University of Colorado
Anschutz Medical Campus

Git: Distributed Version Control

An Introduction

Peter E. DeWitt, Ph.D. & Seth Russell, M.S.
Department of Biomedical Informatics
10 NOVEMBER 2022

Outline

Why? and little bit of How?

Welcome: Why Git?

Getting Started

Basic Git Use

Branching

Closing

Acknowledgements and Other Resources

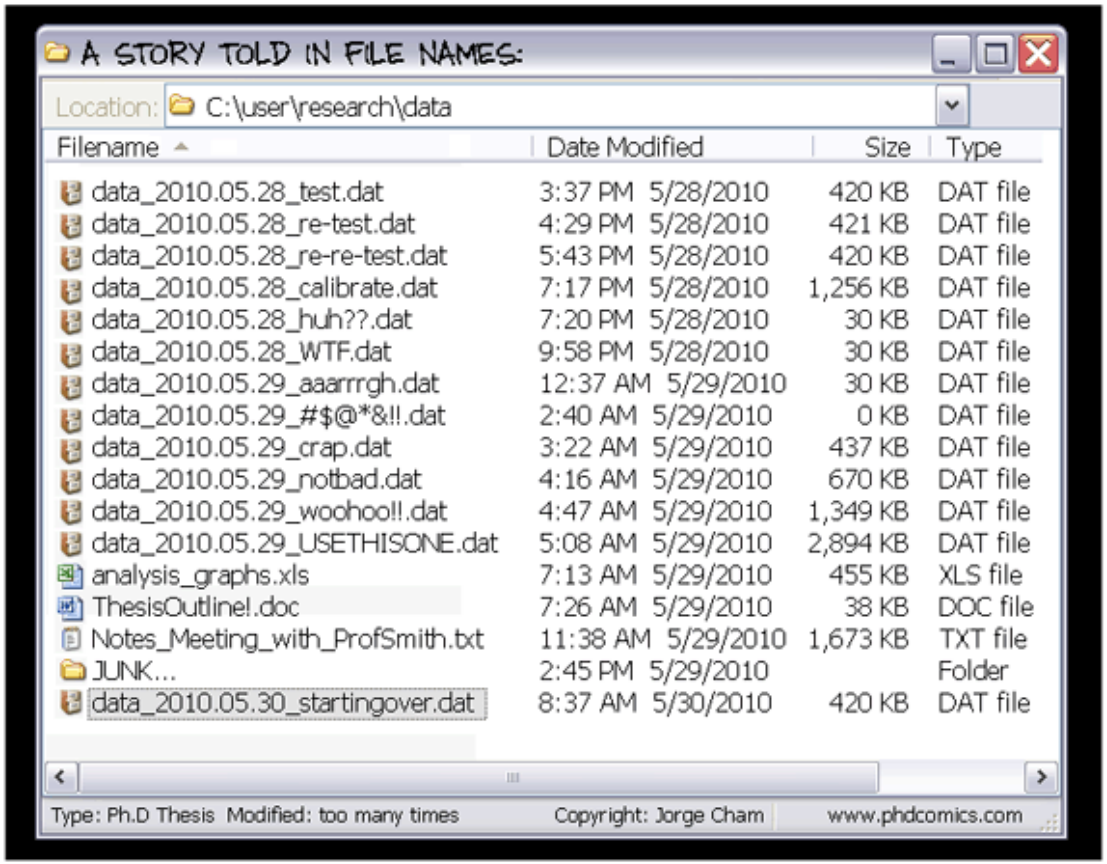
- Primary website: `http://git-scm.com`
- The first part of this presentation was inspired by, and copies from, 'Git and Github' a webinar presented by Hadley Wickham.
- **FREE** book: *Pro Git* 2nd edition, best possible single source reference. `http://git-scm.com/book/en/v2`
- This presentation highlights the first three chapters from *Pro Git*.
- A branching model I subscribe to: `http://nvie.com/posts/a-successful-git-branching-model/`
- References and Cheat-sheets: `https://git-scm.com/docs`
- Git Videos: `http://git-scm.com/videos`

Git

- Can be difficult and frustrating to learn.
- Can be incorporated into your workflow.
- Your workflow may improve as you adapt to the git paradigm.
- Part of reproducible reporting.
- Payoffs: **Safety** and **Community**.
- Works okay for binary files
  - .docx, .pdf, .xlsx, ...
- Works very, very, very, well for flat files:
  - .sas, .tex, .md, .txt, .R, .Rnw, .Rmd, .html, .c, .cpp, .js, .java, .csv, .dat, ...

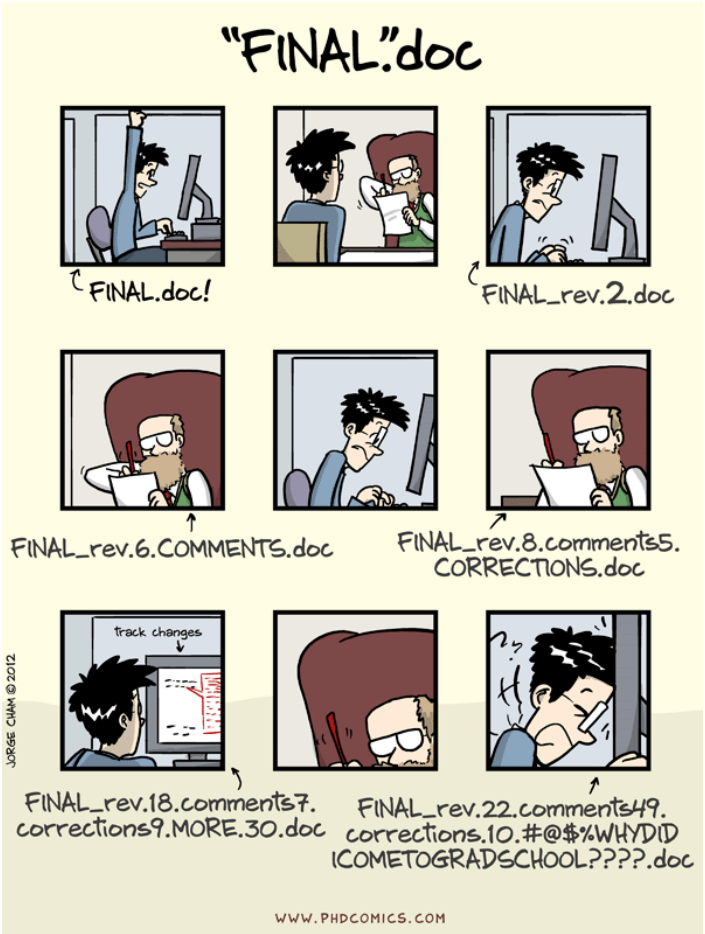
"Failures, repeated failures, are finger posts on the road to achievement. One fails forward towards success." – C.S. Lewis

