# HOW TO USE GIT

James E. Stine, Jr.
Edward Joullian Endowed Chair in Engineering
Oklahoma State University
Electrical and Computer Engineering Department
Stillwater, OK 74078 USA
james.stine@okstate.edu

# About Git

- Created by Linus Torvalds, creator of Linux in 2005
  - Came out of Linux development community and his dissatisfaction that there was no good open-source tool (actually, read more of the history!)
  - Designed to handle version control on Linux
- Goals of git
  - Speed
  - Support for non-linear development (could be thousand of branches)
  - Fully distributed
  - Able to handle large projects efficiently
- Etymology
  - A “git” is a cranky old man and Linux was alluding to himself in a funny sort of way.

# 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 Scientists:
    - http://eagain.net/articles/git-for-computer-scientists/
- At command line: (where verb = config, add, commit, etc.)
  - `git help verb`
- https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
  - You may need to add a key to your GitHub site
    - Create a GitHubID at http://www.github.com
    - https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

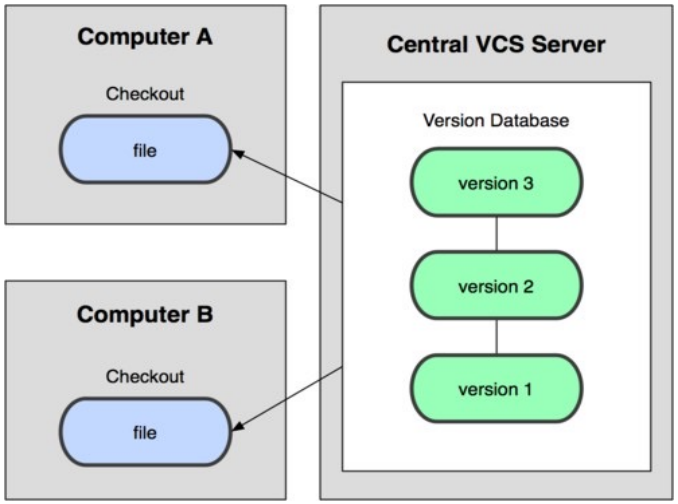
# ”Here’s the story…”

- `git` is not the first program to handle version control
  - Version control is nothing more than being able to work on a computer project within a team.
- Some former version control programs
  - RCS
  - SVN (sub-version)
- All of these tools really help people collaborate on a project as well as annotate each-other’s code.
- In reality, the best tool here is one where all parties contribute!
  - However, I believe `git` has evolved to be a better program than previous versions.
  - There are also web interfaces:
    - http://github.com
    - http://gitlab.com


[Paramount Home Entertainment]

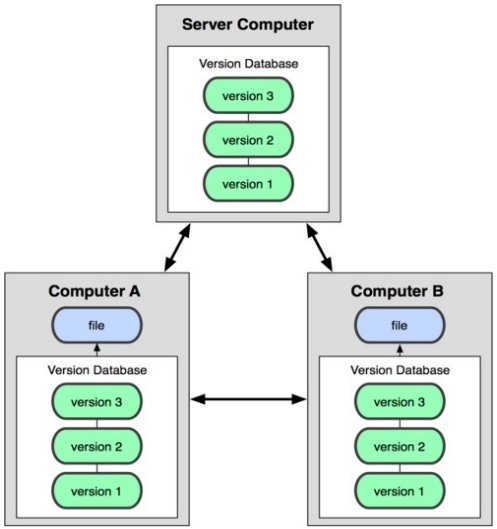
# Centralized VCS

- In subversion, CVS, Perforce, …
  - A central server repository (repo) holds the “official copy” of the code
  - The server maintains the sole version of the repo!
- You make ”checkouts” of it to your local copy
  - You make local modifications
  - Your changes are not versioned
- When you are done, you “check in” back to the server
  - Your ”check in” increments the repo’s version



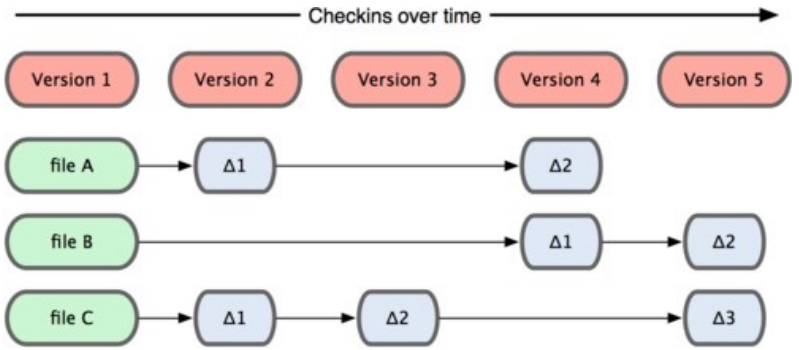
# Distributed VCS (Git)

