Lecture 18
Git Version Control

CS211 – Fundamentals of Computer Programming II

Branden Ghena – Spring 2023

Slides adapted from:
Pat Pannuto & Marcus Darden (Michigan), Max Goldman & Rober Miller (MIT), Michael Ernst (Washington)

Administrivia

- Projects are due on Friday!
  - Submissions to Gradescope are available now
  - It will check that your submitted code compiles properly
    - Submit as many times as you want to
    - Make sure to submit `Resources/` files!!
- Evaluation guide will be due over the weekend
  - For each spec item, tell us how to confirm you did it
  - Also identify your "non-trivial" model tests
- Quiz today at 3pm (warn your absentee friends)

Today’s Goals

- Understand concepts behind Version Control Systems
  - Why are they important?
  - How do we use them?

- Describe one specific Version Control System: Git
  - How does Git work conceptually?
  - How do we use Git?

  - Where does Github fit into this?

Guides for learning Git

- Understanding Git commits and branches
  - https://learngitbranching.js.org/

- Remember git commands
  - https://education.github.com/git-cheat-sheet-education.pdf

- Learning more about Git
  - https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud
  - https://git-scm.com/book/en/v2

Outline

- **Version Control Overview**
- Systems for Version Control

- Git Commit Structure
- Using Git
- Using Github
- Best Practices

Simplest ideas of version control

- Undo/Redo in programs
  - Keeps track of prior actions and lets you go back to them

- Manual file renaming

Project Report

Project Report v2

Project Report v3

Project Report final

Project Report final v2

- What if I told you there was a better way?

You mean there’s a better way?

Building a better system: backup naming

- Start with a system capable of doing the file rename for you
  - When you choose to “commit” the file, the system makes a backup copy
  - Backup copies are kept with metadata
  - Examples:
    - What time was this version saved
    - Who made the changes to the file
    - Message from the user about what changed

Need some way to “revert” to an old version

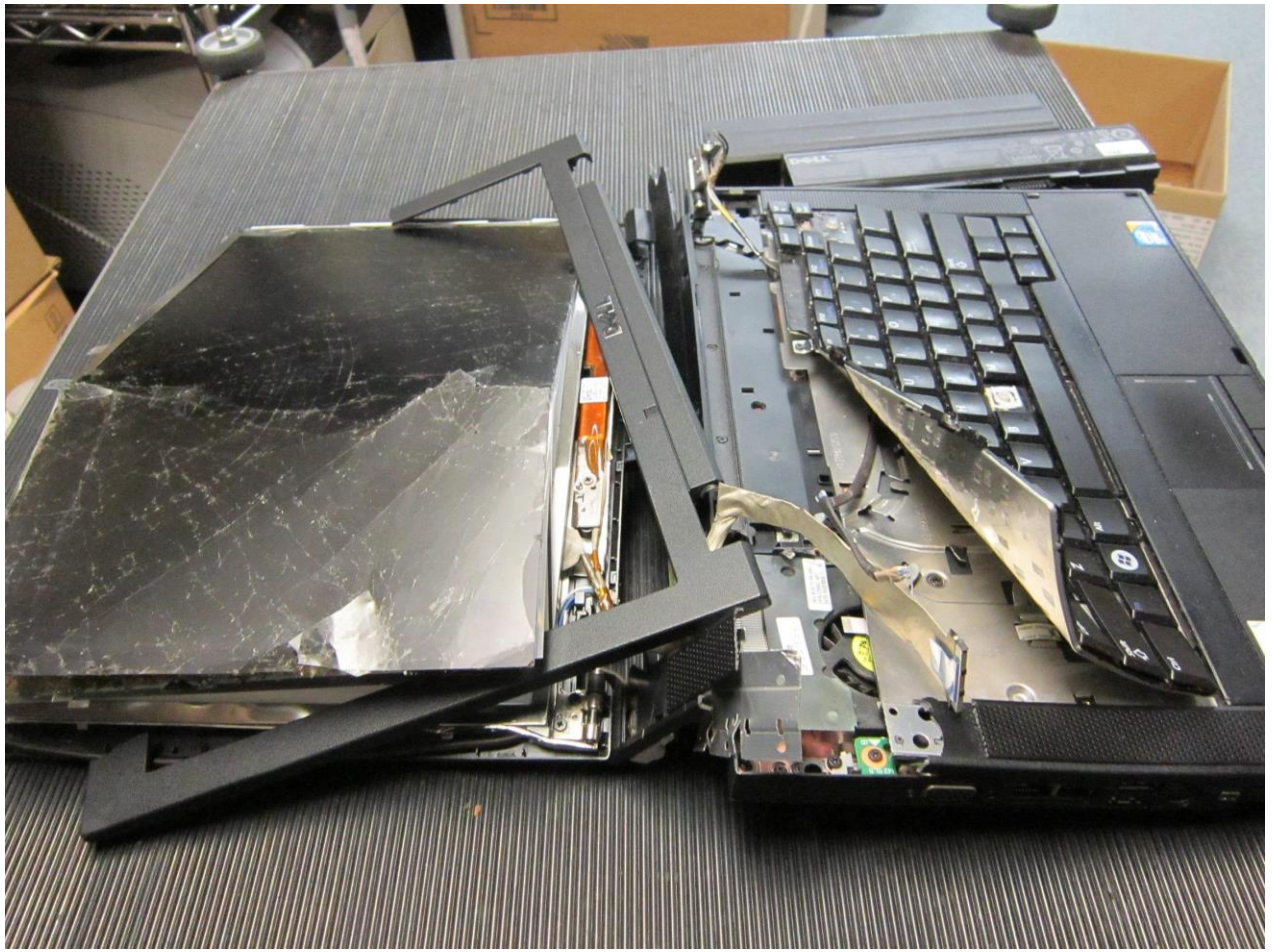
- Oh no! This most recent version broke something

- Change back to a previous version of the file
  - Or maybe several versions ago

- Might also ask to see what changed since previous version
  - One of those lines must be what broke it

  - How we do this depends on the file
    - Code: line-by-line comparison
    - Word documents: more complicated…

How do we survive tragic computer accidents?



- Need backups on another computer

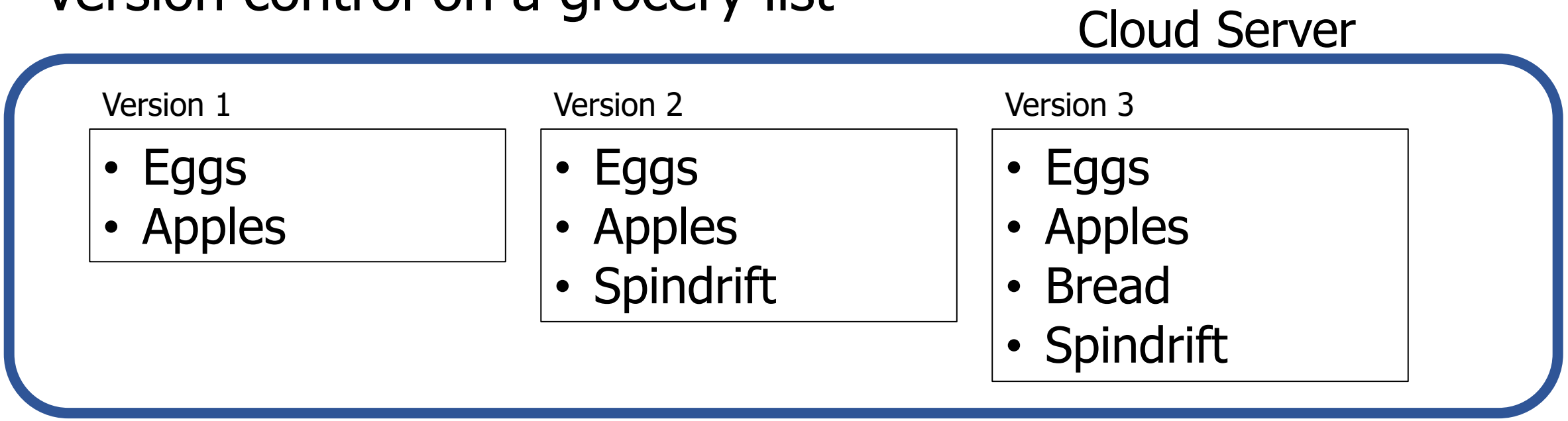
Improve reliability with cloud backups

- Sync files up to the cloud
  - Includes all versions of all files
  - Probably does some stuff to optimize space
    - Only keep the changes, not the whole file

