# Working with Git and GitHub

Research Computing
UNIVERSITY OF COLORADO **BOULDER**

Be Boulder.

# Working with Git and GitHub

Instructor: Brandon Reyes

- Website: www.rc.colorado.edu
- Helpdesk: rc-help@colorado.edu

# My Goal

- Convince you that basic Git/GitHub fluency is:
  - Easy
  - Practical
  - An extremely important tool in your tool belt!

# Learning Goals

- Understand the basics of version control
- Differences between Git, GitHub
- Basic Git fluency
- How to collaborate with Git

# Outline

- Brief overview of Git and GitHub
  - What is version control?
- Creating your own repository locally
- Pushing local changes to GitHub
- Collaboration

# Have you set up Git/GitHub?

This is meant to be a mostly hands on tutorial. If you haven't yet, you may still be able to get everything set up in time using the link:

https://github.com/ResearchComputing/Summer_Camp_2023/blob/main/Day_Three/Using_git/README.md

# Git vs GitHub

- Git: version control system
  - the actual software

- GitHub: Cloud-based storage website
  - Hosts repositories (“repos”)
  - Provides a GUI for many Git features
  - Allows for easy collaboration
    - Issues, pull requests

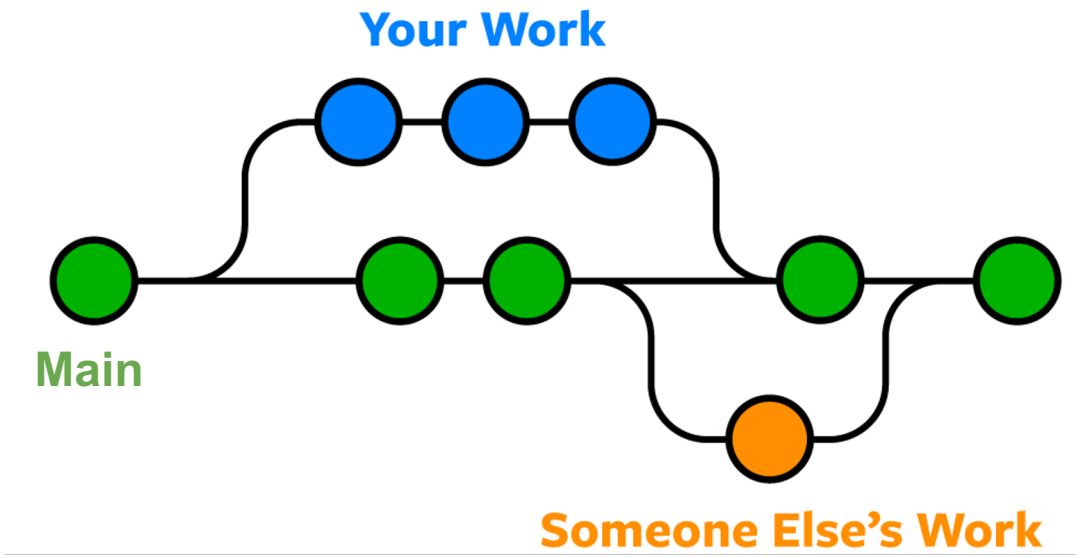
# What is version control?

Version control is the practice of tracking and managing changes to files.

- Why do I need it?
  - Revert to various states of files
    - You can think of this as a backup
  - Allows you to modify items without harming the original copy
  - Not limited to code
    - documents, images, etc…

# Additional benefits of version control

- Using version control provides
  - Clear tracking of the repo's history
  - Management and view of different branches (work)
  - Collaboration through merging of branches




Images: nobledesktop.com

# Different Version Control Systems

- Subversion (svn)
- Mercurial
- CVS
- etx

- We’re going to stick to Git
  - industry standard
  - widely known
  - most resources



Images from Wikipedia

# Getting Started with Git (local)

# Setting Git up locally

Many systems have Git installed; however, you may need to download it on your local machine

- See https://git-scm.com/book/en/v2/Getting-Started-Installing-Git for more information

Today we are going to stick with using Git on a login node

# Logging into RC via Terminal

To login to an RC login node:

```
$ ssh <username>@login.rc.colorado.edu
```

- Supply your IdentiKey password and your Duo app will alert you to confirm the login
- Confirm Git has been configured (by you using the README)

```
$ git config --list
```

# Hands on tutorial

Goal: Create a simple project that contains some Python code

First let's create a new directory for our project:

```
$ cd /projects/$USER
$ mkdir git-tutorial
$ cd git-tutotial
```

# Git Repository (Repo)

A Git repository tracks and saves the history of all changes made.