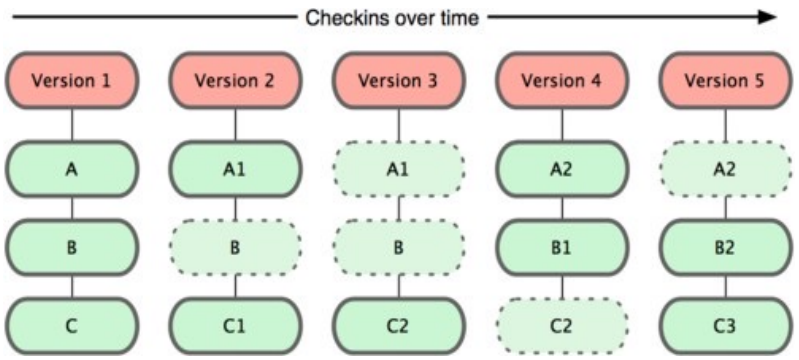
- In git, mercurial and more modern repos, you do not ”check out” from a central repo
  - You ”clone” it and “pull” changes from it.
  - Your local repo is a complete copy of everything on the remote server.
- Many operations are local:
  - Check in/Check out to local repo
  - Commit changes to local repo
  - Local repo keeps version history
- When you are ready, you can “push” changes back to the server



# Git snapshots

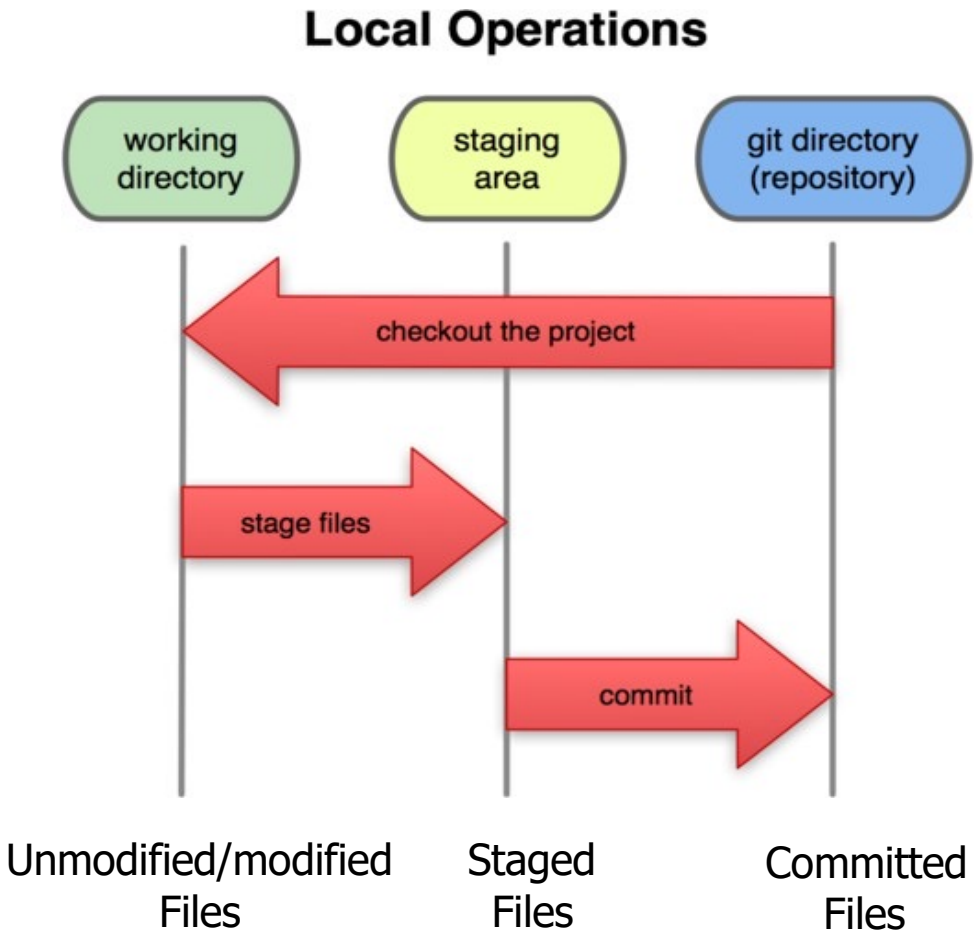
- Centralized vcs like subversion track version data on each individual file.
- Git keeps ”snapshots” of the entire state of the project.
  - Each check in version of the overall code has a copy of each file in it
  - Some files change on a given check in, some do not
  - More redundancy (space), but much faster and more efficient (imho).



