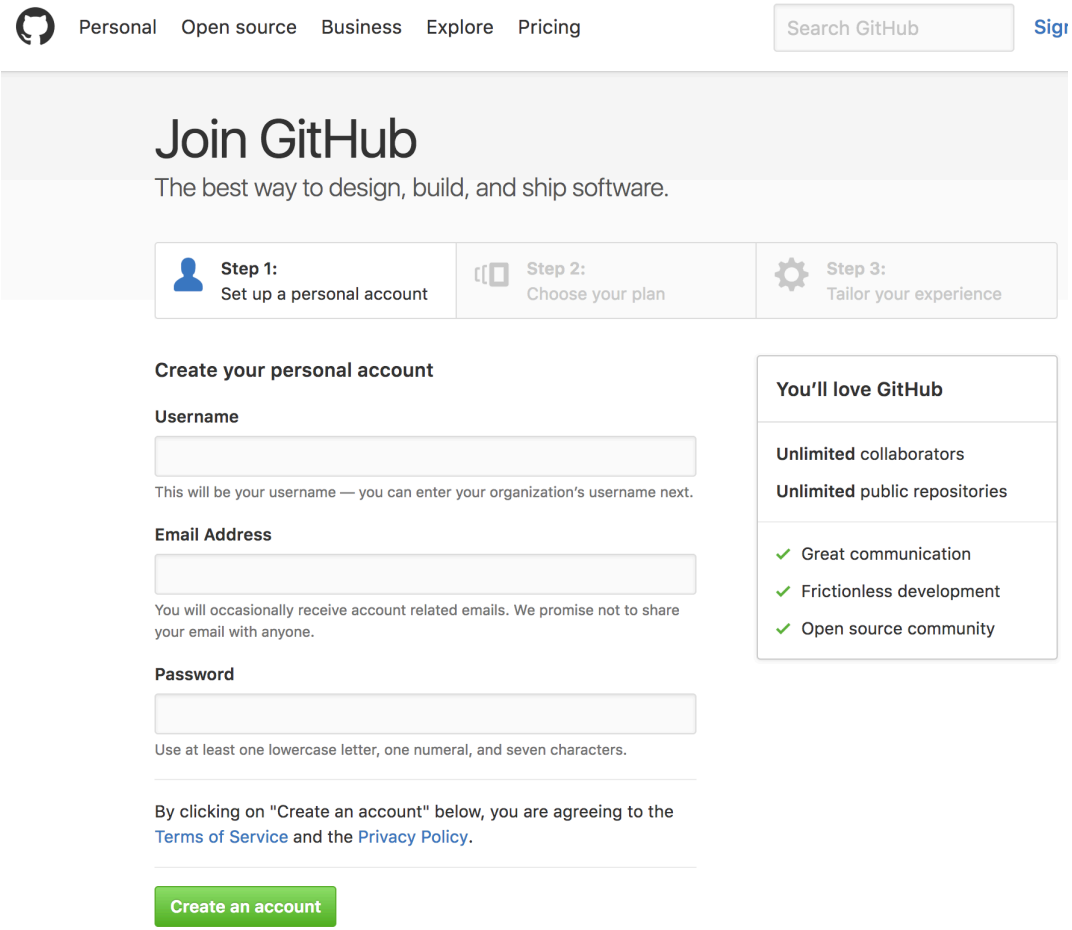

John
Jane
Local repo
master
C2
C1
C0
Public repo
master
C4
C3
C2
C1
C0
Local repo
git push
master
C4
C3
C2
C1
C0


John
Jane
Local repo
git pull
master
C4
C3
C2
C1
C0
Public repo
master
C4
C3
C2
C1
C0
Local repo
master
C4
C3
C2
C1
C0

Try it yourself!