A Story in Pictures



http://www.phdcomics.com/comics/archive.php?comicid=1323

A Story in Pictures



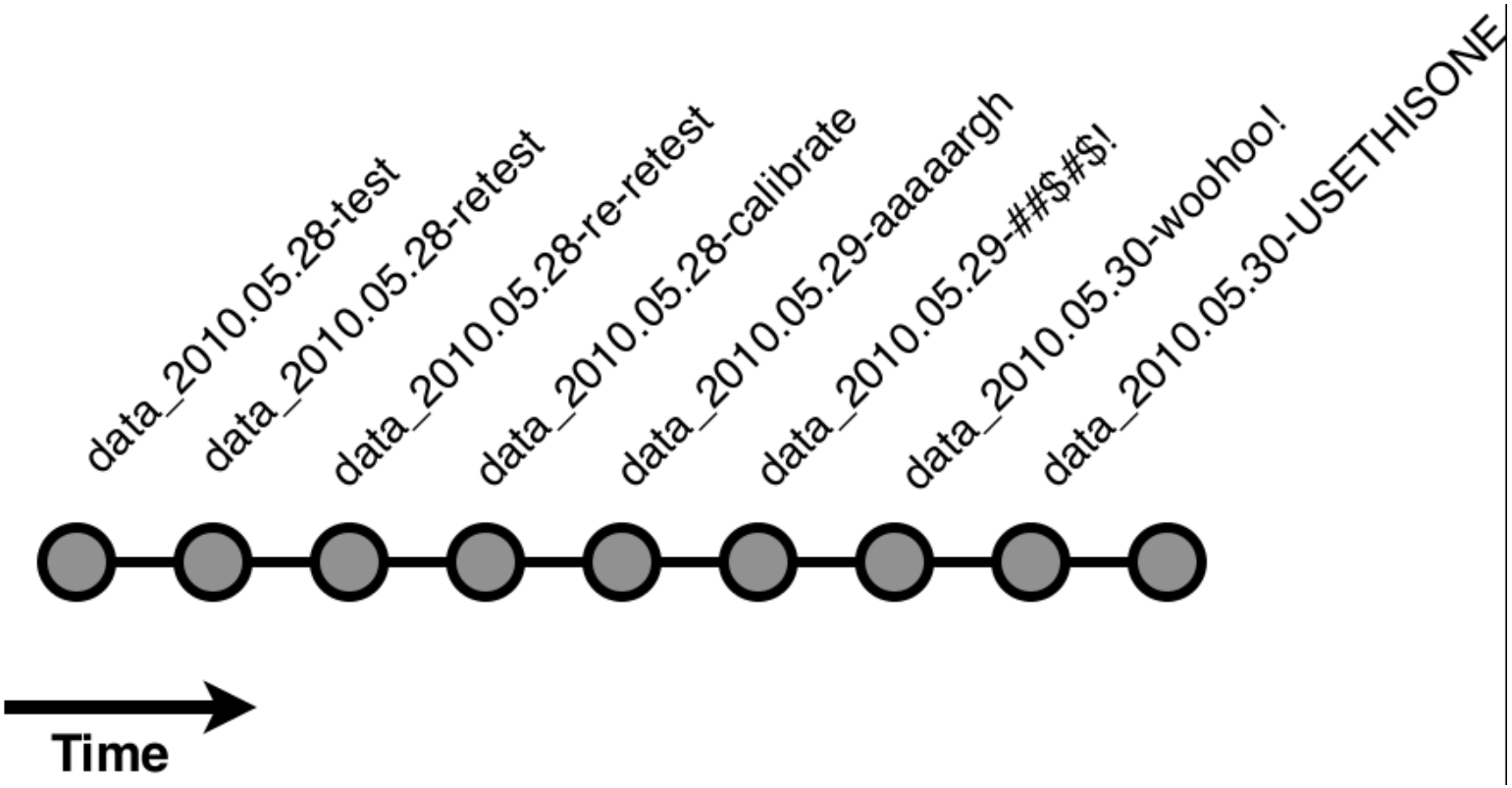
http://www.phdcomics.com/comics/archive.php?comicid=1531

A Story in Pictures

http://xkcd.com/1296/

Changes

The history of a project can be viewed as a series of changes:



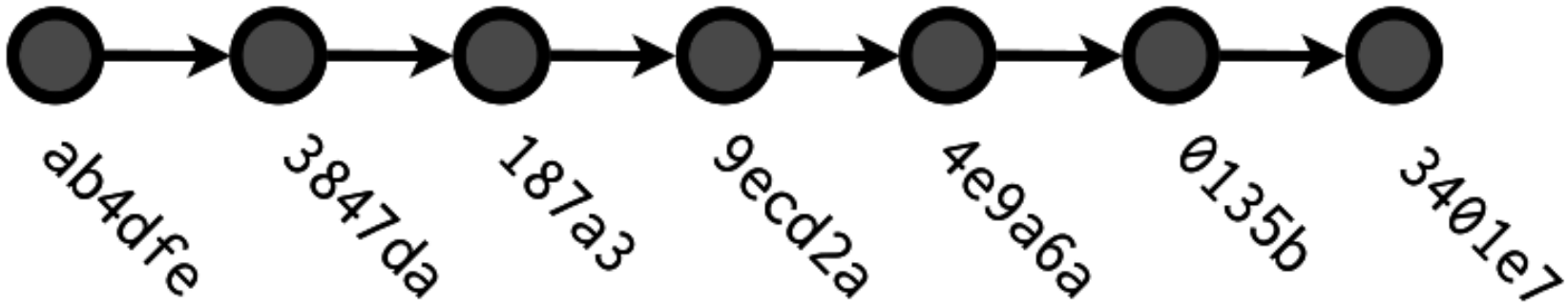
Changes

- A unique identifier
- What changed?
- When did it change?
- Who changed it?
- Why did it change?

