# Course Introduction

Isaac Griffith

Fall 2021

Idaho State University | Software Engineering

SE 5520 - Software Construction and Configuration Management

## Outcomes

At the end of Today's Lecture you will be able to:

-

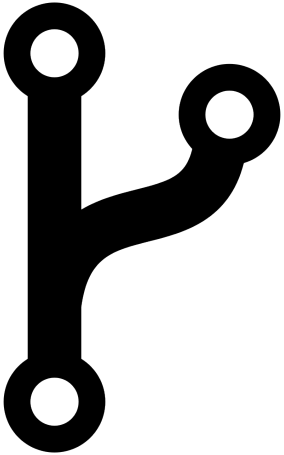
## Inspiration

## Why Version Control?

**SE 5520**

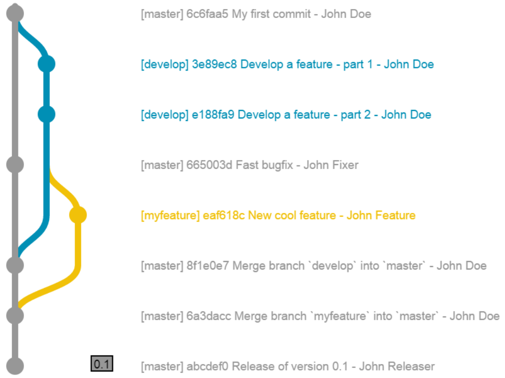
## Why Version Control

- 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



# Keeping Track

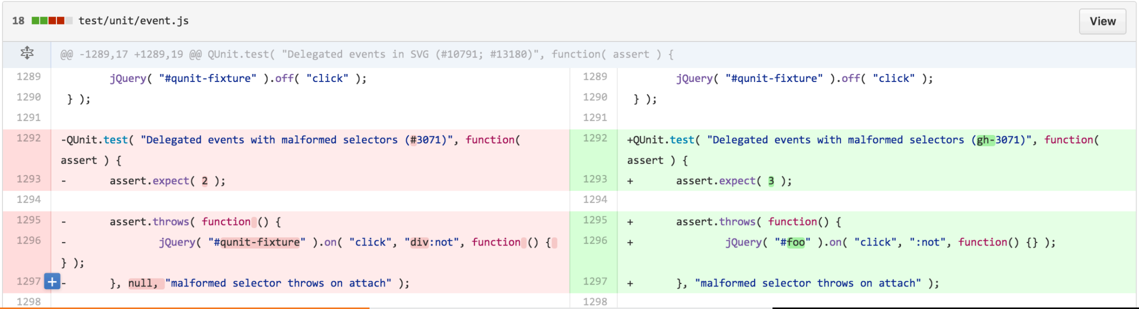
- Making copies of the work
- Something that was removed only to be added later
- Keeping historical copies is elementary version control (primitive)
  - Who did what and why is lost



[master] 6c6faa5 My first commit - John Doe
[develop] 3e89ec8 Develop a feature - part 1 - John Doe
[develop] e188fa9 Develop a feature - part 2 - John Doe
[master] 665003d Fast bugfix - John Fixer
[myfeature] eaf618c New cool feature - John Feature
[master] 8f1e0e7 Merge branch `develop` into `master` - John Doe
[master] 6a3dacc Merge branch `myfeature` into `master` - John Doe
[master] abcdef0 Release of version 0.1 - John Releaser

# Comparing Files

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



18 test/unit/event.js View

```
@@ -1289,17 +1289,19 @@ QUnit.test( "Delegated events in SVG (#10791; #13180)", function( assert ) {
```