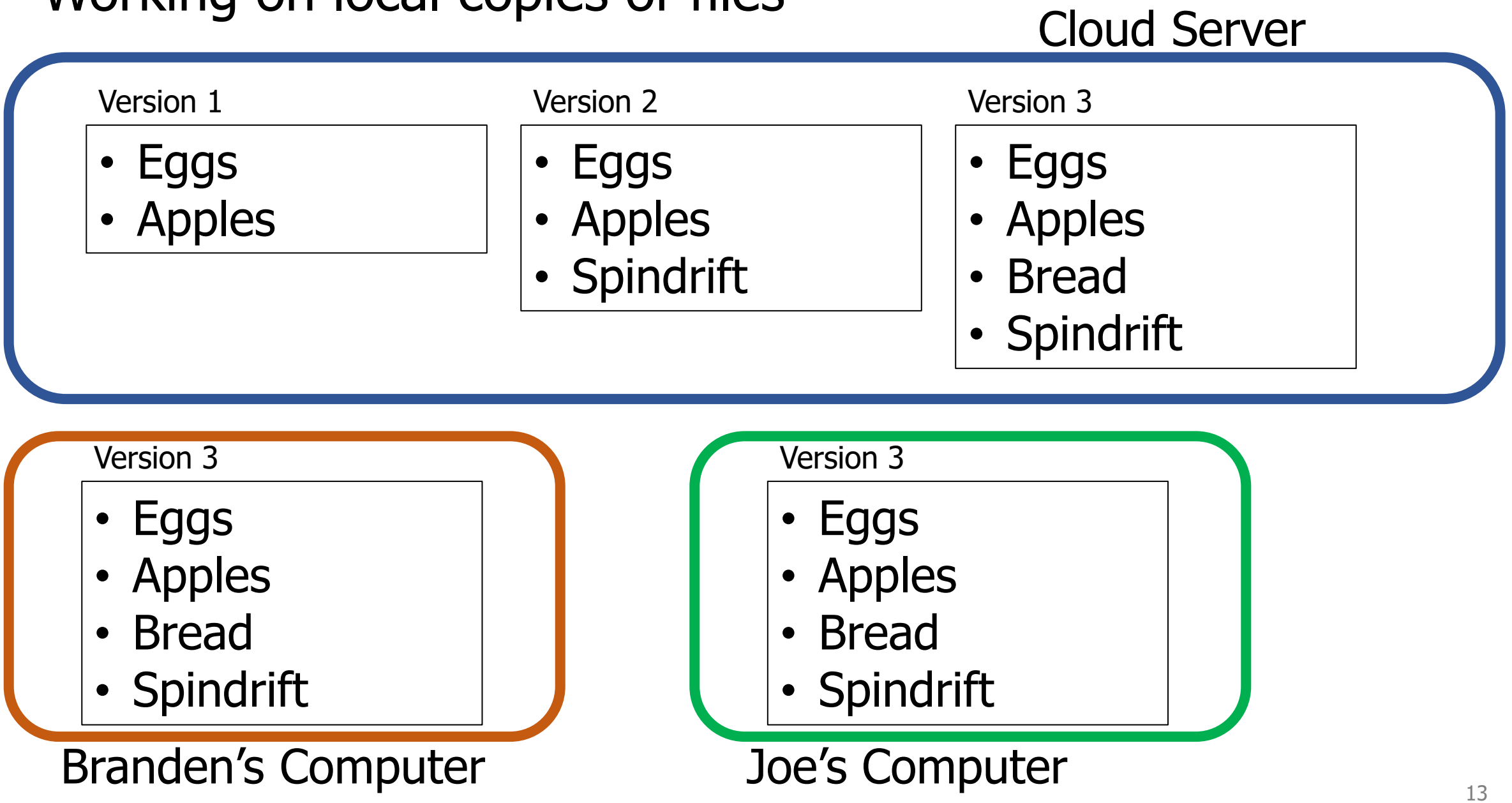
- Download files from the cloud
  - Provides the most recent file (“Head”)

  - Enables sharing files across multiple computers!
    - Across one or multiple people

Version control on a grocery list



Working on local copies of files



Making a new version of a file

Cloud Server

Version 1

- Eggs
- Apples

Version 2

- Eggs
- Apples
- Spindrift

Version 3

- Eggs
- Apples
- Bread
- Spindrift

Version 4

- Eggs
- Oranges
- Bread
- Spindrift

Branden's Computer

Version 3

- Eggs
- Apples
- Bread
- Spindrift

Joe's Computer

Conflicts when editing files!

Cloud Server

Version 1

- Eggs
- Apples

Version 2

- Eggs
- Apples
- Spindrift

Version 3

- Eggs
- Apples
- Bread
- Spindrift

Version 4

- Eggs
- Oranges
- Bread
- Spindrift

Branden's Computer

Version 4

- Eggs
- Apples
- Spindrift

Joe's Computer

Problem: simultaneous edits

- Multiple editors can lead to file conflicts!

- Whoever commits first wins, loser has to handle the problem

- How does the system handle “merging” the files?
  - Sometimes just ask the human to figure it out

  - Sometimes realize that changes are to different parts of the file and just apply both

Fundamental version control operations

- **Commit** file(s)
  - Save a new version of them
- **Revert** file(s)
  - Return to a previous version of them
- **Compare** file(s) across version
- **Push** file(s) to a server
- **Pull** file(s) from a server
- **Merge** changes to a file and handle conflicts

Version Control Systems are essential

- Not just for software, any files!
  - Code
  - Documents
  - Data files
  - Lecture slides

- Often designed with source code in mind
  - Work particularly well on human-readable text files
    - Comparisons can happen line-by-line (diff)
    - Text is easily compressed for transfer and storage

Outline

- Version Control Overview
- **Systems for Version Control**

- Git Commit Structure
- Using Git
- Using Github
- Best Practices

Version control terminology

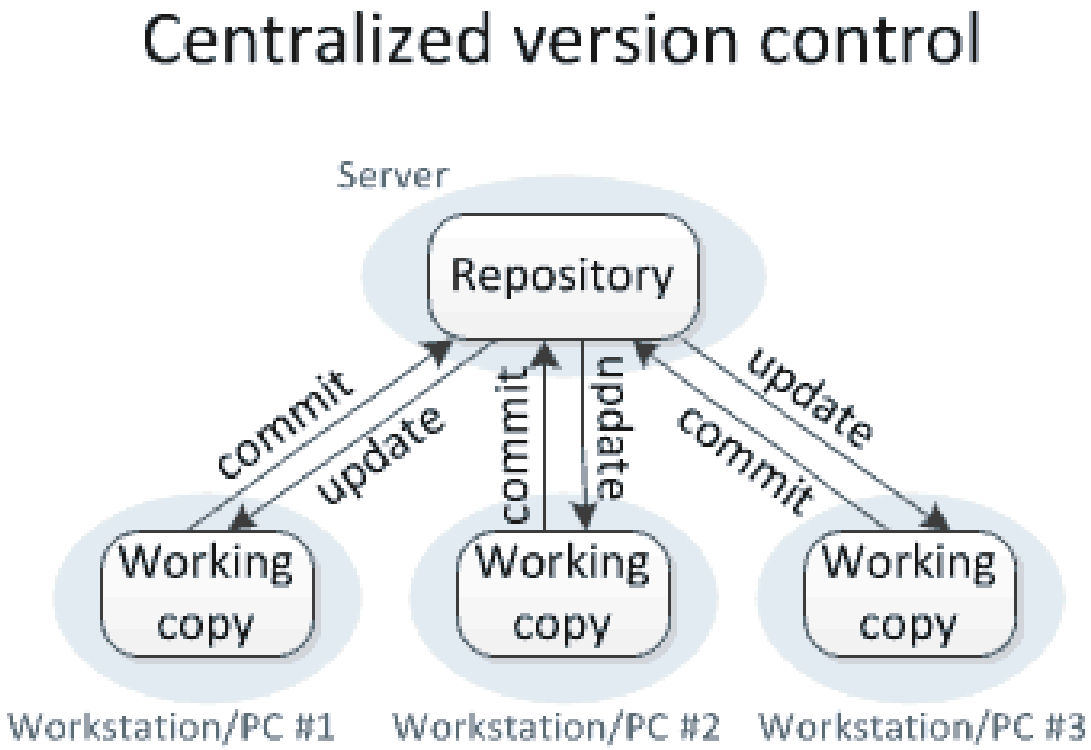
- Repository (a.k.a. repo)
  - Holds all of the versions of all of the files for a project
  - You commit files to a repo
    - And push to it, if it's on a different computer

- Local versions of files are known as the "Working Copy"
  - You can edit these files and then commit them to the repo

- The most recent version of a file is known as the "Head"

Older systems: Centralized version control

- Local computers only ever have a working copy
  - Must request version information from repo on a server



Centralized version control systems

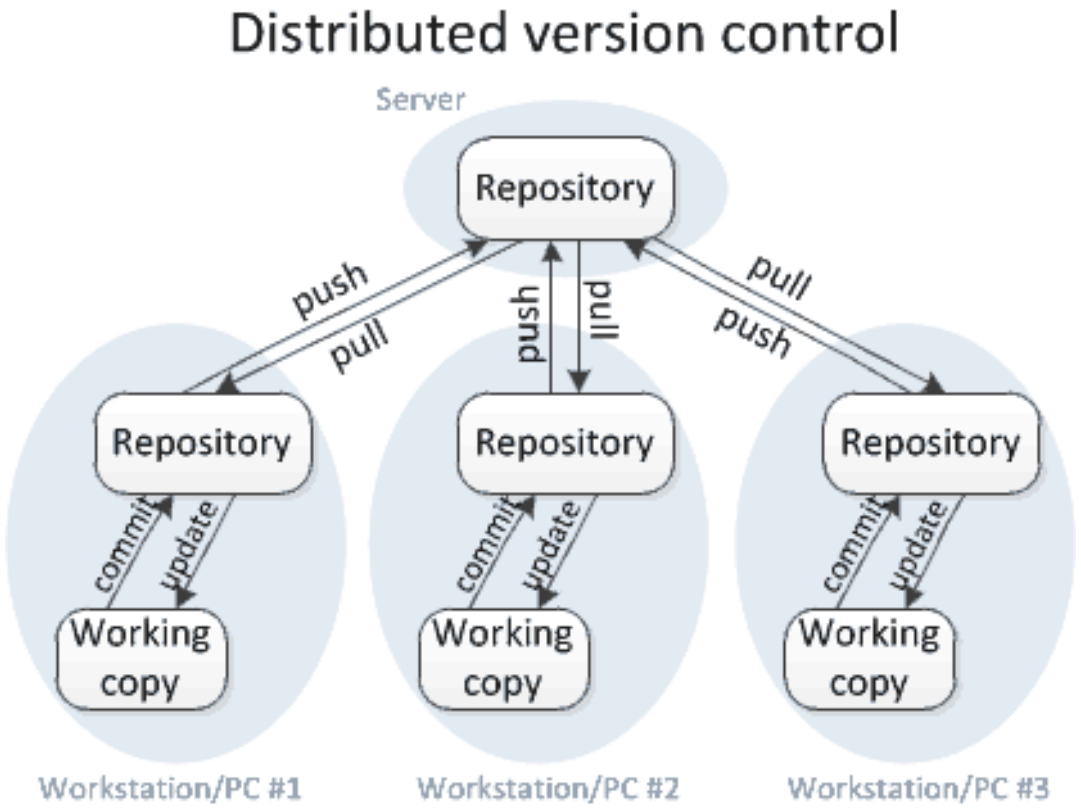
- Revision Control System (RCS), 1982
  - Basic idea of versioning for single files
- Concurrent Versions System (CVS), 1990
  - Expands version to an entire project of files
- Subversion (SVN), 2000
  - A "Commit" includes changes across multiple files
    - Often multiple source/header files might be changed together!
  - Ensures *atomic* changes to the repo