- Difficult to coordinate with multiple files

Changes

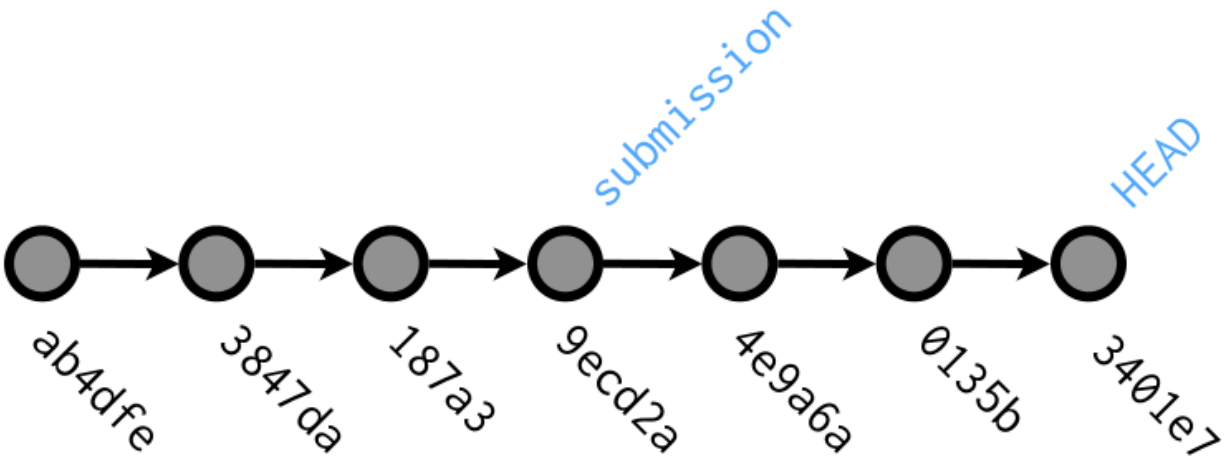
With git, each change (**commit**) is given a unique identifier, a **sha**.



- sha is a 40 digit long hexadecimal value.
- sha is a key into a database that provides the author, date, and a description of the changes.
- The sha is unique to the whole history of the repository.

Changes

You can also name individual commits.

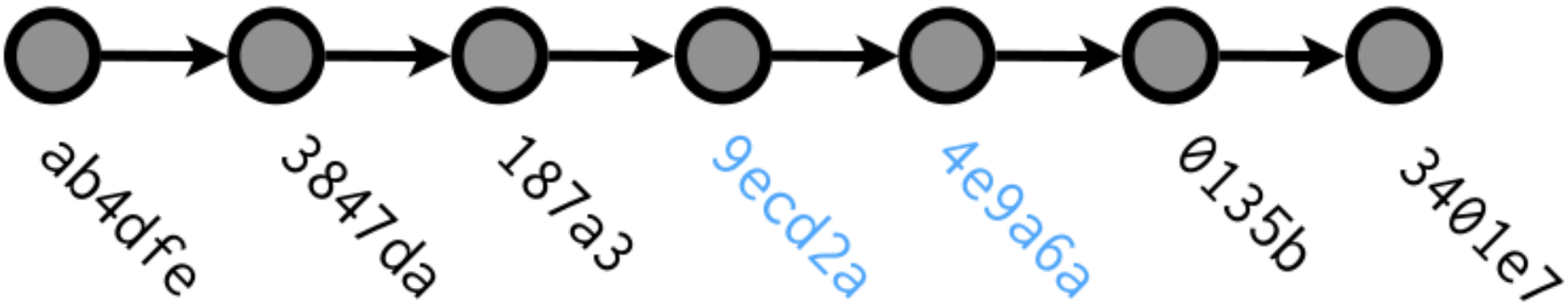



```
git tag submission 9ecd2a
```

- `HEAD` is the location the working directory is set to.
- `submission` is an example of a **tag**

Changes

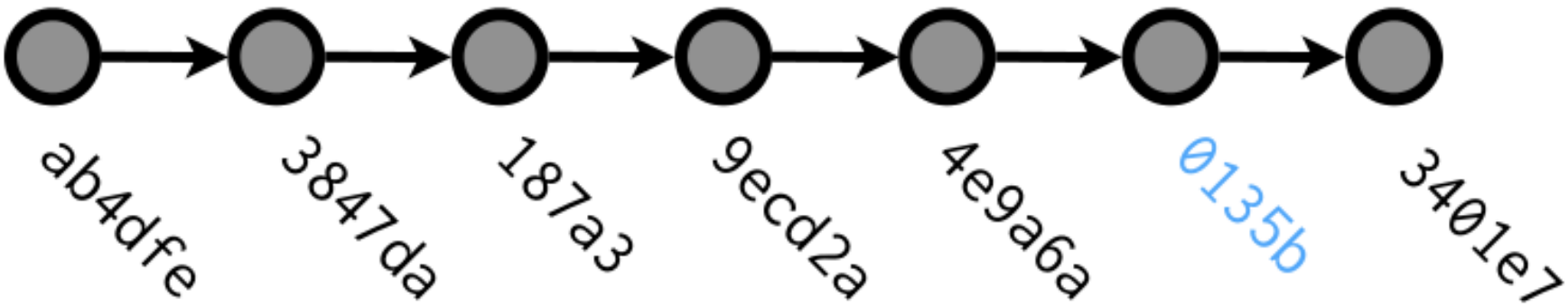
Then see exactly what was changed




```
git diff 9ecd2a..4e9a6a
```

