Lecture 4

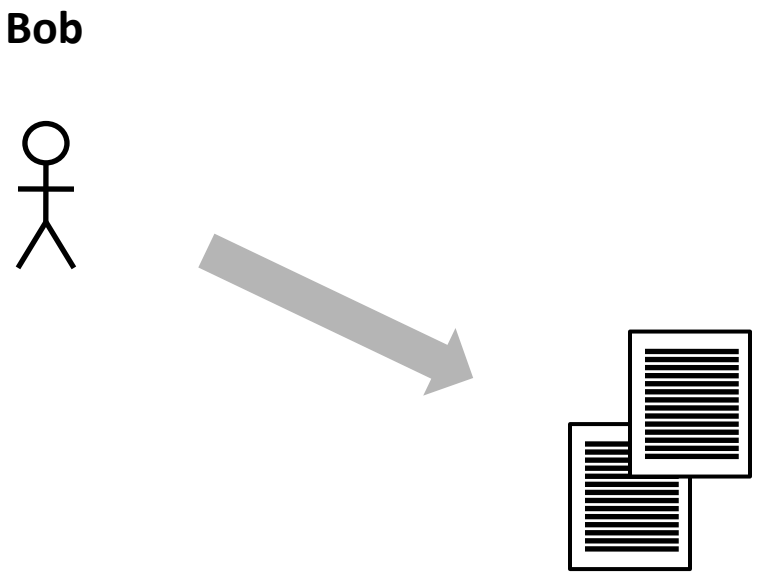
Distributed Version Control and Git

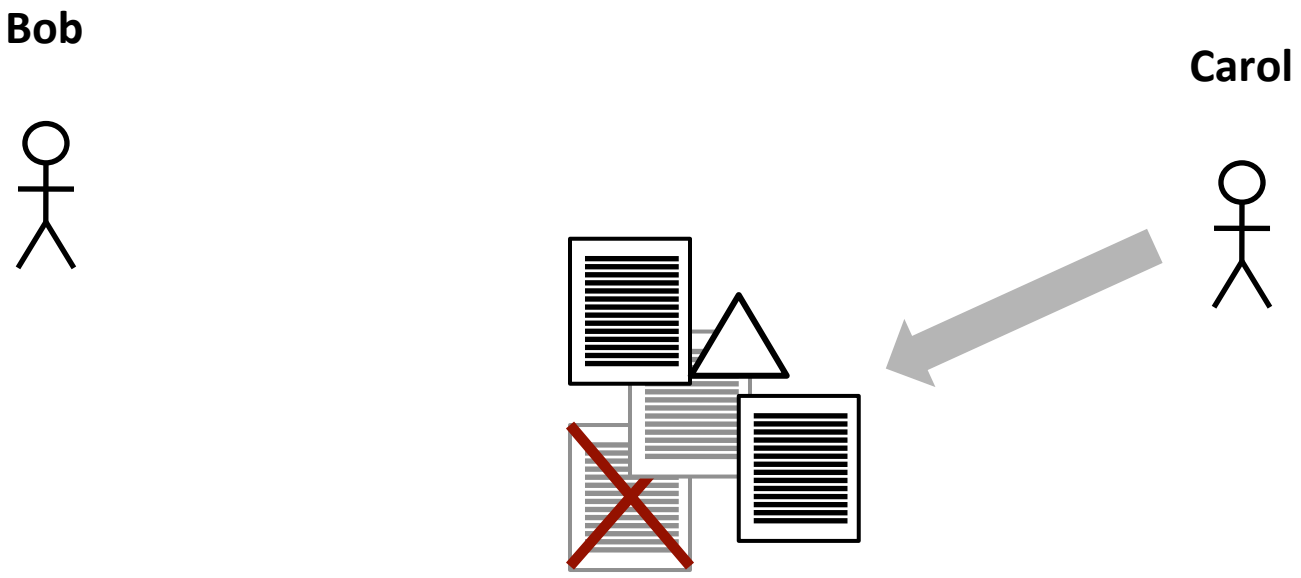
Outline

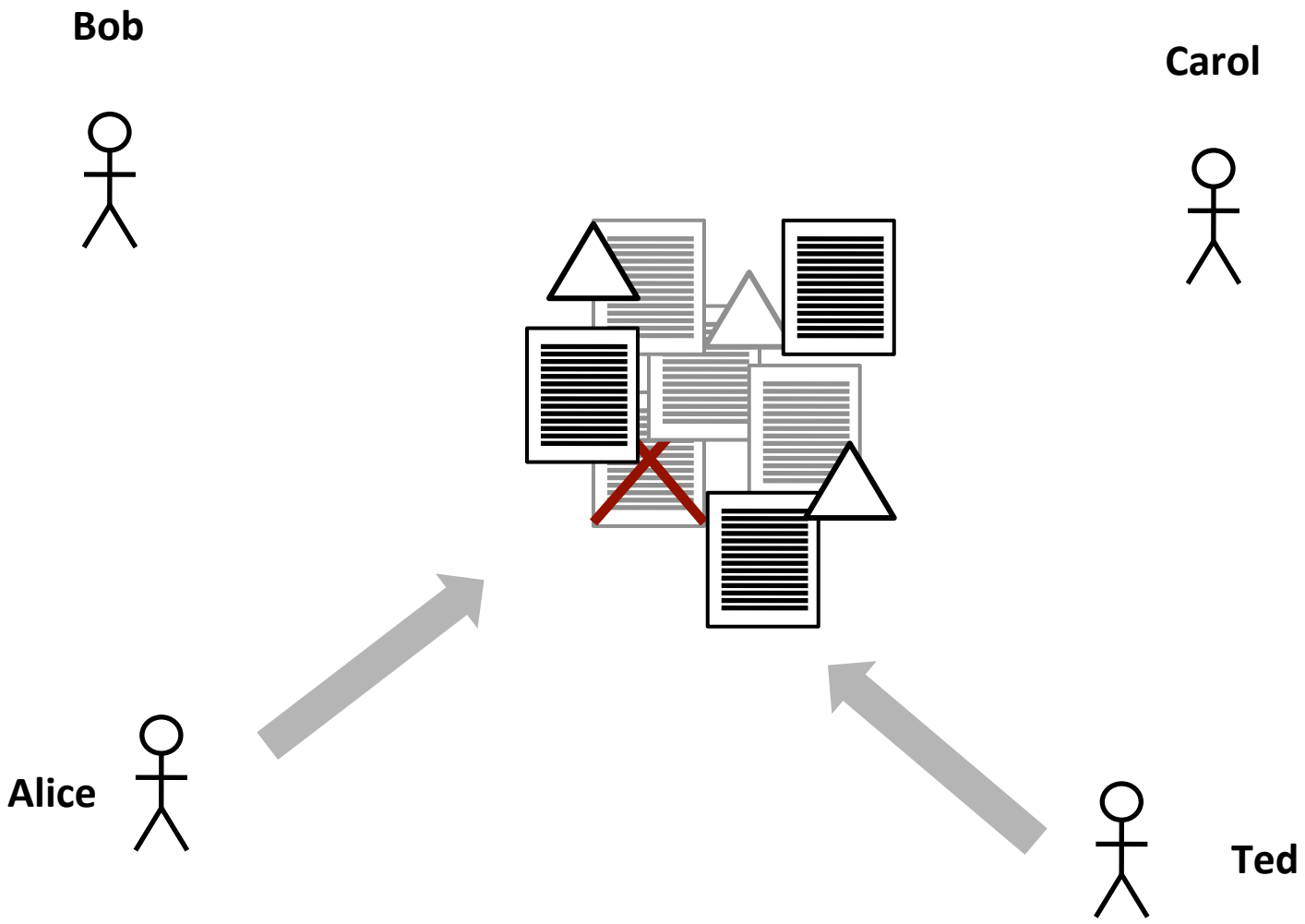
- Introduction to Version Control Systems
  - Terminology
  - Centralized VS Distributed Version Control
  - Data Storage in SVN vs Git

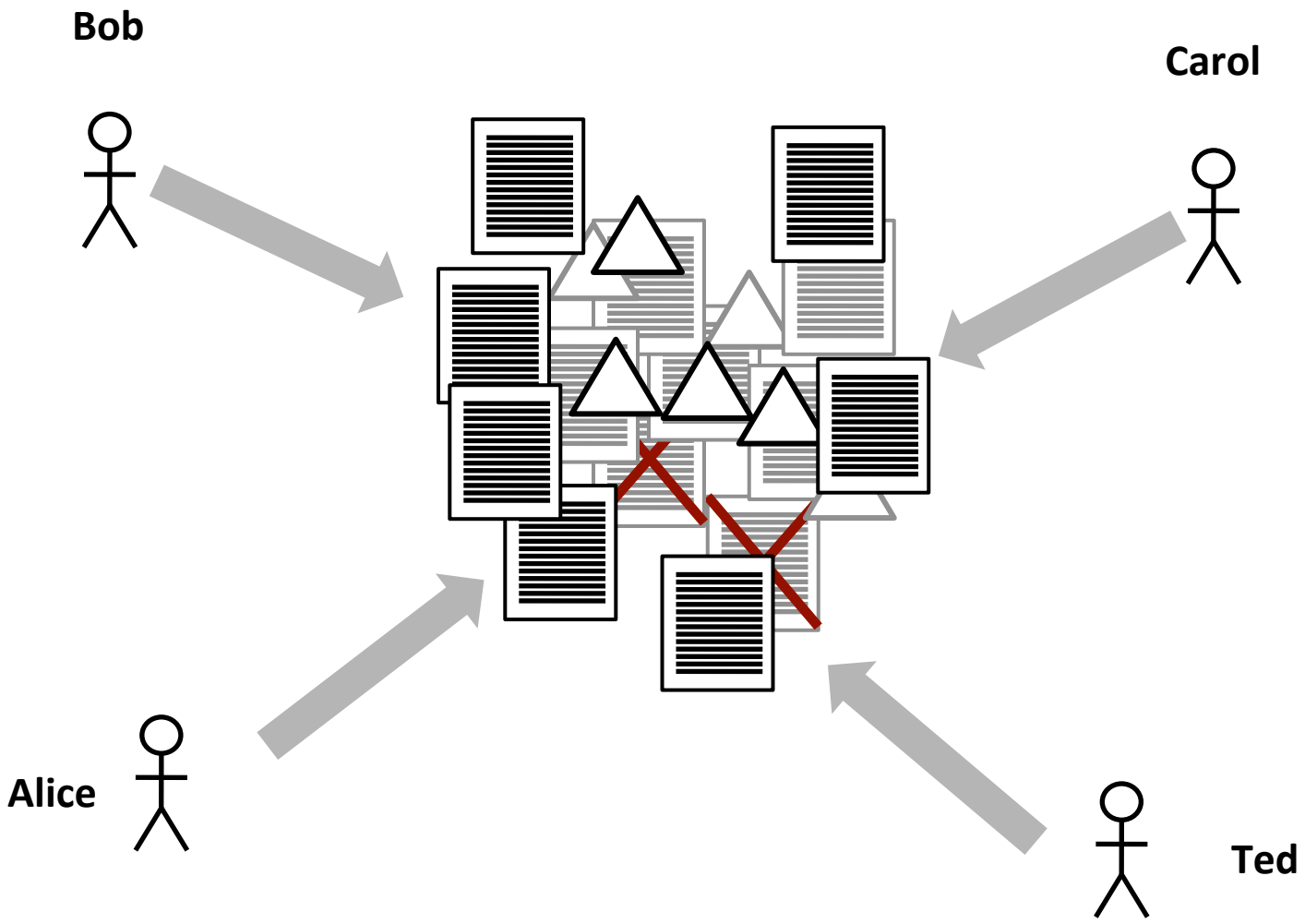
- Git
  - Object Model
  - Basic Operations
  - Collaboration