# Local git areas

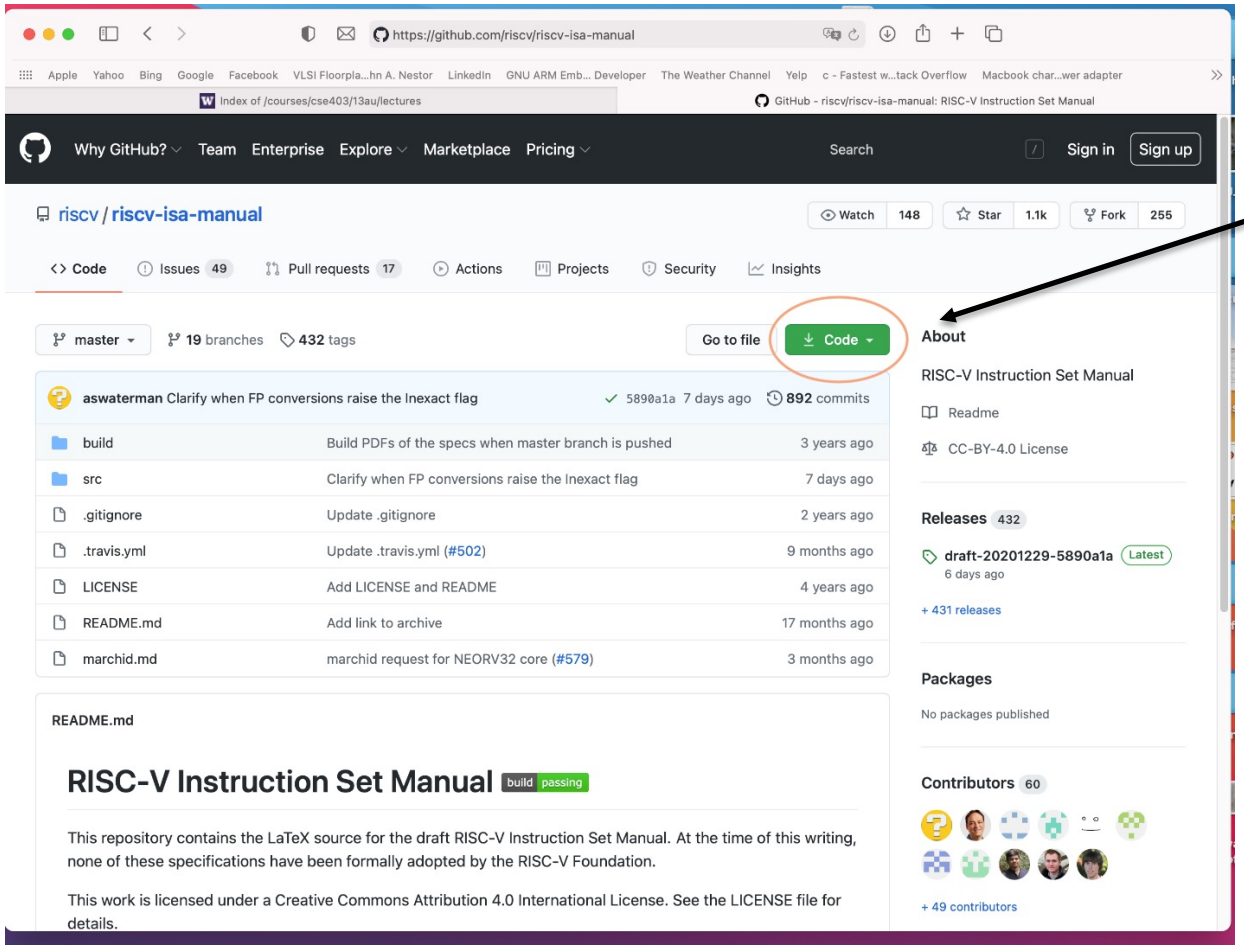
- In your local copy on git, files can be:
  - In your local repo (i.e., committed)
  - Checked out and modified, but not yet committed (i.e., working copy)
  - Or, in-between, in a ”staging” area (i.e., staged files are ready to be committed)
    - A commit saves a snapshot of all staged files!



