

# Version Control Systems (aka VCSs)

Athanasis Zolotas

# Outline

- Introduction
  - Why VCS?
- History
  - Which VCS?
- Let's try Git
  - 3 parts

# Introduction

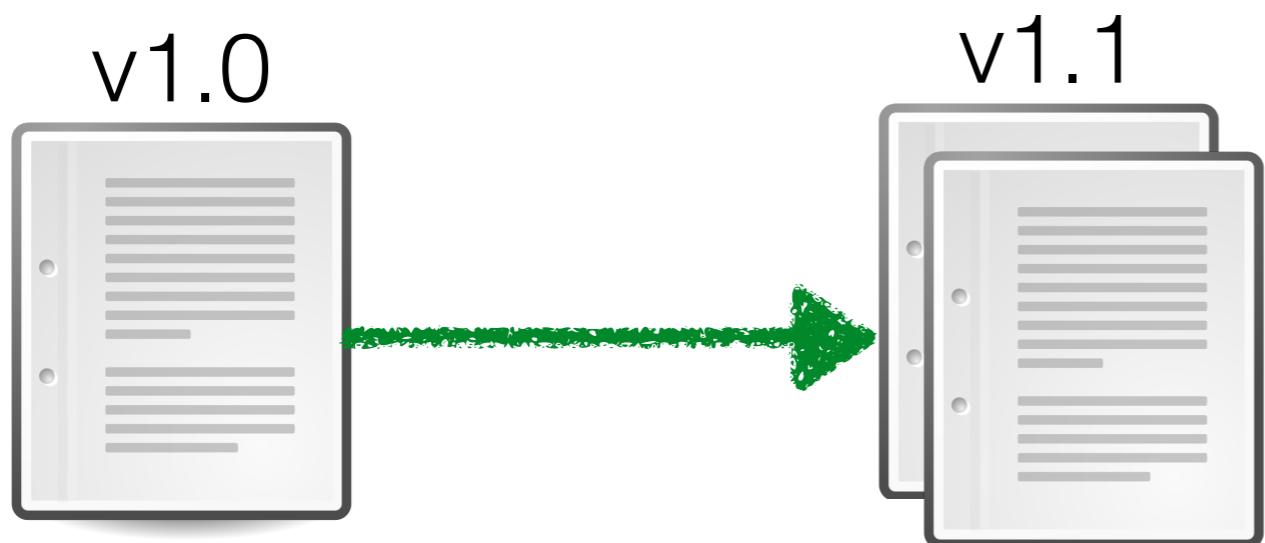
- As the name suggests its a technology to control versions of an artefact
  - Most probably code
  - It's quite common for text project

# Introduction

v1.0



# Introduction



# Introduction



# Introduction



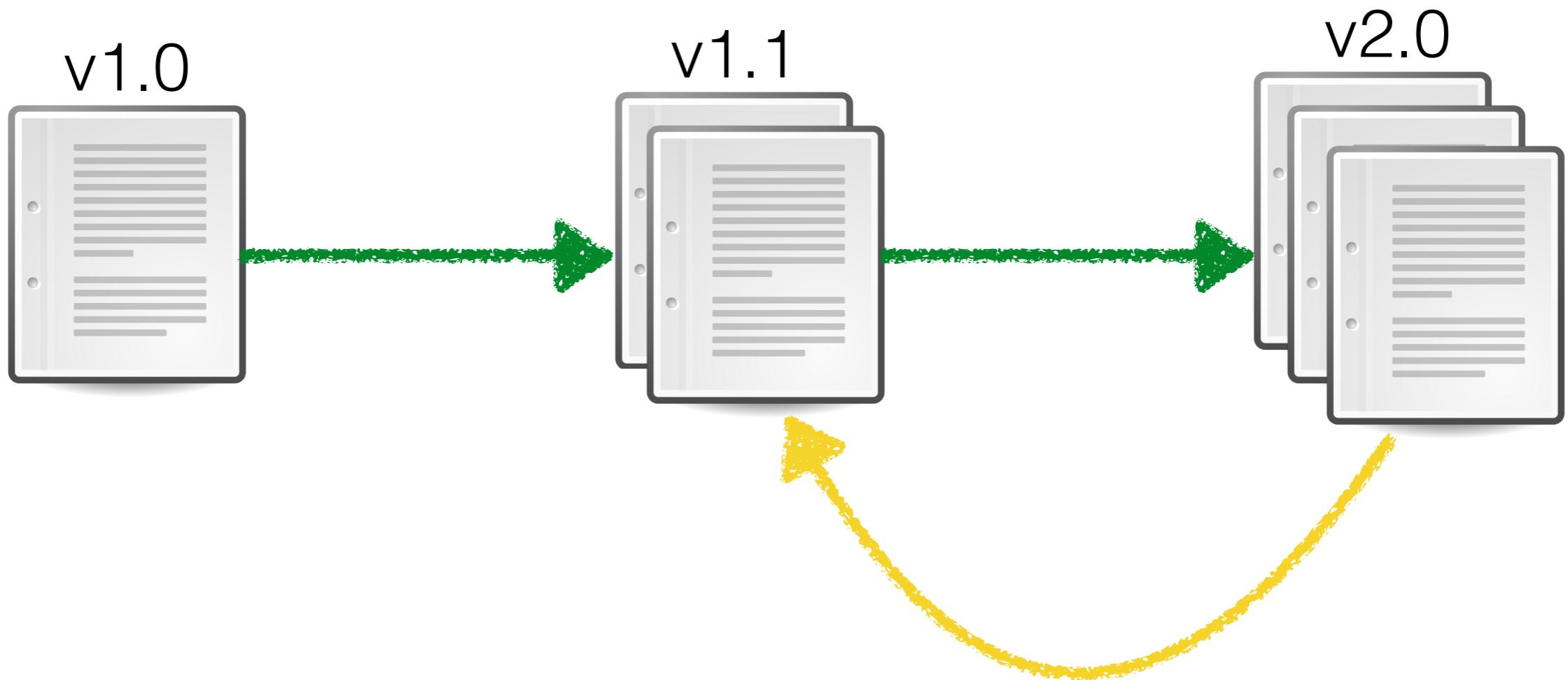
v1

0

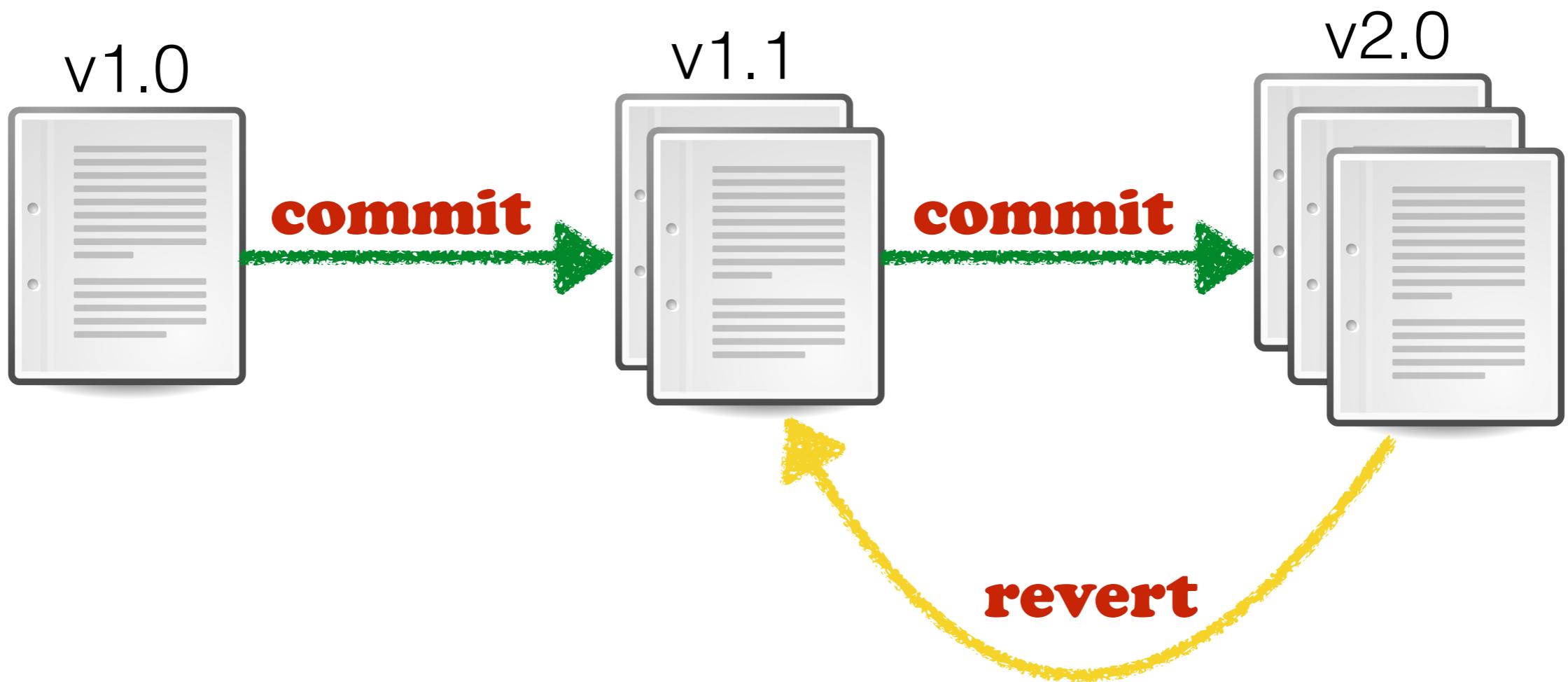
# Introduction



# Introduction



# Introduction



# Why VCS?

- Control / Store different versions, historical data of your application / documents
- Bug fixes (find at which step the error happened)
- But VCSs have more uses:
  - Enable collaboration
  - Work as a backup

# History

# History

- Filenames:
  - project\_191015.docx → project\_211015.docx
  - project\_v1.zip → project\_v2.zip

# History

- Filenames:
  - project\_191015.docx → project\_211015.docx
  - project\_v1.zip → project\_v2.zip
- Diff utilities
  - Find changes between two files

# History

- Filenames:
  - project\_191015.docx → project\_211015.docx
  - project\_v1.zip → project\_v2.zip
- Diff utilities
  - Find changes between two files
- Post-its or cups

# History

# History

- Source Code Control System (SCCS) - 1972

# History

- Source Code Control System (SCCS) - 1972
- Revision Control System (RCS) - 1981

# History

- Source Code Control System (SCCS) - 1972
- Revision Control System (RCS) - 1981
- IBM Rational ClearCase - 1990

# History

- Source Code Control System (SCCS) - 1972
- Revision Control System (RCS) - 1981
- IBM Rational ClearCase - 1990
- Source Safe - 1991

# History

- Source Code Control System (SCCS) - 1972
- Revision Control System (RCS) - 1981
- IBM Rational ClearCase - 1990
- Source Safe - 1991
- Microsoft Delta - 1994

# History

- Source Code Control System (SCCS) - 1972
- Revision Control System (RCS) - 1981
- IBM Rational ClearCase - 1990
- Source Safe - 1991
- Microsoft Delta - 1994
- ...

# Modern VCSSs

# Modern VCSs

- Apache Subversion

# Modern VCSs

- Apache Subversion
- Darcs

# Modern VCSs

- Apache Subversion
- Darcs
- Git

# Modern VCSs

- Apache Subversion
- Darcs
- Git
- Mercurial

# Modern VCSs

- Apache Subversion
- Darcs
- Git
- Mercurial
- Bazaar

# Modern VCSs

- Apache Subversion
- Darcs
- Git
- Mercurial
- Bazaar
- ...

# Modern VCSs

- **Apache Subversion - SVN**
- Darcs
- **Git**
- Mercurial
- Bazaar
- ...