Motivation

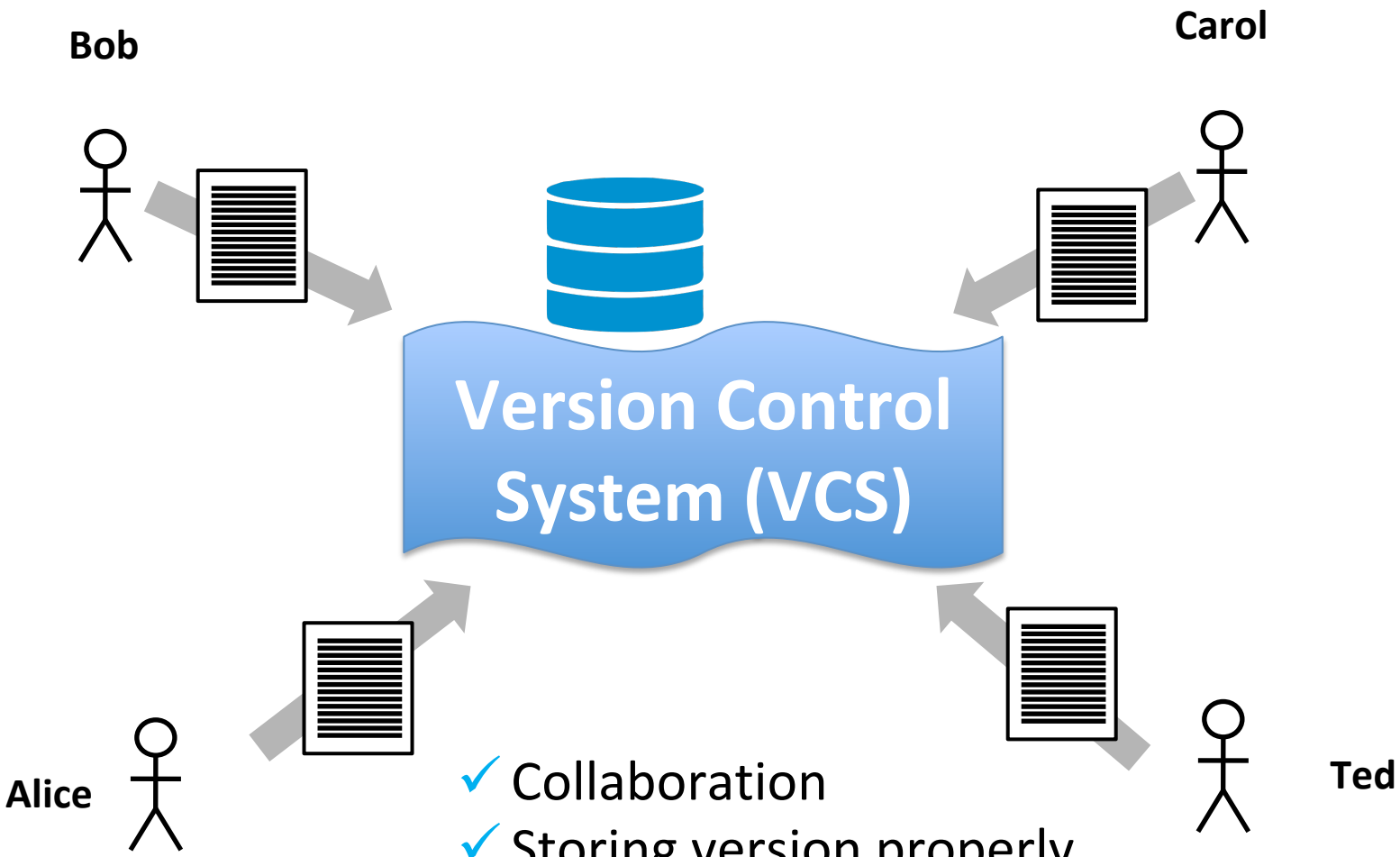

Bob


Bob
Carol


Bob
Carol
Alice
Ted



A recipe for disaster!



- ✓ Collaboration
- ✓ Storing version properly
- ✓ Restoring previous versions
- ✓ Understanding what happened
- ✓ Backup

Some Terminology

Repository

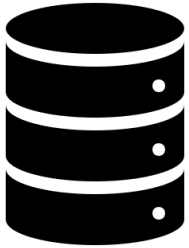
Branches

Working Copy

Commit (Check in) & Update (Check out) Changes

Some Terminology

Repository



- The place where developers store all their work
- Not only stores files but also the history of changes
- Accessed over the network

Branches

Working Copy

Commit (Check in) & Update (Check out) Changes

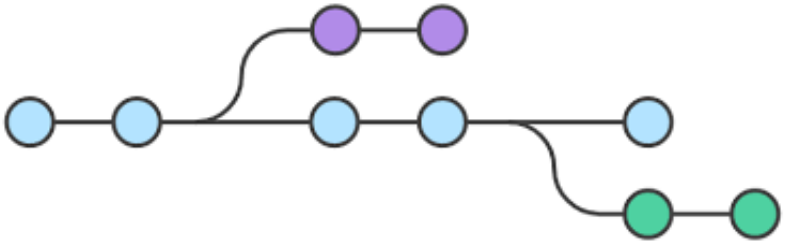
Some Terminology

Repository

Branches

- Used to create another line of development, e.g. when you want your development process to fork into 2 different directions

Working Copy



Commit (Check in) & Update (Check out) Changes

Some Terminology

Repository

Branches

Working Copy

- A snapshot of the repository
- Private workplace where developers can do their work remaining isolated from the rest of the team

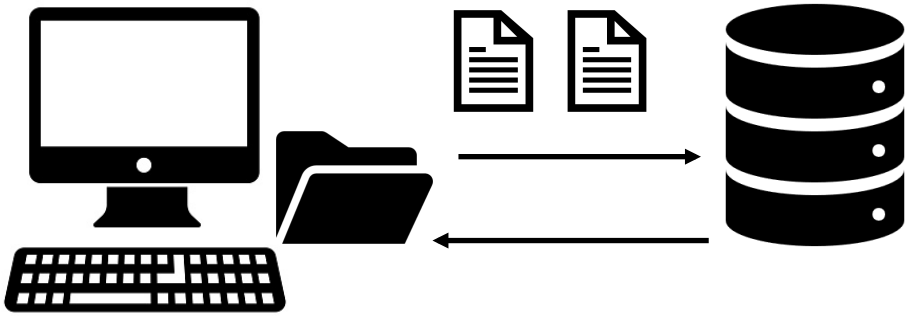
Commit (Check in) & Update (Check out) Changes

Some Terminology

Repository

Branches

Working Copy

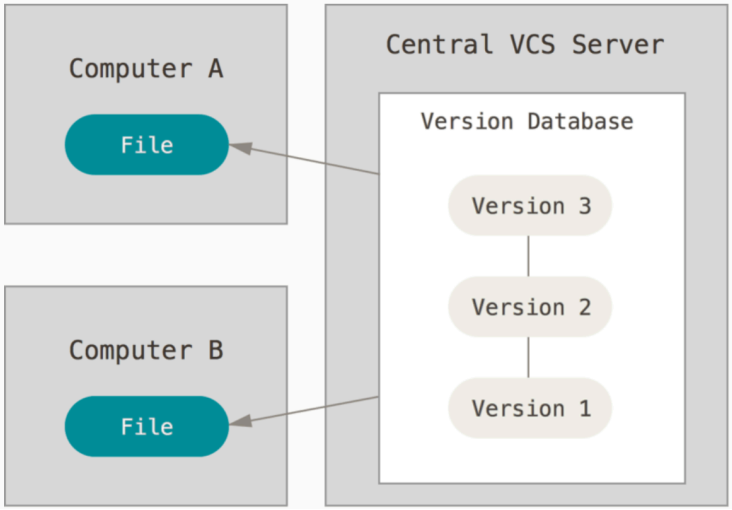


Commit (Checkin) & Update (Checkout) Changes

- Checkin: stores changes from private workplace to the repository
- Checkout: updates the files in the working directory to match the version stored in a branch

Centralised Version Control Systems

- Examples: Subversion (SVN), CVS, Perforce
- A central server repository (repo) holds the "official copy" of the code
  - the server maintains the sole version history of the repository
- You make "checkouts" of it to your local copy
  - you make local modifications
  - your changes are not versioned

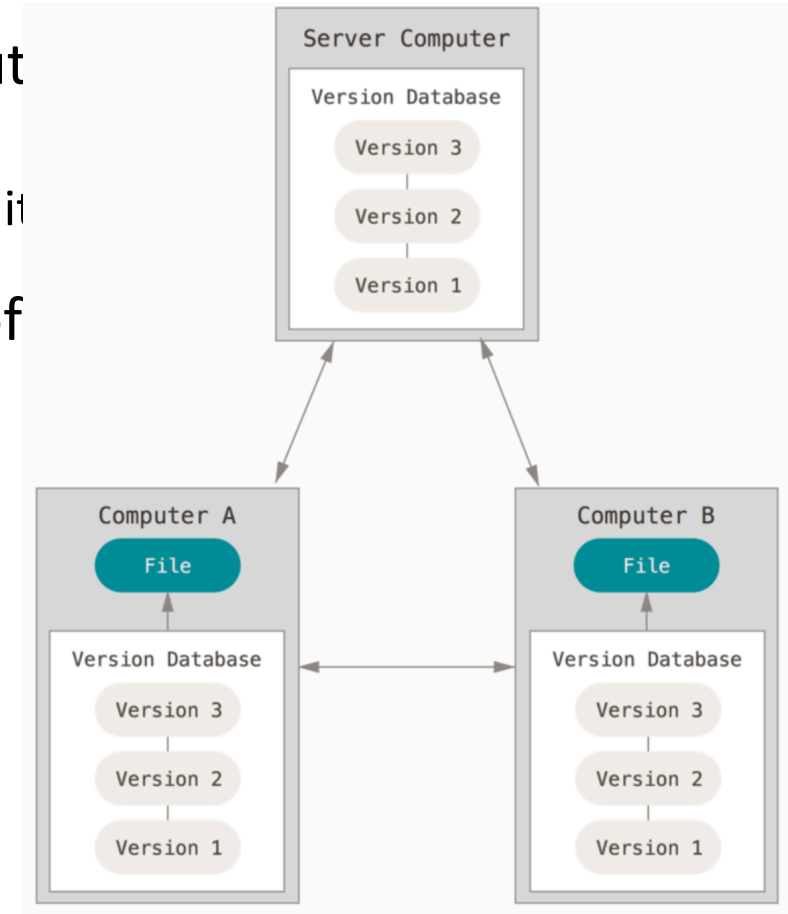


- When you're done, you "check in" back to the server
  - your checkin increments the repo's version

Single point of failure

Distributed Version Control Systems

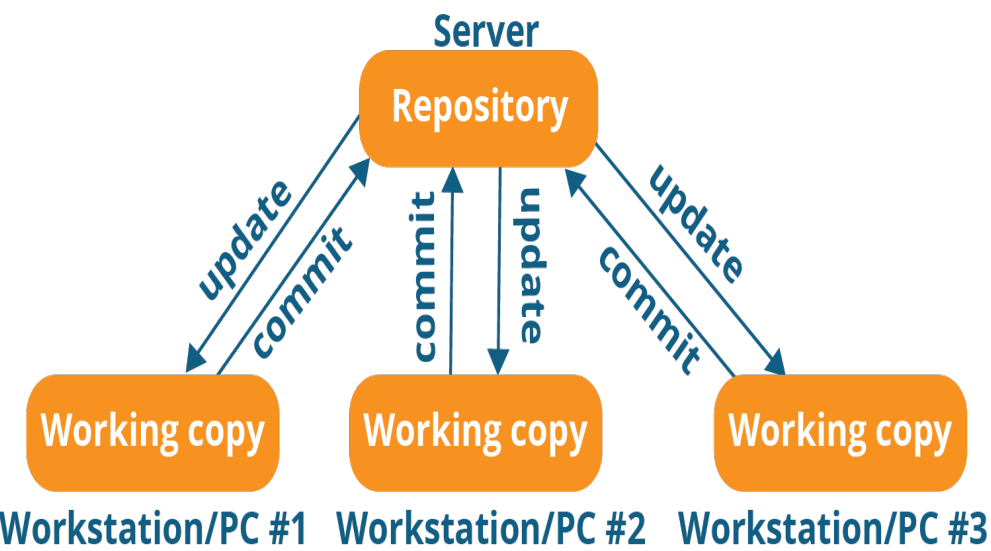
- In Git, Mercurial you don't "checkout from a central repo
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from local repo
  - commit changes to local repo
  - local repo keeps version history
- When you're ready, you can "push" changes back to server

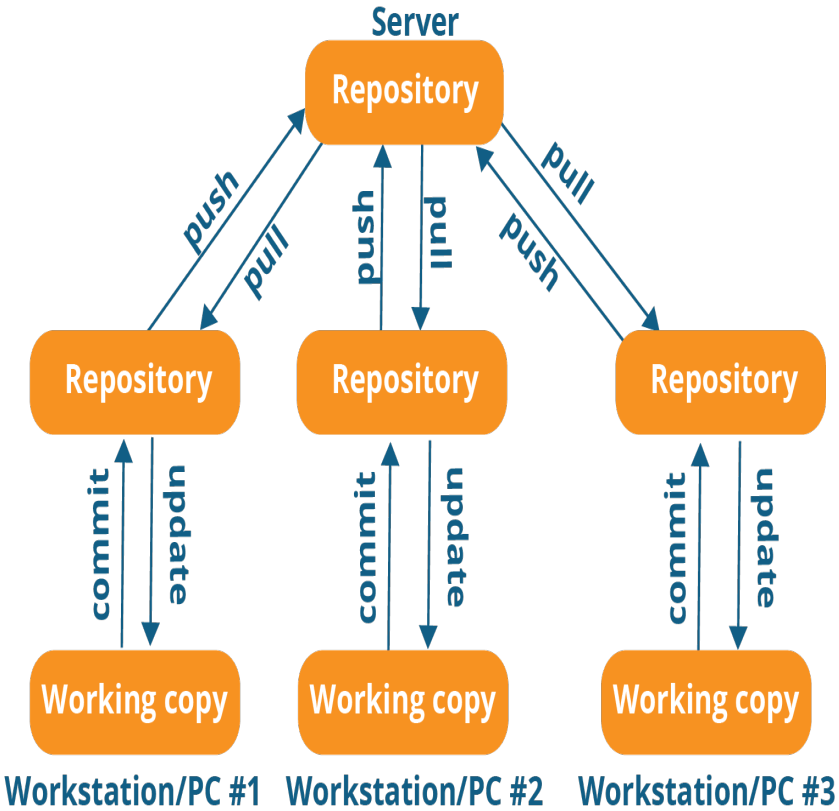


Most operations seem to be executed almost instantaneously.

Version Control Systems

Centralized vs Distributed





About Git

- Distributed VCS
- Originally developed by Linus Torvalds, creator of Linux, in 2005
  - Designed to do version control on Linux kernel
- Goals of Git:
  - Speed
  - Support for non-linear development (thousand of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

(A “git ” is a cranky old man. Linus meant himself.)