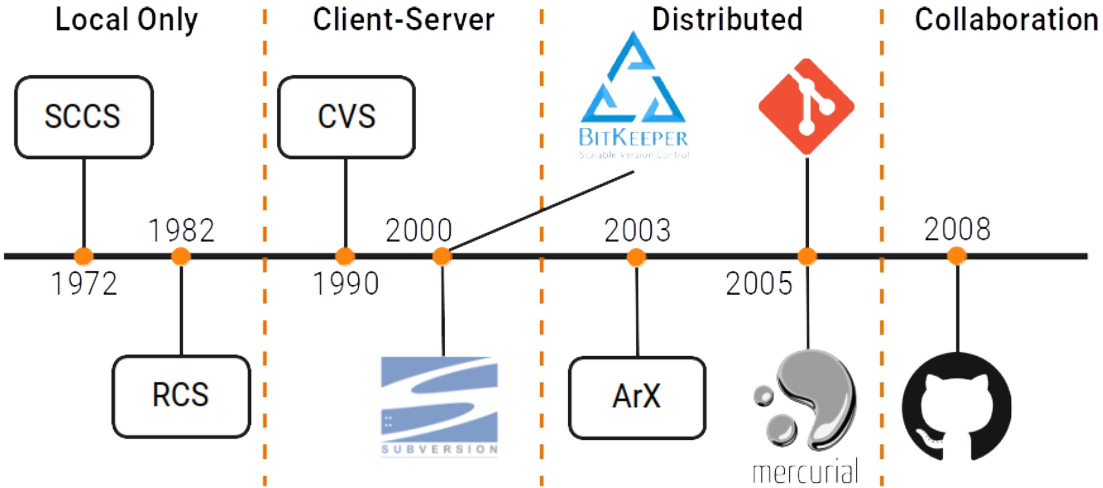
```
        jQuery( "#qunit-fixture" ).off( "click" );
 } );

-QUnit.test( "Delegated events with malformed selectors (#3071)", function( assert ) {
-       assert.expect( 2 );

-       assert.throws( function () {
-               jQuery( "#qunit-fixture" ).on( "click", "div:not", function () { } );
-       }, null, "malformed selector throws on attach" );
```

```
        jQuery( "#qunit-fixture" ).off( "click" );
 } );

+QUnit.test( "Delegated events with malformed selectors (gh-3071)", function( assert ) {
+       assert.expect( 3 );

+       assert.throws( function() {
+               jQuery( "#foo" ).on( "click", ":not", function() {} );
+       }, "malformed selector throws on attach" );
```

## 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

## Version Control History



## GitHub

SE 5520

## What is GitHub?

- GitHub.com is a site for online storage of Git repositories.
- Many open source projects use it, such as the Linux Kernel.
- You can get free space for open source projects or you can pay for private projects.

**Question**: Do I have to use github to use Git? **Answer**: No!

- you can use Git completely locally for your own purposes, or
- you or someone else could set up a server to share files, or
- you could share a repo with users on the same file system.

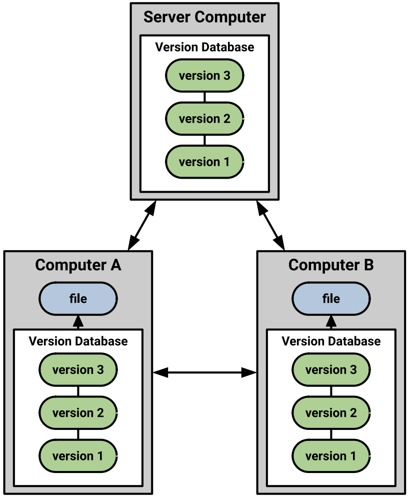
## Git

SE 5520

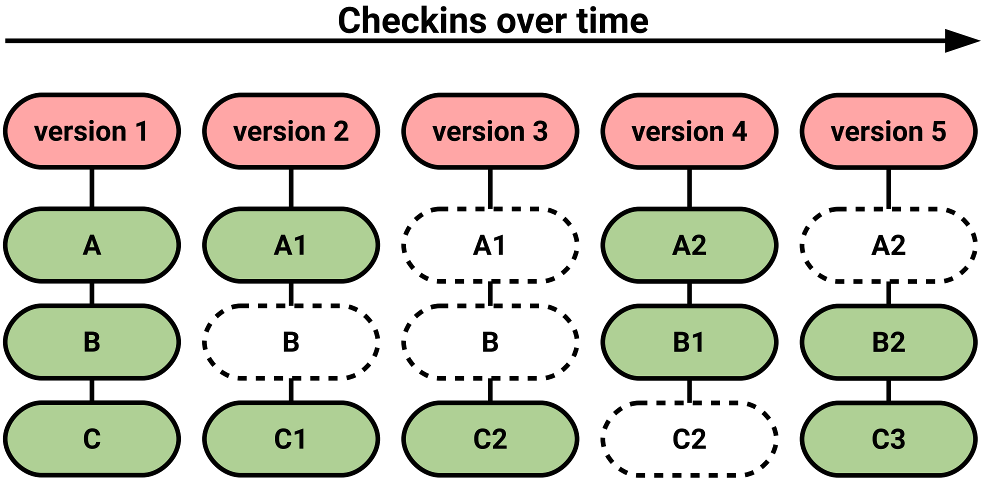
## Git History

- Came out of Linux development community
- Linus Torvalds, 2005
- Initial goals:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like Linux efficiently