# Example

- Find a repo you wish to clone
  - This can be a version stored in a directory somewhere
    - Directly copying the repo can be dangerous, as if you modify any permissions it wipes out data information and thus ruins the repo.
  - It is far easier to pull it from a web-based repo (with security in mind, of course)
- Here is an example of the RISC-V documentation:
  - https://github.com/riscv/riscv-isa-manual
  - It is also easier to perform a clone at the terminal than using some form of GUI
- Most online repos will have a "CODE" button that gives you the location that you can insert into a clone command:
  - `git clone https://github.com/riscv/riscv-isa-manual.git`

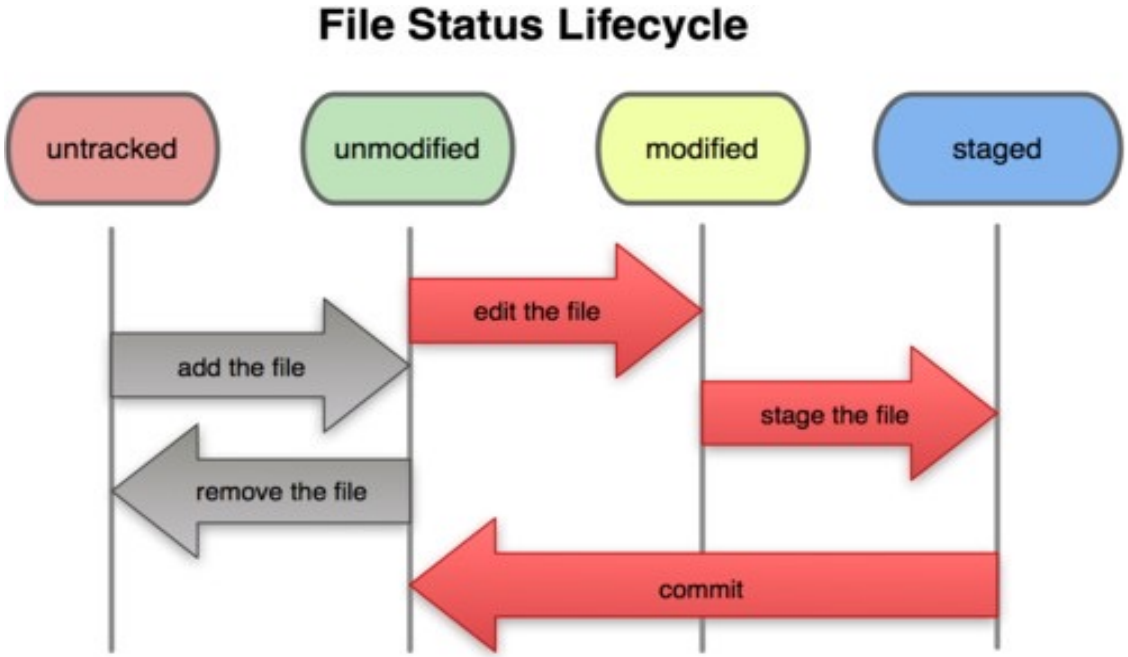
# github site




Click here to see clone option

# Git procedure on adding/modifying

1. Modify files in your working directory
2. Stage files, adding snapshots of them to your staging area
3. Commit, which takes the files in the staging area and stores that snapshot permanently to your git directory



# How does git know where it is?

- The original idea in subversion is to increment the version # of the overall repo.
  - In git, each user has their own copy of the repo and commits changes to their local copy of the repo before pushing it to the central server
  - So, git generates a unique SHA-1 hash (40-character string of hex digits) for each commit.
  - Refers to commits by this ID rather than a version number
- However, sometimes you will only see the first 7 characters of this hash.
  - `1677b2d Edited first line of readme`
  - `258efa7 Added line to readme`
  - `0e52da7 Initial commit`
- You can see the most recent hash by the following command:

```
jstine@iq9:riscv-o3$ git rev-parse HEAD
f87b48ebd6c67ad0f8c842ecef0da6f60dedef07
```

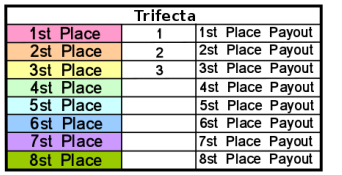
# Initial Git configuration

- Set the name and email for Git to use when you commit:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs.bunny@gmail.com`
  - You can call `git config --list` to verify these are set.
- Set the editor that is used for writing commit messages (set by EDITOR environmental variable by default):
  - `git config --global core.editor nano`
  - The editor is important when you commit
    - `git commit` (This will invoke your editor to allow you more room to give more information on your commit message)
    - `git commit -m "Message"` (This will bypass invoking the editor)
- The config command is also useful to see the original clone location of the repository.
  - `git config --list`