# Approaches

- Centralised
- Decentralised (or Distributed)

# Terminology

# Terminology

**Repository:** Where everything is stored (files & historic data).

# Terminology

**Repository:** Where everything is stored (files & historic data).

**Commit [v]:** Store a change to the repository.

# Terminology

**Repository:** Where everything is stored (files & historic data).

**Commit [v]:** Store a change to the repository.

**Commit [n]:** A revision in the repository. A version.

# Terminology

**Repository:** Where everything is stored (files & historic data).

**Commit [v]:** Store a change to the repository.

**Commit [n]:** A revision in the repository. A version.

**Branch :** The project is forked. New paths that share the same code base.

# Terminology

**Repository:** Where everything is stored (files & historic data).

**Commit [v]:** Store a change to the repository.

**Commit [n]:** A revision in the repository. A version.

**Branch :** The project is forked. New paths that share the same code base.

**Merge :** Combine two paths.

# Terminology

**Repository:** Where everything is stored (files & historic data).

**Commit [v]:** Store a change to the repository.

**Commit [n]:** A revision in the repository. A version.

**Branch :** The project is forked. New paths that share the same code base.

**Merge :** Combine two paths.

**Conflict :** Changes performed at the same files cannot be resolved automatically.

# Approaches

# Approaches

- Centralised:
  - Unique, central repository
  - Changes are committed to this central copy of the project

# Approaches

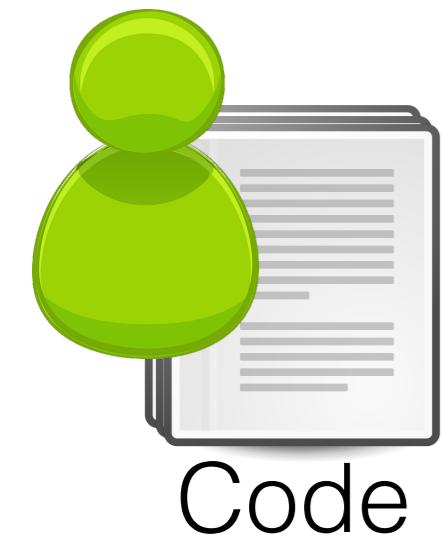
- Centralised:
  - Unique, central repository
  - Changes are committed to this central copy of the project
- Decentralised / Distributed:
  - Different copies, each copy is equally valid
  - Commits are happening to your local repository

# SVN

- Apache Subversion
  - Is a centralised VCS
  - When a new version is committed, its id increments sequentially (e.g. 1, 2, 3, ...)