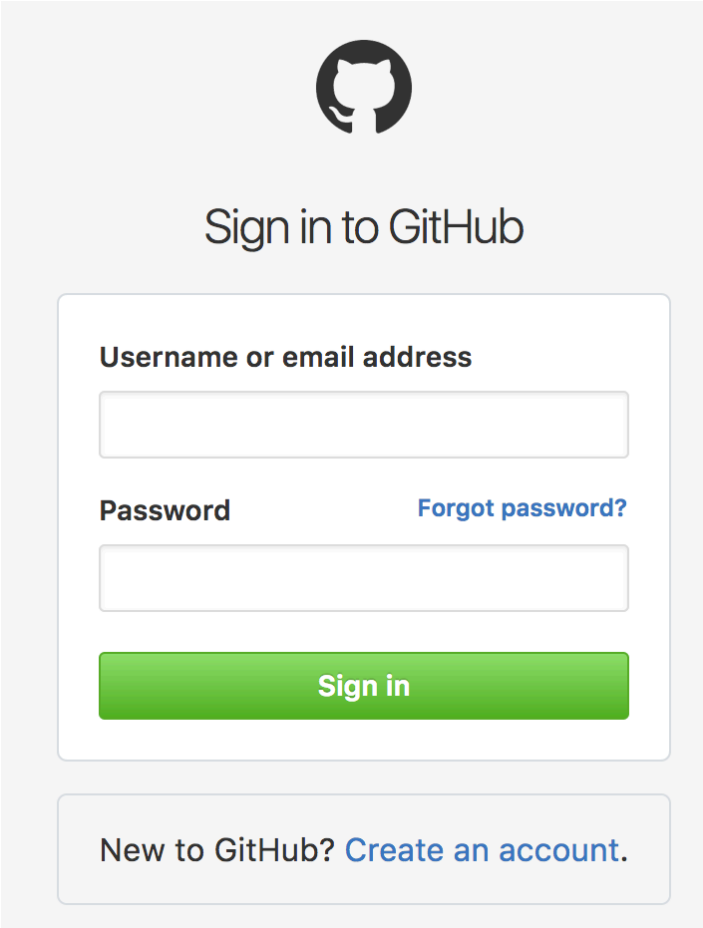
Create a Github account

- Go to https://github.com/join?source=login
- Fill the fields and click on “Create an account”