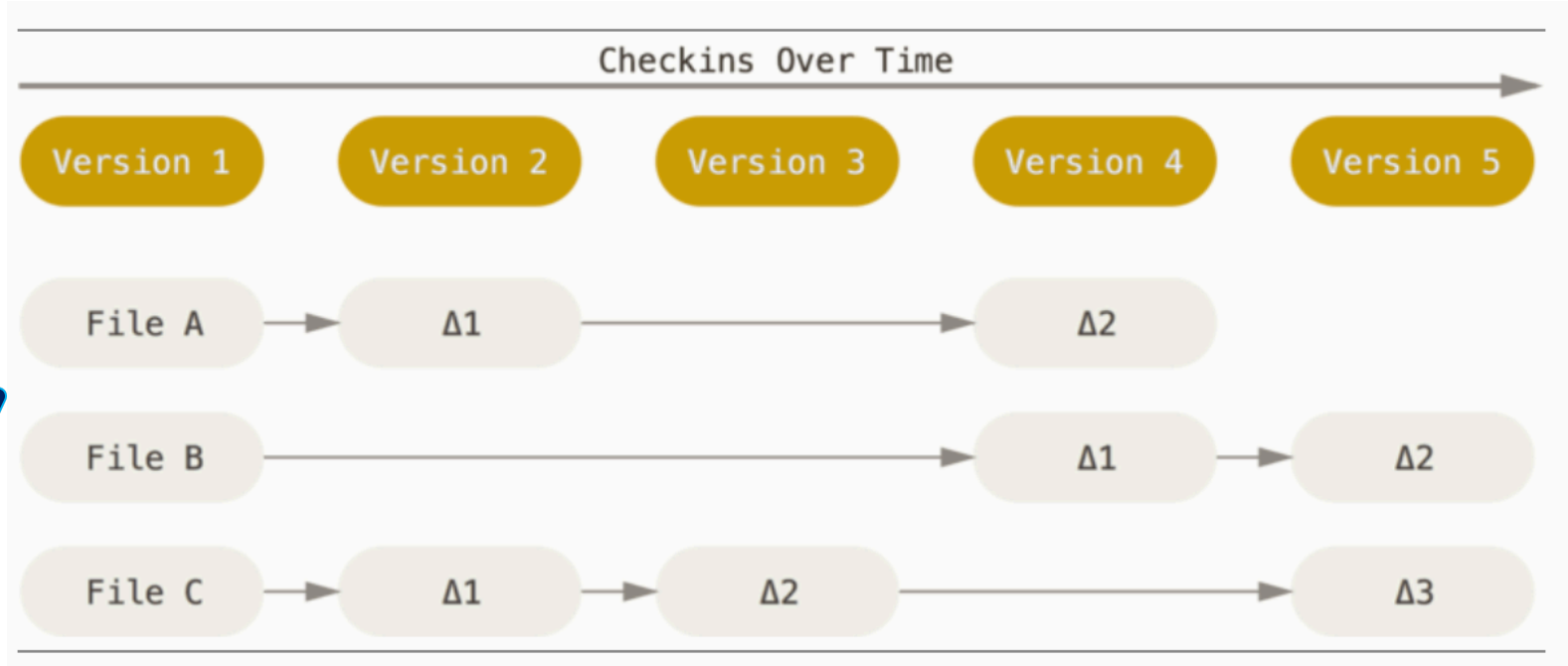
Installing/Learning Git

- Git website: http://git-scm.com
  - Free on-line book: http://git-scm.com/book
  - Reference page for Git: http://gitref.org/index.html
  - Git tutorial: http://schacon.github.com/git/gittutorial.html
  - Git for Computer Scientist http://eagain.net/articles/git-for-computer-scientists
  - Git Cheat Sheet https://education.github.com/git-cheat-sheet-education.pdf

- At command line (where verb = config, add, commit, etc.)
  - `git help` *`verb`*

How Data Are Stored

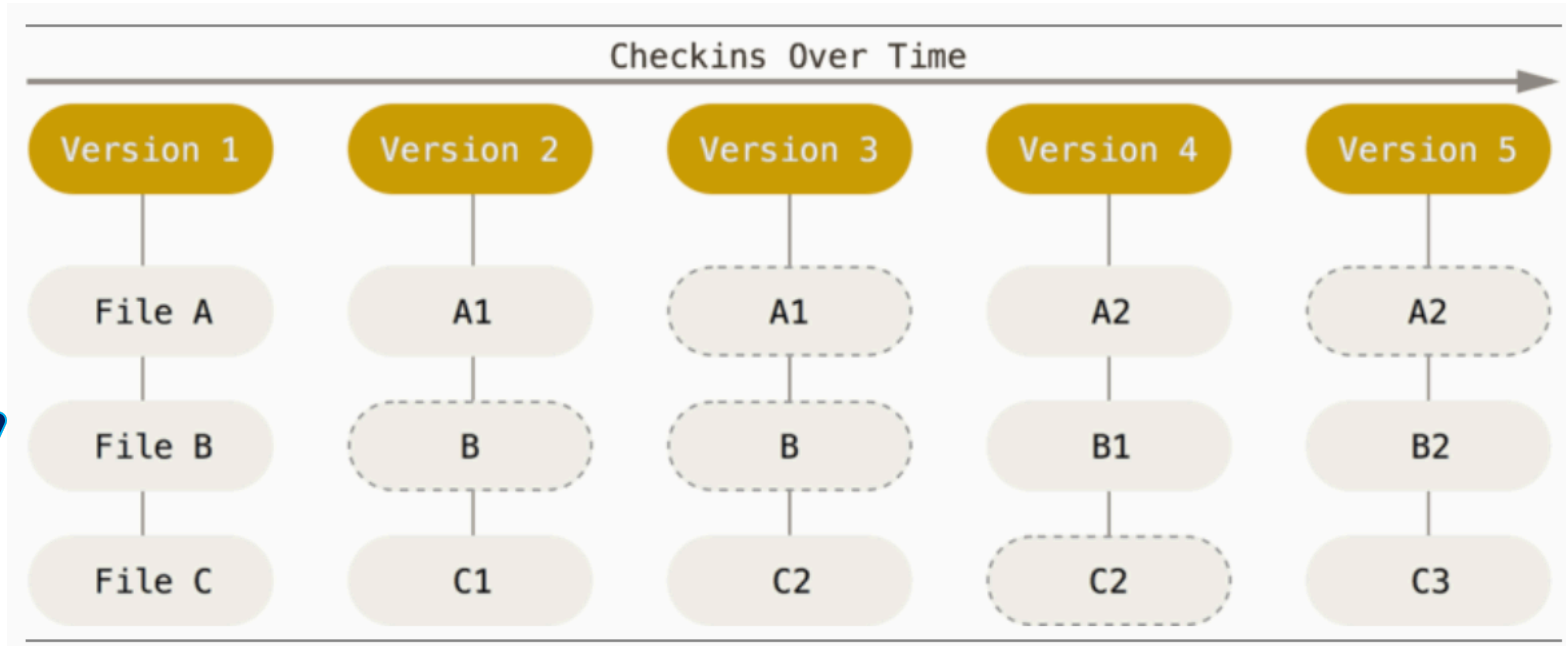
SVN: Delta storage model



Data are stored as changes to a base version of the same file

How Data Are Stored

Git: Snapshot storage model



Data are stored as snapshots of the project over time.

- Some files change on a given check in, some do not.
  - More redundancy, but faster
  - If not changed, a link to the previous identical file it has already stored

Most Important Thing to Remember About Git

A File can be in **3 main states:**

- **Modified:** the file has been changed but not yet committed.
- **Staged:** the file is marked as modified and will be included in the next commit snapshot. => Every commit is "hand-crafted”
- **Committed:** the data is safely stored in your local database.

Git Workflow

```
    Working                    Staging              .git directory

   Directory                    area                 (Repository)

       |                          |                      |

       |      git checkout        |                      |

       | <========================================== |
```

Modified

```
       |                          |                      |

       |                          |                      |

       |                          |                      |

       |                          |                      |
```

Git Workflow

```
 Working                    Staging              .git directory

Directory                    area                 (Repository)

    |                         |                       |

    |      git checkout       |                       |

    | <=========================================== |

    |                         |                       |

    |     git add             |                       |

    | ======================>|                       |
                              Staged
    |                         |                       |

    |                         |                       |
```

Git Workflow

```
 Working                    Staging              .git directory

Directory                    area                 (Repository)

     |                        |                       |

     |       git checkout     |                       |

     | <========================================== |

     |                        |                       |

     |      git add           |                       |

     | ====================>|                       |

     |                        |                       |

     |                        |   git commit          |

     |                        | ===================>|
```

Committed

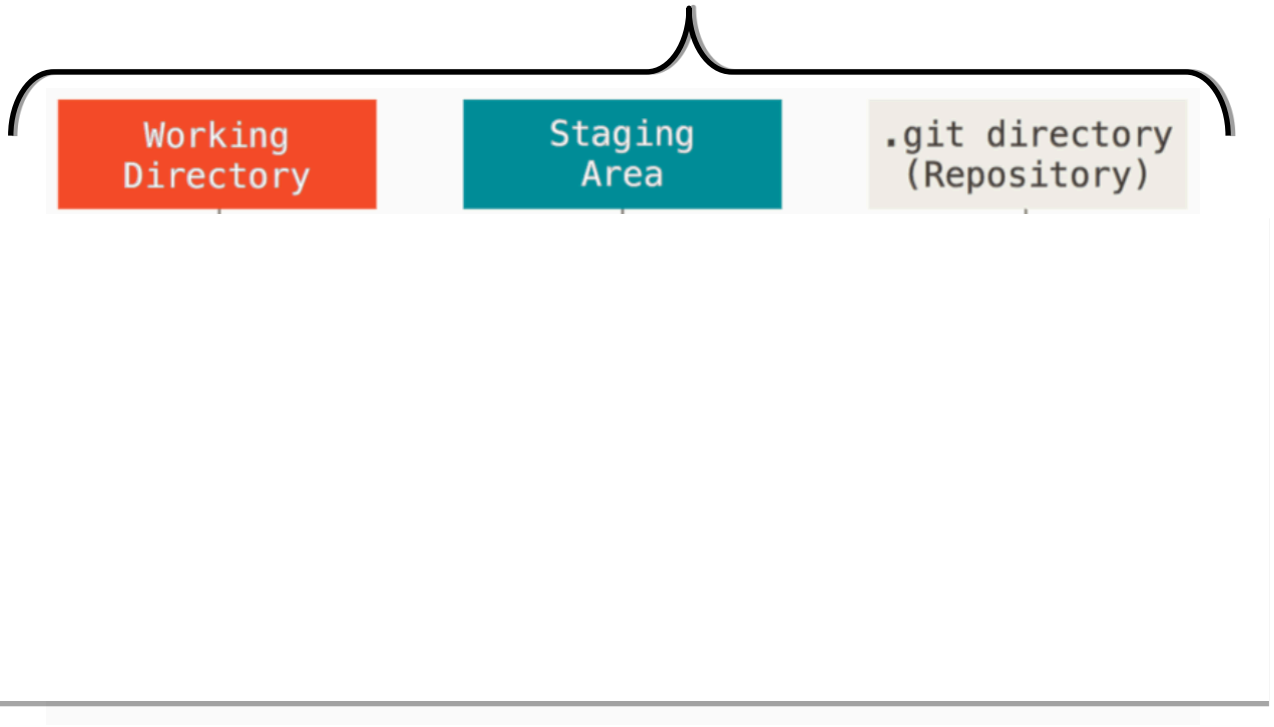
```
     |                        |                       |
```

Most Important Thing to Remember About Git

A File can be in 3 main states:

- **Modified:** the file has been changed but not yet committed.
- **Staged:** the file is marked as modified and will be included in the next commit snapshot.
- **Committed:** the data is safely stored in your local database.

Main Sections of a Git project
(created locally on your machine)



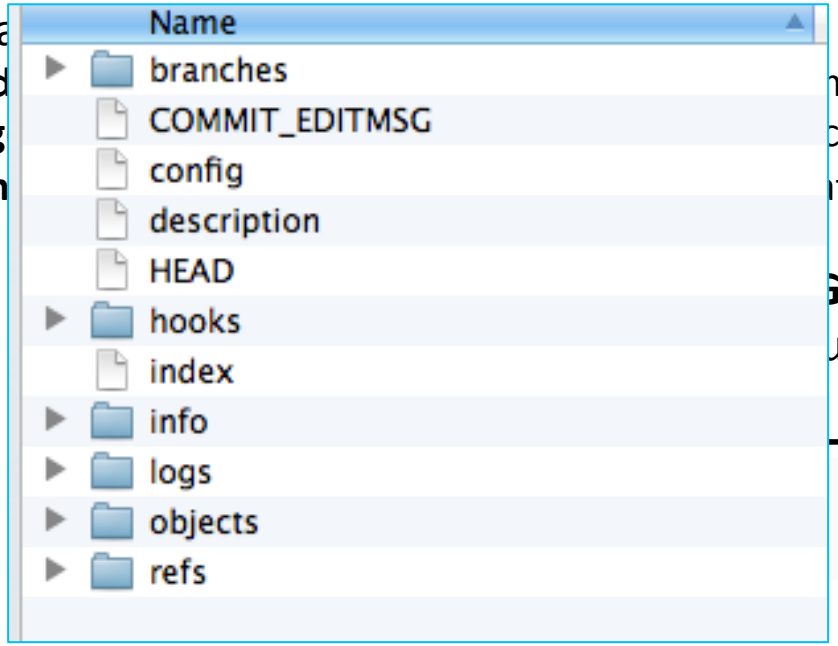
Most Important Thing to Remember About Git

A File ca

- **Mod** nmitted.
- **Stag** cluded in the next commit snapshot.
- **Com** tabase.