# SVN Example

# SVN Example



Repository 0



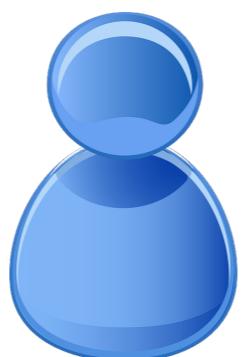
# SVN Example



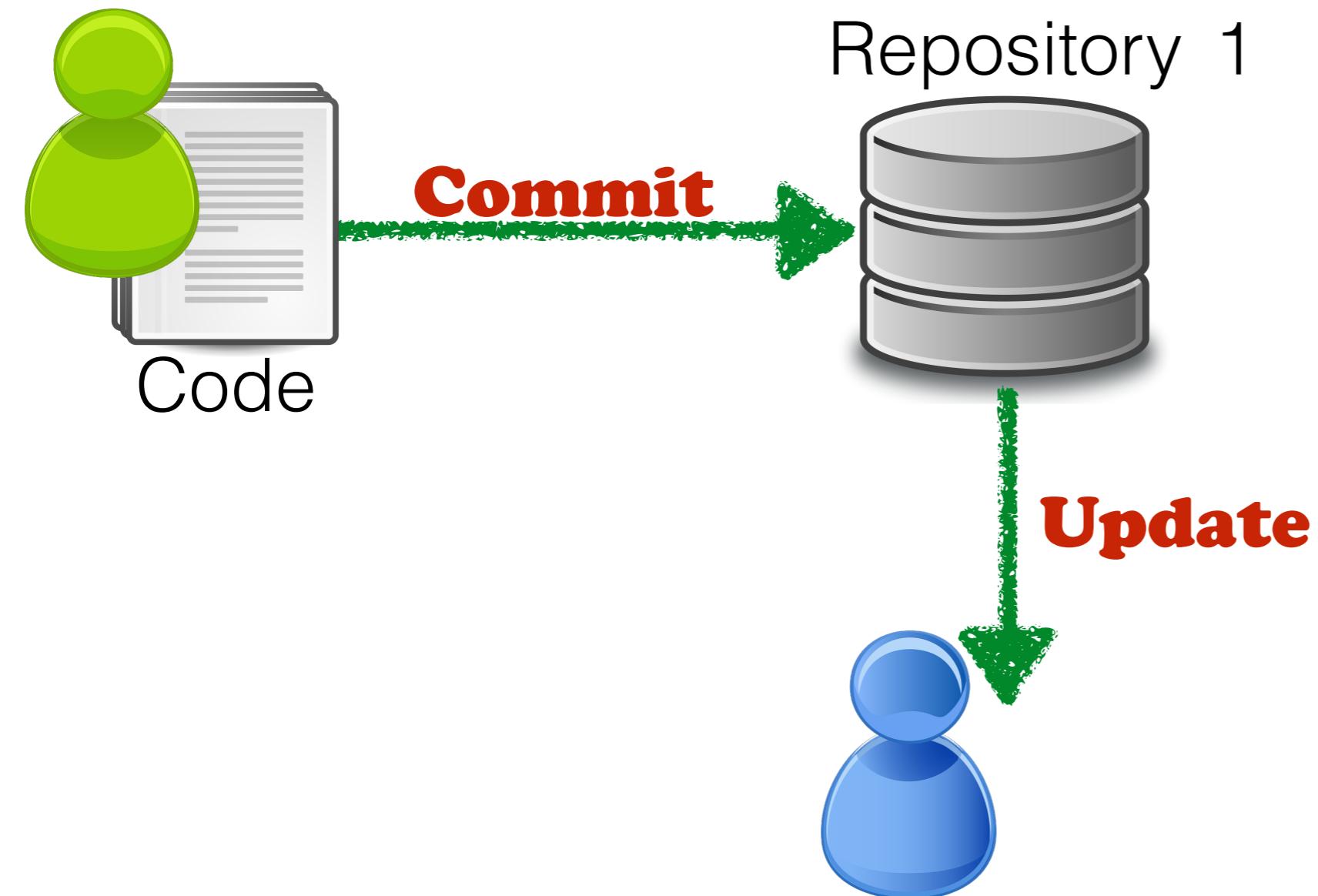
# SVN Example



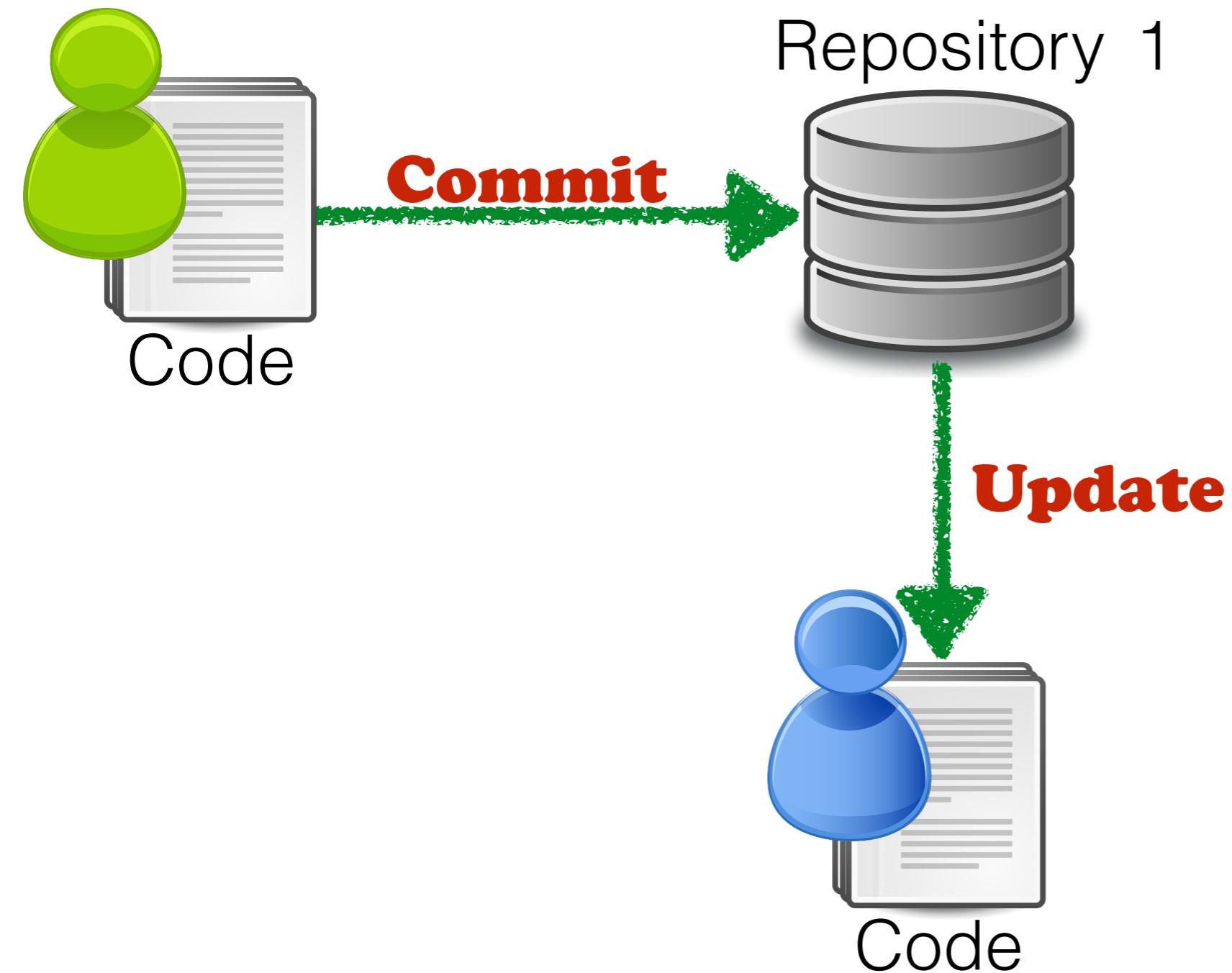
# SVN Example



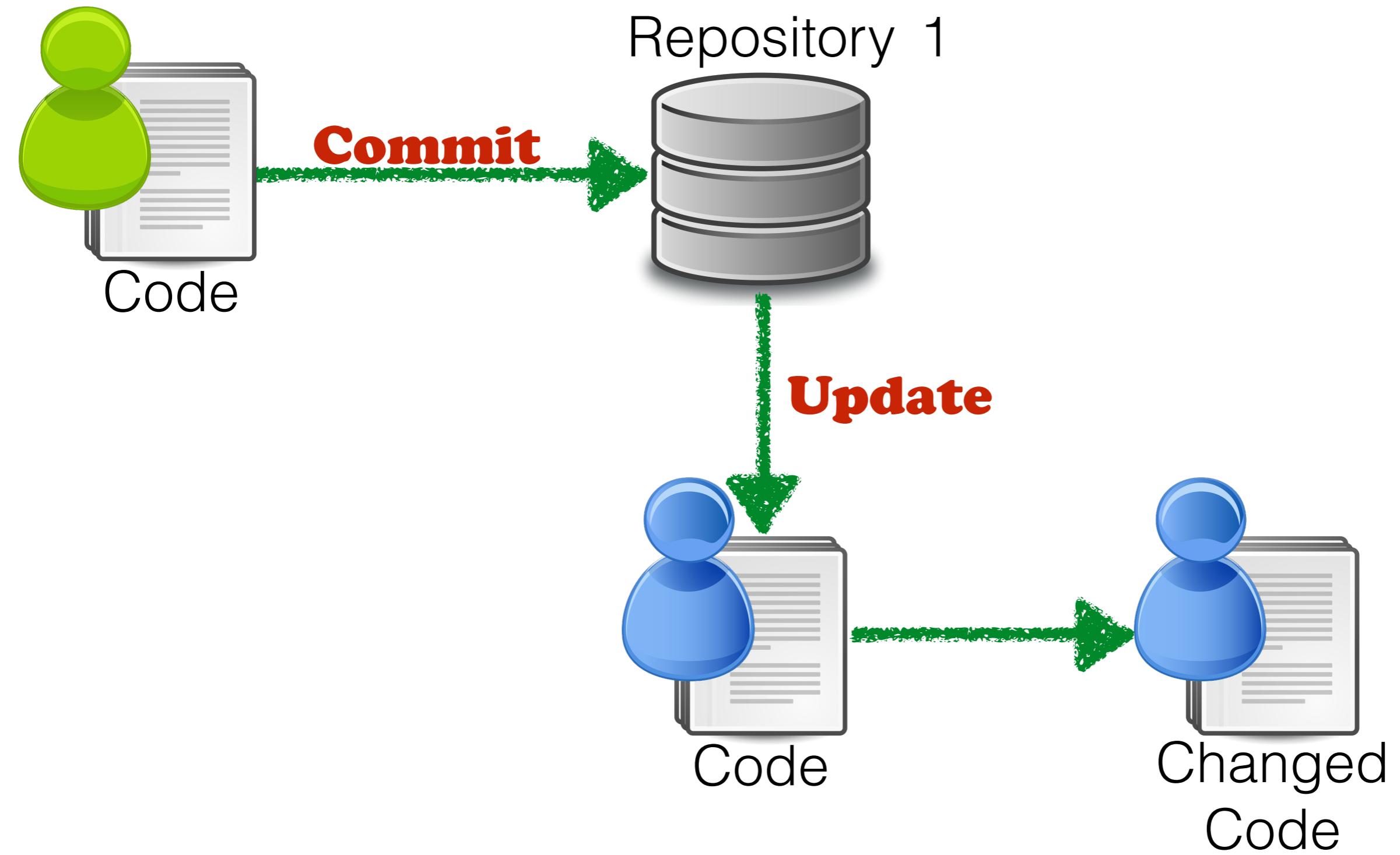
# SVN Example



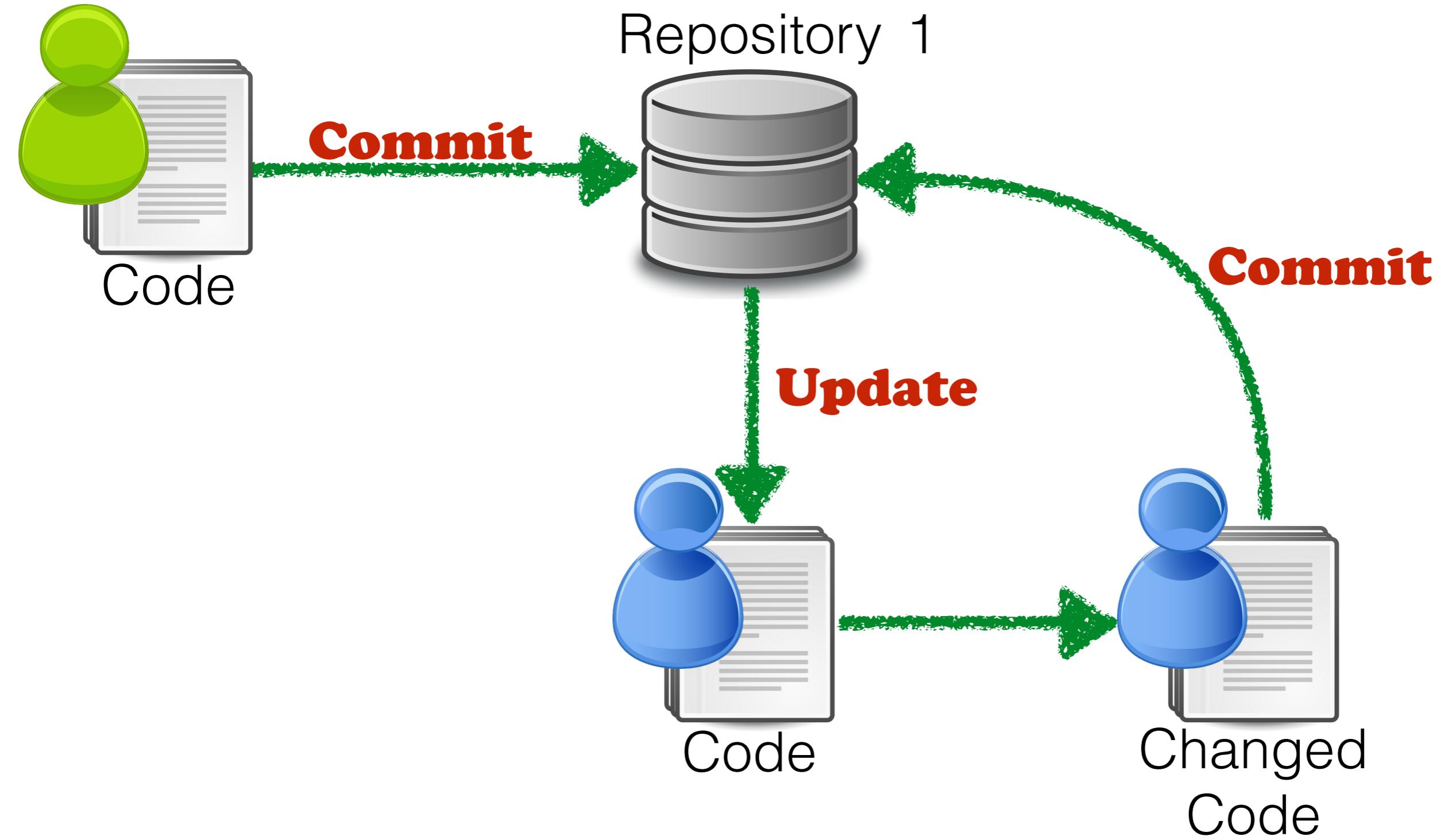
# SVN Example



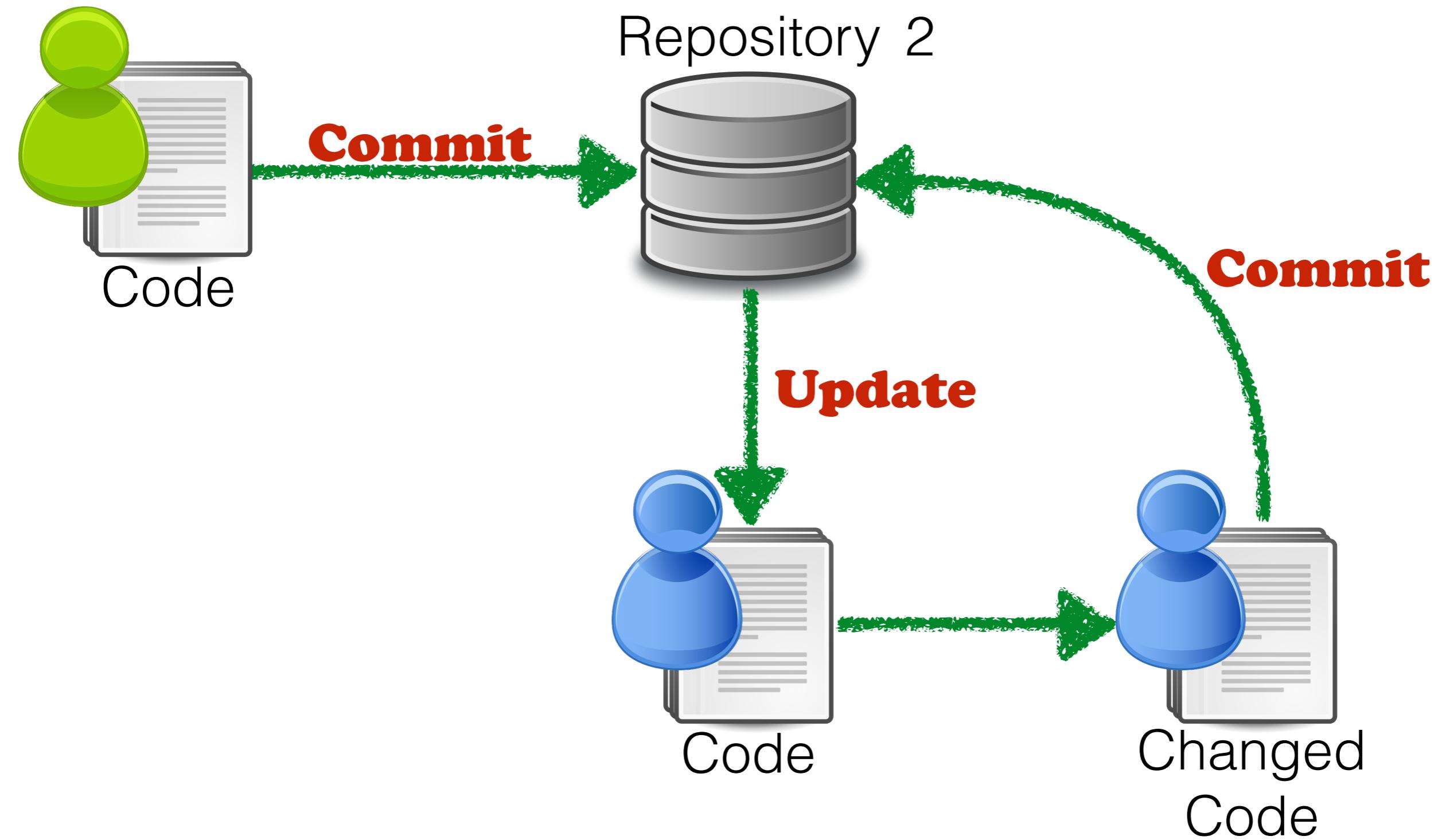
# SVN Example



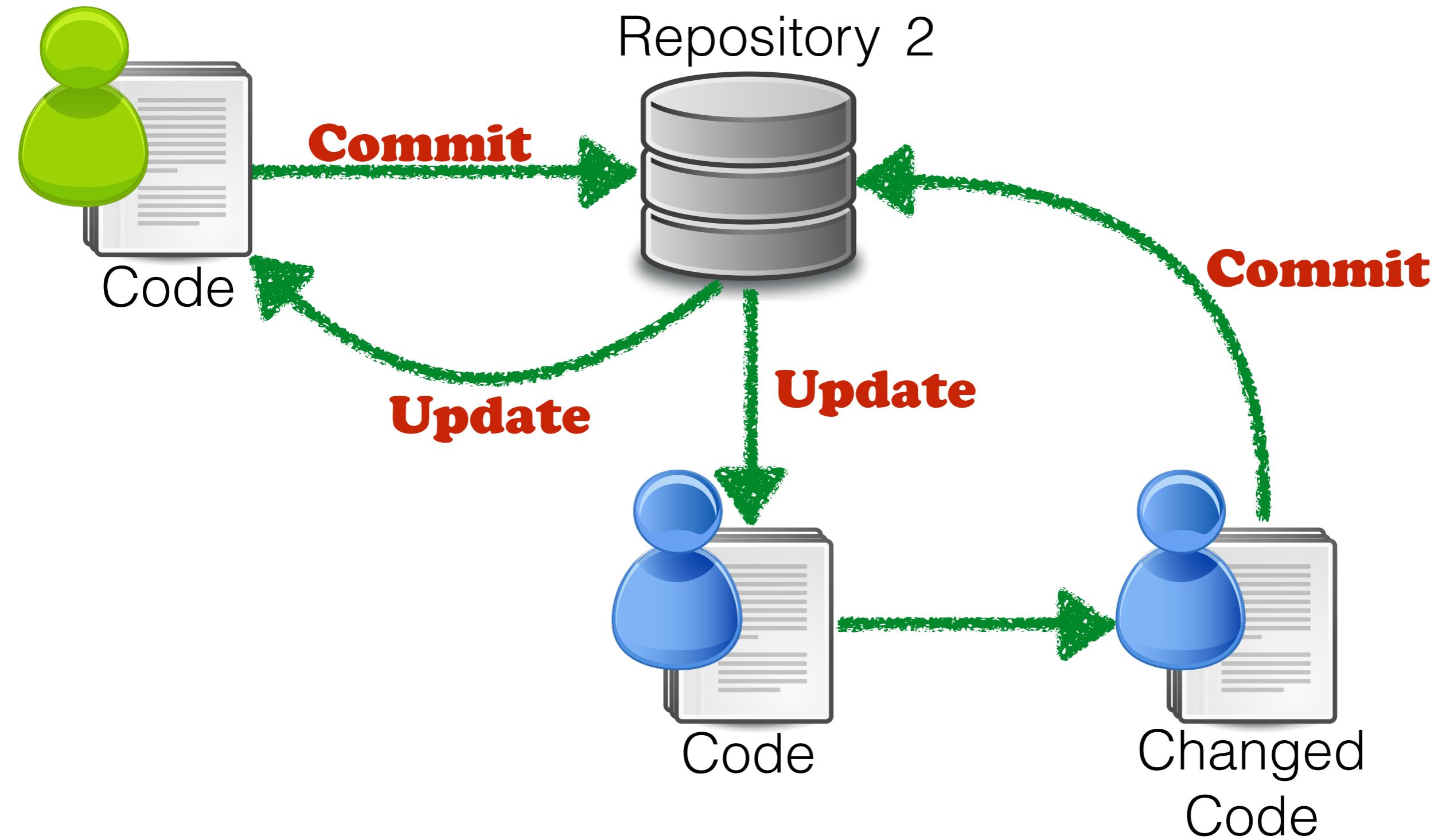
# SVN Example



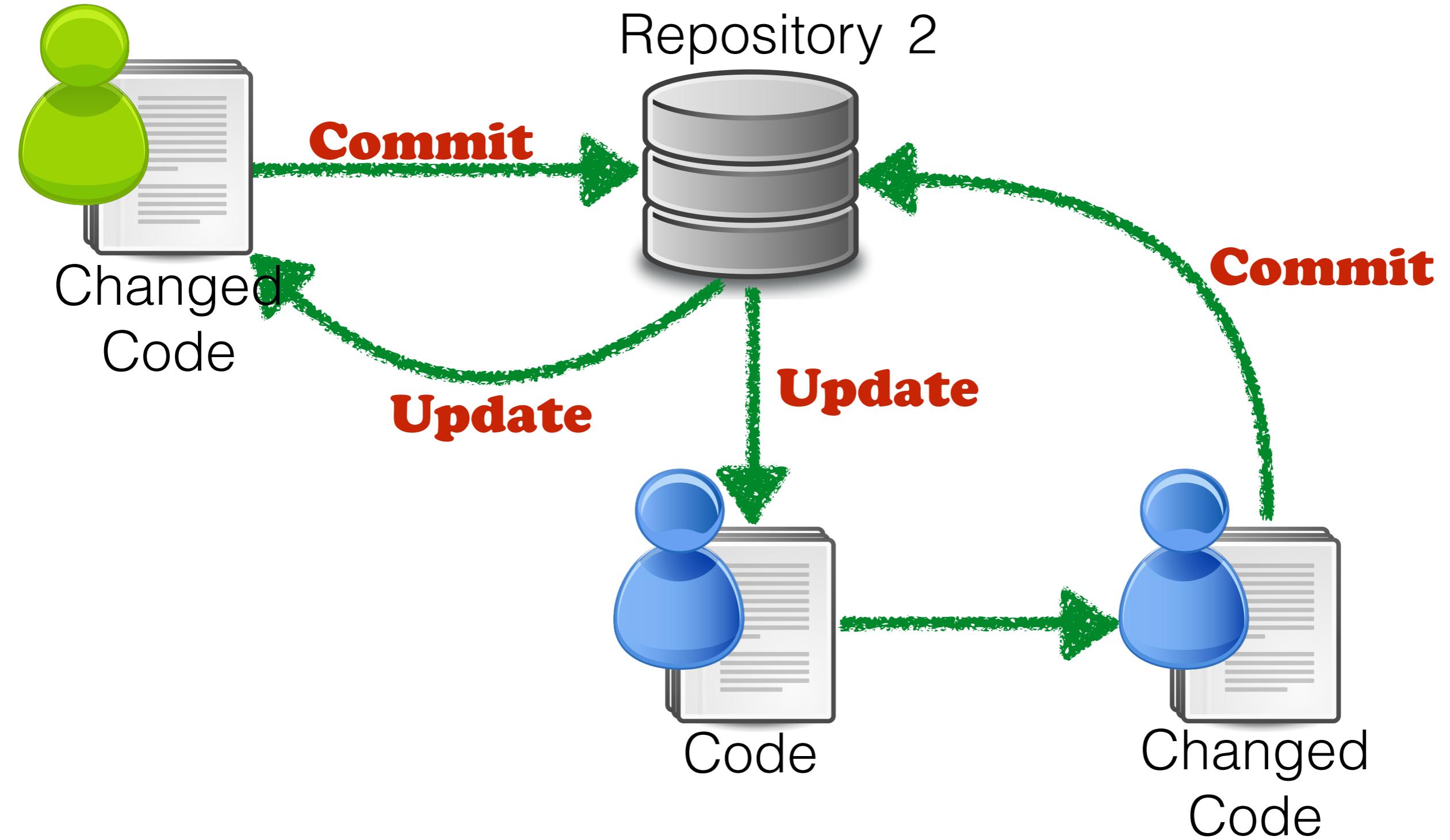
# SVN Example



# SVN Example



# SVN Example



# SVN Software

- Windows:
  - **TortoiseSVN** - free
- Mac OS:
  - Versions - \$
  - **SmartSVN** - free/\$
- Linux:
  - RapidSVN - free
- Or Eclipse:
  - **Subversive** - free
  - Subclipse - free

# SVN Pros & Cons

# SVN Pros & Cons

- ✓ Works as you expect

# SVN Pros & Cons

- ✓ Works as you expect
- ✓ Only one copy so everyone knows on what code the changes are made

# SVN Pros & Cons

- ✓ Works as you expect
- ✓ Only one copy so everyone knows on what code the changes are made
- ✓ Widely used - Plenty of learning resources