- All of this information is stored in ".git", which is the repository folder

We can make a directory (folder) a Git repo using "git init"

# Git Init

In your “git-tutorial” directory run

```
$ git init
```

- Git creates the "hidden" directory called “.git”

```
$ ls -a
```

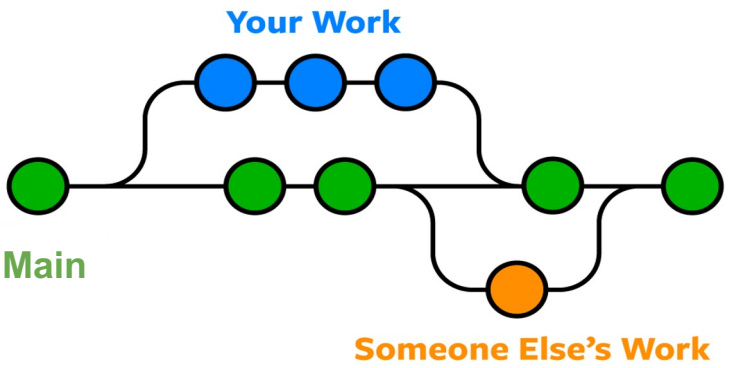
- Your directory is now a repo!
  - Git is now ready to to be used
  - Allows us to tell Git what items to watch

# Create the main branch

Now that we have a repo, we can create branches. Branches are a version of the repository.

- It is customary to name the primary branch “main”
- This can be done as follows (after an init)

```
$ git checkout -b main
```



# Let's add a file!

It is customary to add a README.md

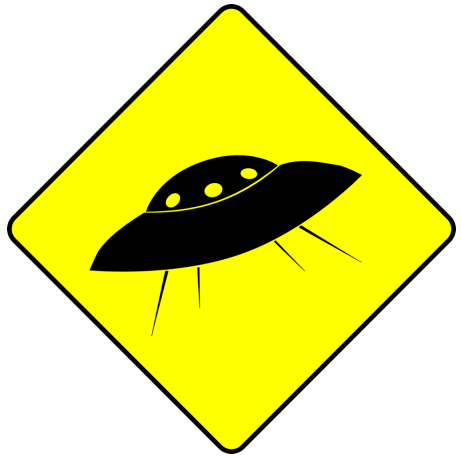
- Description of repo and any helpful information

To add a README.md, in "git-tutorial" create and edit the file using nano (or an editor of your choice)

```
$ nano README.md
```

- Add anything you would like!
- Be sure to save the file when you exit.


Git does not know about README.md yet!!



# Areas of Git

Working Area

- Items that you are currently working on
- Are not tracked by Git!
- Exists locally

Staging Area

- When Git starts tracking and saving your work
- Exists locally
- Items are added to this area by using “git add”

GitHub

- Exists locally and on GitHub!
- Items are added to this area using “git push”

# Git Status

The git status command **displays the state of the working and staging area**.

Let's see what area README.md is in

```
$ git status
```

- We see it is an untracked file, so it is in the working area

# Git Ignore

- In you `.gitignore` you can choose to ignore output files:

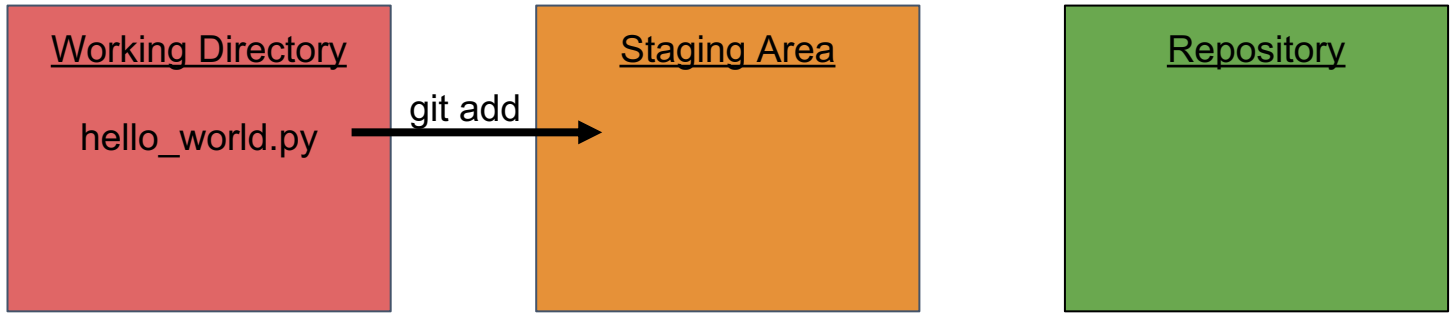
```
*out        # globbing, will get all files that end with “out”
```

# Git Add

- The git add command **adds a change in the working directory to the staging area** (getting the “picture” ready for a snapshot)
- It tells Git that you want to include updates to a particular file.

```
$ git add hello_world.py   # “git add .” to add all files
$ git status
```

**git add doesn't affect the repository - changes are not actually recorded until you run git commit



# Git Ignore

- You may have some files that you don't want tracked
  - secret keys (passwords, API tokens, etc)
  - build files
  - data sets
- Create a ignore.txt file

```
$ echo "ignore this file!" > ignore.txt
```

- Create a .gitignore file

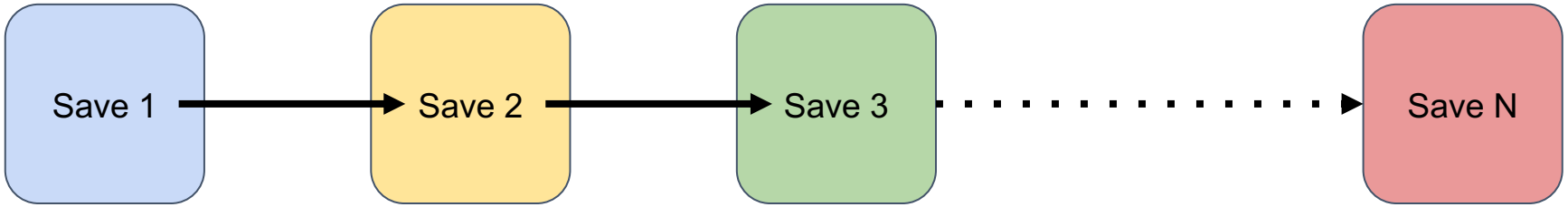
```
$ vim .gitignore
```

- list any files/directories you don't want tracked:

```
ignore.txt
```

# Your Git timeline

- Git commits are like savepoints or snapshots of your project

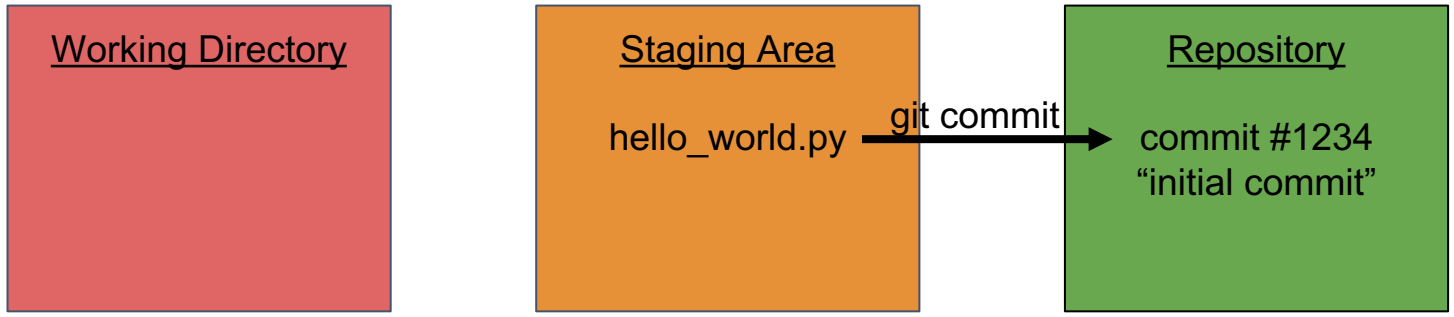


# Git Commit

- The git commit command **captures a snapshot of the project's currently staged changes.**
- Committed snapshots can be thought of as “safe” versions of a project.
- Commits are logged with a brief message of what was changed