Git project

ur machine)



.git directory
(Repository)

- Metadata and object database for the project.
- This is what is copied when a repository is cloned from another computer.

Most Important Thing to Remember About Git

A File can be in 3 main states:

- **Modified:** the file has been changed but not yet committed.
- **Staged:** the file is marked as modified and will be included in the next commit snapshot.
- **Committed:** the data is safely stored in your local database.

Main Sections of a Git project
(created locally on your machine)

- Single checkout of one version of the project.
- These files are pulled out of the compressed database in the .git directory and placed on disk for you to use or modify.

Most Important Thing to Remember About Git

A File can be in 3 main states:

- **Modified:** the file has been changed but not yet committed.
- **Staged:** the file is marked as modified and will be included in the next commit snapshot.
- **Committed:** the data is safely stored in your local database.

Main Sections of a Git project
(created locally on your machine)

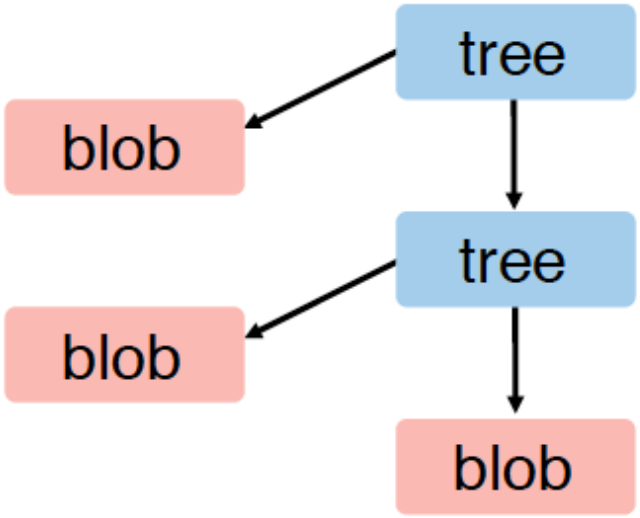
- It is a file, generally contained in your .git directory, that stores information about what will go into your next commit.

The Git object model

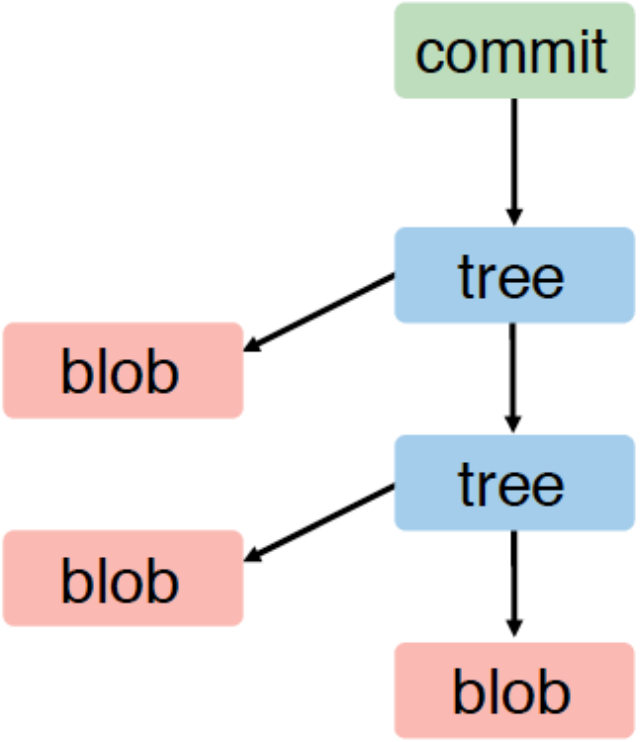
A “blob” is *content* under version control (a file)

blob

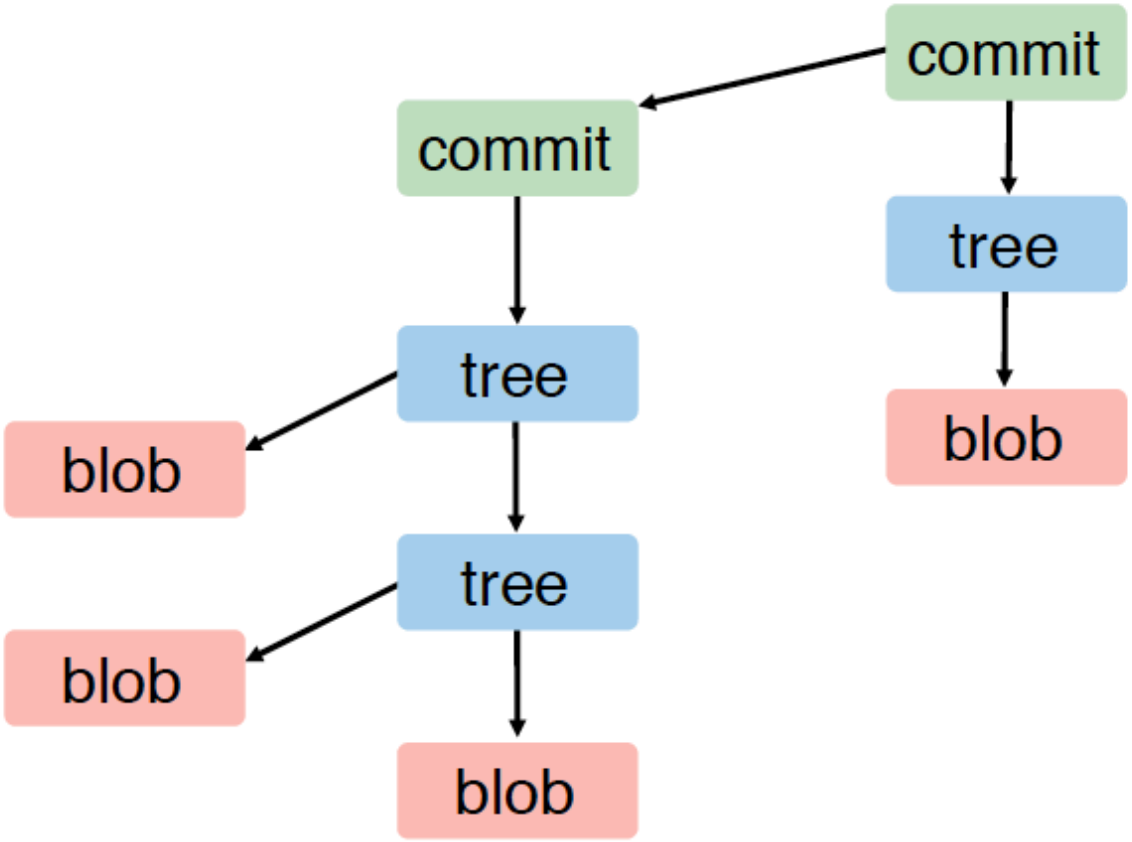
You can have *trees* of blobs
(directories of files)



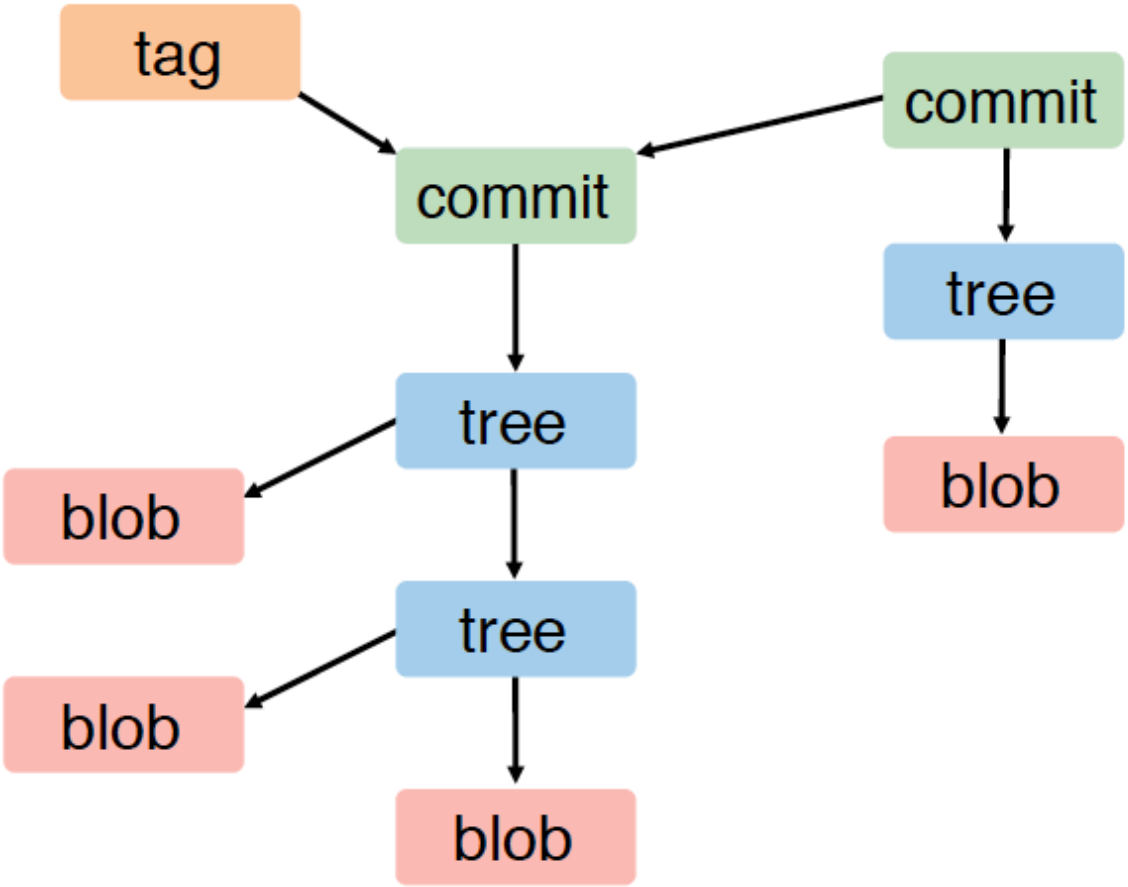
A “commit” is a tree of blobs (a set of changes)



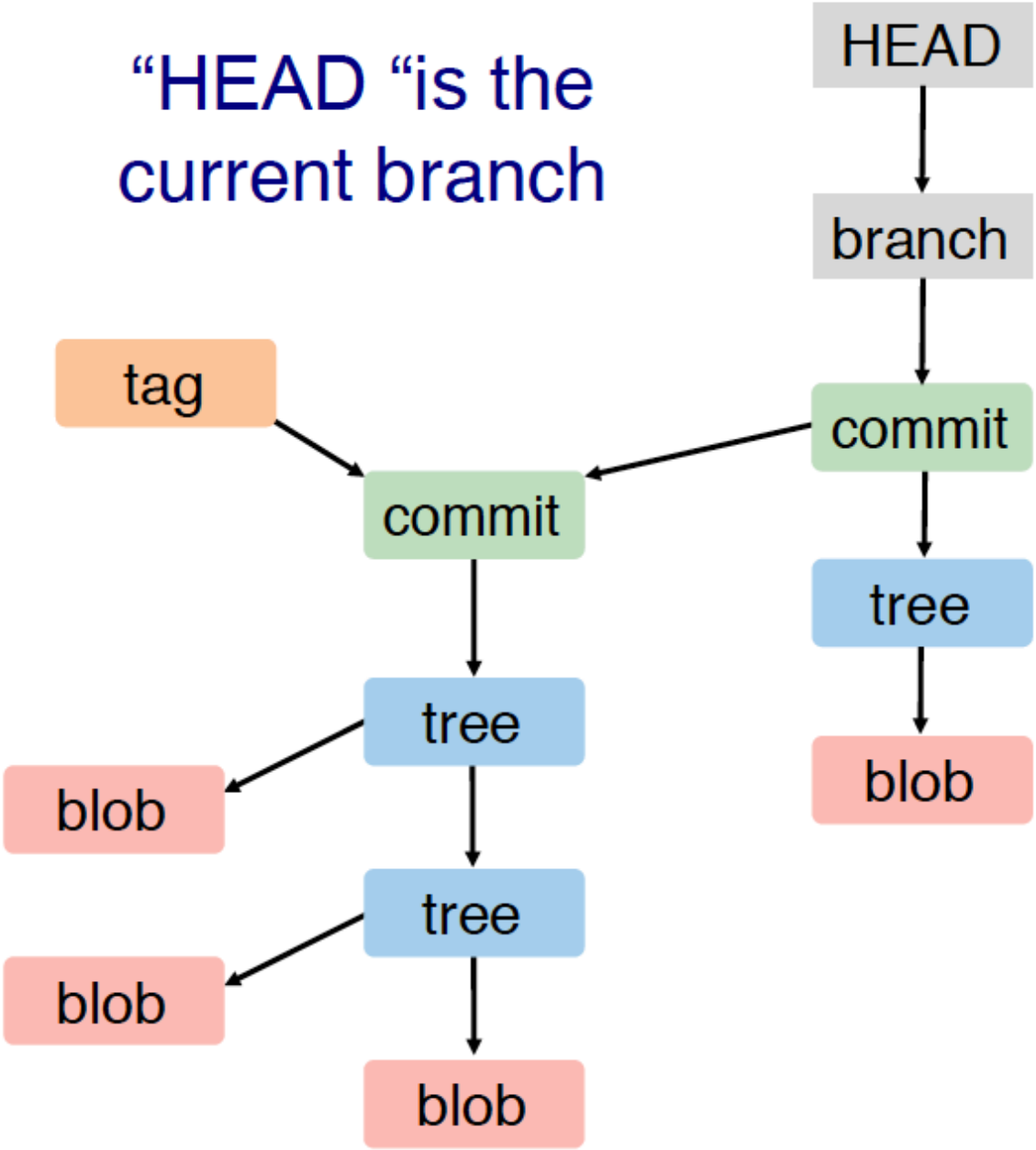
Most commits modify (or merge) earlier commits

