# Leveraging version control to accelerate research

February 25, 2024

# A note on terminology

Github $\subset$ git $\subset$ Version Tracking/Control $\subset$ Configuration Management

# Introduction to version control

**Version Control:** In software engineering, version control is a class of systems responsible for **managing changes to** computer programs, documents, large web sites, or other **collections of information.**

# Introduction to version control

**Version Control:** In software engineering, version control is a class of systems responsible for **managing changes to** computer programs, documents, large web sites, or other **collections of information.**

↓

**Version control manages changes to collections of information.**

# Introduction to version control

- Broadly, version control allows you to:
  - track, manage, and understand changes
  - collaborate effectively
  - store backups
  - create and build upon "known states"
- Version control is often seen as a safety net, which may be why it is often ignored.

# Introduction to version control

However, version control can also accelerate progress!...Well, indirectly at least.

I will argue that effective utilization can

- **Reduce cognitive load**
  Utilizing **pull requests** and **branches** lowers cognitive overhead to start/continue one or multiple projects.

# Introduction to version control

I will argue that effective utilization can

- **Reduce cognitive load**
  Utilizing **pull requests** and **branches** lowers cognitive overhead to start/continue one or multiple projects.

- **Deliver incremental success**
  Working from a *known state* towards a *known state* allows for better project definitions, particularly when a task/project is "done"

# Introduction to version control

I will argue that effective utilization can

- **Reduce cognitive load**
  Utilizing **pull requests** and **branches** lowers cognitive overhead to start/continue one or multiple projects.

- **Deliver incremental success**
  Working from a *known state* towards a *known state* allows for better project definitions, particularly when a task/project is "done"

- **Reduce technical debt**
  Standardized, automated, decentralized testing bakes in reproducibility, creates known states, and alerts you of unintentional changes.

# Introduction to version control

I will argue that effective utilization can

- **Reduce cognitive load**
  Utilizing **pull requests** and **branches** lowers cognitive overhead to start/continue one or multiple projects.

- **Deliver incremental success**
  Working from a *known state* towards a *known state* allows for better project definitions, particularly when a task/project is "done"

- **Reduce technical debt**
  Standardized, automated, decentralized testing bakes in reproducibility, creates known states, and alerts you of unintentional changes.

- **Bake in tenants of Open Science from the start**
  Building a "product" that is incrementally constructed, continuously tested, serving a well-defined goal, reduces work at the end of a project and makes your work easier to understand and build on top of.

# Introduction to git

- git was introduced in 2005 to manage development of the linux kernel.
- Extremely lightweight, pervasive, effectively manages projects of any size.
- There are plenty of ways to use git - I will show one that is probably sufficient.

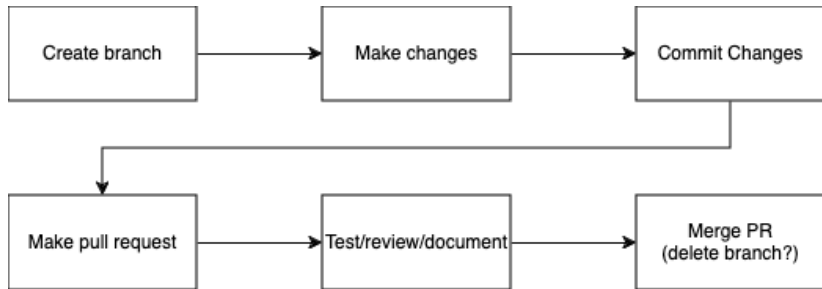# Introduction to github

Github is:

- ▶ Perfect in almost every way ✓
- ▶ ...owned by Microsoft ✗

# Introduction to github

Github is a cloud-based service that

- **is free for students** (and many others)
- **synchronizes git repositories**
  Collaborate with others or yourself, and/or deploy to cloud computing and/or HPC
- **integrates with many toolchains** I alluded to earlier (continuous integration, documentation, project management, etc)
- Has a wonderful getting started tutorial

# The git(hub) workflow & branches



```
Create branch → Make changes → Commit Changes
                                      ↓
Make pull request → Test/review/document → Merge PR
                                            (delete branch?)
```

- ▶ **Branch**: An isolated environment to make changes in, without effecting other branches
- ▶ **Commit**: A single batch of changes, usually quite small
- ▶ **Pull Request**: (PR) A formal request to merge two branches, usually consisting of peer review/tests/documentation of changes

# Use cases

- **Branches**: topic branch, development branch, bug-fix, integration test

# Use cases

- **Commits**: commits should happen frequently. Finish that function? Commit. Get another test to pass? Commit. Prove that theorem? Commit.

# Use cases

- **Pull Requests**: PRs should happen when a coherent block of functionality is complete. Fix the bug you were hunting? Open a PR. Finish the task? Open a PR.

  Further, PRs are a great place to document the state of the work and handle review comments. Add figures, justify the approach taken, document any hang ups and review comments/follow ups.

# Quick example

# Github Actions

Github actions are configurable, automated scripts that

- ▶ run at pre-determined times (e.g., on PRs/merges)
- ▶ automate process like continuous integration (unit/integration testing), build and deploy documentation, codecoverage analysis, etc
- ▶ can post results in PRs upon completion

Example: You develop on your laptop and use github to sync to the HPC. Use github actions to ensure your code builds in a clean linux environment.

# Github Actions

Github actions are configurable, automated scripts that

- ▶ run at pre-determined times (e.g., on PRs/merges)
- ▶ automate process like continuous integration (unit/integration testing), build and deploy documentation, codecoverage analysis, etc
- ▶ can post results in PRs upon completion

Example: You develop on your laptop and use github to sync to the HPC. Use github actions to ensure your code builds in a clean linux environment.

**Github actions are powerful**