1st Generation

2nd Generation

Modern systems: Distributed version control

- Local computers have their own copy of the repository
  - Along with the working copy that users directly edit



Distributed version control systems

- Bazaar (bzr), 2005
  - No longer developed
- Mercurial (hg), 2005
  - Still developed, not widely used
- Git (git), 2005
  - Most popular version control system

3rd Generation

- All provide methods for enabling distributed version control
  - Changes can be made and tracked locally
  - Sets of changes can be sent to others as needed
    - Often to a central shared server

Sidebar: what happened in 2005?

- Bazaar (bzr), 2005
- Mercurial (hg), 2005
- Git (git), 2005

- Bitkeeper, a proprietary distributed version control system decided to end free access for open-source projects
  - Including Linux!

  - Led to simultaneous development of new systems
    - And the death of Bitkeeper

Break + xkcd

https://xkcd.com/1597/

Outline

- Version Control Overview
- Systems for Version Control

- **Git Commit Structure**
- Using Git
- Using Github
- Best Practices

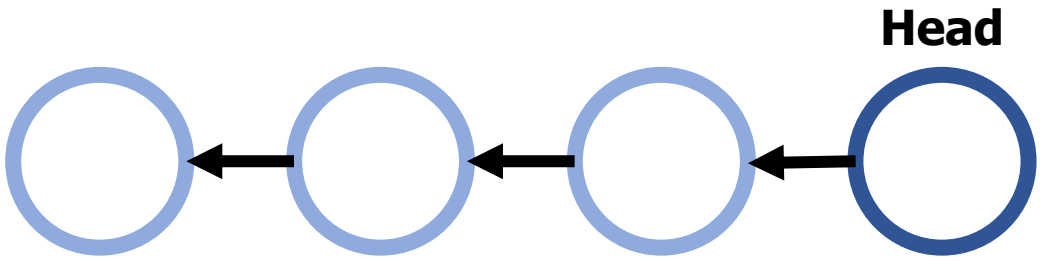
Commits are the units of change in Git

- A Commit contains
  - File modifications
  - Timestamps
  - Author of commit
  - Commit message
  - Parent of Commit

- Git commits each have a name
  - Example: 42e2e5af9d49de268cd1fda3587788da4ace418a
    - 160-bit SHA1 hash of the commit data (guaranteed unique)
    - Usually referred to by first 7 digits (268 million choices, likely unique)
      - Example: 42e2e5a

Every commit has a parent

- Simplest structure of commits can form a sort of linked list



The commit history can branch

- Multiple commits can have the same parent

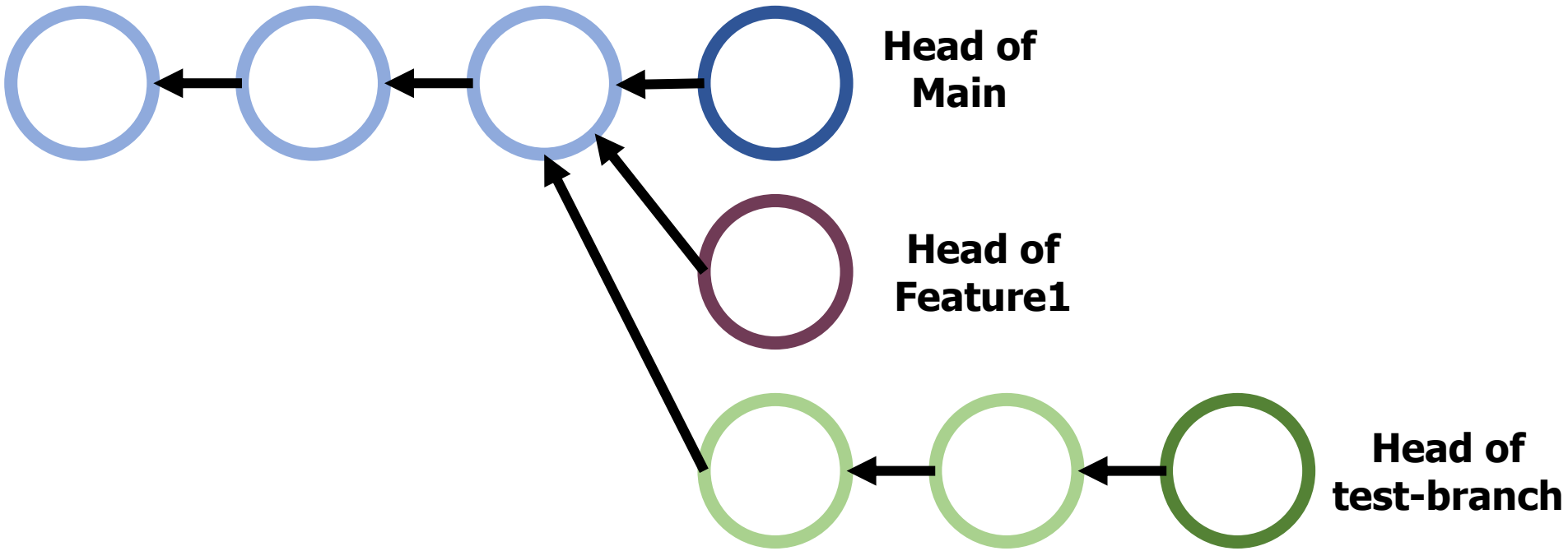
- Branching intentionally forms a new path for commits
  - Starts at a parent from the main “branch”
  - Continues on separately from there

  - Often used for development of new features

- Original code path is known as Main
  - Occasionally referred to as “master” in legacy projects

Some parents have multiple children

- Simplest structure of commits can form a sort of linked-list
- With branches, commits can form a tree structure



Branches may later merge back into Main

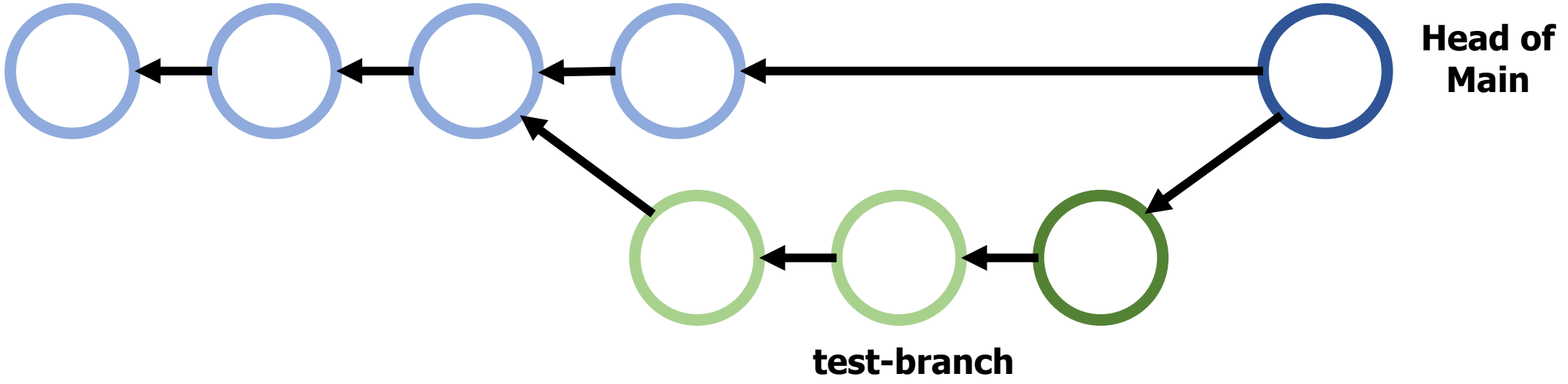
- Combining two branches requires a “merge” operation
  - Might include conflicts if both branches modify the same file!
  - Git is pretty good at automatically resolving conflicts
    - Unless the two branches both modify the same line of code

- Merge commit gets added to list the multiple parent commits

- There is an alternative: a “rebase” operation
  - Change branch’s parent to the current head of Main
    - Probably works as long as nothing major has changed

Git's true nature is a graph structure

- Simplest structure of commits can form a sort of linked-list
- With branches, commits can form a tree structure
- Reality of Git: Directed, acyclical graph
  - Each commit has one *or more* parents
  - There are no cycles (parents that are children of themselves)



Tags let you refer to specific commits from repo's history

- Git history can start to get rather complicated
  - What if you want to point to something other than the Head of Main?