# SVN Pros & Cons

- ✓ Works as you expect
- ✓ Only one copy so everyone knows on what code the changes are made
- ✓ Widely used - Plenty of learning resources
- A single copy stored in a server. If server fails you're not able to do versioning

# SVN Pros & Cons

- ✓ Works as you expect
- ✓ Only one copy so everyone knows on what code the changes are made
- ✓ Widely used - Plenty of learning resources
- A single copy stored in a server. If server fails you're not able to do versioning
- Encourages large commits

# SVN Pros & Cons

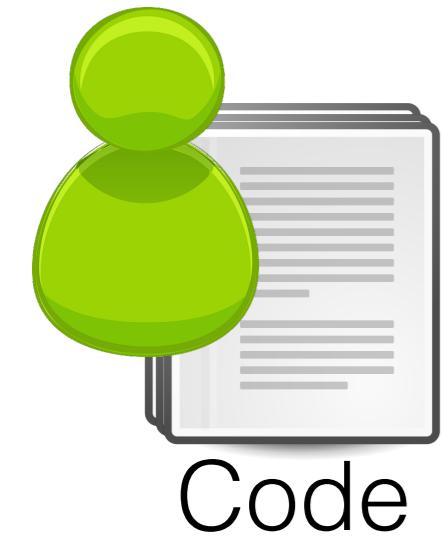
- ✓ Works as you expect
- ✓ Only one copy so everyone knows on what code the changes are made
- ✓ Widely used - Plenty of learning resources
- A single copy stored in a server. If server fails you're not able to do versioning
- Encourages large commits
- Discourages branching

# Git

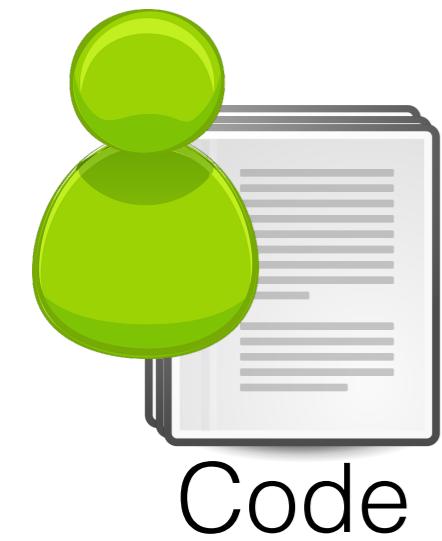
- Is a decentralised (distributed) VCS
- 40 hexadecimal numbers for commit versions (e.g. f3abe64fc121b75f3f0566c73f2f1a4e8ffd68e)
  - Constructed based on author, commit message, previous version, ...

