Version Control

Idaho State University | Computer Science

Dr. Isaac Griffith

CS 2263
Department of Computer Science
Idaho State University

Outcomes

After today's lecture you will:

- Understand why we use version control

Thought Experiment

You are a part of a team of 7 developers, all working on the same project.

You and your co-worker Kim are working in the same part of the code.

- How do you ensure that she sees all your changes and you see all of hers?
- How do you ensure that if both of you change the same file, both changes are seen in the final end product?
- How do we share these changes between all of the team members, and within the larger organization?

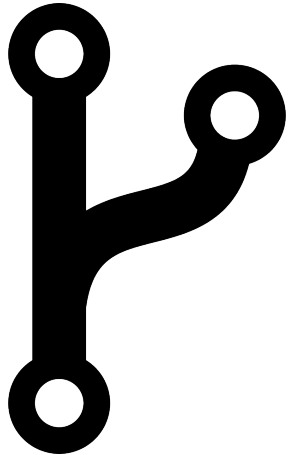
Why Version Control?

CS 2263

ROAR

Why Version Control

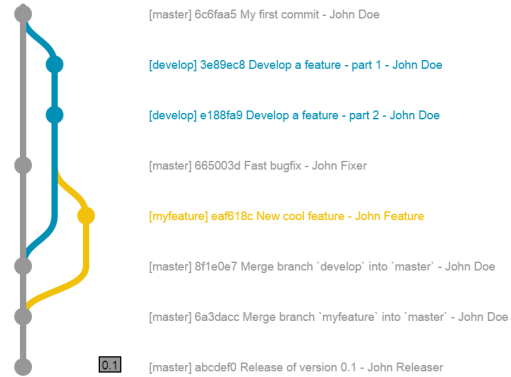
- New changes keep recurring
- We need a means to track the changes
- We need a mechanism to see what changed, overtime, and undo certain changes



Idaho State University
Computer Science

Keeping Track

- Making copies of the work
- Something that was removed only to be added later
- Keeping historical copies is elementary version control (primitive)
  - Who did what and why is lost



ROAR

Comparing Files

- Two copies of the same code from different times
  - Eyeball them?
  - **diff** command
  - **meld**, **kdiff3**, **vimdiff**
  - **patch**

Version Control

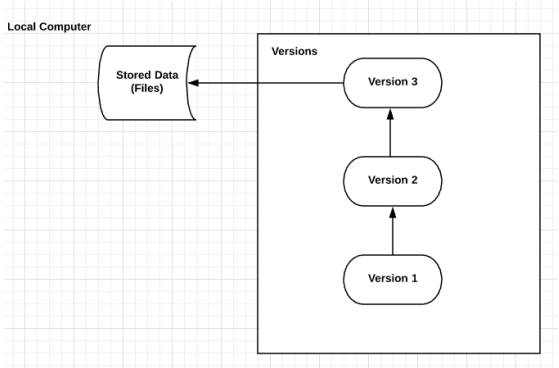
- Keeps track of all the versions
- Helps retrieve past versions and who changed the files and when
- Files are organized in repositories
- A repository can have thousands of contributors

Types of Version Control

- There are three common types of version control system
  - Local Version Control
  - Centralized Version Control
  - Distributed Version Control

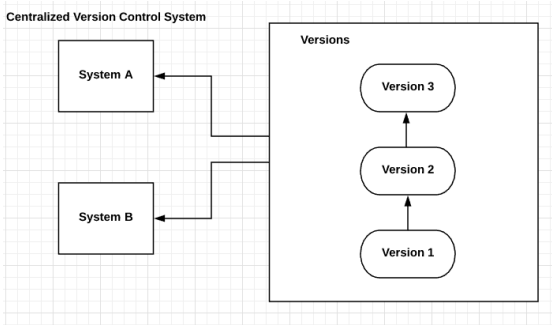
Local Version Control

- Tracks files within a local system
- Common and simple to use
- Error prone



Centralized Version Control

- All changes are tracked by a centralized server
- Clients check files out from the central server
- **Example:** Subversion



Distributed Version Control

- Each client maintains a complete repository with the full history
- If a server fails, clients can copy their repositories to help restore it
- Every clone is a full backup of the data

- **Example:** Git

Version Control Jargon

- **Repository:** Heart of a version control system.
  - Stores the files that are shared between users
  - Maintains history of each file
- **Trunk:** directory where all development takes place.
- **Tags:** snapshots of the project
  - provides a means by which versions can be named in the repo
- **Branches:** forks in the repository which create new lines of development

Version Control Jargon

- **Working copy:** the snapshot the repository where the developer is actually working on it.
  - Each developer has their own working copy.
  - Changes made to the working copy are eventually merged into the trunk.
- **Commit changes:** the act of storing changes from the working copy into the repository.
  - This is an atomic action and thus it either occurs or is rolled back, there are not partial changes in a repository.

Popular Systems

- SVN - Apache Subversion, a centralized version control system.
- Git - Distributed version control system emphasizing speed and performance.
- Mercurial - distributed version control system focused on ease of use, customization, and scalability.