- Tags are alternative names given to specified commits
  - Can be reverted to
  - Can be compared against
  - Usually for major versions of your project

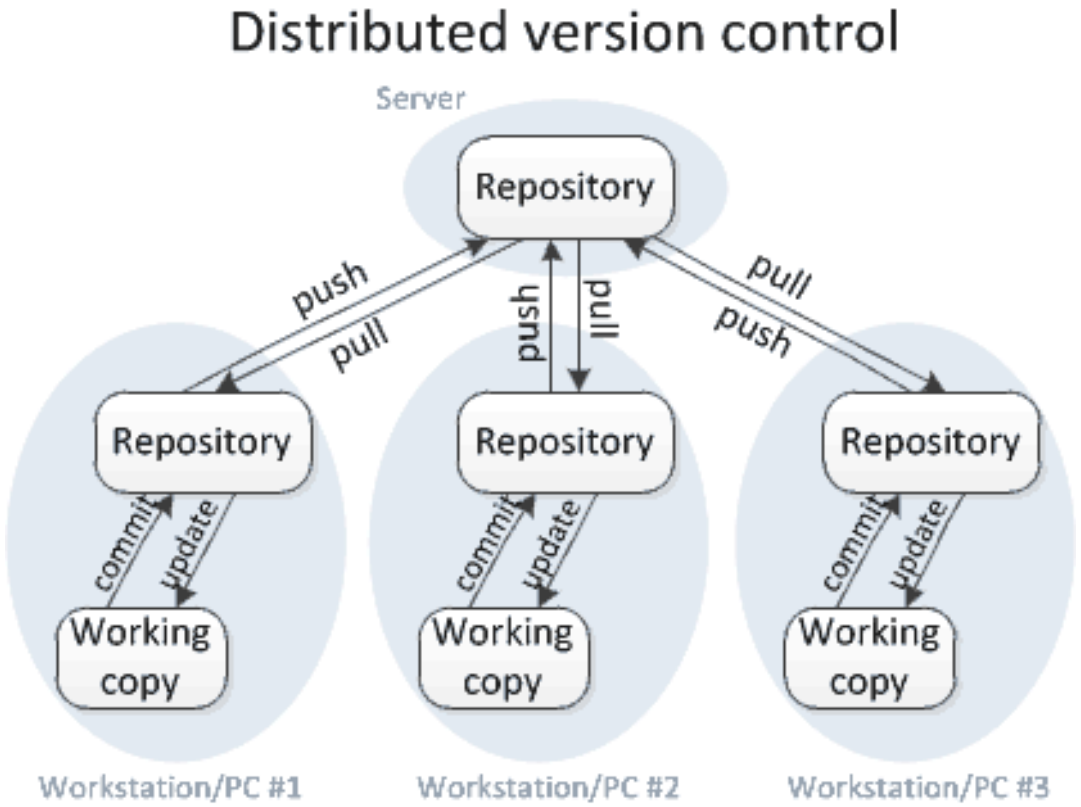
Outline

- Version Control Overview
- Systems for Version Control

- Git Commit Structure
- **Using Git**
- Using Github
- Best Practices

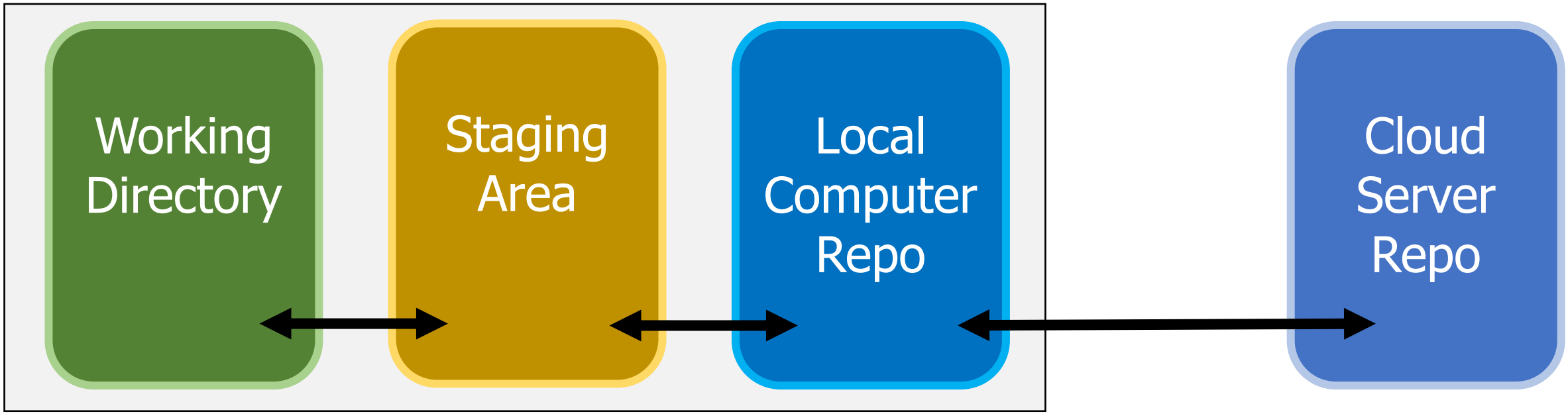
Git is a distributed version control system

- There will be one (or more!) server repositories
- Each user will also have their own local repository clone



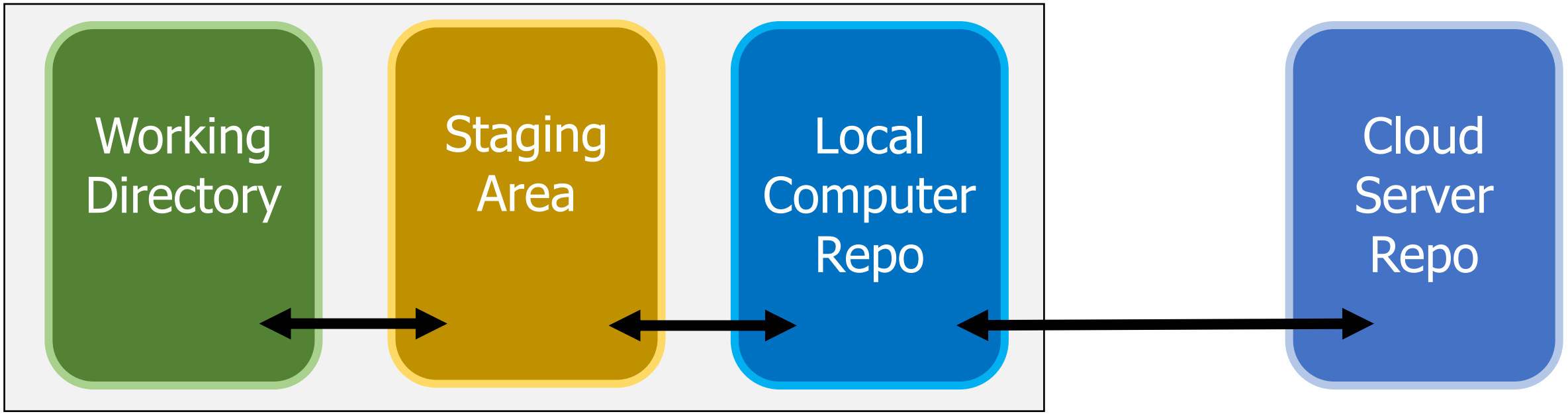
Git splits the local computer into several parts

- Local computer repo
  - A copy of the repo from the cloud
  - Might have various local commits/branches that aren't on the cloud



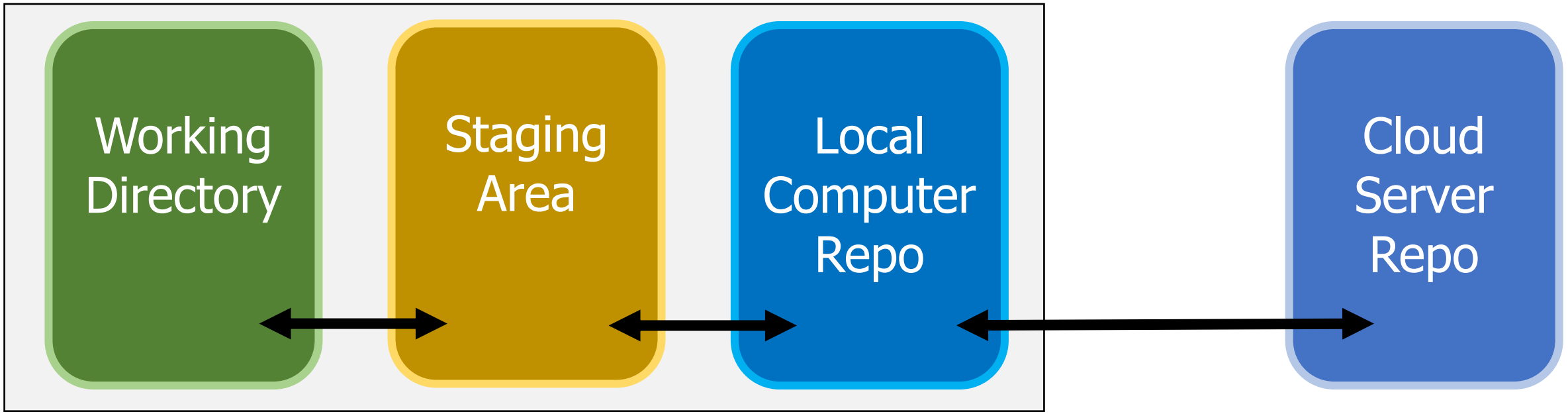
Git splits the local computer into several parts