# Git Example

# Git Example



# Git Example



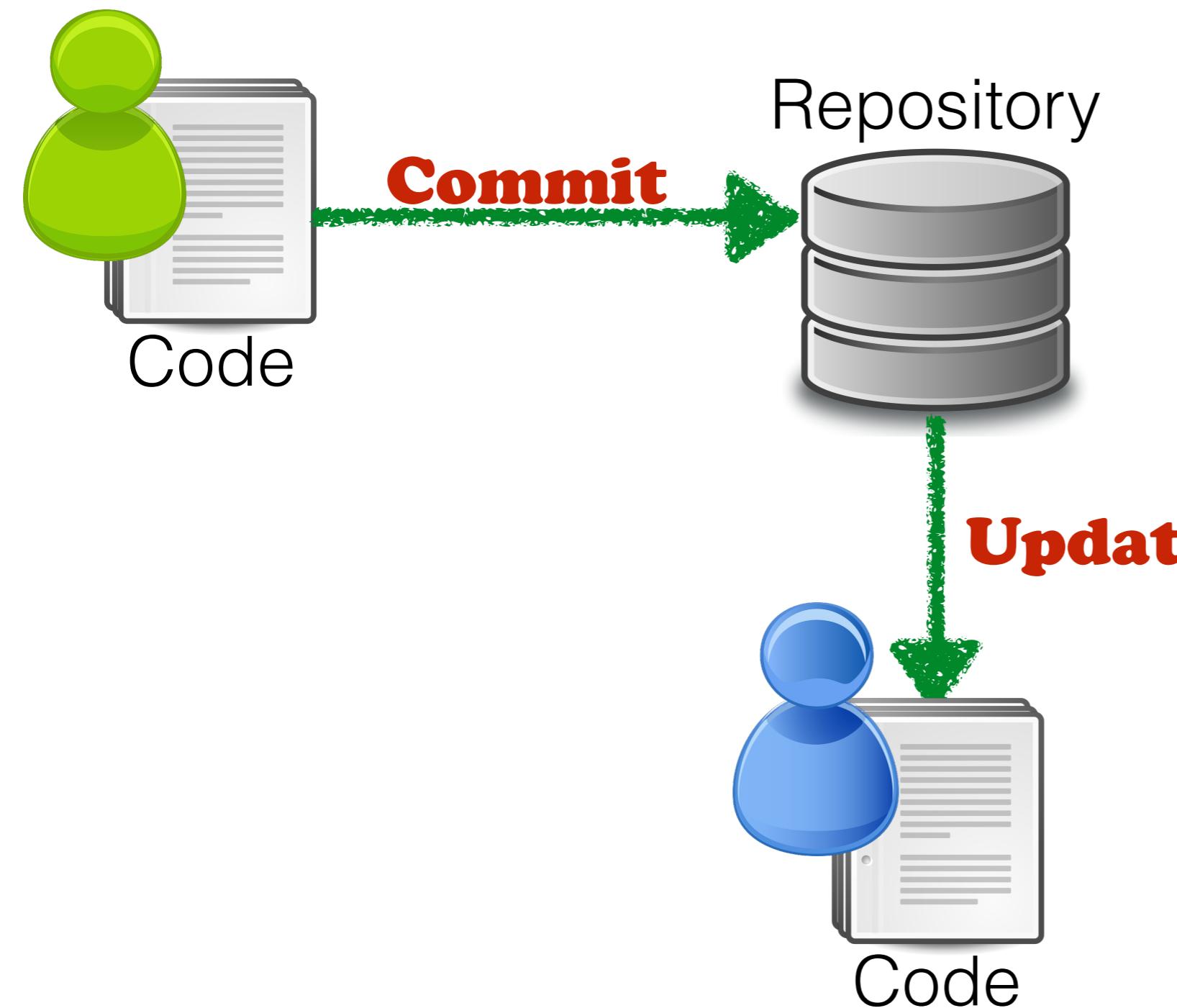
Repository



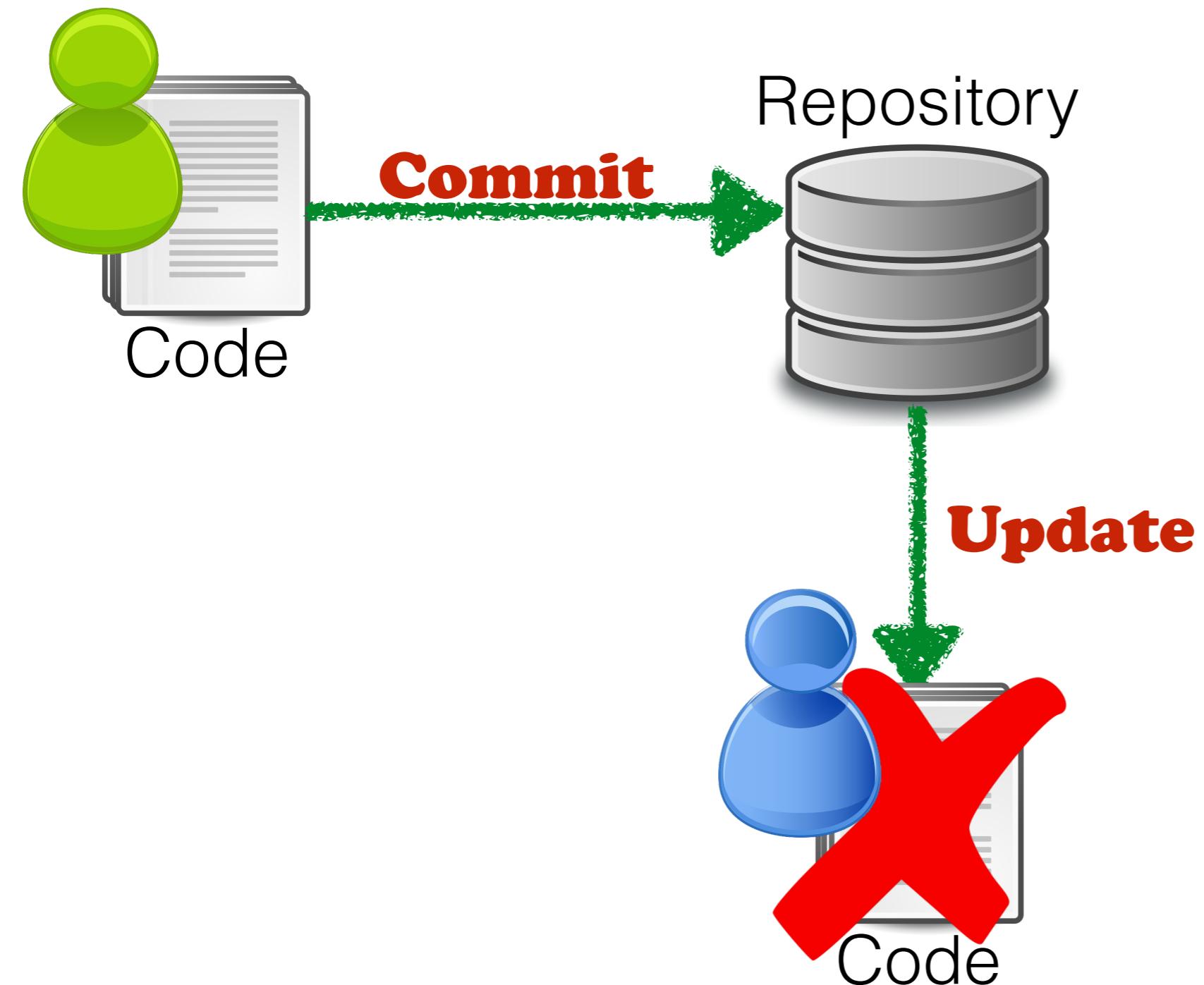
# Git Example



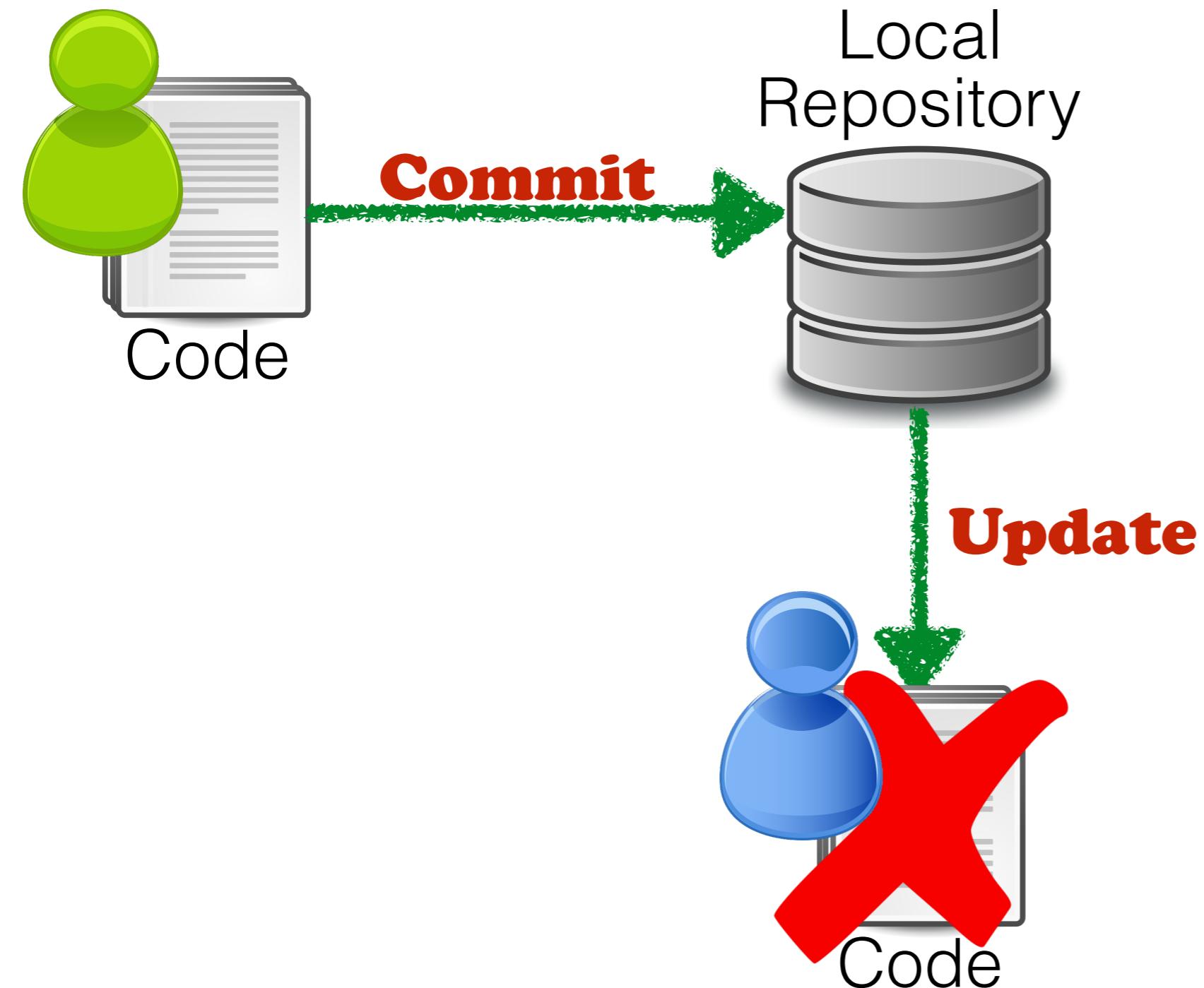
# Git Example



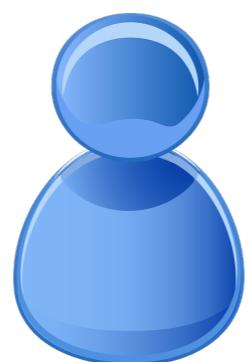
# Git Example



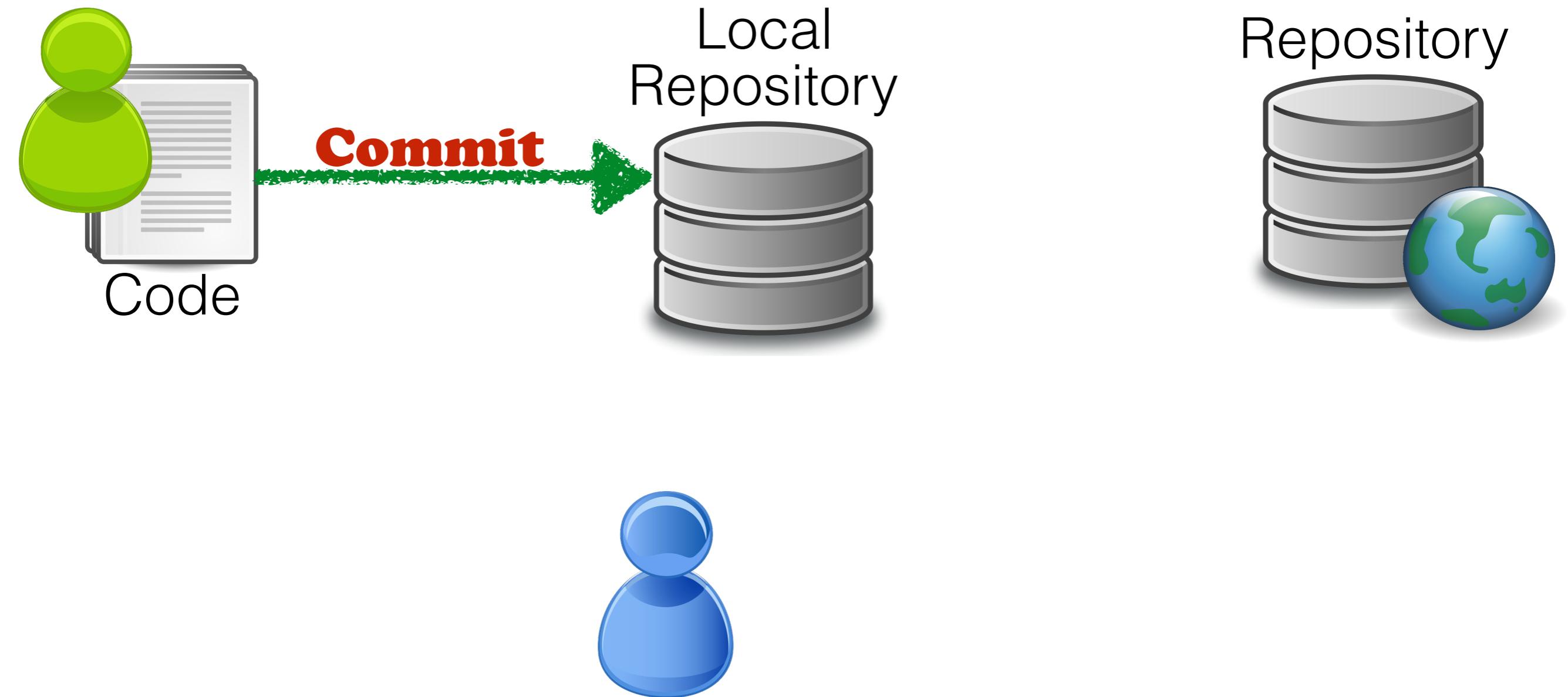
# Git Example



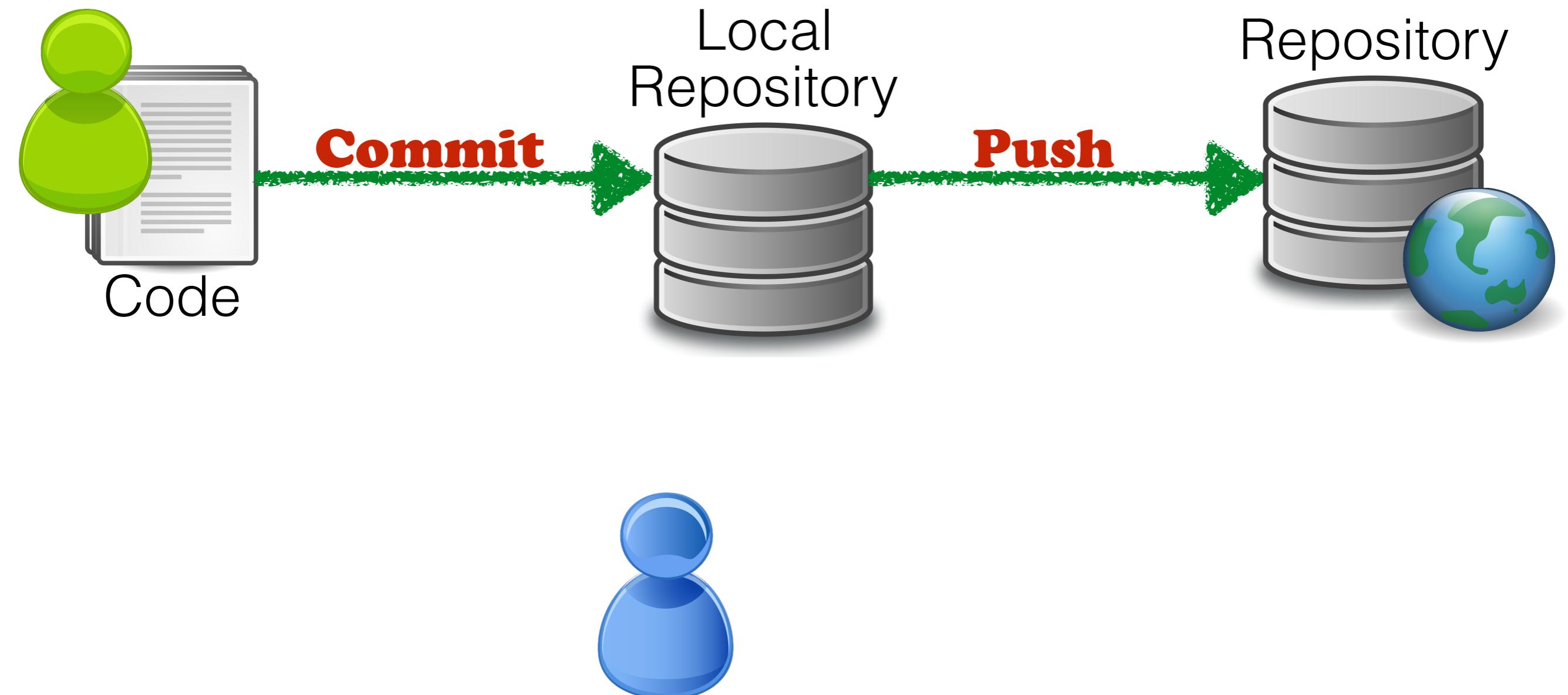
# Git Example



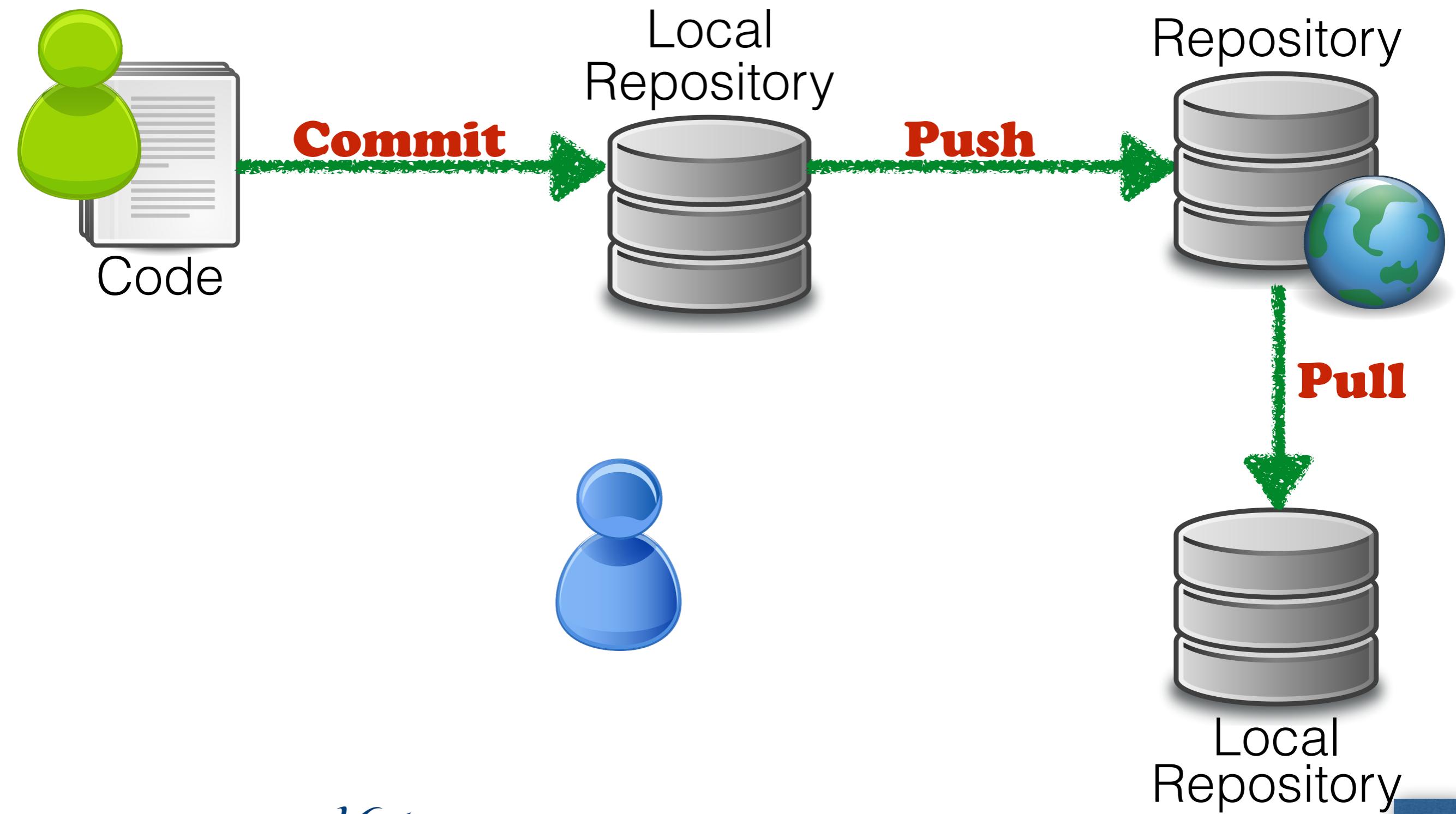
# Git Example



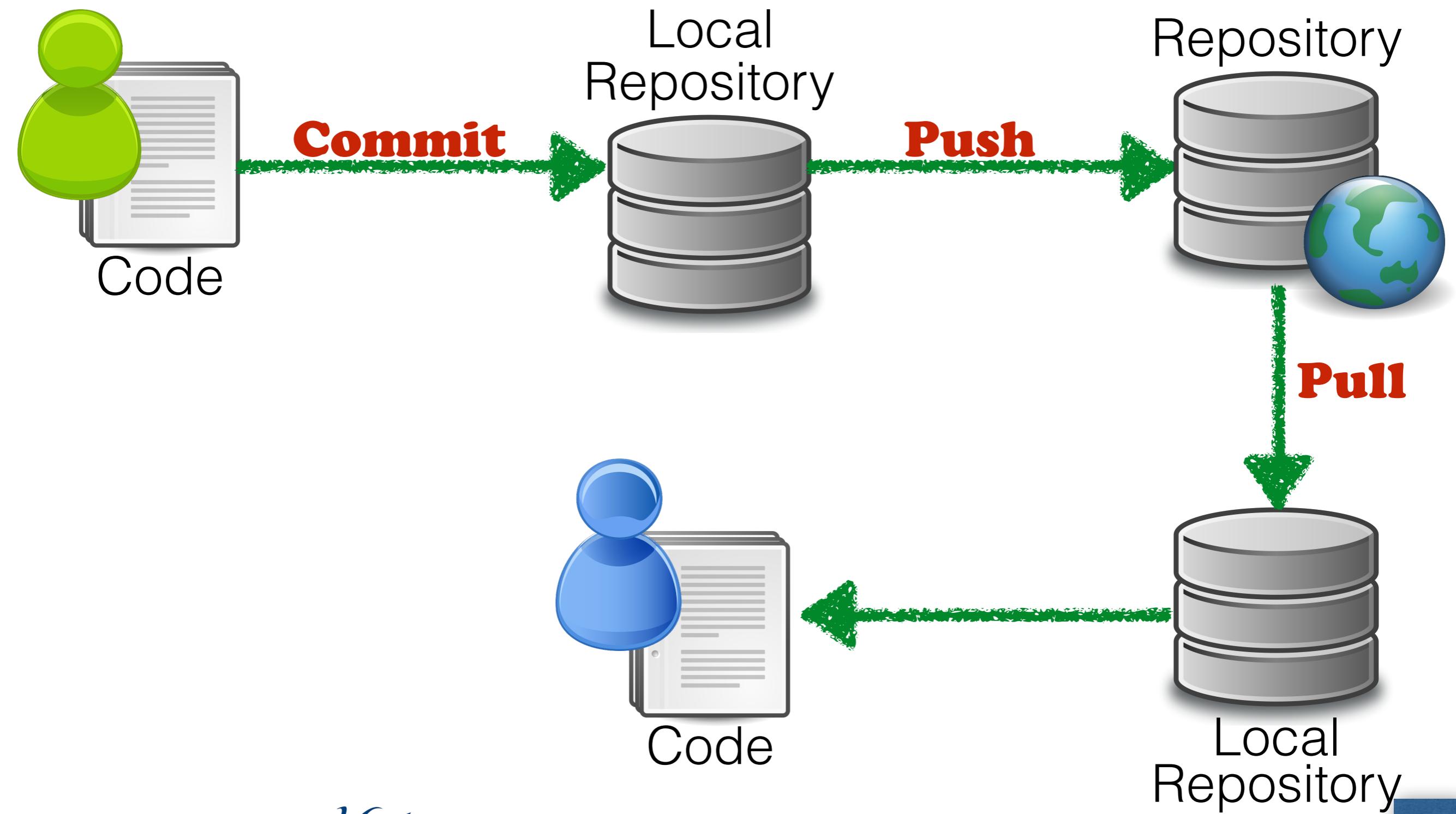
# Git Example



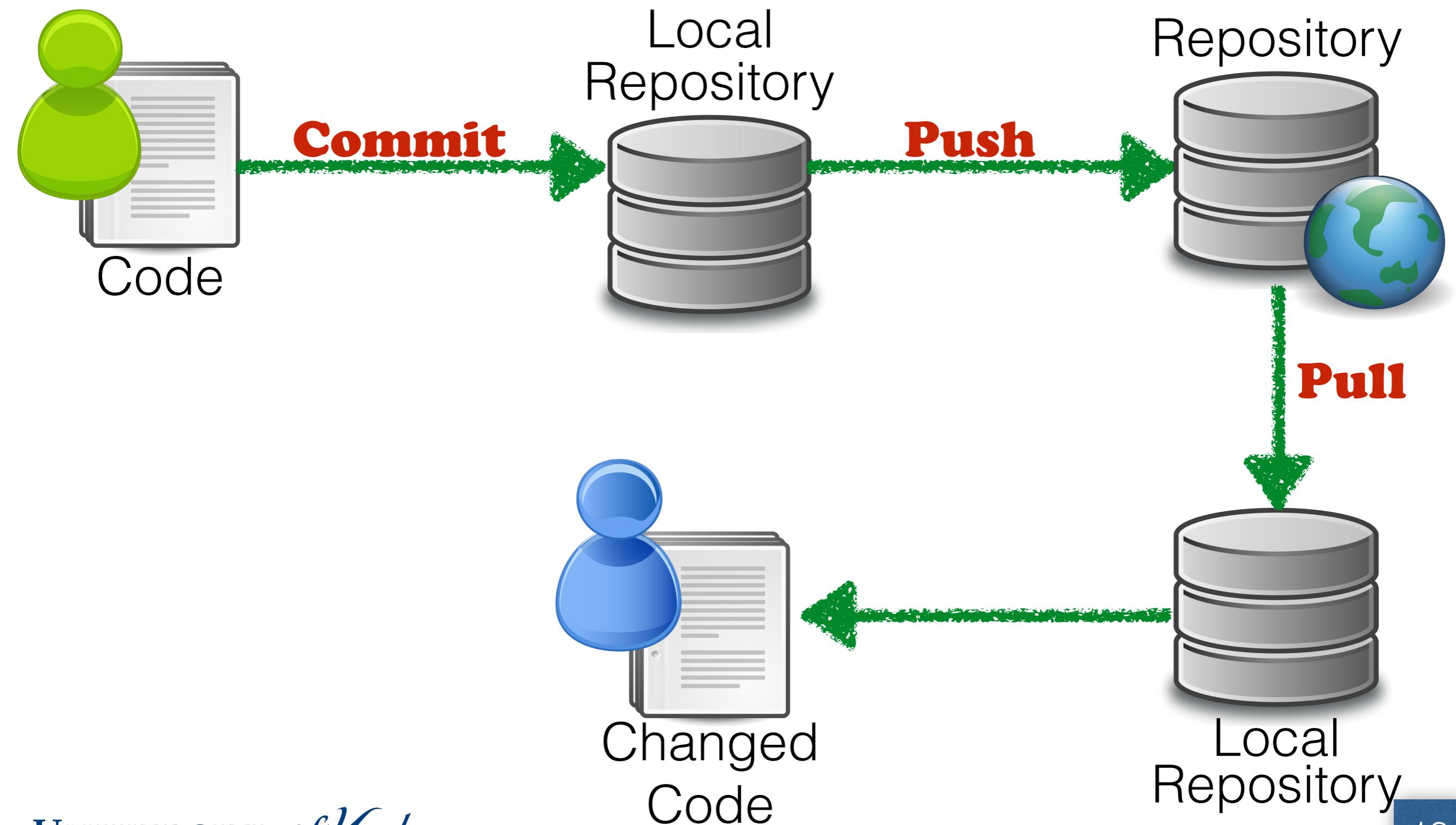
# Git Example



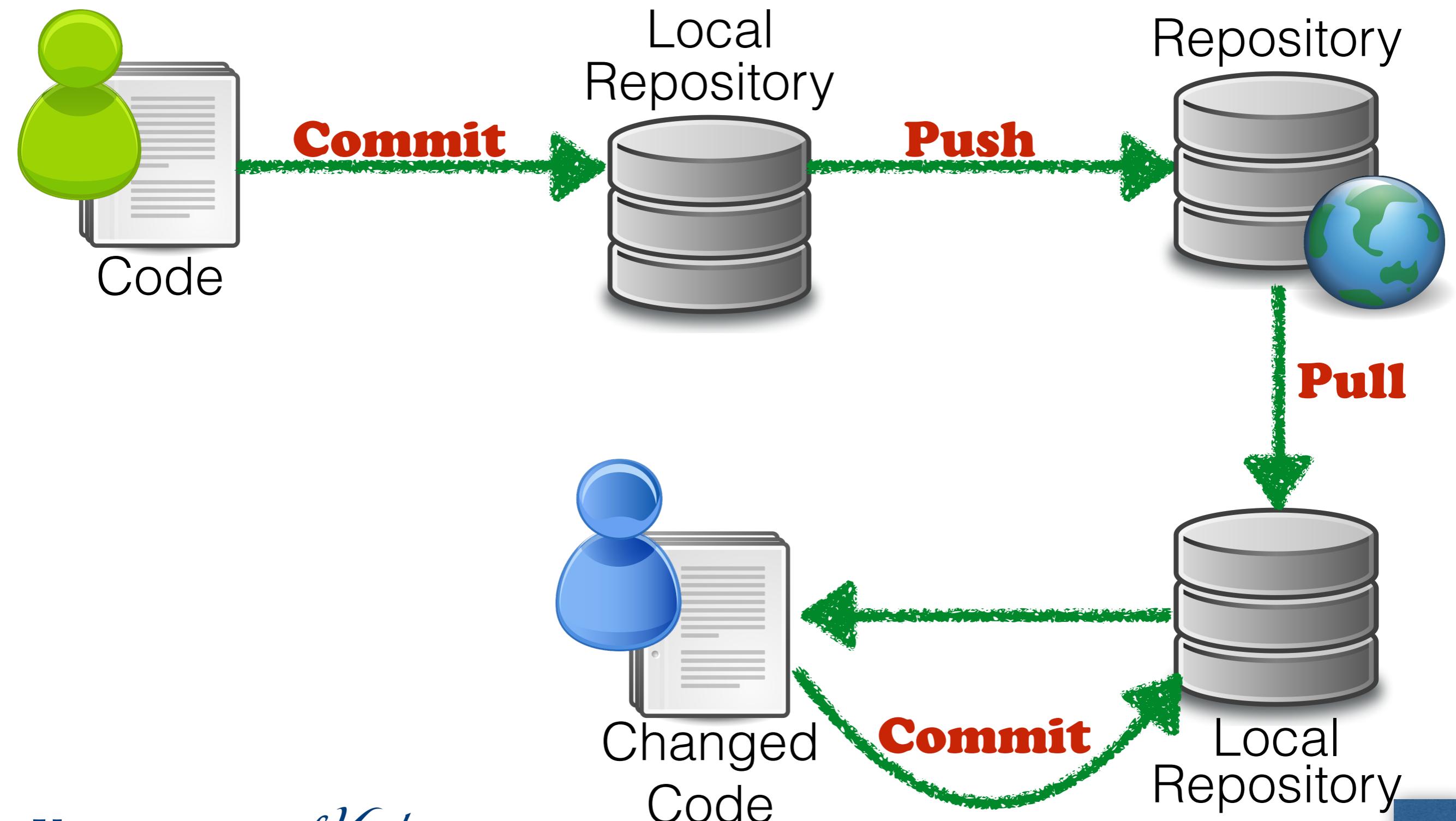
# Git Example



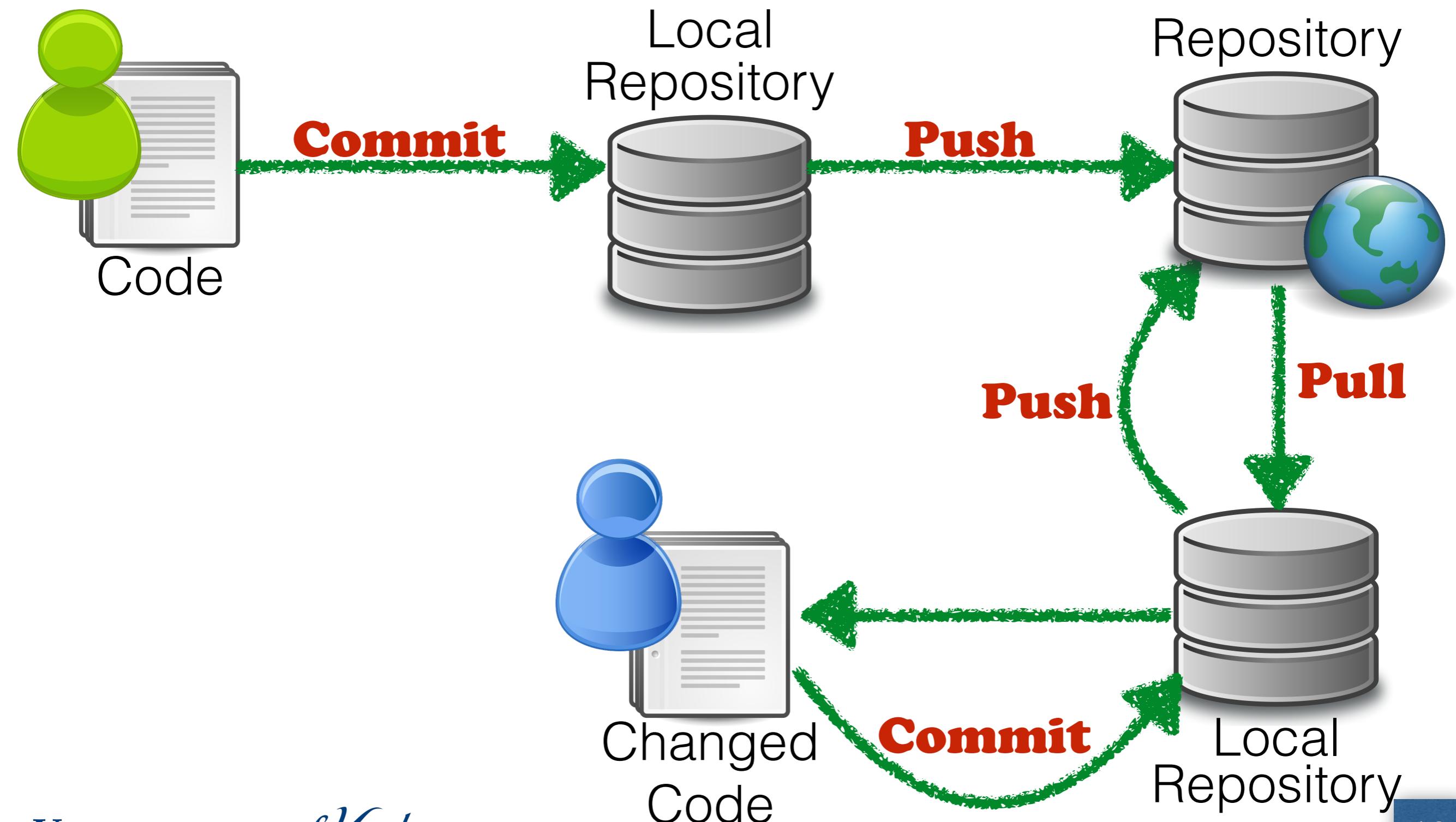
# Git Example



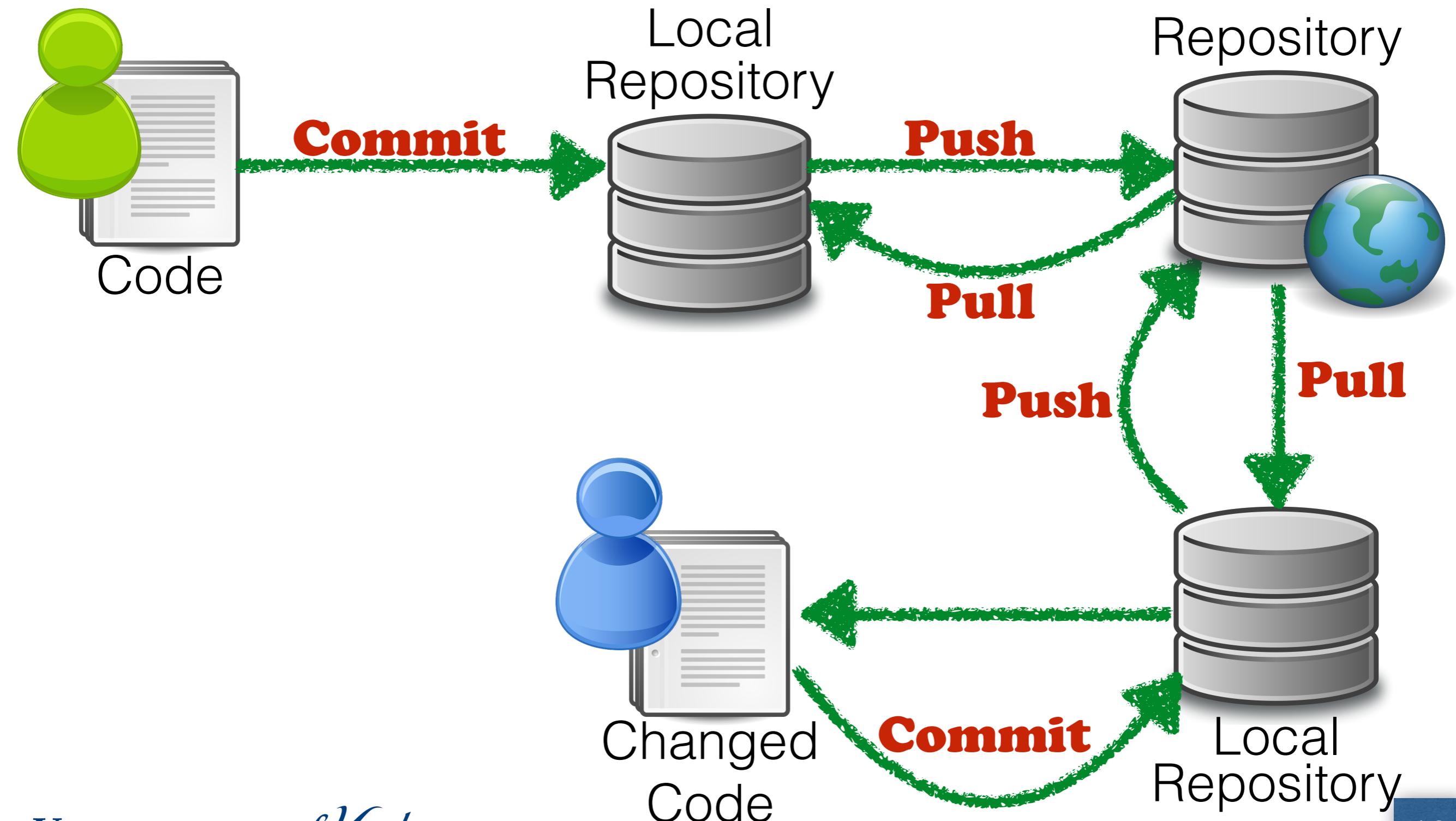
# Git Example



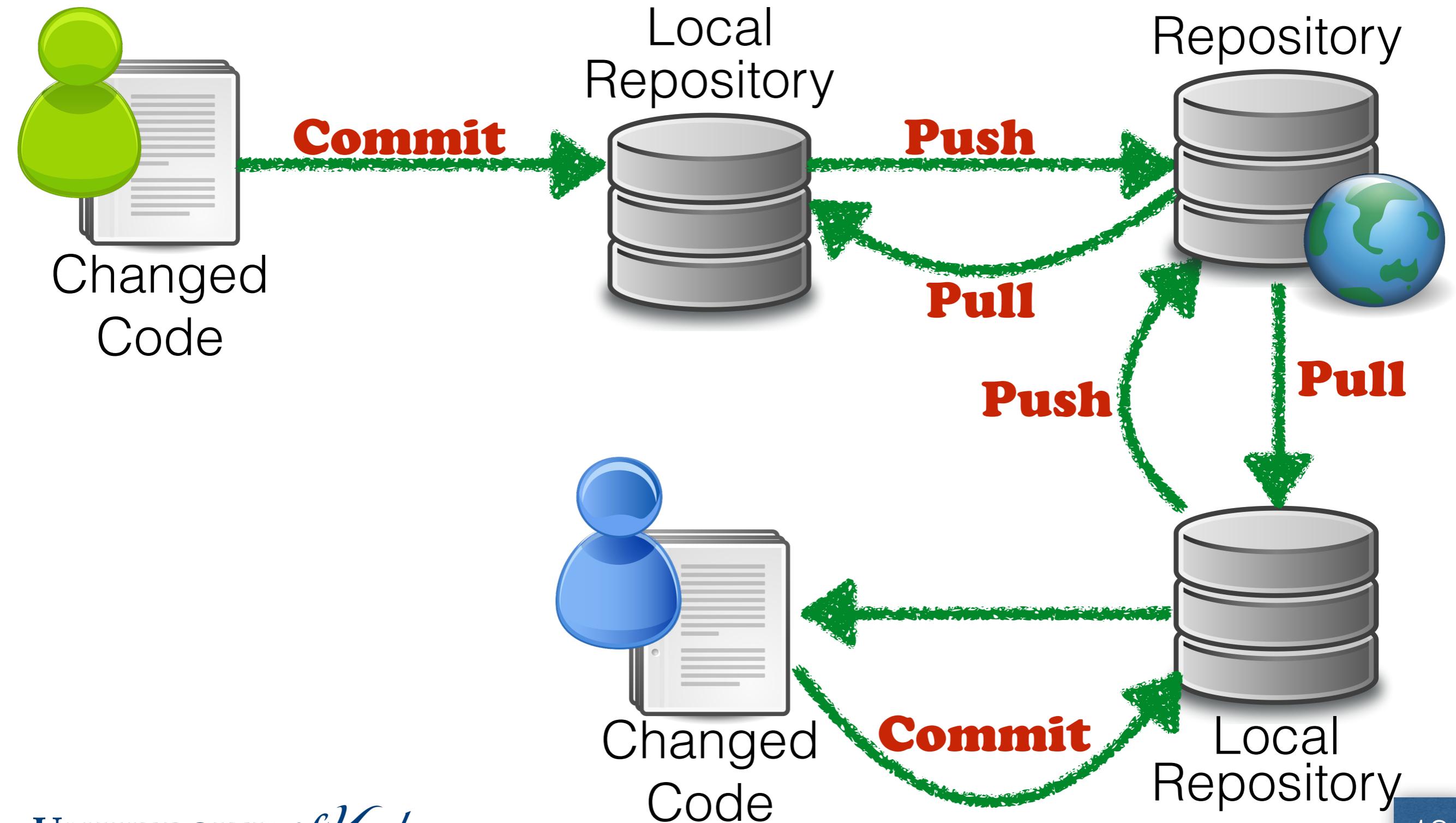
# Git Example



# Git Example



# Git Example



# Decentralised



Local  
Repository