Changes

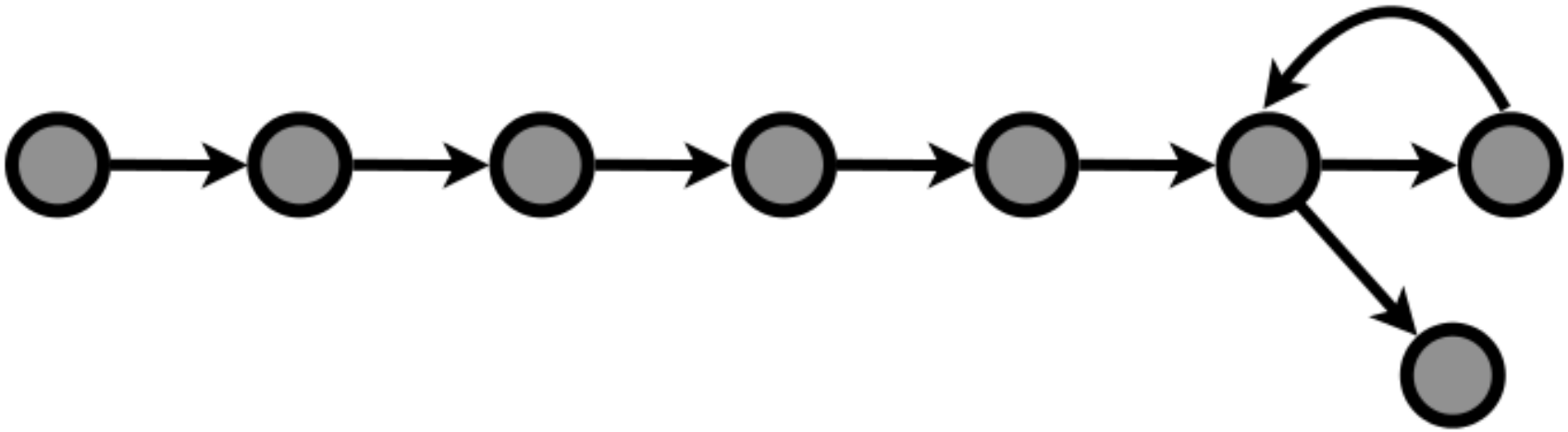
You can revert to a previous change with git checkout




```
git checkout 0135b
```

Changes

That allows you to undo mistakes



Outline

Why? and little bit of How?

Welcome: Why Git?