## Git Uses a Distributed Model



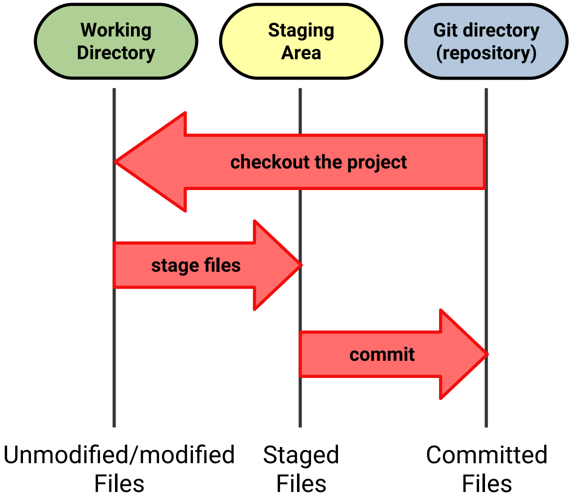
# Git Takes Snapshots



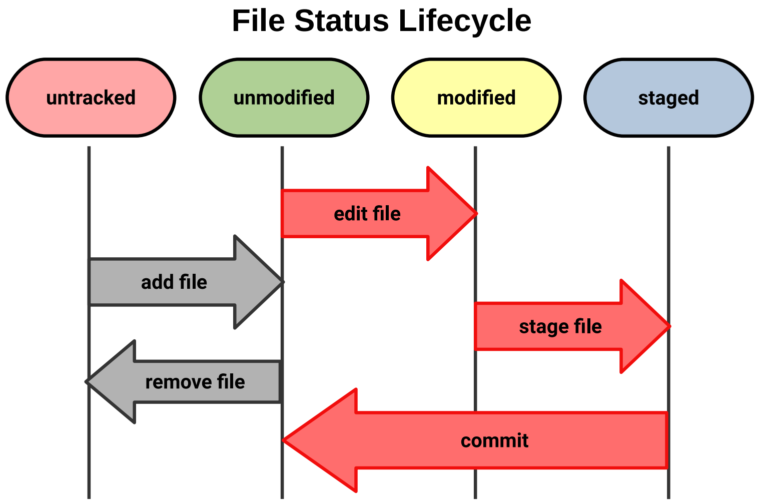
## Git Uses Checksums

- Git generates a unique SHA-1 hash for every commit
  - 40 character string of hex digits
- Refer to commits by this ID rather than a version number
- Often we only see the first 7 characters:
  - `1677b2d Edited first line of readme`
  - `258efa7 Added line to readme`
  - `0e52da7 Initial commit`

# Local Projects



# Git File Lifecycle



## Basic Workflow

Basic Git workflow:

1. **Modify** files in your working directory.
2. **Stage** files, adding snapshots of them to your staging area.
3. Do a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

- Notes:
  - If a particular version of a file is in the **git directory**, it's considered **committed**.
  - If it's modified but has been added to the **staging area**, it is **staged**.
  - If it was **changed** since it was checked out but has not been staged, it is **modified**

## Using Git

SE 5520

# Get Ready to Use Git!

1. Set the name and email for Git to use when you commit:

```
$ git config --global user.name "Your Name"
$ git config --global user.email youremail@whatever.com
```

- You can call `git config -list` to verify these are set.
- These will be set globally for all Git projects you work with.
- You can also set variables on a project-only basis by not using the `--global` flag.
- You can also set the editor that is used for writing commit messages:

`$ git config --global core.editor emacs` (it is vim by default)

## Create a Local Copy

❷ Two common scenarios: (only do one of these)

- To **clone an already existing repo** to your curent directory: `$ git clone <url> [local dir name]`
  This will create a directory named local dir name, containing a working copy of the files from the repo, and a **.git** directory (used to hold the staging area and your actual repo).
- To **create a Git repo** in your current directory: `$ git init`
  This will create a **.git** directory in your current directory. Then you can commit files in that directory into the repo:

```
$ git add file1.Java
$ git commit -m "initial project vesion"
```