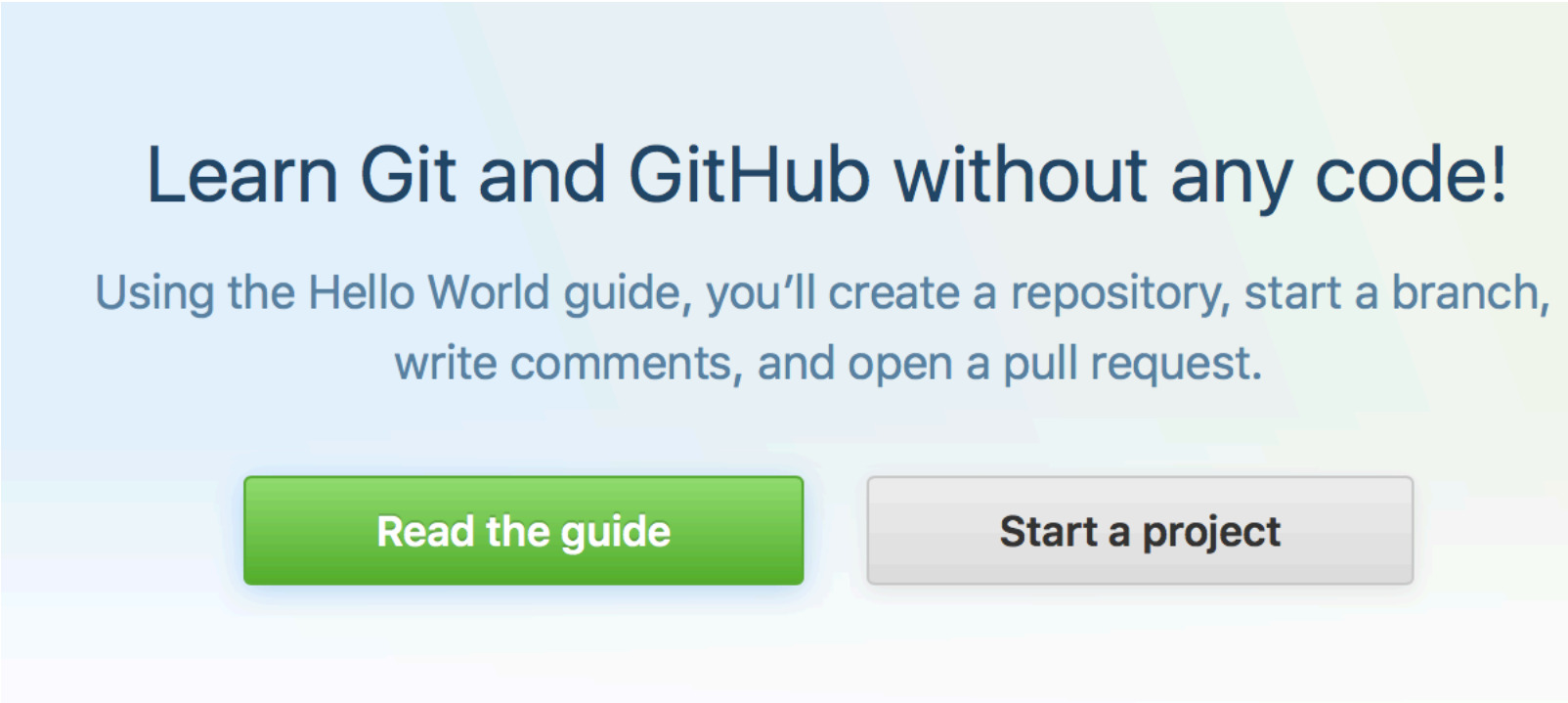
Login to Github

- Go to https://github.com/login
- Insert the username and password you used to sign up



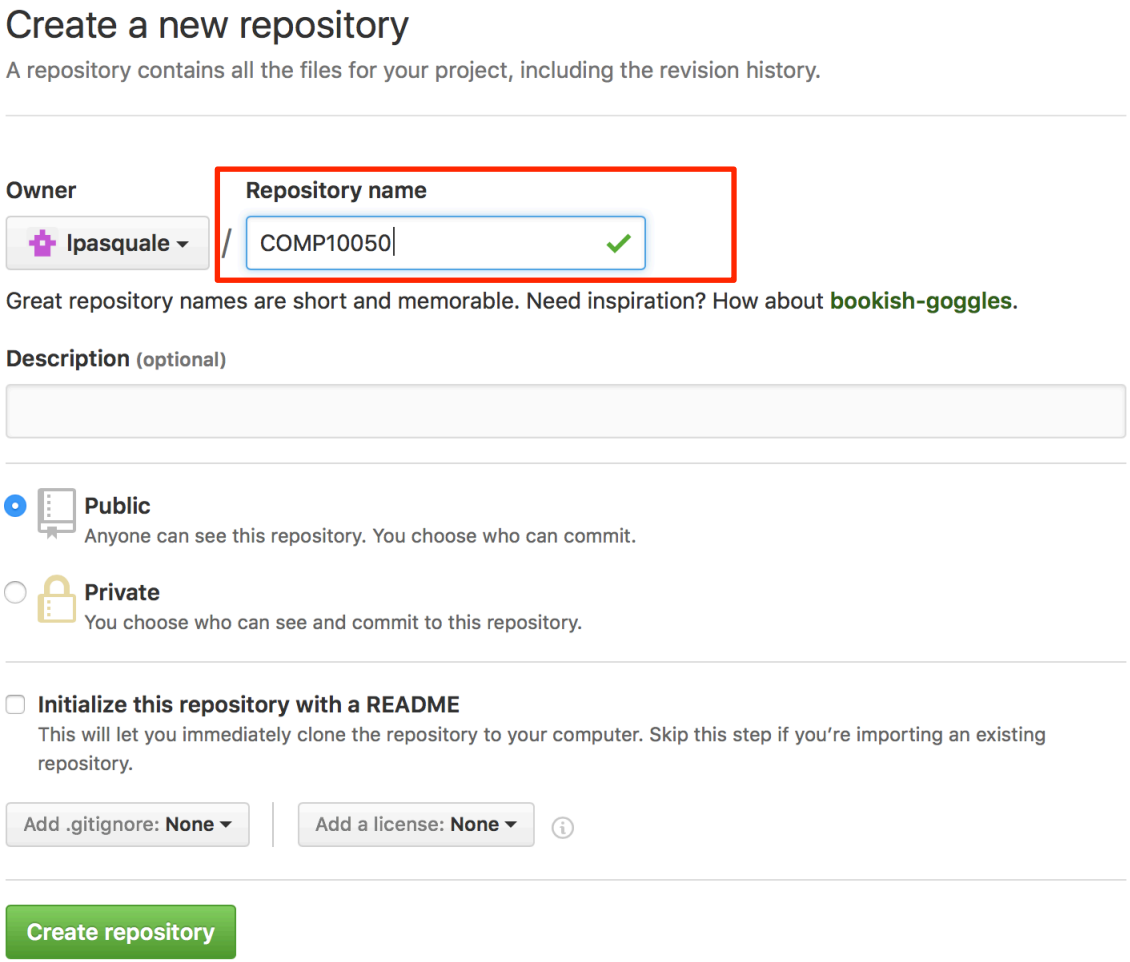
Create a New Repository

- Click to “Start a Project”