Getting Started

Basic Git Use

Branching

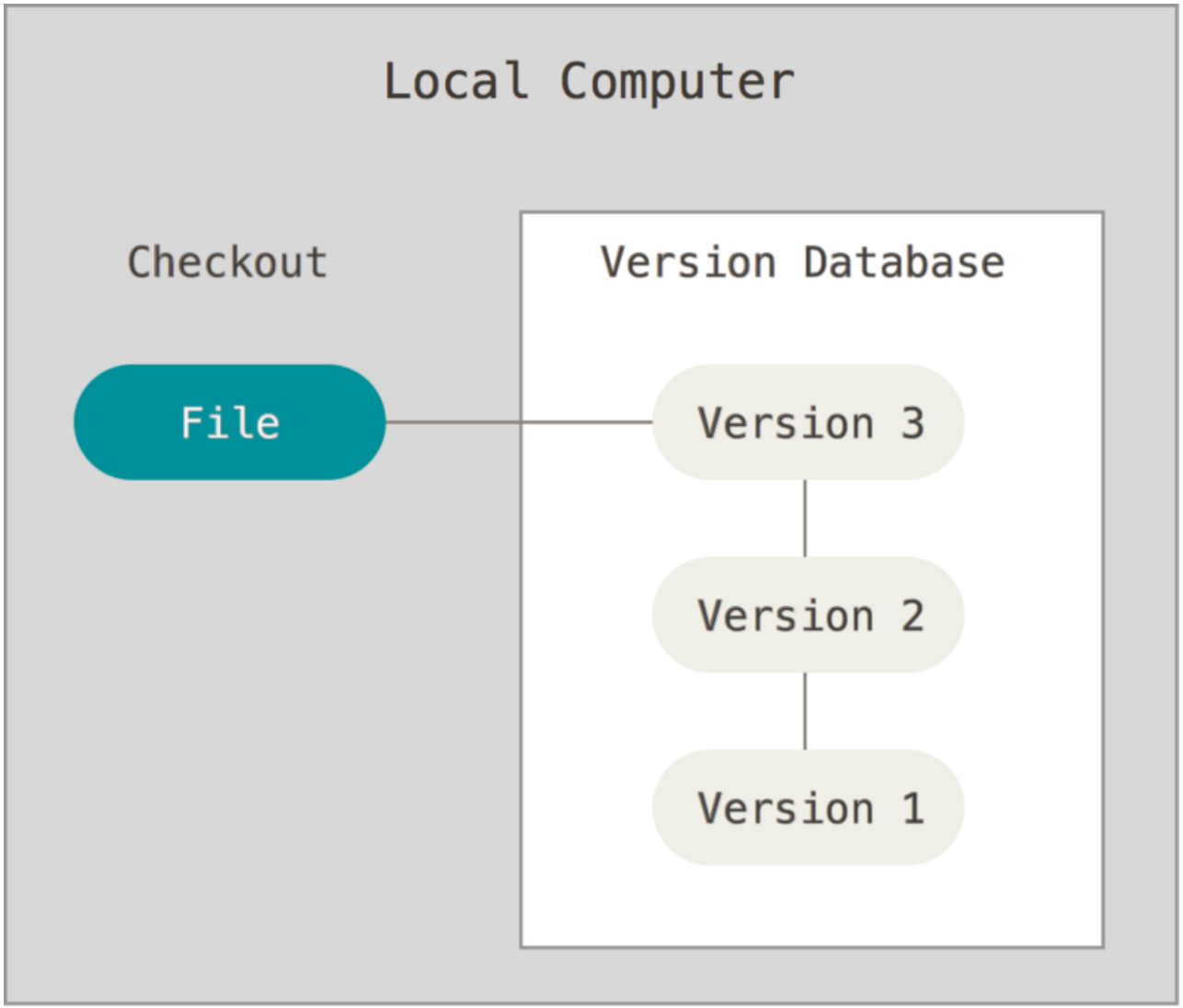
Closing

A summary of Chapter 1 from *Pro Git*

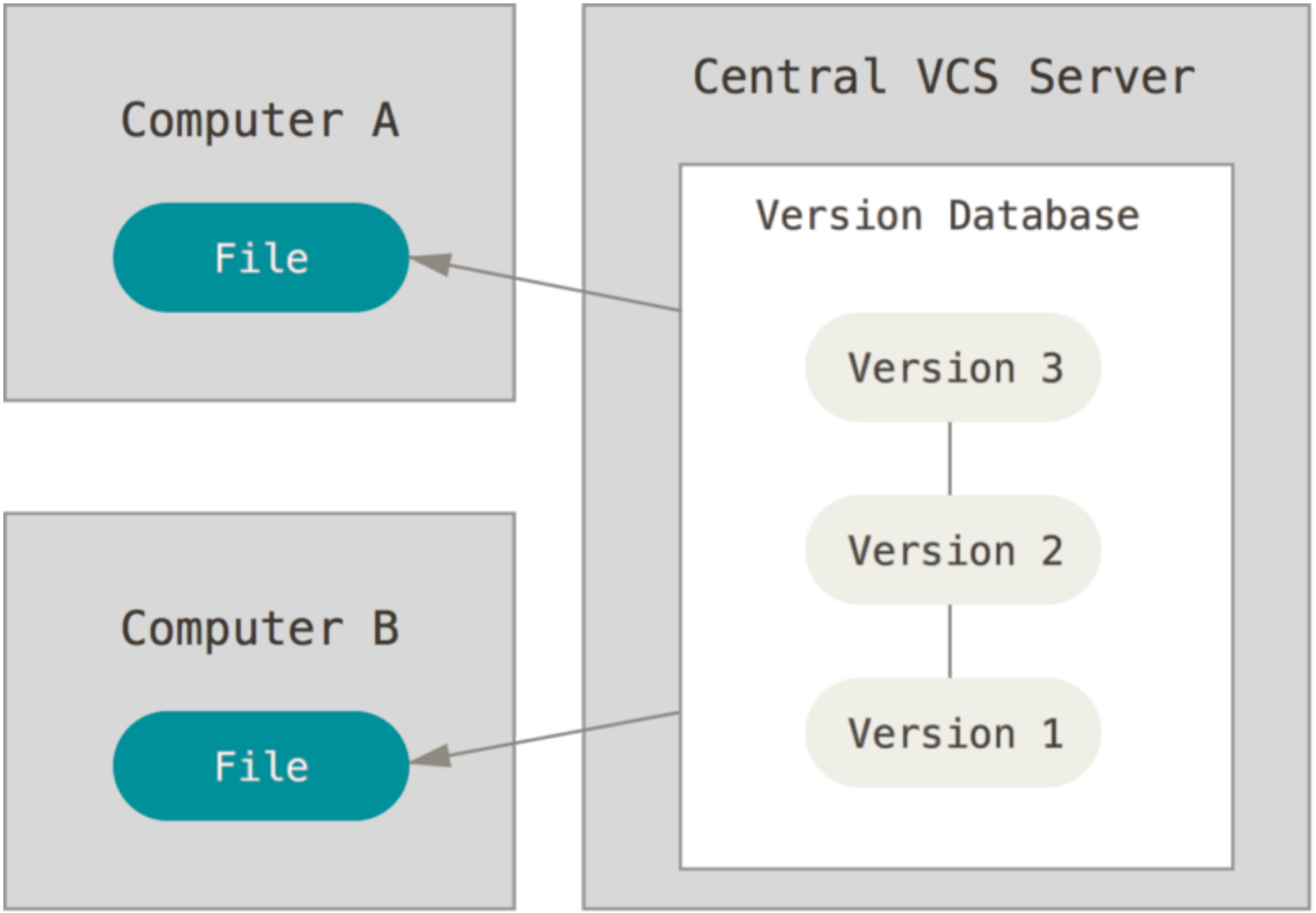
- Version control paradigms.
- What is Git?
- How does Git work?

- How to use Git will be discussed in the next section.