$ git commit -m “initial commit”

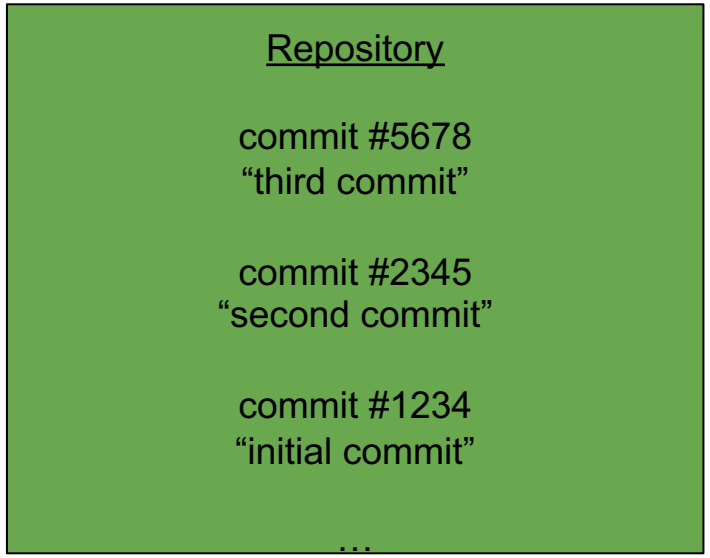
$ git status # clean working directory



# Git Log

- git log **lists the commits made in that repository** in reverse chronological order; the most recent commits show up first

  $ git log



# Getting Started with GitHub (remote)

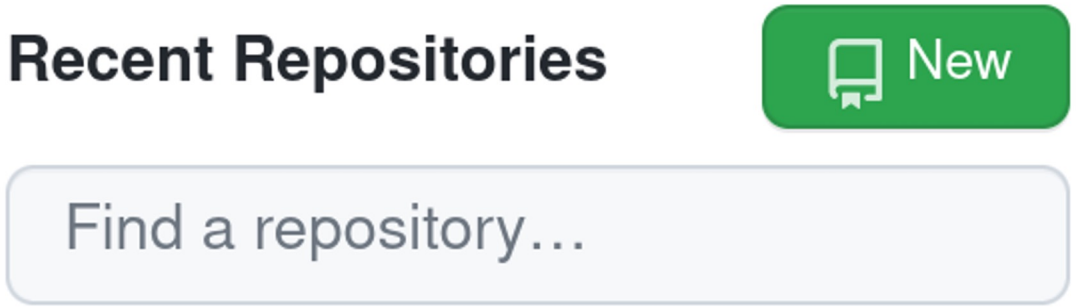
# GitHub

- GitHub: Cloud-based storage (repository, or “repo”) site
  - a common/shared area to host projects
  - many Git features as a web GUI

- We’re going to demonstrate how to work with remote repositories using GitHub

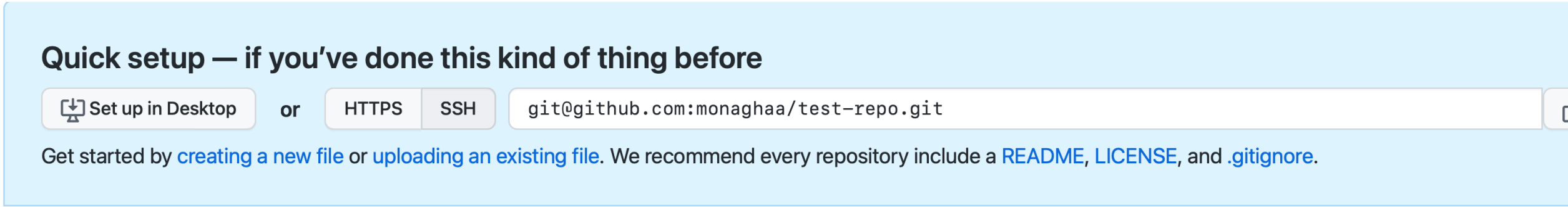
# GitHub

- Go to: https://github.com
- Sign in (or create an account)
- Click on “Create New Repository” or just “New”