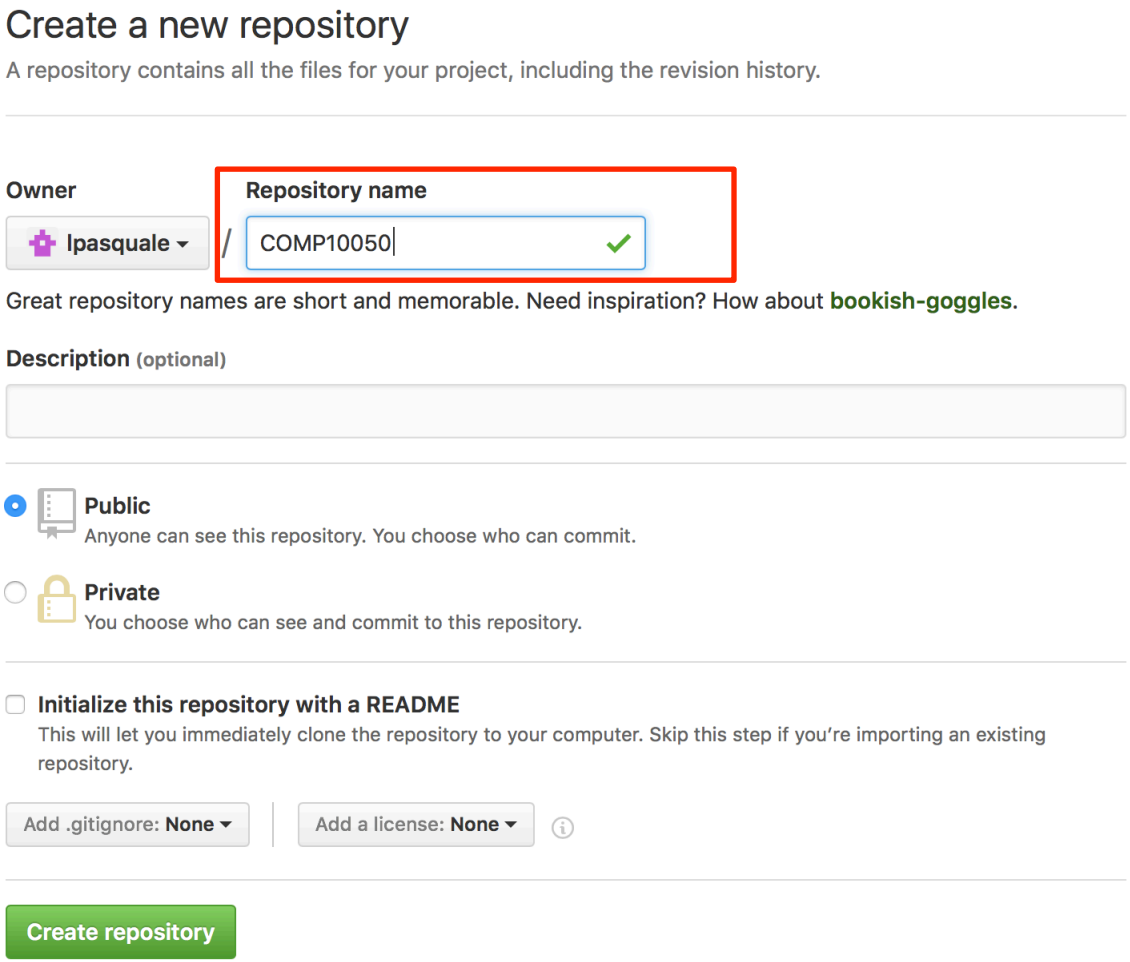
Create a New Repository

- Insert a repository name
- Click “Create Repository”



Create a New Repository

- Insert a repository name
- Click “Create Repository”



Create a New Repository

lpasquale / COMP10050

Watch 0 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/lpasquale/COMP10050.git

We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# COMP10050" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/lpasquale/COMP10050.git
git push -u origin master
```

- Link to the git repository

Initializing a Repository

- Create a directory where you want to place your repository
- Go to the directory you created using the command line
- Type the following