Semantic Versioning

CS 2263

ROAR

Semantic Versioning

Version numbers for releases should follow the Semantic Versioning 2.0.0 approach:

- Each version number is specified as: MAJOR.MINOR.PATCH
- We increment:
  1. MAJOR version when you make incompatible API changes
  2. MINOR version when you add functionality in a backwards compatible manner
  3. PATCH version when you make backwards compatible bug fixes
- Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format

Project Documentation

CS 2263

Project Documentation

- Typically a GitHub project is documented in a few ways
  - Changelogs
  - Readme
  - Project Wiki
  - GitHub Pages
- We will discuss the first two, and I will leave the latter for your own discovery

Keeping a Changelog

- Normally keps as the file `CHANGELOG.md` in the project root folder
- A **changelog** is simply a file containing a curated ordered list of notable changes for each version of a project
- Provides documentation so that other contributors know what happened in the project
- All projects need a changelog

Changelog Guiding Principles

- Changelogs are for humans, not machines.
- There should be an entry for every single version.
- The same types of changes should be grouped.
- Versions and sections should be linkable.
- The latest version comes first.
- The release date of each version is displayed.
- Mention whether you follow Semantic Versioning.

Types of Changes

- `Added` for new features
- `Changed` for changes in existing functionality
- `Deprecated` for soon-to-be removed features
- `Removed` for now removed features
- `Fixed` for any bug fixes
- `Security` in case of vulnerabilities

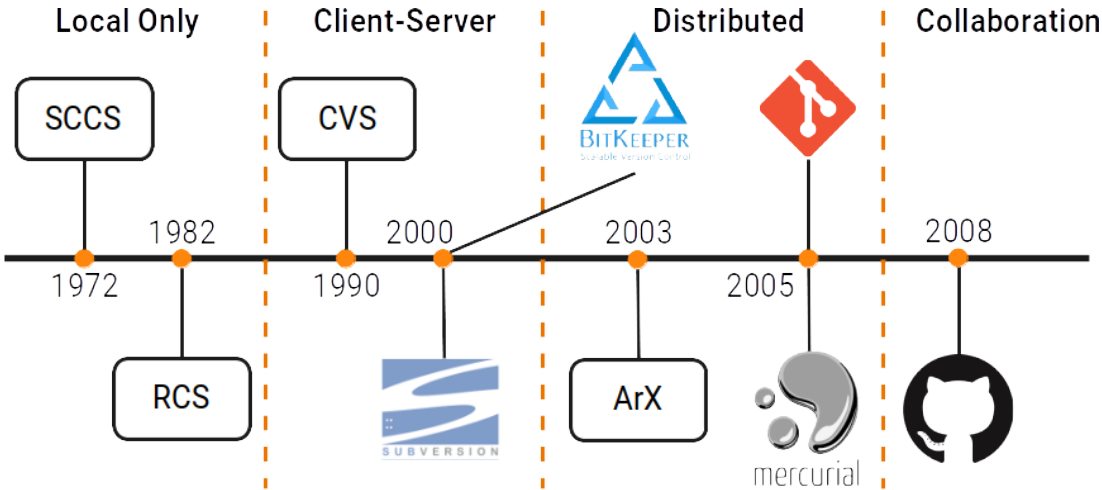
Reducing Effort

- You should keep a section titled `Unreleased` which tracks upcoming changes
- Serves two purposes:
  - Allows people to see changes that are expected in upcoming releases
  - Allows developers to simply move the `Unreleased` section to the next released version

Project README.md

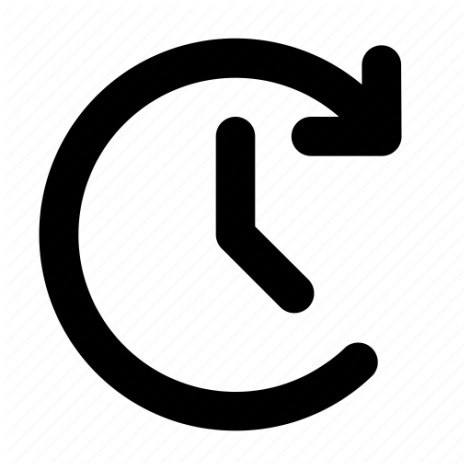
- Project readme's follow a specific format in order to immediately orient developers to the most important aspects of a project.
  - Normally kept as `README.md` in the project's root folder
- This format is as follows:
  - **Project Name** - the project name, and the first thing they will see
  - **Description** - A clear and concise description of the importance of your project and what it does
  - **Table of Contents** - Optional, but allows for quicker navigation
  - **Installation** - Informs users how to locally install your project (use pictures or an animated gif to improve)
  - **Usage** - Describes how to use the project once it has been installed (screenshots help)
  - **Contributing** - Describes how others may contribute to the project
  - **Credits** - Highlights and links to authors of the project
  - **License** - License of the project (may be a link to another file)

Version Control History



For Next Time

- Review the Git Book Chapter and Article
- Review this Lecture
- Come to Class
- Start Homework 02
- Read Git Book Chapter 2



Are there any questions?