## Git Commands

| command | description |
|---|---|
| `git clone` *`url`* *`[dir]`* | copy a git repository so you can add to it |
| `git add` *`files`* | adds file contents to the staging area |
| `git commit` | records a snapshot of the staging area |
| `git status` | view the status of your files in the working directory and staging area |
| `git diff` | shows diff of what is staged and what is modified but unstaged |
| `git help` *`[command]`* | get help info about a particular command |
| `git pull` | fetch from a remote repo and try to merge into the current branch |
| `git push` | push your new branches and data to a remote repository |
| others | `init`, `reset`, `branch`, `checkout`, `merge`, `log`, `tag` |

## Committing Files

- The first time we ask a file to be tracked, and **every time before we commit a file** we must add it to the staging area:

```
$ git add README.txt hello.java
```

This takes a snapshot of these files at this point in time and adds it to the staging area.

- To move staged changes into the repo we commit:

```
$ git commit -m "Fixing bug #22"
```

```
Note: To unstage a change on a file before you have committed it:
`$ git reset HEAD -- filename`
```

```
Note: To unmodify a modified file:
`$ git checkout -- filename`
```

**Note:** These commands are just acting on **your local version of repo**

## Status and Diff

- To view the **status** of your files in the working directory and staging area:

`$ git status` or `$ git status -s` (-s shows a short one line version)

- To see what is modified but unstaged:

```
$ git diff
```

- To see staged changes:

```
$ git diff --cached
```

# Viewing Logs

To see a log of all changes in your local repo:

- `$ git log`
- `$ git log --oneline` (to show a shorter version)

```
1677b2d Edited first line of readme
258efa7 Added line to readme
0e52da7 Initial commit
```

- `$ git log -5` (to show only the 5 most recent updates, etc.)

Note: changes will be listed by commitID #, (SHA-1 hash)

Note: changes made to the remote repo before the last time you cloned/pulled from it will also be included here

# Pulling and Pushing

Good Practice:

1. **Add** and **Commit** your changes to your local repo
2. **Pull** from remote repo to get most recent changes (fix conflicts if necessary, add and commit them to your local repo)
3. **Push** your changes to the remote repo

---

To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory

```
$ git pull origin master
```

To push your changes from your local repo to the remote repo:

```
$ git push origin master
```

Notes: `origin` = an alias for the URL you cloned from `master` = the remote branch you are pulling from/pushing to, (the local branch you are pulling to/pushing from is your current branch)

# Branching

To create a branch called experimental:

- `$ git branch experimental`

To list all branches: (* shows which one you are currently on)

- `$ git branch`

To switch to the experimental branch:

- `$ git checkout experimental`

Later on, changes between the two branches differ, to merge changes from experimental into the master:

- `$ git checkout master`
- `$ git merge experimental`

Note: `git log --graph` can be useful for showing branches.

Note: These branches are in your local repo!

## Do This:

1. `$ git config --global user.name "Your Name"`
2. `$ git config --global user.email youremail@whatever.com`
3. `$ git clone https://github.com/grifisaa/gitflowtest`

Then try:

1. `$ git log`, `$ git log --oneline`
2. Create a file named userID.txt (e.g., grifisaa.txt)
3. `$ git status`, `$ git status -s`
4. Add the file: `$ git add userID.txt`
5. `$ git status`, `$ git status -s`
6. Commit the file to your local repo: `$ git commit -m "added userID.txt file"`
7. `$ git status`, `$ git status -s`, `$ git log --oneline`

**WAIT, DO NOT GO ON TO THE NEXT STEPS UNTIL YOU ARE TOLD TO!!**

## Resources

- Pro Git - **Highly recommended reading**. Chapters 1-5 should teach you most of what you need to use Git proficiently.
- Oh Shit, Git!?! - short guide on how to recover from common git mistakes.
- Git for Computer Scientists - short explaination of git's data model
- Git from the Bottom Up - detailed explanation of git's implementation, for the curious
- How to explain git in simple words
- Learn Git Branching - a browser-based game that teaches you git.

## Summary

## For Next Time

## Are there any questions?