```
$ git init
```

This command creates the .git subdirectory

Initializing a Repository

Start version-controlling existing files in your repository

- Create a text file `Lab1.txt` in the project's directory and type:

```
$ git add Lab1.txt
$ git commit –m 'initial project version'
$ git remote add origin <link to your git
repository>
$ git push -u origin master
```

Checking the Status of Your Files

- Go to directory of the repository that you initialized and type:

```
$ git status
```

  It should show that your branch is up-to-date:
  - no tracked files (i.e. files staged for commit)
  - no modified files (files modified but not yet staged for commit)
  - no untracked files (newly added files not staged for commit)

Checking the Status of Your Files

- Now, add a new file to your project (a simple README file)

```
$ echo 'My Project' > README
```

- Type the status command again to see your untracked file:

```
$ git status
```

Tracking New Files

- Go to your project directory and type

```
$ git add README
```

- Type the status command again to see that the README file is now tracked and staged to be committed

```
$ git status
```

- You can tell that it is staged because it is under “Changes to be committed” heading.
- If you commit at this point, the version of the file at the time you ran `git add` is what will be placed in the historical snapshot.

Staging Modified Files

Let's change a new file that was already tracked.

- Create file MODIFIED.md and add it for tracking.
- Then modify it and run the `status` command:

```
$ git status
```

The `MODIFIED.md` file appears under a section named "Changes not staged for commit". You will need to run the `add` command again to include the changes in the next commit.

```
$ git add MODIFIED.md
$ git status
```

Committing Your Changes

Remember that only the files that are staged (i.e., you have run `git add` on since you edited them) will go into this commit.

```
$ git commit
```

- Doing so launches your editor of choice. This is set by your $EDITOR environment variable. You can also configure it with `git config --global core.editor`.
- You can type your commit message. The default message contains the output of the `git status` command.

- You can type your commit message inline with the `commit` command by specifying it after a `-m` flag.

```
$ git commit -m "My first commit"
```

Removing Files

To remove a file from Git, you have to remove it from your tracked files (i.e. remove it from the staging area) and then commit. The `git rm` command does that and also removes the files from your working directory.

- Create a new text file `PROJECT.md`, add it to the tracked files and commit. Then type the following:

```
$ rm PROJECT.md
$ git status
```

If you simply remove the file it shows up under "Changed but not updated" (that is the unstaged area).

- Type the following to stage the file's removal

```
$ git rm PROJECT.md
```

The file will be removed in the next commit

Removing Files

To remove a file (e.g., `README`) from Git, but still keeping it in your working directory. Type:

```
$ git rm --cached README
```

Viewing the Commit History

- Clone project
  `https://github.com/schacon/simplegit-progit`

- List the commits made in that repository in reverse:

```
$ git log
```

- Show the differences introduced in each commit, limiting the output to the last 2 entries

```
$ git log -p 2
```

- Show each commit in a single line:

```
$ git log --pretty=oneline
```

- Or in a specific format:

```
$ git log --pretty=format:"%h - %an, %ar : %s"
```

Pushing to a remote repository

Git push is essntially the transfer of your local information to the remote repository

```
$ git push
```

Once a git commit is complete, git push will send all the local changes to the remote branch

Pulling from repository

Incorporates changes from a remote repository into the current branch

```
$ git pull
```

“git pull” is shorthand for “git fetch” followed by
“git merge FETCH_HEAD”

- Git will show which files will be merged (over writtern), allowing the user to prevent losing files or allow it to run

Cloning

Incorporates changes from a remote repository into the current branch

```
$ git pull
```

“git pull” is shorthand for “git fetch” followed by
“git merge FETCH_HEAD”

- Git will show which files will be merged (over writtern), allowing the user to prevent losing files or allow it to run

Cloning an Existing Repository

Get a copy of an existing repository you would like to contribute to.

- From a different directory type

```
$ git clone <link to your git repository>
```

  - Every version of every file for the history of the project is pulled down
  - This is the equivalent of the “checkout command” for Subversion

Exercise

- Create a new Git repository in directory A
- Create file README1.txt, add it to the repository and commit
- Push changes to the remote repository
- Clone the repository to directory B
- Create file README2.txt, add it to the repository and commit
- Push changes to the remote repository

- From directory A, pull changes
- You should be able to see files README1.txt and README2.txt