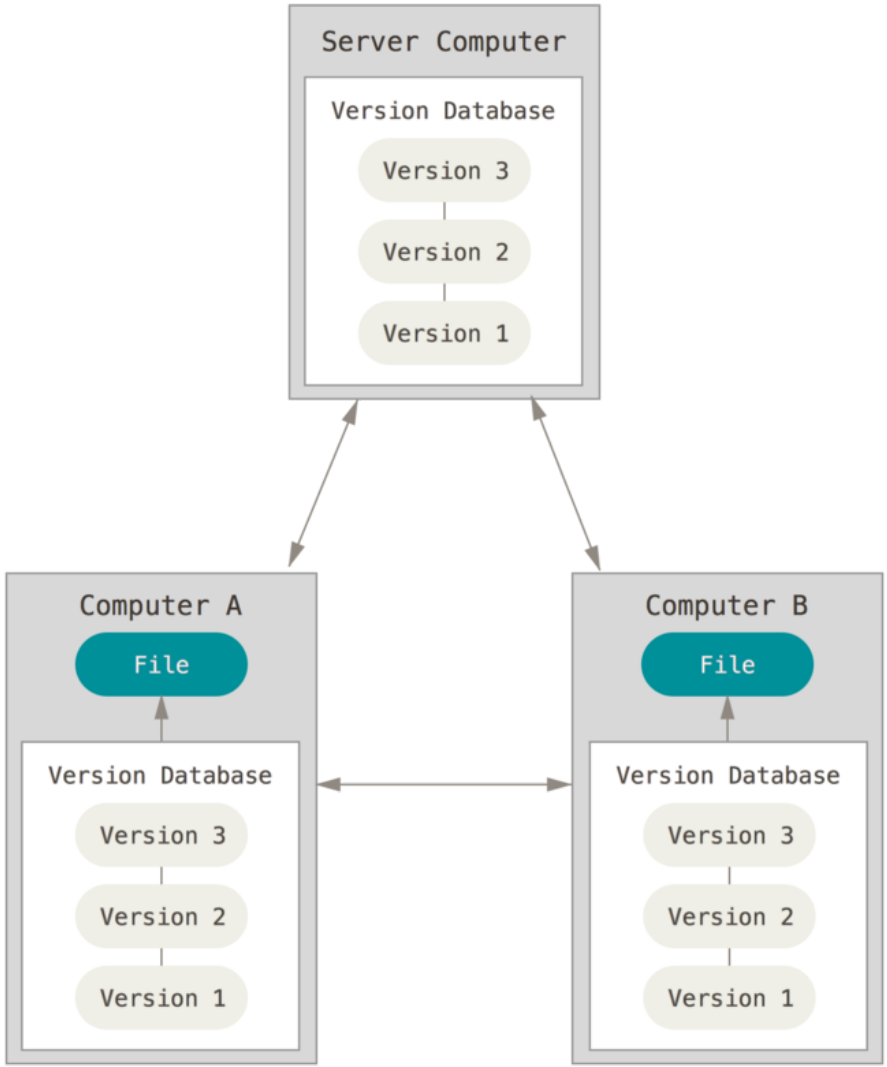
Local Version Control System



Centralized Version Control Systems



Distributed Version Control Systems

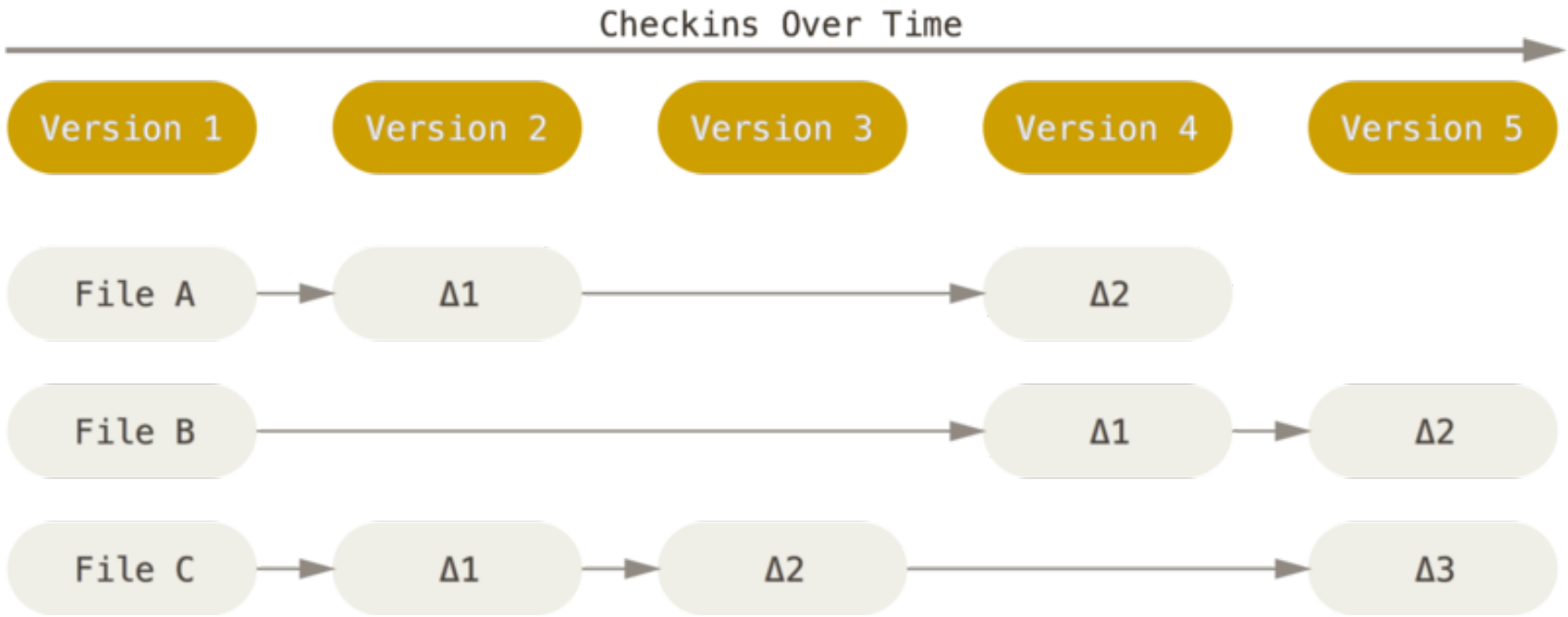


Short History of Git

- Linux (Linus Torvalds) vs BitKeeper
- 2005: Linux development community sets out to develop their own DVCS with goals for the new system of
  - speed,
  - simple design,
  - strong support for non-linear development (thousands of parallel branches),
  - fully distributed, and
  - be to handle large projects, like the Linux kernel, efficiently.