- Staging Area
  - Where files are held that are ready to be committed
  - User selects files that are ready to commit and first adds them to staging area

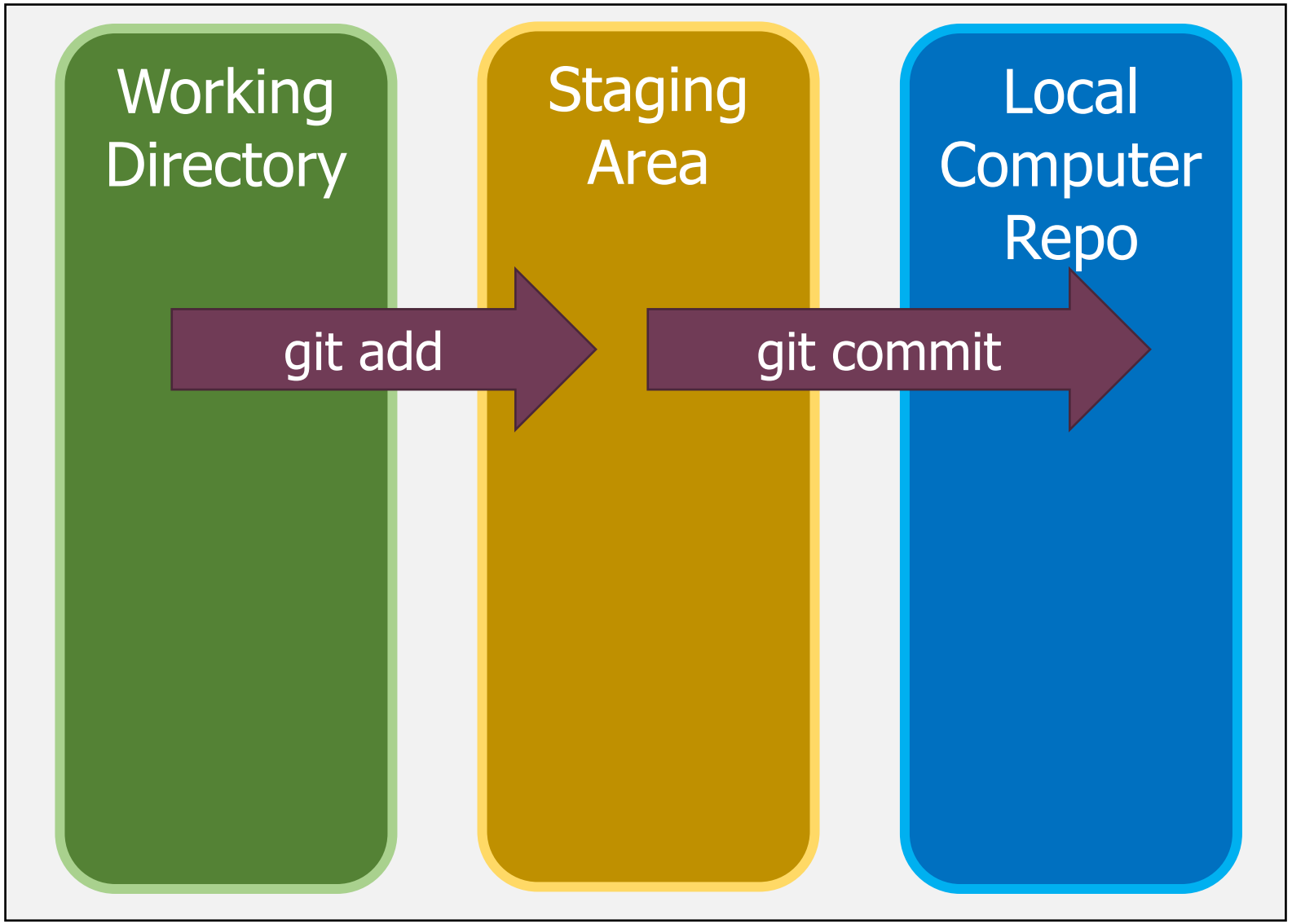


Git splits the local computer into several parts

- Working Directory
  - Files actually in use on the local computer
  - Initially matches a commit in the repo
    - Might include local edits that haven't been staged