# Creating a git repo

- There are two common scenarios (highly recommended)!
- To create a new local git repo in your current directory
  - `git init`
  - This will create a .git directory in your current directory that could cause some Unix permission issues with multiple users.
  - Then you can commit files in that directory into the repo.
  - `git add filename`
  - `git commit –m “commit message”`
  - `git push`
- The better option as indicated previously is to clone a remote repo to your current directory
  - `git clone url  localDirectoryName`
  - This will create the given local directory, containing a working copy of the files from the repo, and a .git directory (used to hold the staging area and your actual local repo)

# Trifecta of commands



| Trifecta | | |
|---|---|---|
| 1st Place | 1 | 1st Place Payout |
| 2st Place | 2 | 2st Place Payout |
| 3st Place | 3 | 3st Place Payout |
| 4st Place | | 4st Place Payout |
| 5st Place | | 5st Place Payout |
| 6st Place | | 6st Place Payout |
| 7st Place | | 7st Place Payout |
| 8st Place | | 8st Place Payout |

[Wikipedia]

- There is a trifecta of commands that makes things easier when you want to move your staged stuff into a commit.
- I try to remember these but its good before you commit to see if anyone else has updated otherwise you may need to manually merge changes (later)
  - To get the most recent version of your repo, first, just ”pull” it
  - `git pull`
- Trifecta of commands!
  - `git add file`
  - `git commit –m “Commit message”`
  - `git push`
- It is highly advisable to add as much information about what you are committing in the commit message to help the next user.
  - Do not create a message like, “Update to files” – make it detailed!

# Common Commands

| command | description |
|---|---|
| `git clone` ***`url [dir]`*** | copy a git repository so you can add to it |
| `git add` ***`file`*** | 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` | |

# Another useful command

- A useful command I use all the time is indicative of what the most recent update to a repository is and who did it
- It also gives the most recent hash!
- `git log --stat`

```
jstine@iq9:riscv-isa-manual$ git log --stat
commit 5890a1a702abf4157d5879717a39d8ecdae0de68 (HEAD -> master, tag: draft-2020
1229-5890a1a, origin/master, origin/HEAD)
Author: Andrew Waterman <andrew@sifive.com>
Date:   Mon Dec 28 18:19:44 2020 -0800

    Clarify when FP conversions raise the Inexact flag

 src/f.tex | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)

commit 66dc1729544fd2e80390ba3dd1937b10a6977903 (tag: draft-20201229-66dc172)
Author: Andrew Waterman <andrew@sifive.com>
Date:   Mon Dec 28 18:07:58 2020 -0800

    Use consistent wording for FP exception text

 src/f.tex | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 42dc13a64887553e1575f65f135d8ce89ddae643 (tag: draft-20201222-42dc13a)
Author: Paul Donahue <48959409+pdonahue-ventana@users.noreply.github.com>
Date:   Tue Dec 22 12:51:22 2020 -0800
```

# Added Note : a hack

- Sometimes, I do not remember what I modified, and things go awry.
- This happens and what I find is best is the following:
  - Move your directory to another name somewhere else
  - Re-clone the repository
  - Either inspect what you did or put back changes into the new clone
- Once you do this, you can commit to the repository and be assured you did not kill something.
- Delete the old directory when you are done.
- This is only a suggestion, but it tends to work for my issues and is probably not the most elegant of solutions.

# Some other commands

- To view status of files in working directory and staging area
  - `git status`
- To see what is modified but unstaged
  - `git diff`
- To see a list of staged changes
  - `git diff --cached`
- To see a log of all changes in your repo
  - `git log`
  - `git log -5` (to show only 5 most recent updates)
  - `git log --stat` (to show which files have changed)
  - `git log -p` (to show what file contents have changed)

# Branching and Merging

- To create a new local branch
  - `git branch branch_name`
- To push a new local branch to the remote, allowing others to access it
  - `git push -u origin branch_name`
- To list all local branches (* = current branch)
  - `git branch -a`
- To switch to a given local branch
  - `git checkout branch_name`
- To merge changes from a branch into the local master
  - `git checkout master`
  - `git merge branch_name`

# Conclusion

- Git is incredibly easy to use although its far simpler at a command line than any web interface.
- Remember the trifecta of commands
- When making an initial git repository, it is far simpler to make it on a web page (e.g., `github.com`), pull the empty repository, add files manually, and then commit them.
- With practice, it will seem quite easy
  - But it requires practice!