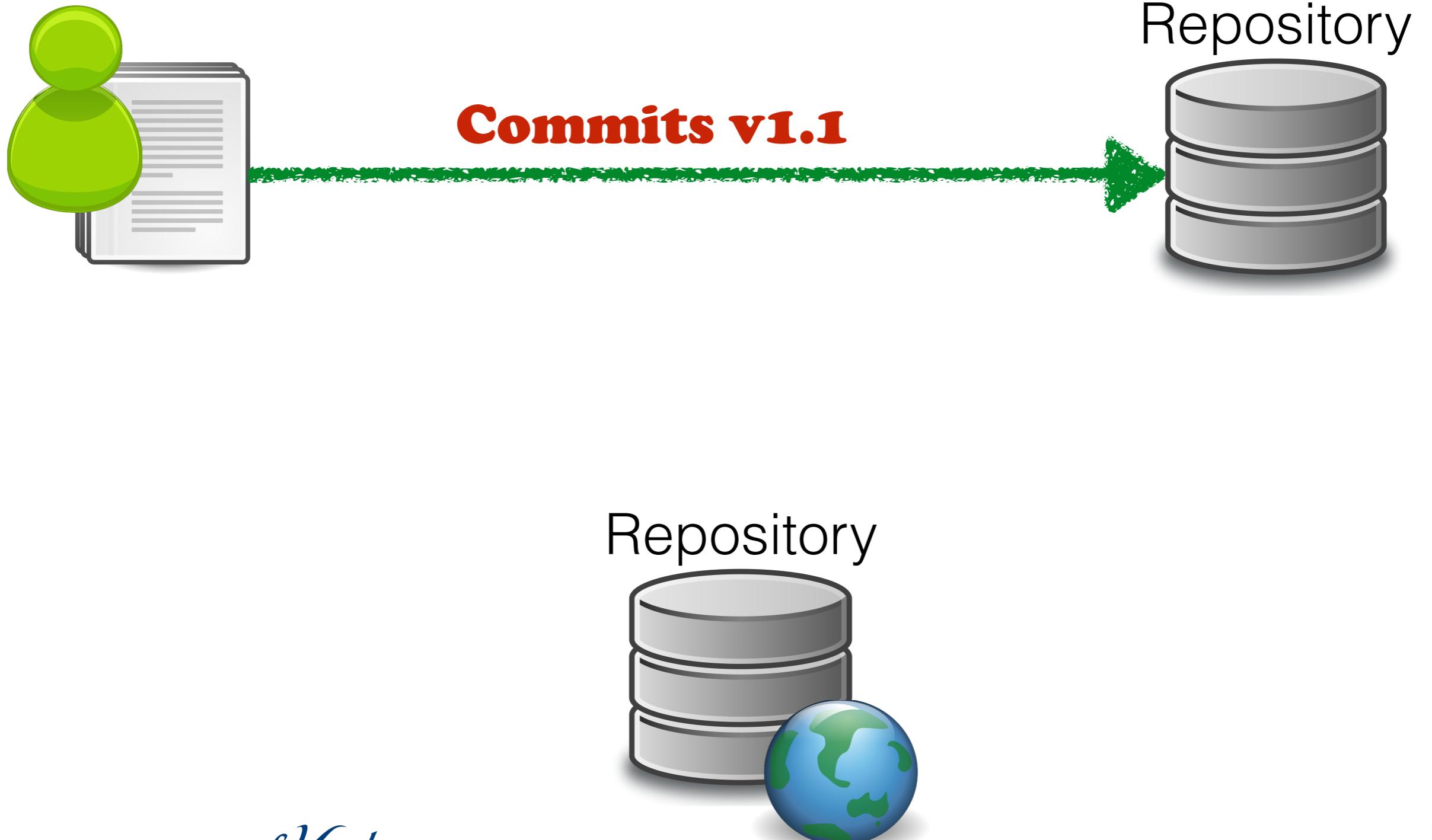
Repository



# Decentralised



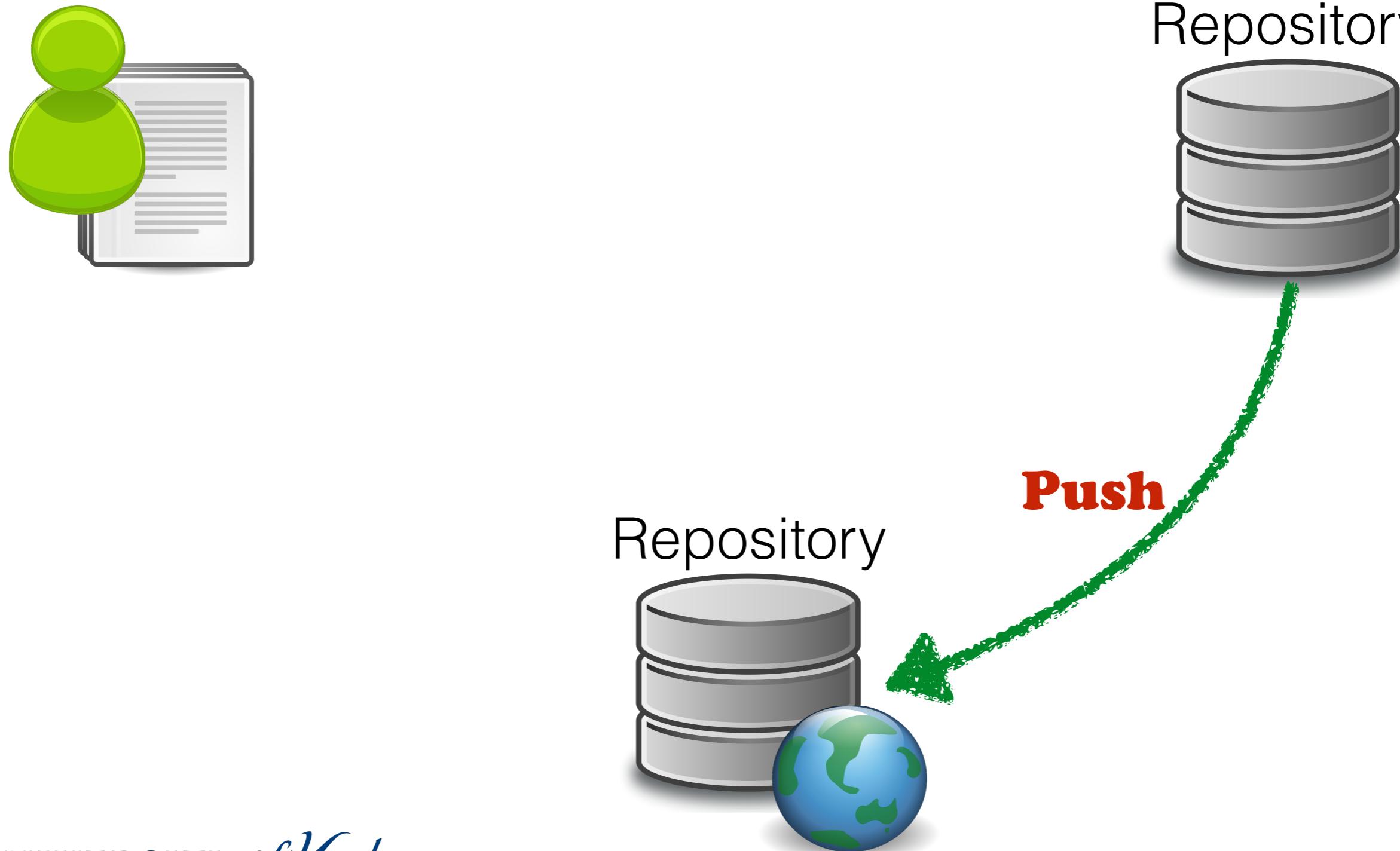
# Decentralised



# Decentralised



# Decentralised



# Decentralised



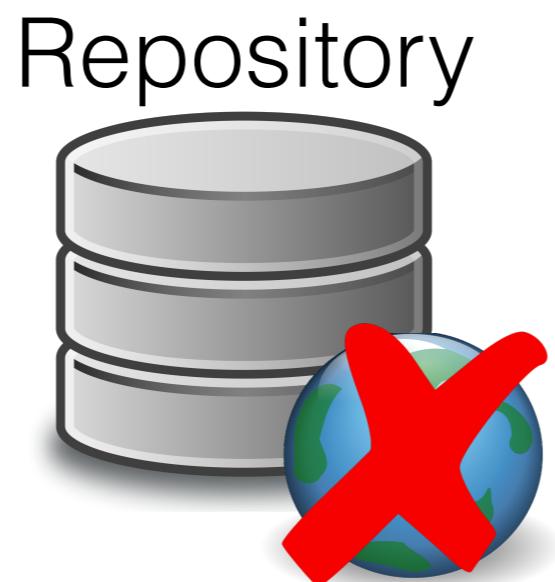
Local  
Repository



Repository



# Decentralised



# Git Software

- Windows:
  - **Git-bash** (command line) - free
  - **SourceTree** - free
- Mac OS:
  - **Terminal** - free
  - **SourceTree** - free
- Linux:
  - Git Cola - free
  - GitK - free
  - Terminal - free
- Eclipse (cross-platform):
  - EGit - free
- GitKraken (cross-platform)

# Git Pros & Cons

# Git Pros & Cons

- ✓ Small (offline) commits

# Git Pros & Cons

- ✓ Small (offline) commits
- ✓ Encourages branching

# Git Pros & Cons

- ✓ Small (offline) commits
- ✓ Encourages branching
- ✓ Widely used (trending VCS)

# Git Pros & Cons

- ✓ Small (offline) commits