# Create Repo in GitHub

- Create a new repo
- Call it whatever you would like
- Ignore directions for you, just change to **ssh** and copy the link
  - e.g. git@github.com:<user>/test-repo.git



# Git Remote

- Git remote tells you **which remote repositories you have linked to your local project**.

```
$ git remote          # should return nothing
```

- To link our remote repository (accepts 2 values):

```
$ git remote add origin git@github.com:<user>/test-repo.git
```

- View remote again

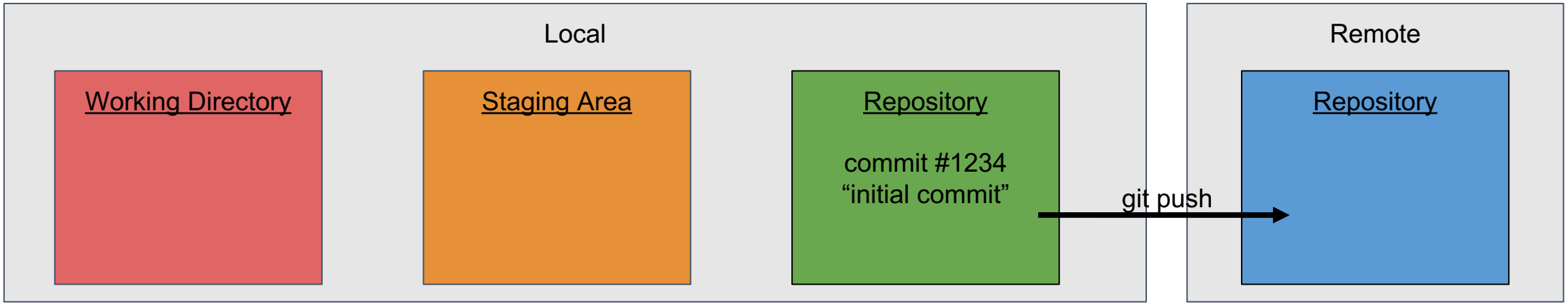
```
$ git remote
$ git remote -v          # view url as well
```

# Git Push

- Sync up local code with remote GitHub repo!
- Git push **uploads a local repositories content to a remote repository**. Pushing is how you transfer commits from your local repo to a remote repo

```
$ git push <name of remote repo> <branch>
```

```
$ git push origin main
```



# GitHub

- Go back to GitHub and refresh your page
  - should see the files we have added (and not the ones we've ignored)

- Some cool features!
  - look at our commits
  - directly edit/commit in the browser

- Let's do that! Let's fix the typo and commit it
  - But now our remote repo is one commit ahead of our local one…

# Git Fetch & Merge

- Git fetch retrieves the changes from the remote repo

```
$ git fetch
```

- Git merge combines two branches

```
$ git merge origin/main
```

- But there's an easier way!

# Git Pull

- Git pull combines the fetch and merge commands
- **Must have clear working directory!**

```
$ git pull origin main
```

# Git Clone

- Git clone **makes a clone or copy of a remote repo at in a new directory, at another location**.

```
$ git clone <url> <optional new name>
```

- Easy way to grab third-party code, or pre-existing code you might need to work on

```
$ cd /projects/$USER
$ git clone https://github.com/ResearchComputing/HPC_software_dev_course
```

# Update your project! (practice)

- Create a new file in your test repo and Add + Commit it
- Then push up to your GitHub repo and ensure your new file is there!

# Review: Learning Goals

1. Understand basics of version control
2. Differences between Git, GitHub
3. Basic Git fluency

# Help! I'm stuck, where do I go?

- **Documentation**: curc.readthedocs.io/
- **Trainings with Center for Research Data and Digital Scholarship (CRDDS)**: https://www.colorado.edu/crdds/
- **Software Carpentries tutorial:** https://swcarpentry.github.io/git-novice/index.html
- **Helpdesk**: rc-help@colorado.edu

# Questions

# Survey and feedback

http://tinyurl.com/curc-survey18