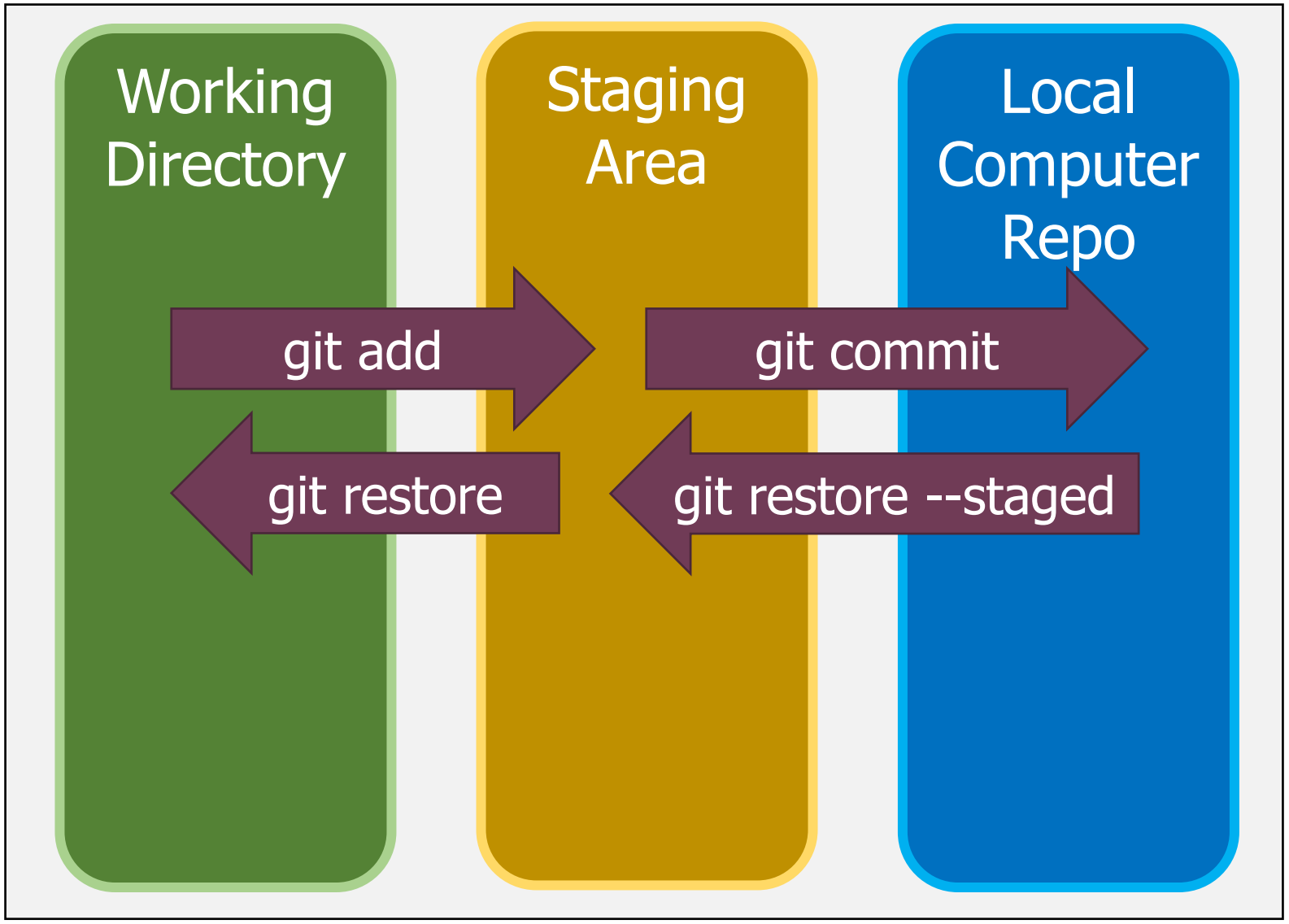


Git commands modify files in different areas



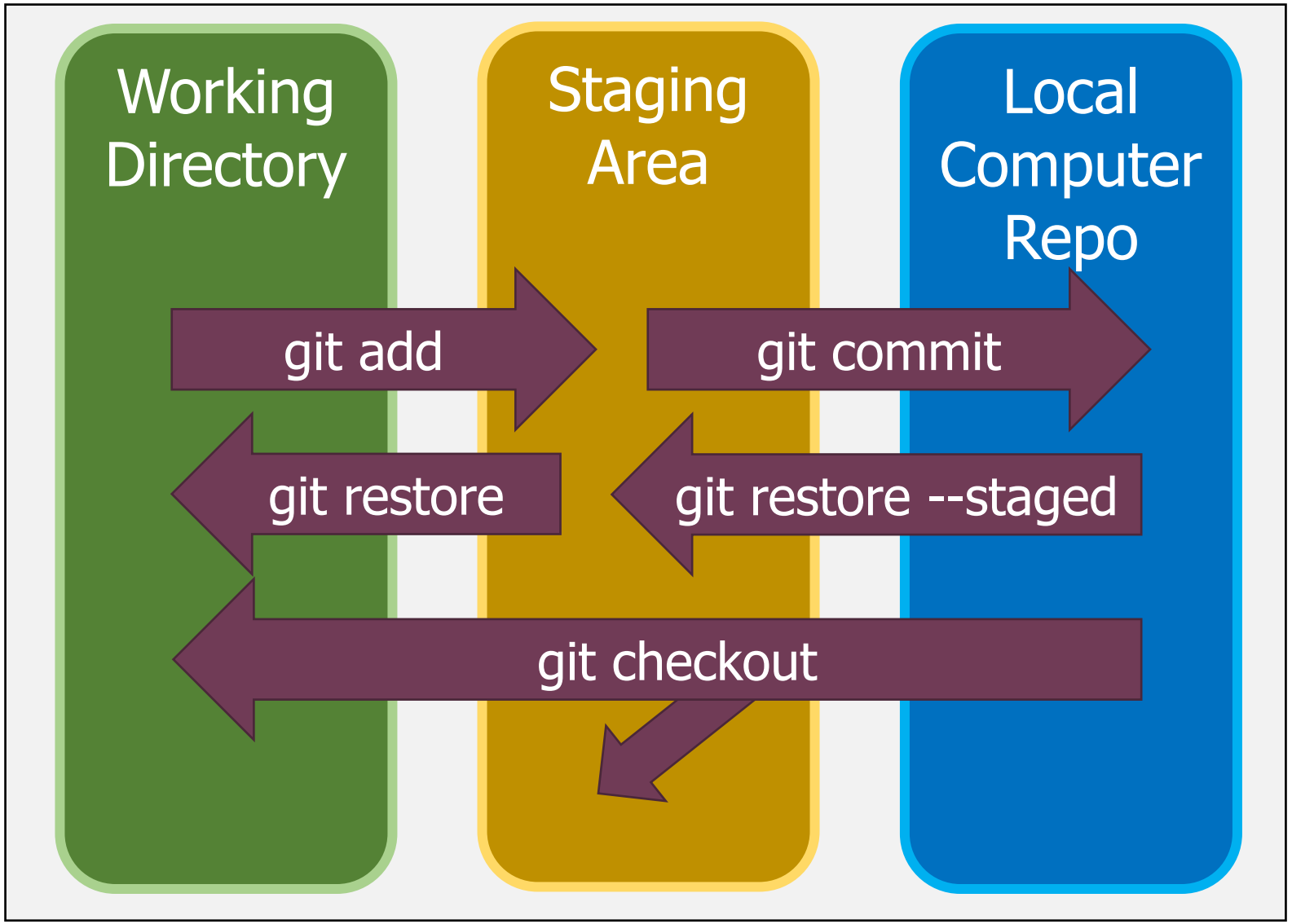
Cloud Server Repo

Git commands modify files in different areas



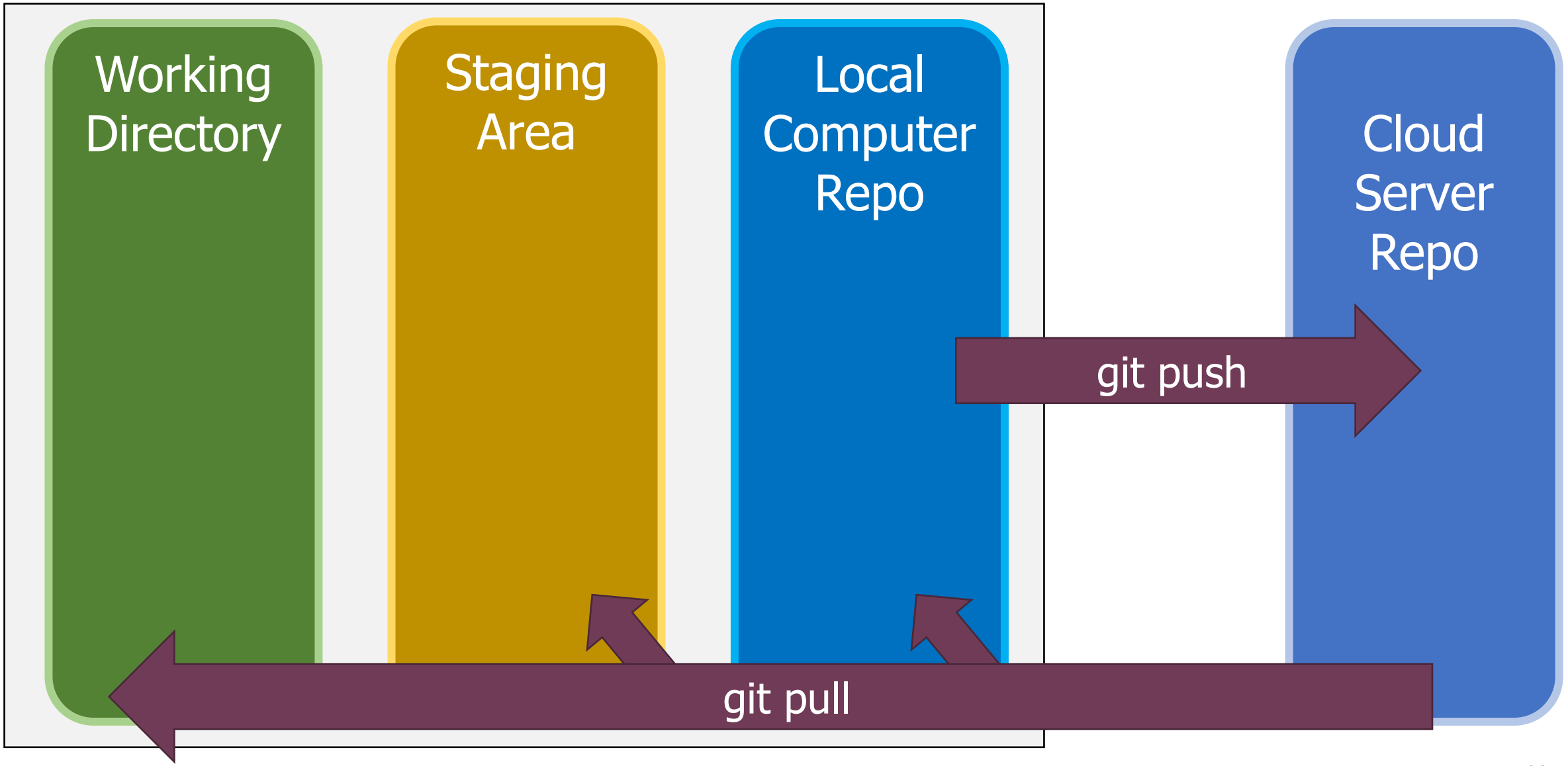
Cloud Server Repo

Git commands modify files in different areas

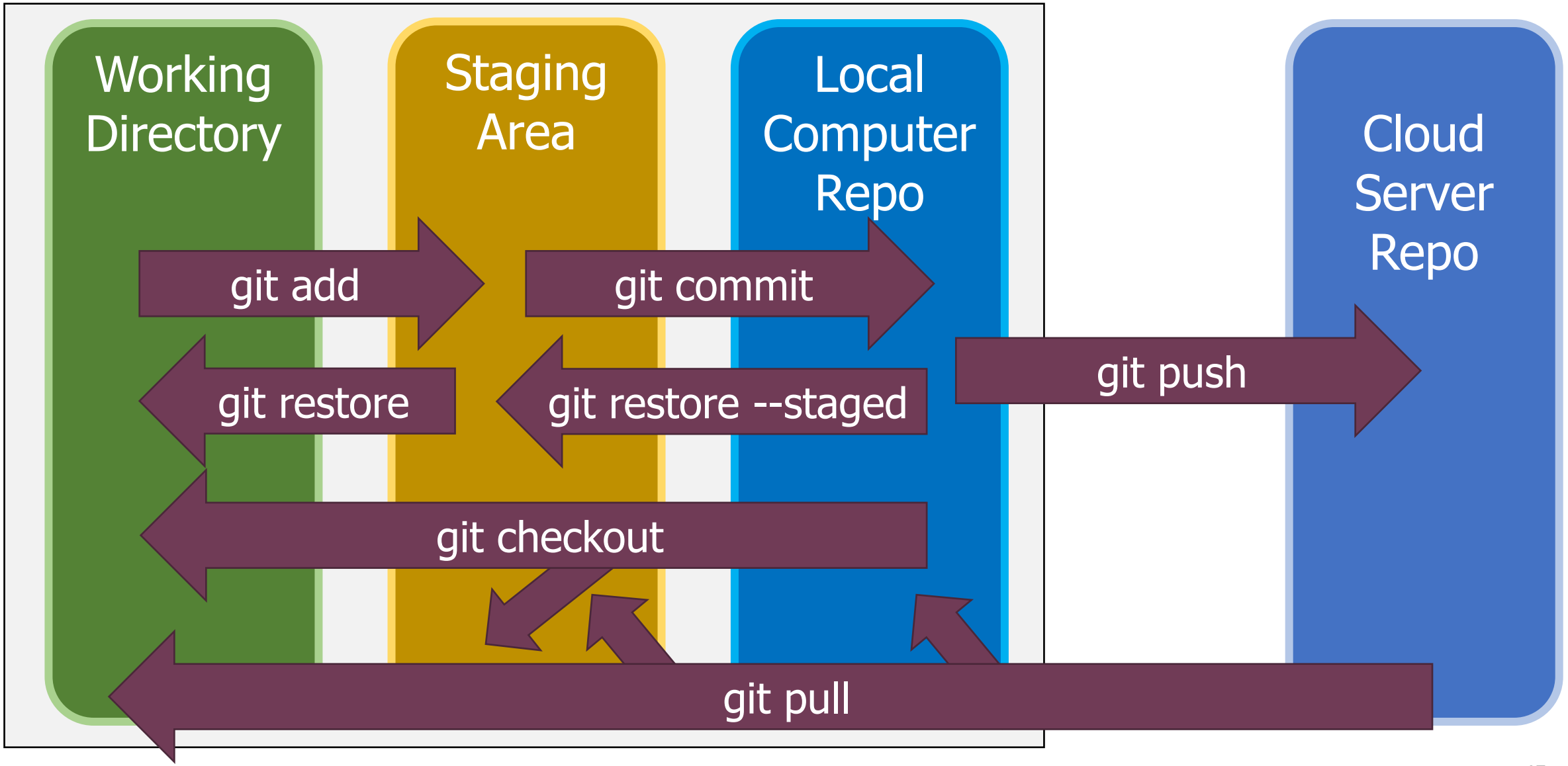


Cloud Server Repo

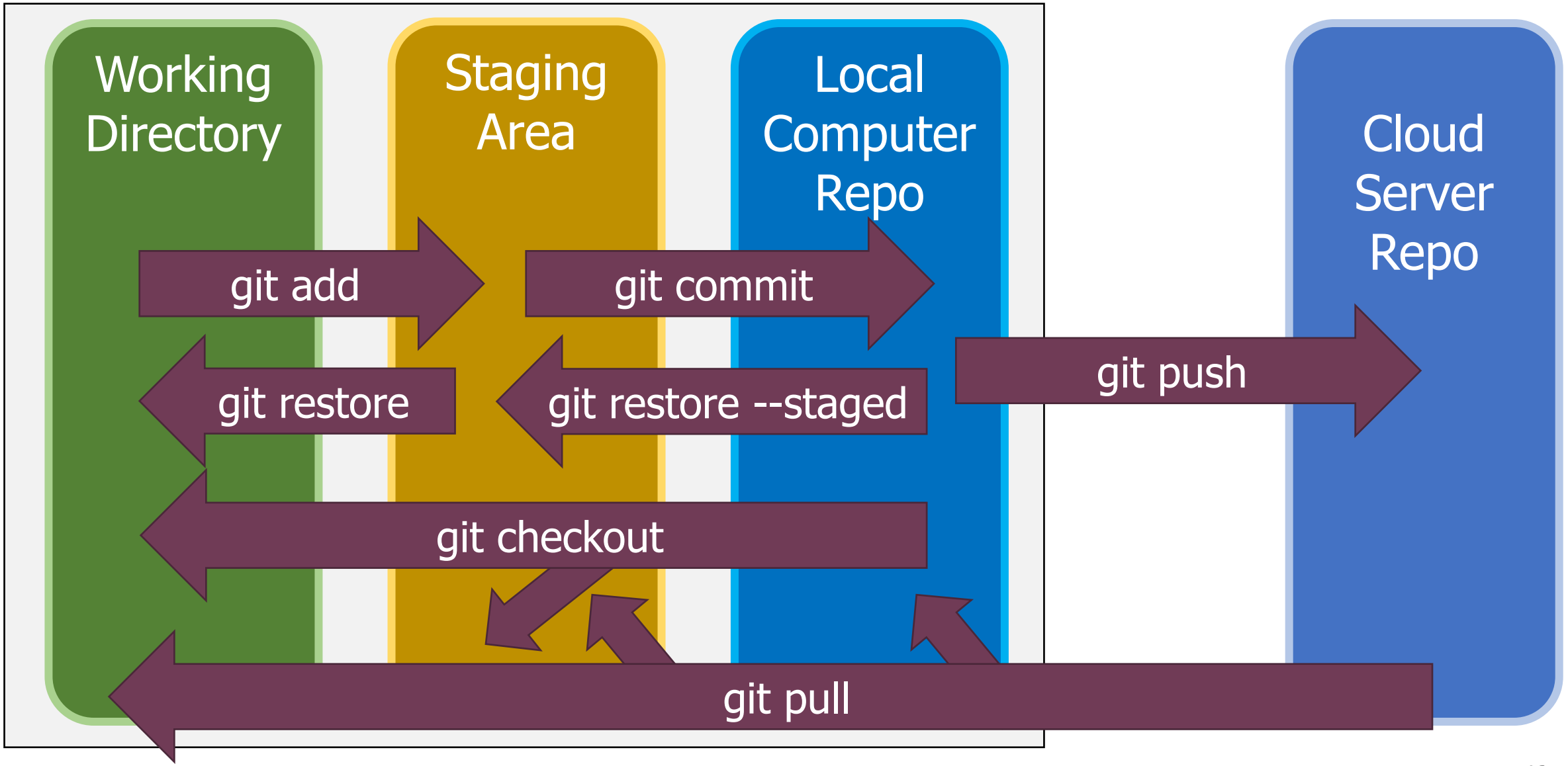
Git commands modify files in different areas



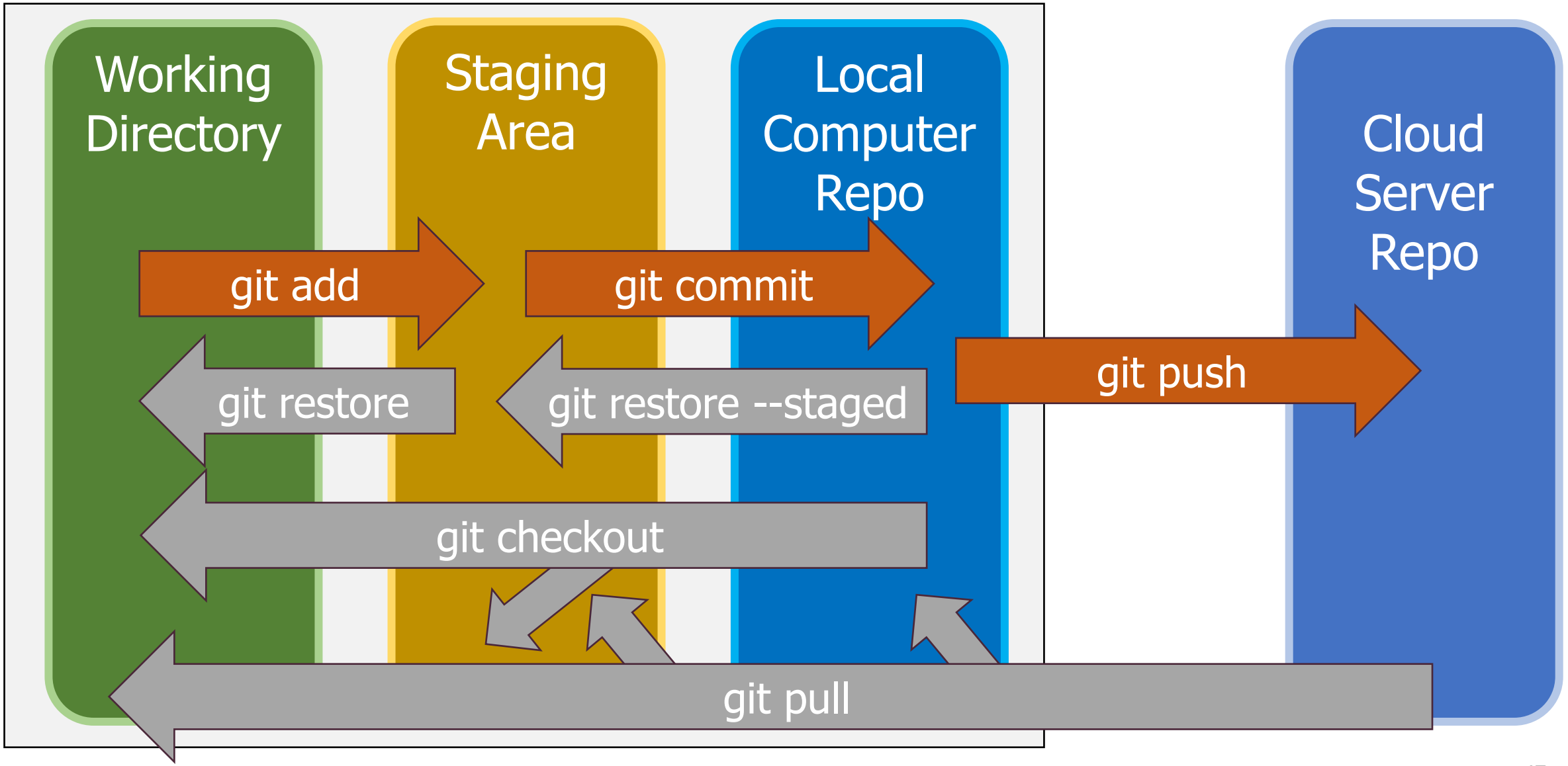
Overview of moving files



Break + Question: how would I share a new file with everyone?



Break + Question: how would I share a new file with everyone?



Keeping your file revisions safe

- Many of these commands will fail rather than overwrite changes in your working directory or staging area

- Exception: `git restore` will overwrite whatever is in your Working Directory with the version from the Staging Area

Other important commands

- `git clone`
  - Makes a local repo that's a copy of some remote server repo

- `git status`
  - Lists all modified Working Directory files
  - Lists all files currently in the Staging Area

- `git diff`
  - Lists all modifications from the Staging Area for all files
  - `git diff FILE` lists differences for a single file

Other important commands

- `git branch`
  - Creates a new branch with a parent of the current commit

- `git checkout`
  - Changes which commit or branch is the current one

- `git log`
  - Lists commit history previous to the current commit
  - `git log -N` lists details from the last N commits

Outline

- Version Control Overview
- Systems for Version Control

- Git Commit Structure
- Using Git
- **Using Github**
- Best Practices

What is Github?