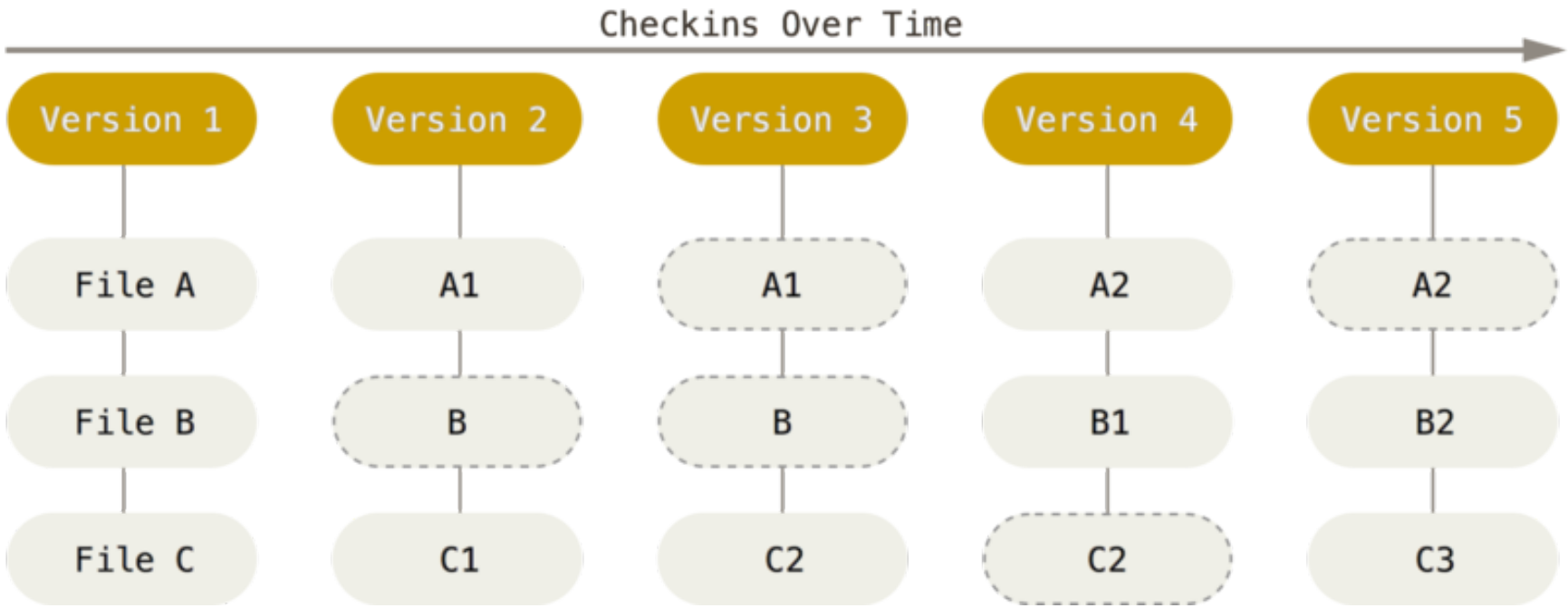
Snapshots, Not Differences

The Subversion model: file-based changes.



Snapshots, Not Differences

The Git model: a stream of snapshots.



Nearly Every Operation is Local

- Most operations are local.
  - No need to talk to other computers on a network.
  - The entire project history is on your local disk.
- You can work offline.
- You can work off VPN.
- Asynchronous
  - Multiple people can work on file at the same time
  - Git can merge non-conflicting changes easily
  - Git has tools and methods for resolving conflicts

Git Has Integrity

- Everything is check-summed (remember the sha?)
- You can't lose information in transit nor get a file corruption without Git being able to detect it.
- Git stores everything in its database not by file name but by the hash value of its contents.

Git Generally Only Adds Data

- Nearly all actions in Git add data to the Git database.
- It is difficult to do anything that is not undoable.
- You can lose/corrupt un-committed changes.
- It is very difficult to lose anything after a commit, especially with frequent pushes to other repositories.

The Three States

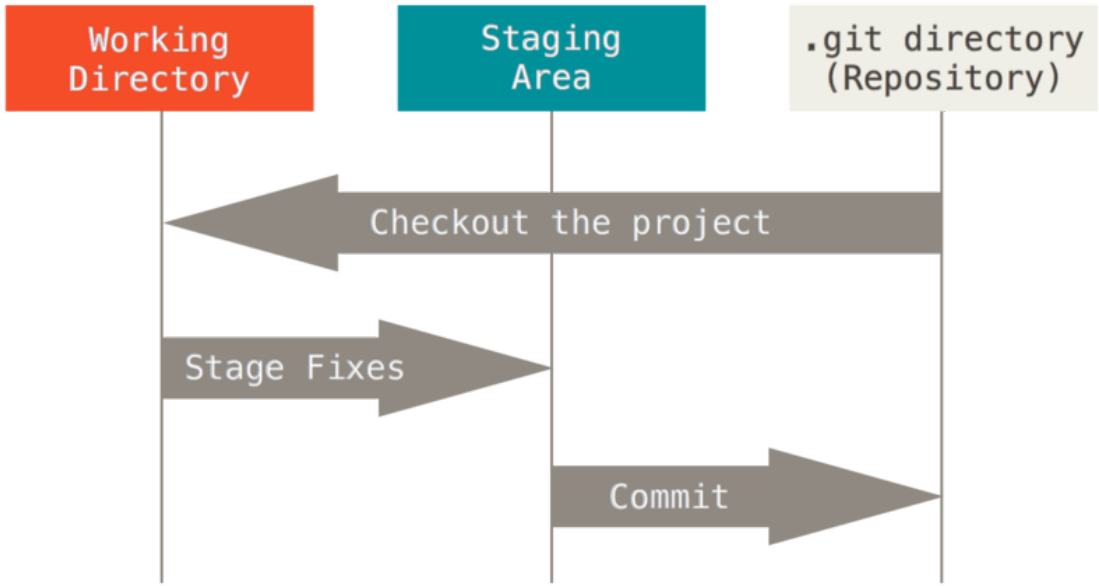
Git has three main states that files reside in.

1. Committed
   - data is safely stored in the local database.
2. Modified
   - changed the file(s) but have not committed to the database yet.
3. Staged
   - Marked modified file(s) in its current version to go into the next commit snapshot.

This leads to three main sections of a Git project:

1. the Git directory,
2. the working directory, and
3. the staging area.