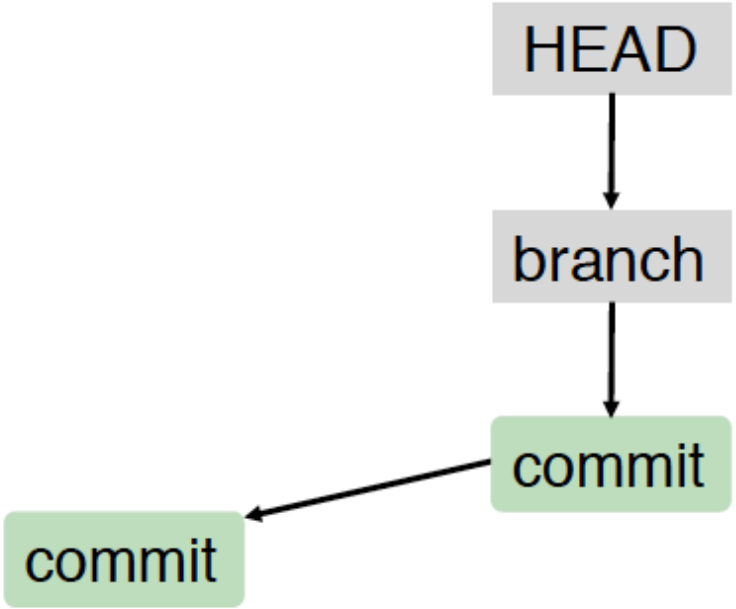


You can “tag” an interesting commit



“HEAD “is the current branch



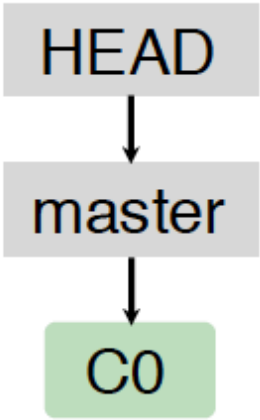


We will focus on commits only for one branch

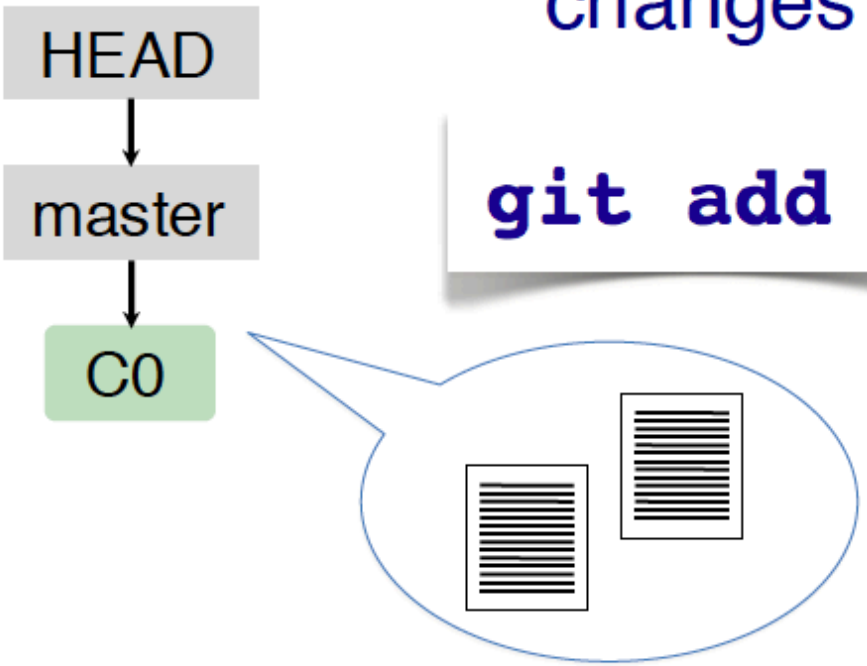
Git Basic Operations

Create a git repo

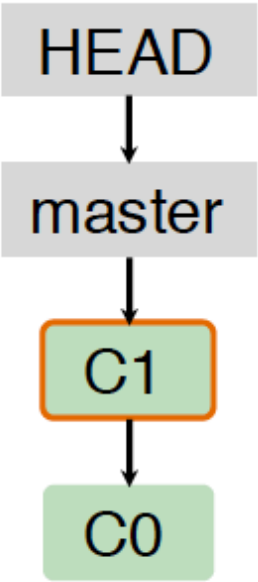
```
mkdir repo
cd repo
git init
```



Tell git to “stage” changes



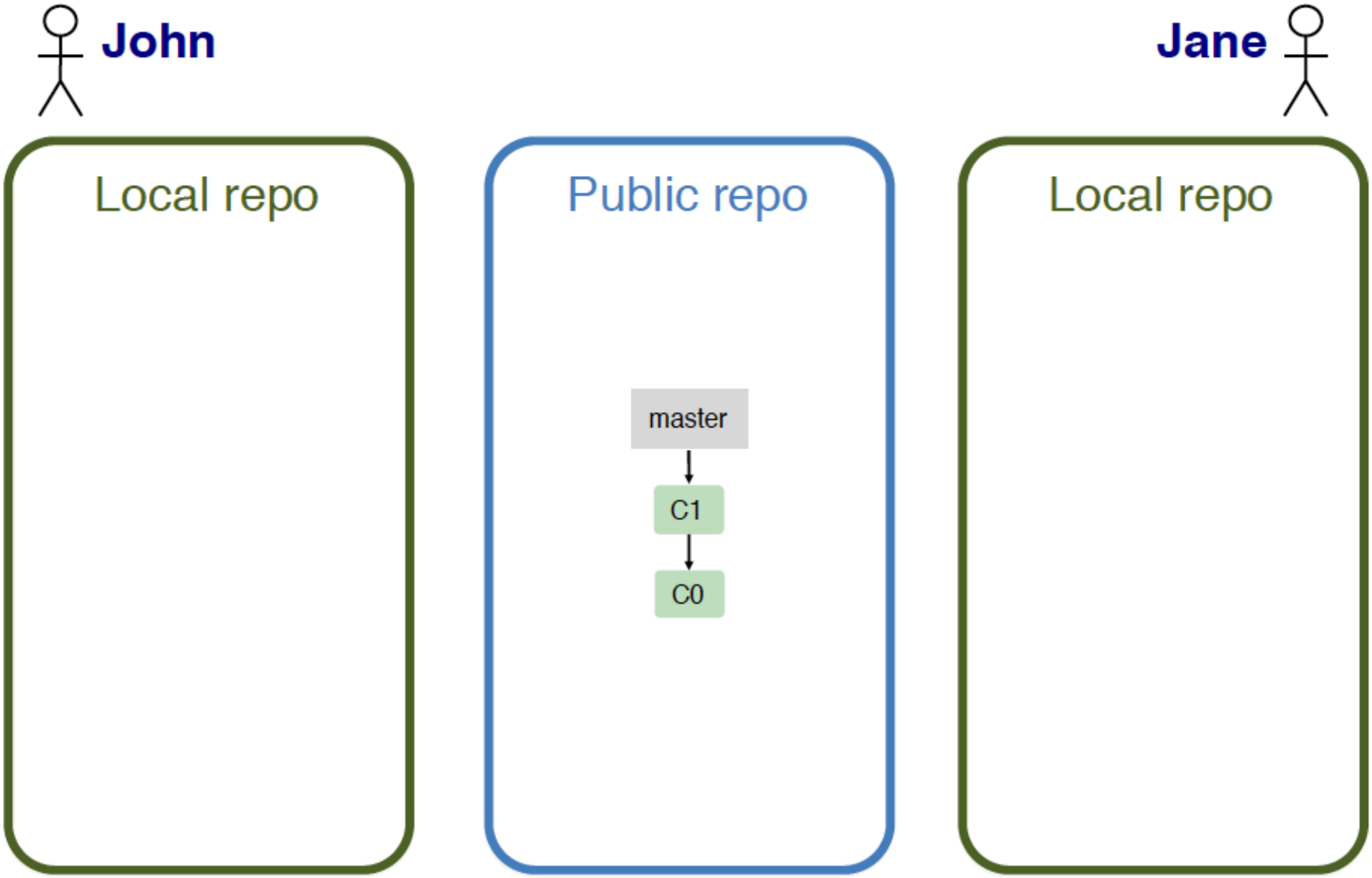

```