- ✓ Encourages branching
- ✓ Widely used (trending VCS)
- ✓ More features than SVN

# Git Pros & Cons

- ✓ Small (offline) commits
- ✓ Encourages branching
- ✓ Widely used (trending VCS)
- ✓ More features than SVN
- Can be difficult

# Git Pros & Cons

- ✓ Small (offline) commits
- ✓ Encourages branching
- ✓ Widely used (trending VCS)
- ✓ More features than SVN
- Can be difficult
  - Yes, more difficult than SVN (but after a while it is going to be your best friend in group projects)

# Practical

- Let's try Git
- 1st part: single-user exercises
- 2nd part: branches, collaboration & conflicts
- 3rd part: work with your teams

# Part 1

- Navigate to this practical's git repo: <https://github.com/louismrose/VersionControlLecture>
  - <http://tinyurl.com/git-york> (or this that will redirect you to the above)
- Follow the instructions under Part 1
- IF YOU'VE FINISHED TAKE A BREAK. WE WILL BE BACK AT **10:30** FOR PART 2.

# Part 2

- Collaboration through GUI (SourceTree). I will:
  - Create a repo on GitHub
  - Clone it locally
  - Create files, stage them and commit
  - Invite a collaborator
  - Work together
  - Create and resolve conflicts

# Part 3 (optional)

- Having a VCS for your team project is not a requirement
- ... however, I encourage you to do or at least give it a try
- If you are thinking of having one then spent the remaining time on:
  - Choosing which is the best for your needs
  - Experiment with different GUIs and choose which fits you best (or reject them all and use terminal)
  - If you pick Git, then create a repository online (Github or Bitbucket are good services)
  - Add the files you have so far created for your project
  - Try to collaborate
  - If conflicts appear don't panic - try to resolve them