- A website that hosts remote repos
  - Can be the one shared cloud repo that everyone pulls from and pushes to
  - Could just be a copy of that repo for public access

- There are many websites that provide this
  - Github
  - Bitbucket
  - Gitlab

- Often provide additional features as well
  - Viewing history of the repo in a GUI
  - Managing community input about a project

Public and Private Repos

- A repo hosted by Github can be Public or Private
  - Public repos are accessible by anyone
  - Private repos are accessible by only specified users

- Generally, want to make repos Public if possible
  - That's how you share neat work with others and build off their work

- Specifically, make sure any class files are in Private repos
  - Otherwise, you could be accused of academic dishonesty

Pull Requests

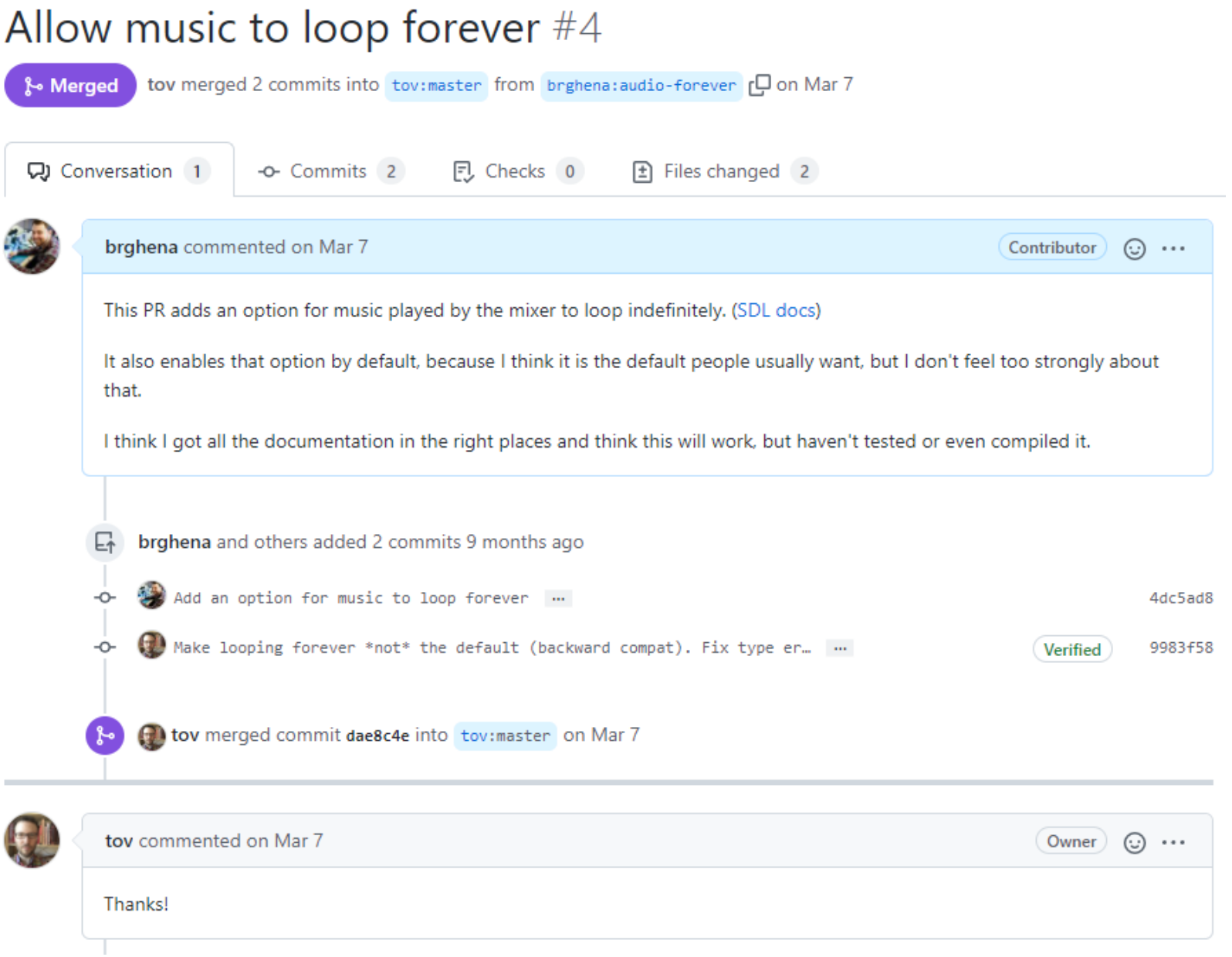
- Pull Requests are a feature of hosting websites
  - Literally: a request for the remote repo to pull from your copy

- Repo maintainers can review pull requests, comment on them, make changes, and eventually pull them to add the commits to their project

- These are how you contribute to open source projects
  1. Copy their repo
  2. Make changes to it
  3. Pull request so they can get your changes

Example Pull Request

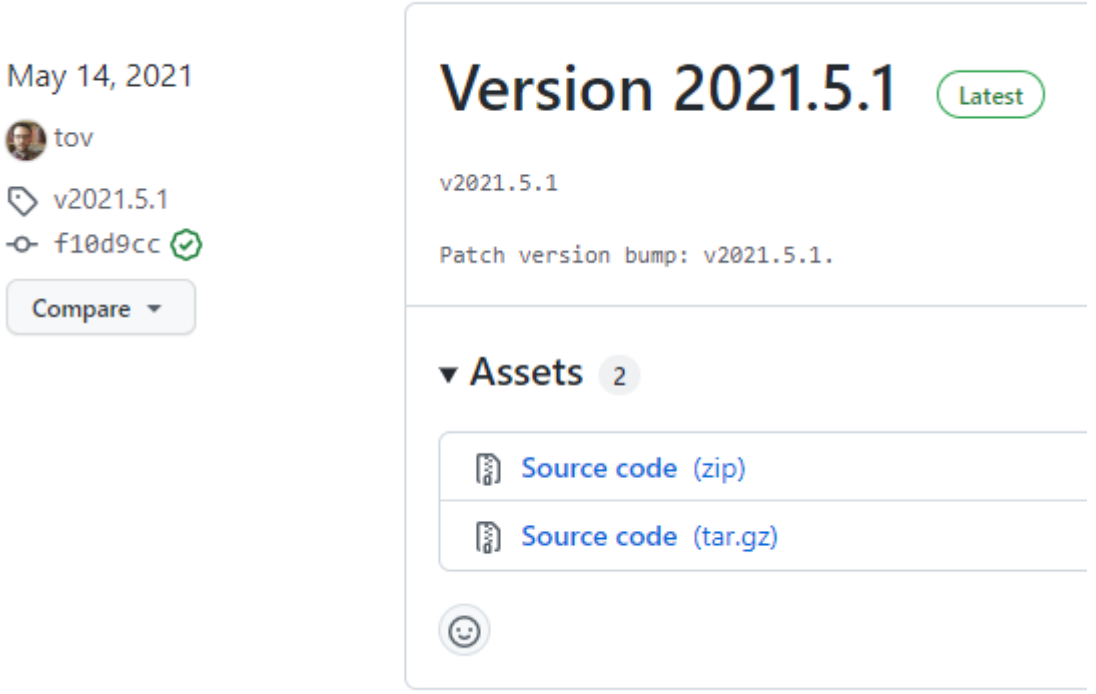
- Adds a commit and explains why it is useful
- Maintainer adds their own commit and then merges



https://github.com/tov/ge211/pull/4

Github also hosts releases

- A release is
  - A tagged commit
  - Plus the built files that you want to distribute
- Users can clone the repo and checkout the code for that particular release
- Or, they can just download the pre-compiled files



https://github.com/tov/ge211/releases

Outline

- Version Control Overview
- Systems for Version Control

- Git Commit Structure
- Using Git
- Using Github
- **Best Practices**

Pull often

- Especially when working on group projects
  - Don’t want to be working with old versions of the files
  - Might end up fixing the same bug someone else already did

- Helps to avoid conflicts
  - If you’re working on the most up-to-date version of a file, you’ll only conflict if someone else modifies it while you do

Commits should be a sensible unit

- Commits should include every file related to a change
  - And should NOT included files with unrelated changes

- The goal is that any commit is valid and compiles
  - Otherwise collaborators will get upset when they pull…

- Advanced users can re-write commits to combine or split them allowing them to fix this later
  - Git has all kinds of crazy features for rewriting local history before sending up to the remote repo

Don’t commit generated files

- Source code and build system should be committed

- Built artifacts should not be committed
  - .o files
  - Executables
    - Any user can regenerate them whenever they are needed

  - Non-text files don’t play well with version control
    - Can only detect if anything changed, not what
    - Often hard to compress

Use .gitignore files

- Enable you to list which files should NEVER be committed
  - Example from one of my repos:
    - build/
    - *.pdf
    - *.tgz
    - .DS_Store
    - *~
    - .idea/
- Definitely use these to keep accidents from happening

Don’t force anything

- If the version control system doesn’t let you do something, there’s usually a good reason

- Example: you cannot push commits that don’t align with the history of the branch in the remote repo
  - Because that would mess things up for anyone else using it

  - If you know that no one else is using it, then you can force push to overwrite the old commits with the new ones

Git demo

- https://github.com/brghena/git-example

- Clone the repo
  - Check the log of commits
  - Diff what changed in those commits

- Make some modifications
  - Demonstrate adding and restoring

- Commit files

- Push changes

Outline

- Version Control Overview
- Systems for Version Control

- Git Commit Structure
- Using Git
- Using Github
- Best Practices