git add …
```

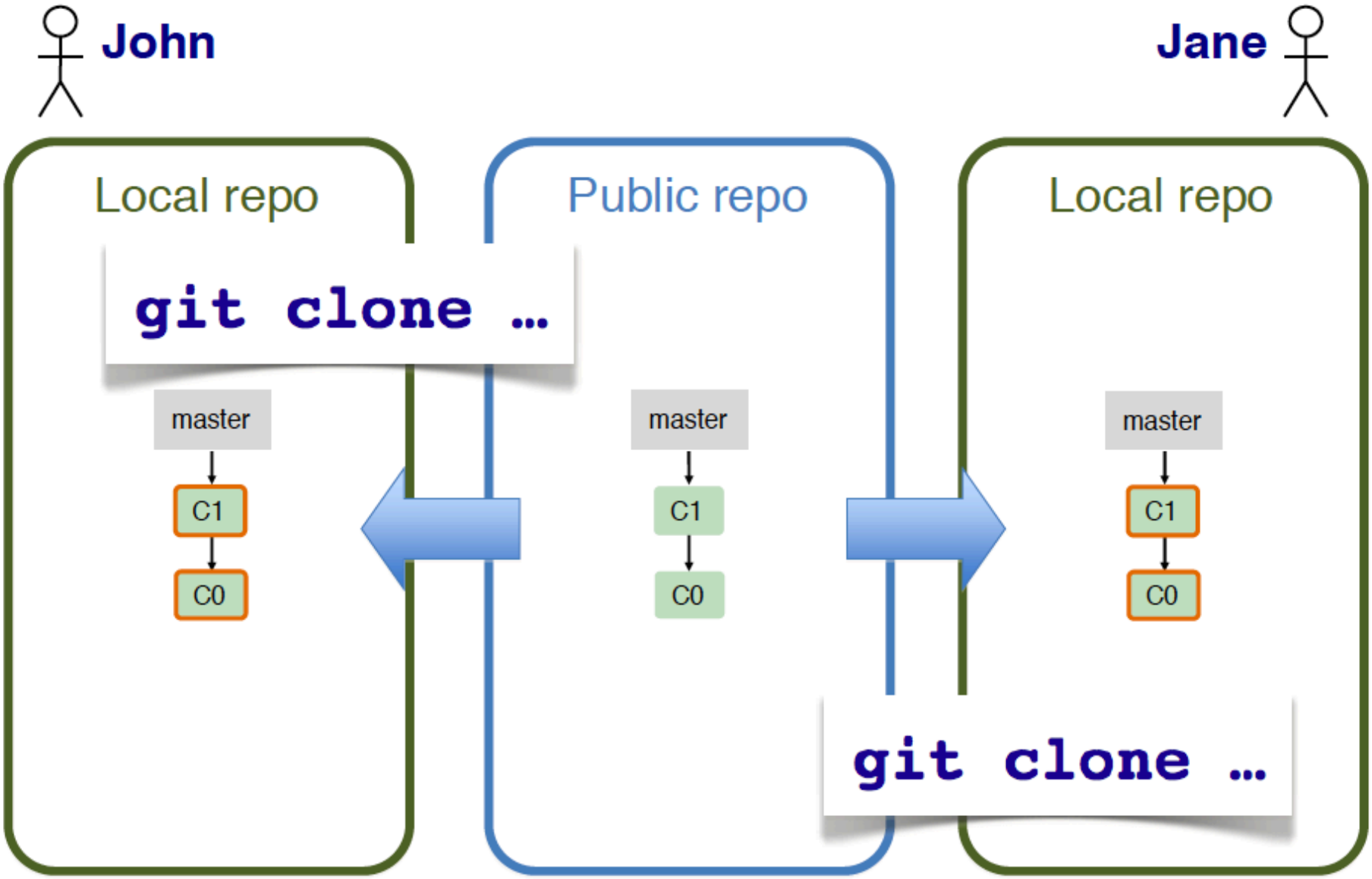


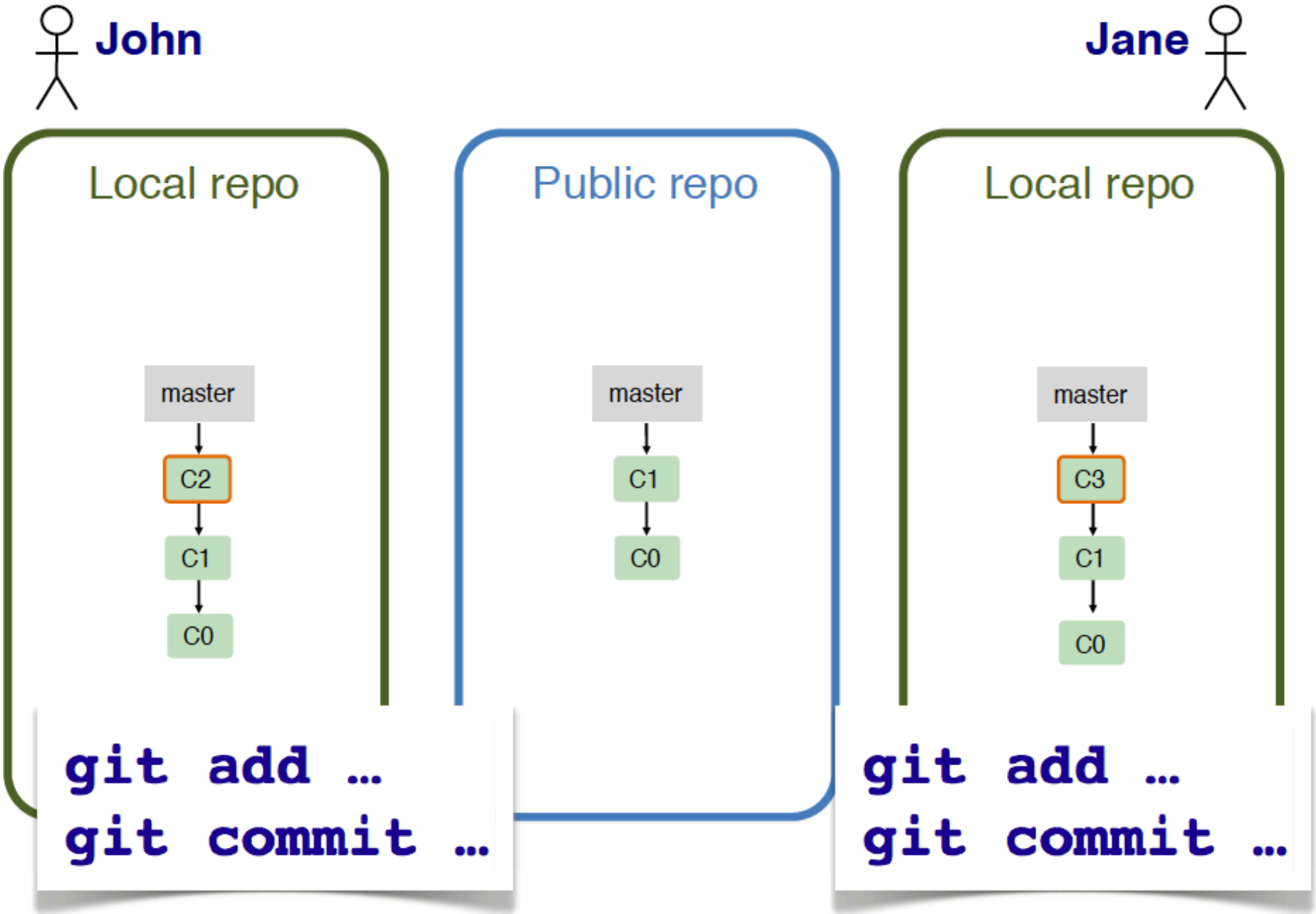
Commit your changes

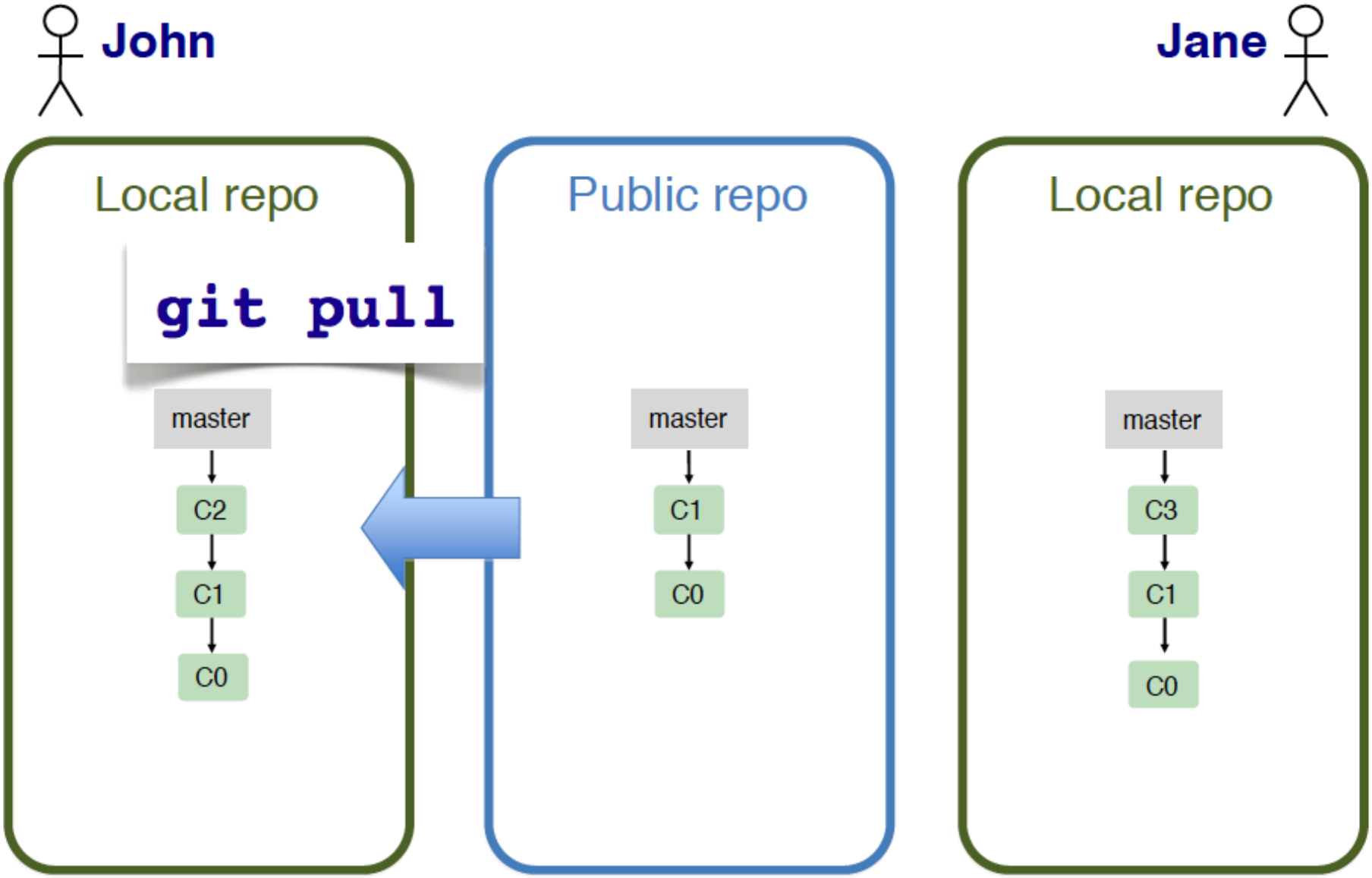
```
git commit …
```

Collaborating

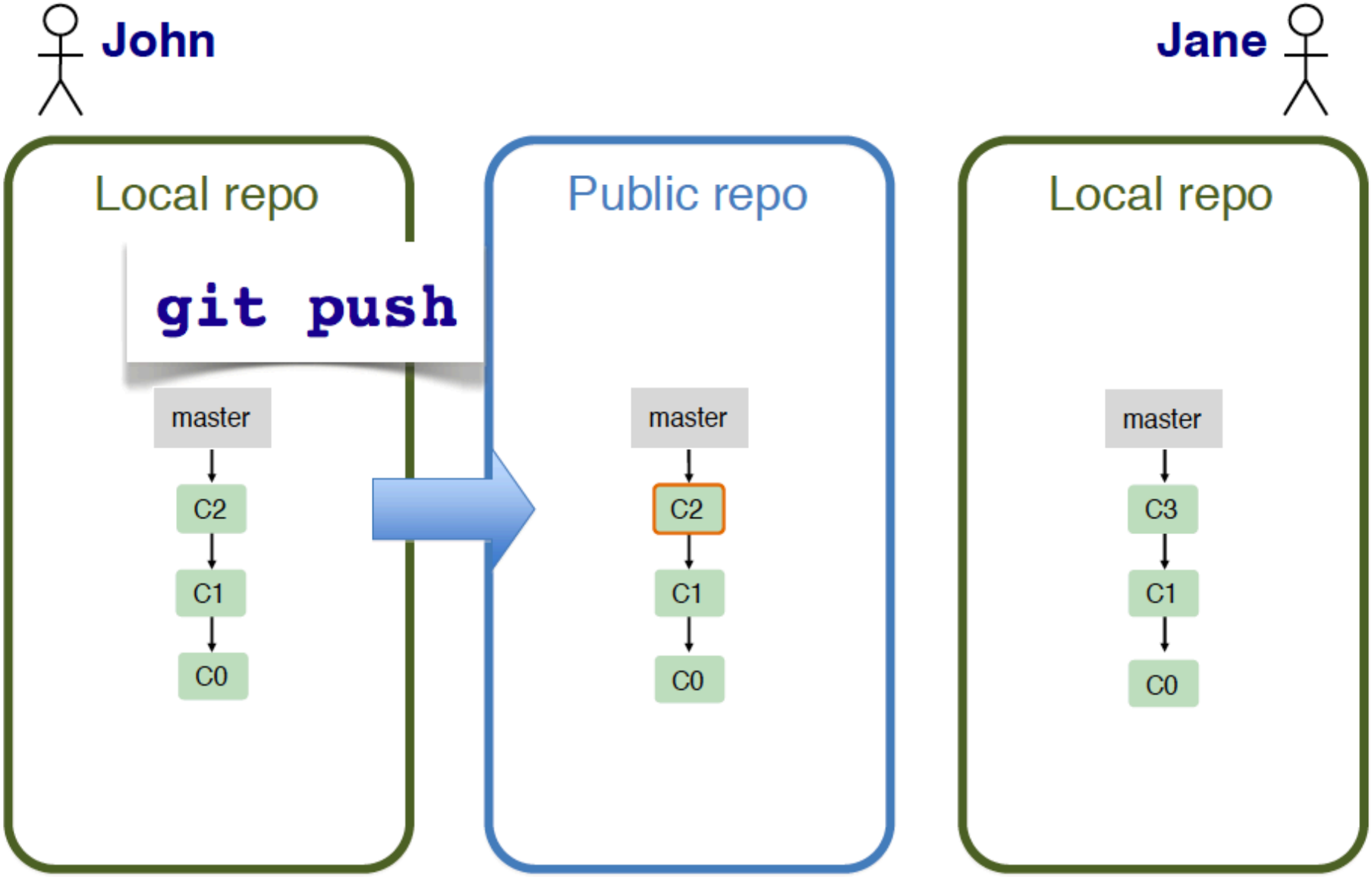

John
Jane
Local repo
Public repo
Local repo
master
C1
C0

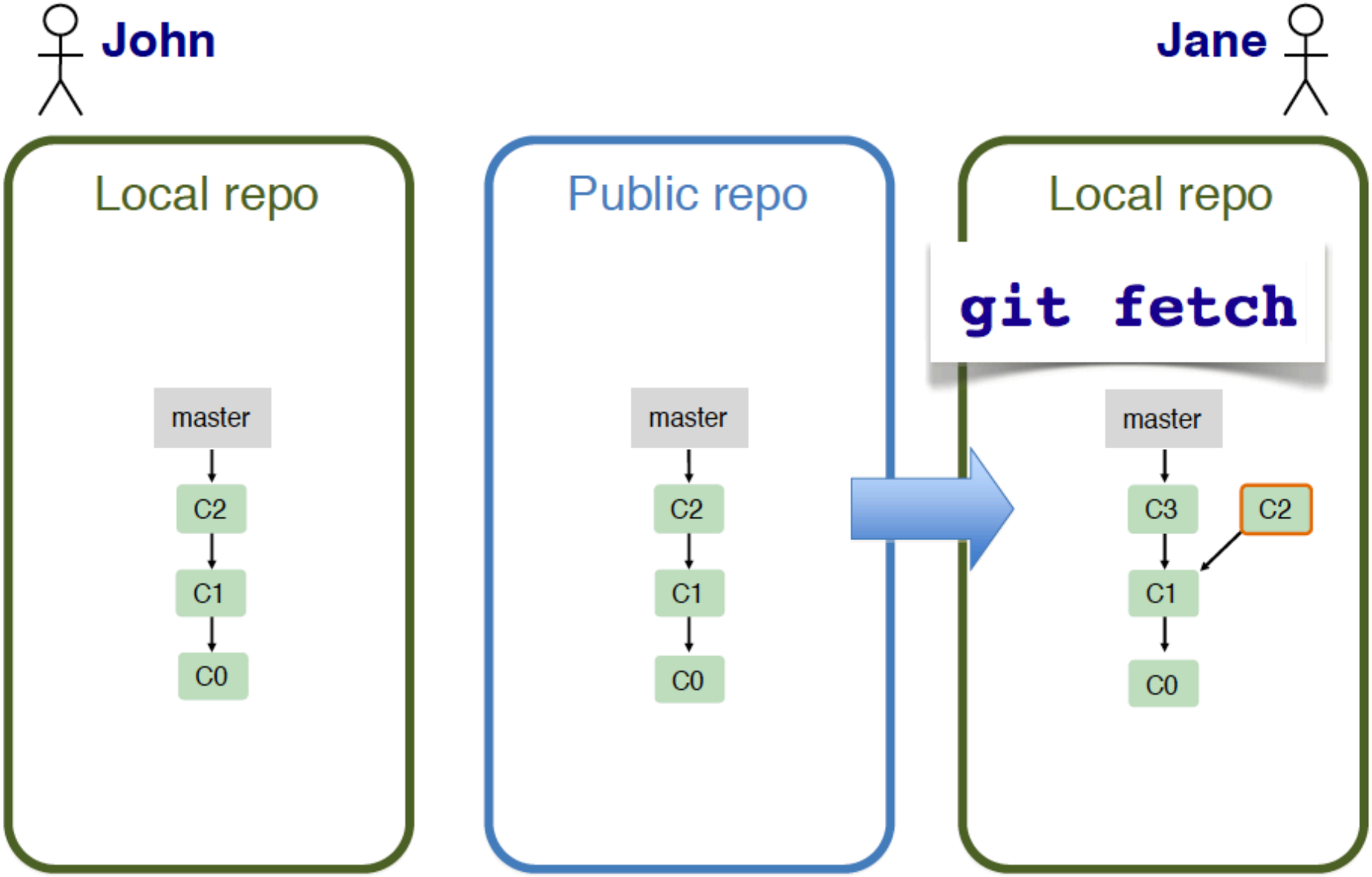

John
Jane
Local repo
Public repo
Local repo
git clone …
master
C1
C0
master
C1
C0
master
C1
C0
git clone …