The Three States



Basic Git workflow:

1. Modify files in working directory.
2. Stage files, adding snapshots of them to staging area.
3. Commit. Takes the files as they are in the staging area and stores that snapshot permanently in your Git directory.

Different ways to use Git

- Original command line tools.
- Many graphical user interfaces (GUIs) with varying capabilities and other Clients are available.
- Command line allows you to use **all** Git commands.
- GUIs allow for only a subset of Git commands.
- After install, on Windows, Mac OSx, or Linux, command line will be available.
- GUIs, take your pick. `http://git-scm.com/downloads/guis` has about three doze different clients with different features, OS requirements, . . . ,
- RStudio has a built in Git Client.

Git is Free!

- Linux: install via
  - `yum install git`, or
  - `apt-get install git`.
- Mac: Xcode Command Lines Tools.
- Windows: `http://git-scm.com/download/win`
- Install form source too, if you want.

Set up

- Need to run this once after install. Done via command line, can be done in *some* GUIs.

```
$ git config --global user.name "your name"
$ git config --global user.email "you@school.edu"
```

- Name and email are used in the signature to a commit; we'll know who to blame for a change.
- Extra credit/security - you can sign commits with GPG keys to verify the author of the commit.

Getting Help

You can get information on Git verbs in the terminal via:

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

Outline

Why? and little bit of How?

Welcome: Why Git?

Getting Started

Basic Git Use

Branching

Closing

What will be covered in this section?

- Working on a local repository.
- Working with a remote.
- Working with other people with a common remote.

- Read Chapter 2 of *Pro Git* for more information.

- Examples will be done using the command line.

Working on a Local Repository

Objectives:

We will learn the following Git verbs:

- `init`
- `status`
- `add`
- `commit`
- `diff`
- `log`
- `checkout`
- `mv`
- `rm`

Other helpful things:

- `.gitignore` files

Working with a Remote

Objectives:

Remote Hosting Options

- `github.com`, `gitlab.com`, `bitbucket.com`, and many others.
- Read the end user agreements.
- Public or Private repo: data is stored on a public server.
- You can run remote servers behind firewall

Authentication:

- https
  - Okay, quick and simple
  - May require you provide username/password for each push, fetch
  - Not supported if you have 2FA set up
- ssh
  - Easy to set up
  - Better security, works within 2FA
  - I like to have a unique public/private key pair for each computer I work on. If a machine is compromised I can protect all my work by revoking that specific key.

Working with a Remote

We will learn the following Git verbs:

- `remote`
- `push`
- `clone`
- `fetch`
- `merge`
- `pull` (this is short-hand for `fetch` and `merge` in one step)

Working with a Remote

Working with Others and Remotes

Owners of the repo can set read/write permissions.
We will learn the following Git verbs:

- `merge`
- `diff-tools`

Outline

Why? and little bit of How?

Welcome: Why Git?

Getting Started

Basic Git Use

Branching

Closing

What will be covered in this section?

- Why Branching?
- How to Branch.

- Read Chapter 3 of *Pro Git* for more information.

- Branching and merging are fundamental to the use of Git.
  - In other VC software system branching and merging are either impossible or very, very difficult.
  - Git makes it easy!
- I'm hoping to have time to talk about this, but I expect to have run out of time. It's okay, working on just the master branch for now is a good starting place. Learn all the verbs needed to work on one branch and then the details to learn branching will come as a natural next step.

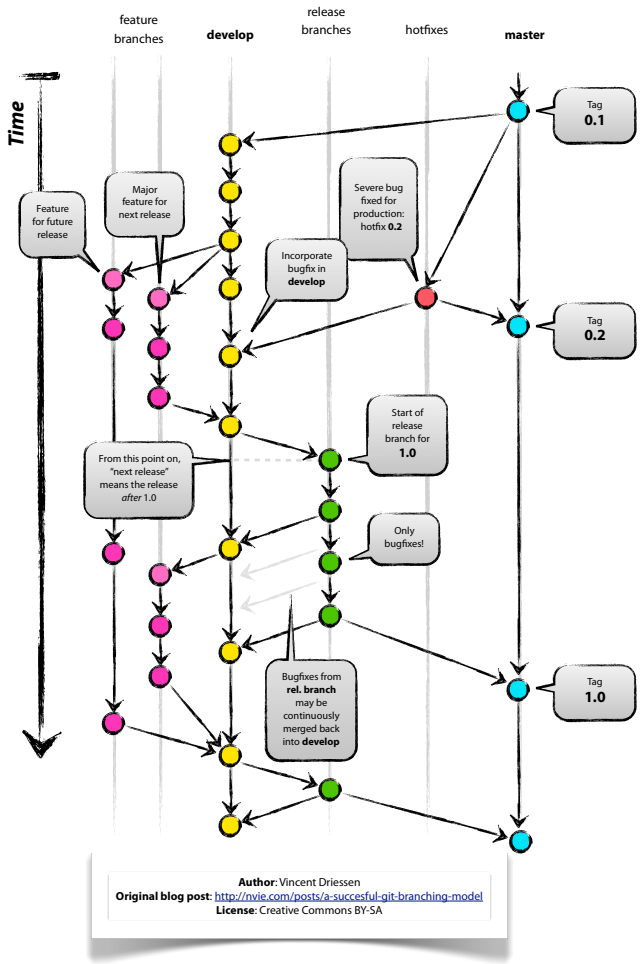
Why is Branch?

- Development off of the master branch.
- Work in a sandbox for features.
- What's down this rabbit hole?

- "I've got this great idea! Wait, #@!!?%, it's not going anywhere. Well, maybe? I'd done know." ← Make a branch! :)

Branching Model

http://nvie.com/posts/a-successful-git-branching-model/



Outline

Welcome: Why Git?

Getting Started

Basic Git Use

Branching

Closing

Closing

Thank you for your time.

- Slides are available from `https://github.com/magic-lantern/Computational-Husbandry-2022.git`
- Direct questions:
  - peter.dewitt@cuanschutz.edu
  - seth.russell@cuanschutz.edu

The floor is open for questions and comments.