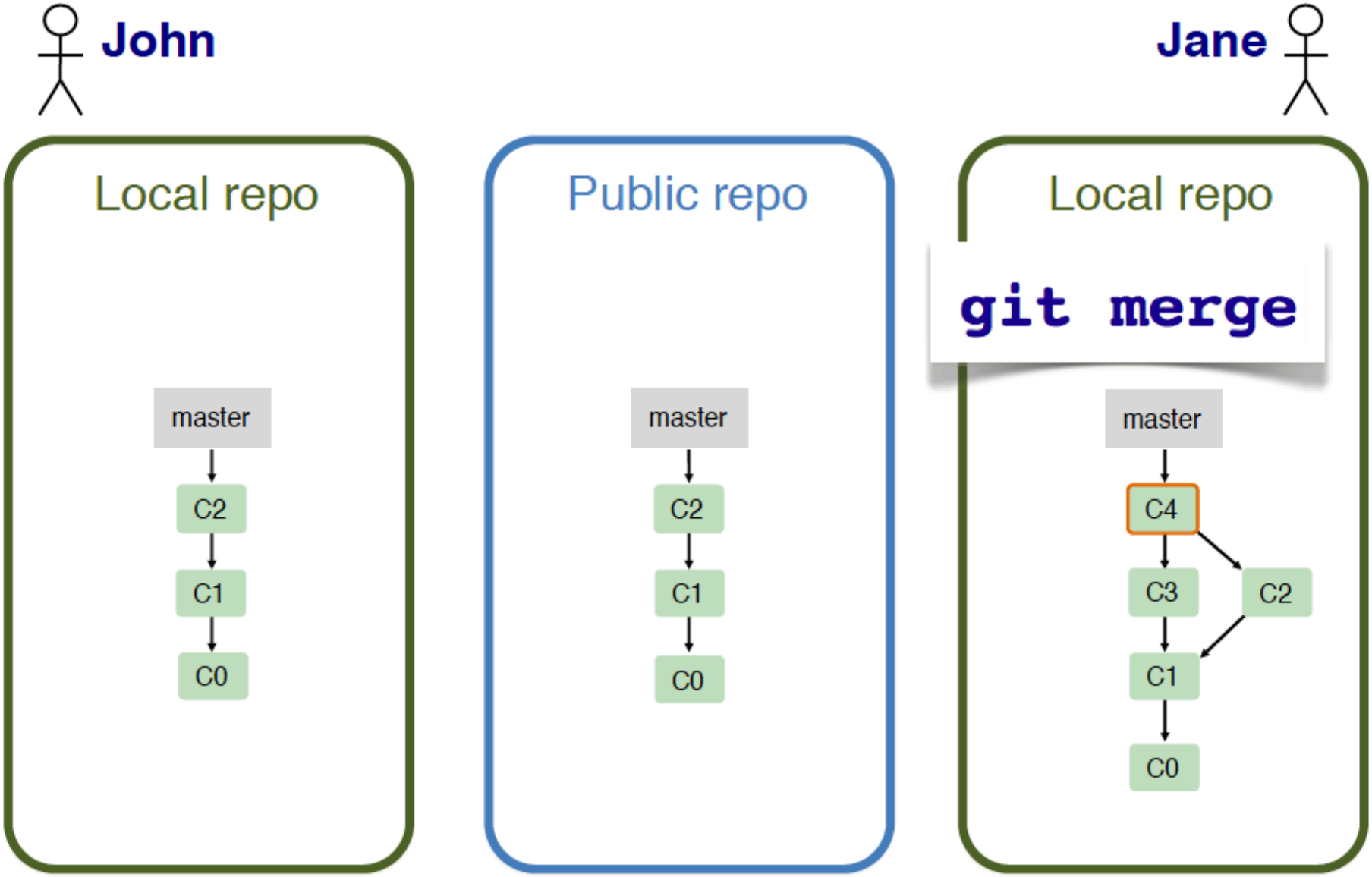

John
Jane
Local repo
Public repo
Local repo
master
C2
C1
C0
master
C1
C0
master
C3
C1
C0
git add …
git commit …
git add …
git commit …



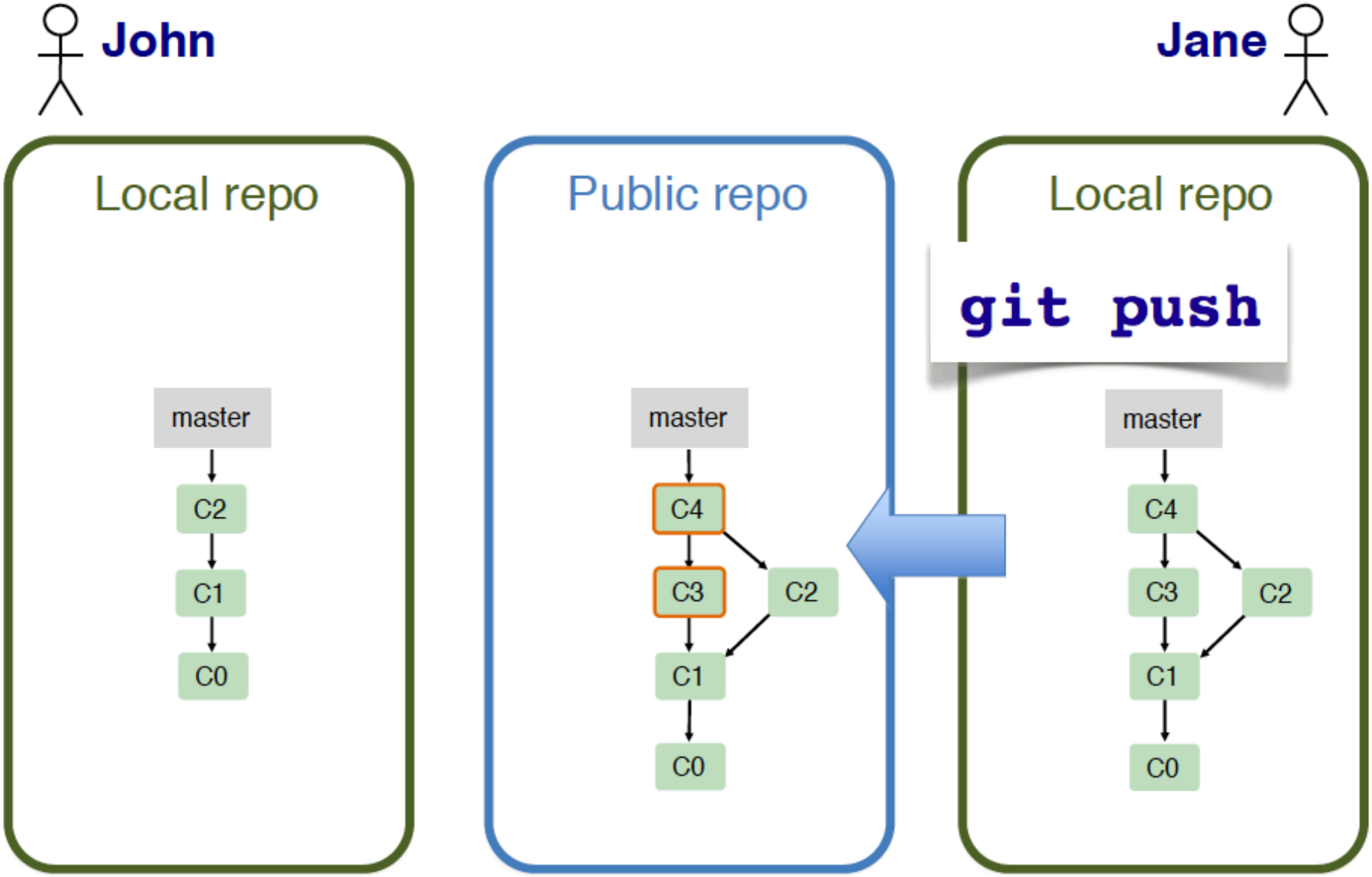
(nothing new to pull)

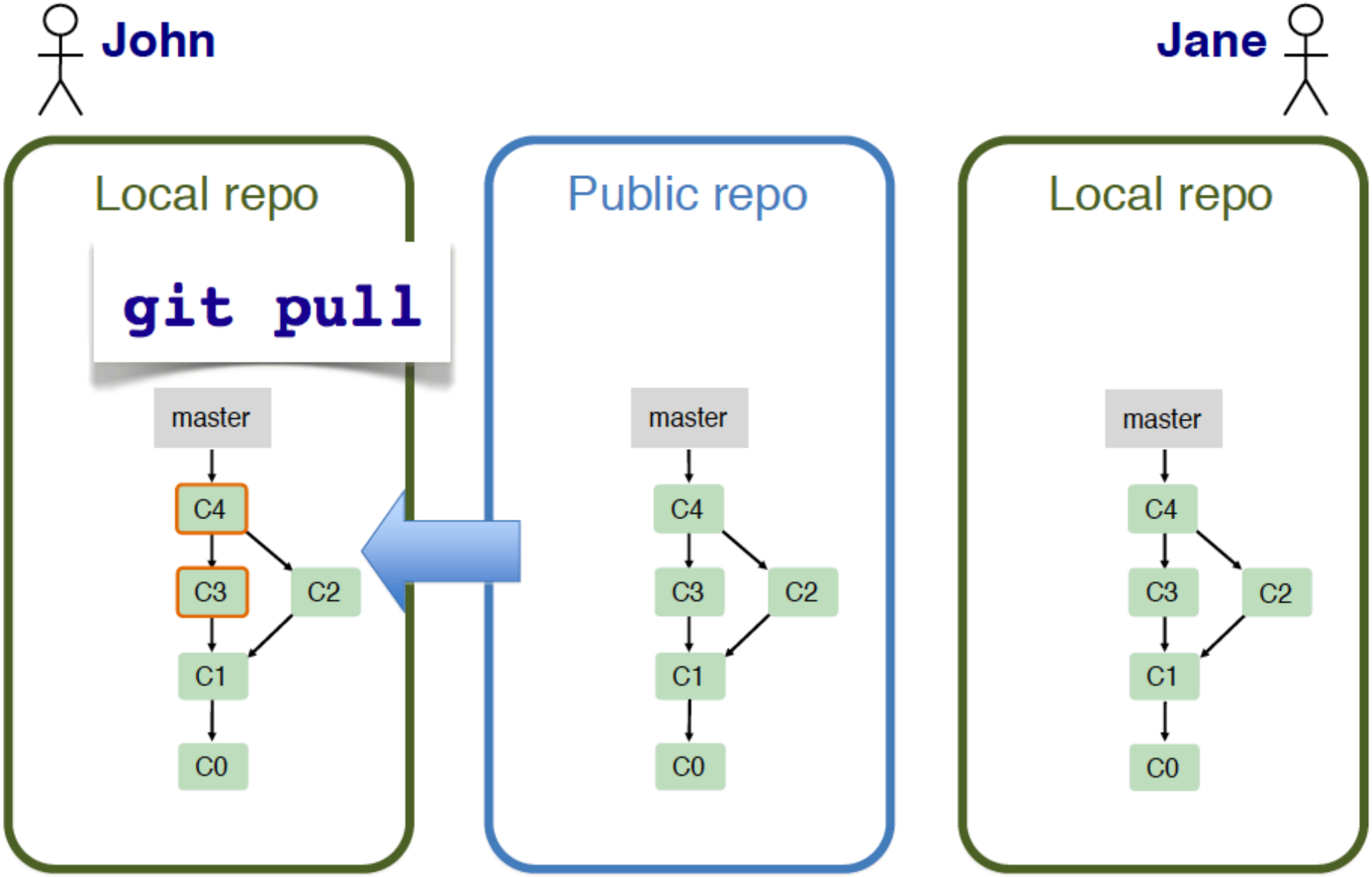

John
Jane
Local repo
git push
master
C2
C1
C0
Public repo
master
C2
C1
C0
Local repo
master
C3
C1
C0


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

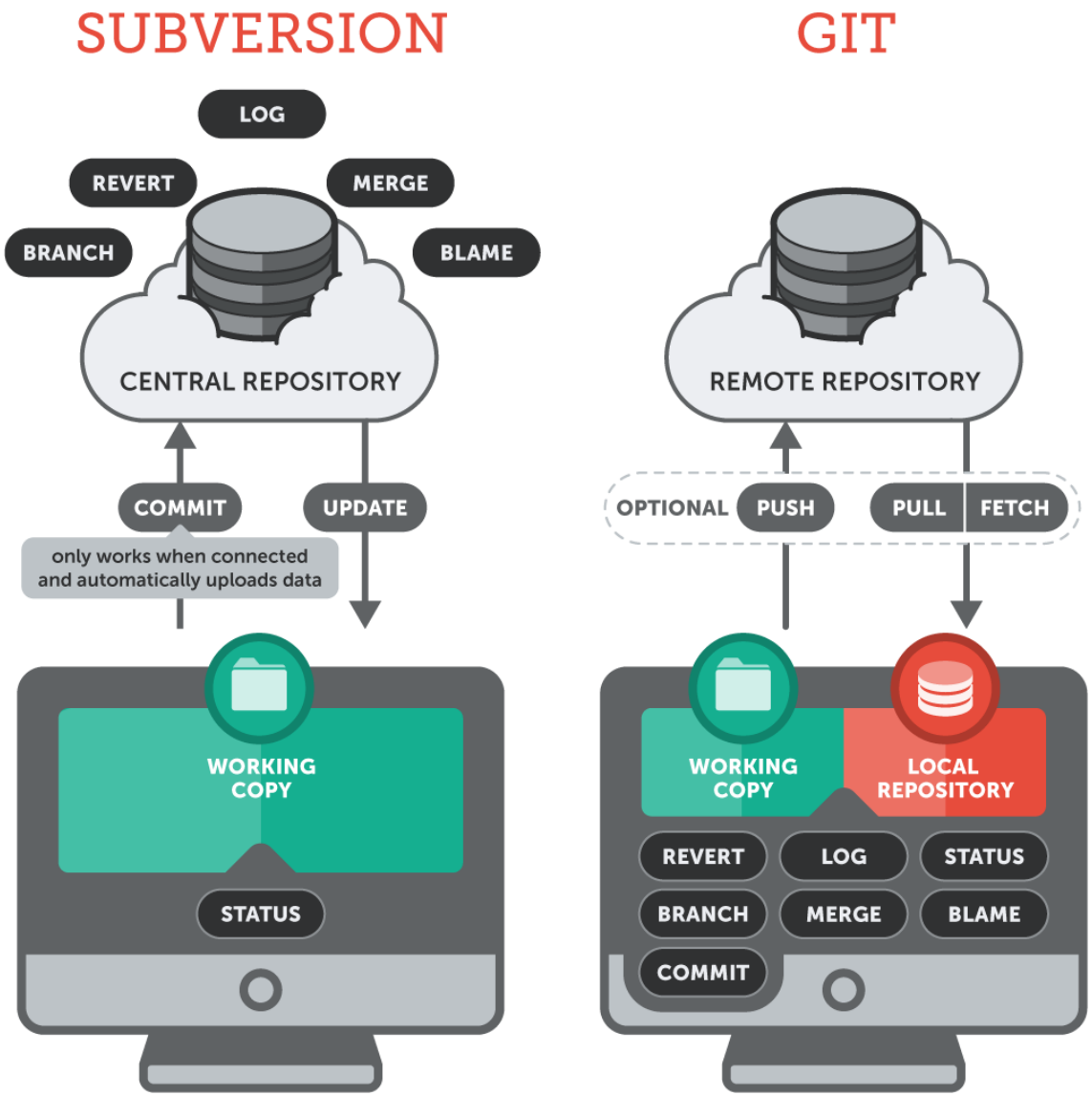


NB: **git pull** = fetch + merge


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

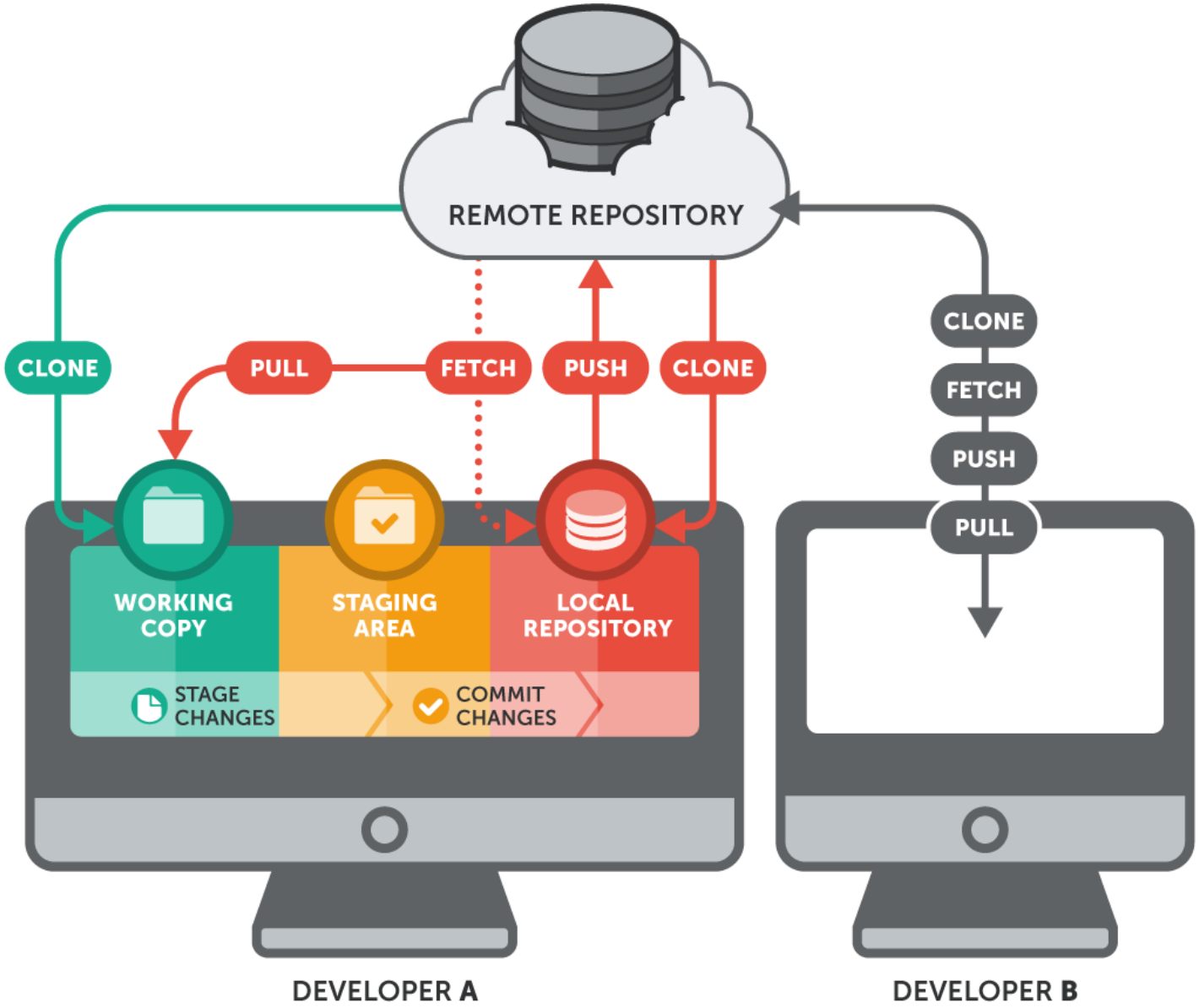

John
Jane
Local repo
git pull
master
C4
C3
C2
C1
C0
Public repo
master
C4
C3
C2
C1
C0
Local repo
master
C4
C3
C2
C1
C0

Wrap Up



https://www.git-tower.com/learn/git/ebook/en/desktop-gui/appendix/from-subversion-to-git

Wrap Up



https://www.git-tower.com/learn/git/ebook/en/command-line/remote-repositories/introduction

GitLab and GitHub

GitLab.com and GitHub.com are sites for online storage of Git repositories:

- You can create a remote repository and push code to it.
- You can get space for open source projects.

Question: Do I always have to use GitHub or GitLab to use git?

- Answer: No. You can still use git locally for your own purposes.
- Alternatively, you or someone else could set up a server to share files.

More to git …

- Merging and merge tool
- Squashing commits when merging
- Resolving conflicts
- User authentication with ssh
- Smartgit and other graphical interface for git commands
- git configure — remembering your name
- git remote — multiple remote repos
- gitlab — an open source public repo
- …

More to git …

Git Cheat Sheet

by Jan Krüger <jk@jk.gs>, http://jan-krueger.net/git/
Based on work by Zack Rusin

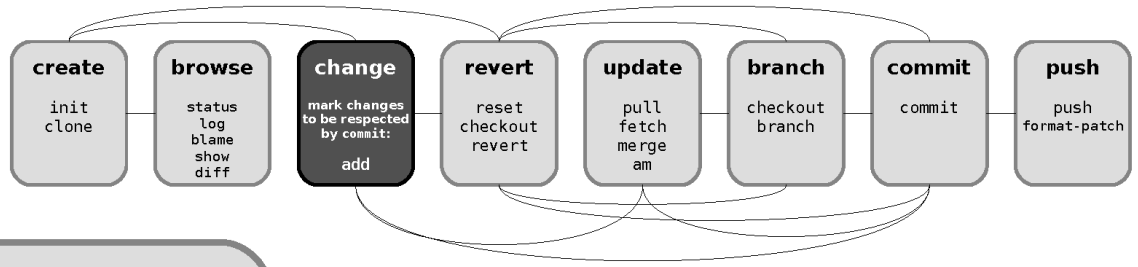
Basics

Use git help [*command*] if you're stuck.

`master`	default devel branch
`origin`	default upstream branch
`HEAD`	current branch
`HEAD^`	parent of HEAD
`HEAD~4`	great-great grandparent of HEAD
`foo..bar`	from branch *foo* to branch *bar*

Command Flow

(left to right)



Create

From existing files

```
git init
git add .
```

From existing repository

```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

Revert

In Git, `revert` usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
```
(**reset** to last commit)

```
git revert branch
git commit -a --amend
```
(**replaces** prev. commit)

```
git checkout id file
```

Publish

In Git, `commit` only respects changes that have been marked explicitly with add.

```
git commit [-a]
```
(-a: add changed files automatically)

```
git format-patch origin
```
(create set of diffs)

```
git push remote
```
(push to origin or *remote*)

```
git tag foo
```
(mark current version)

Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

Branch

```
git checkout branch
```
(switch working dir to *branch*)

```
git merge branch
```
(merge into current)

```
git branch branch
```
(branch current)

```
git checkout -b new other
```
(branch *new* from *other* and switch to it)

Useful Tools

```
git archive
```
Create release tarball

```
git bisect
```
Binary search for defects

```
git cherry-pick
```
Take single commit from elsewhere

```
git fsck
```
Check tree

```
git gc
```
Compress metadata (performance)

```
git rebase
```
Forward-port local changes to remote branch

```
git remote add URL
```
Register a new remote repository for this tree

```
git stash
```
Temporarily set aside changes

```
git tag
```
(there's more to it)

```
gitk
```
Tk GUI for Git

Conflicts

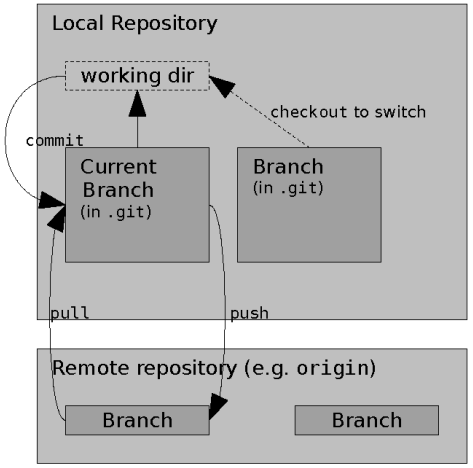
Use add to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```

Tracking Files

```
git add files
git mv old new
git rm files
git rm --cached files
```
(stop tracking but keep files in working dir)

Structure Overview



Resources

http://git-scm.com/

git community book

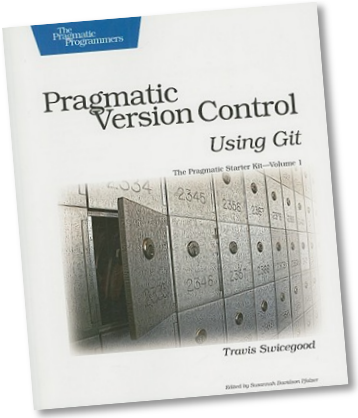
http://book.git-scm.com/index.html

Getting Git

Scott Chacon

GitLab

https://gitlab.com/



http://www.slideshare.net/chacon/getting-git

http://oreilly.com/