



# Working with Git and GitHub

# Working with Git and GitHub

Instructor: Brandon Reyes

- Website: [www.rc.colorado.edu](http://www.rc.colorado.edu)
- Helpdesk: [rc-help@colorado.edu](mailto: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/D  
ay\\_Three/Using\\_git/README.md](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



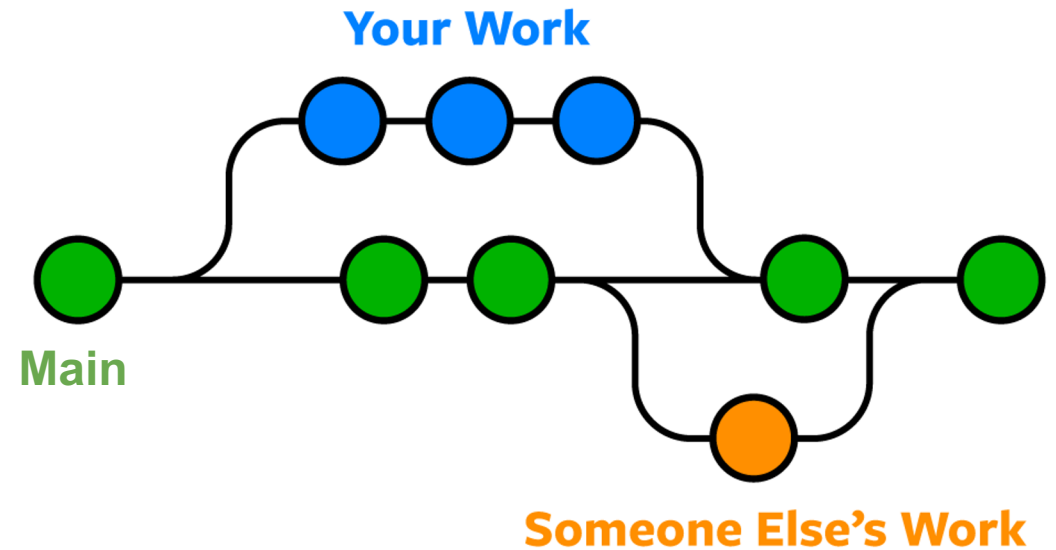
# What is version control?

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-tutorial
```

# 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
```

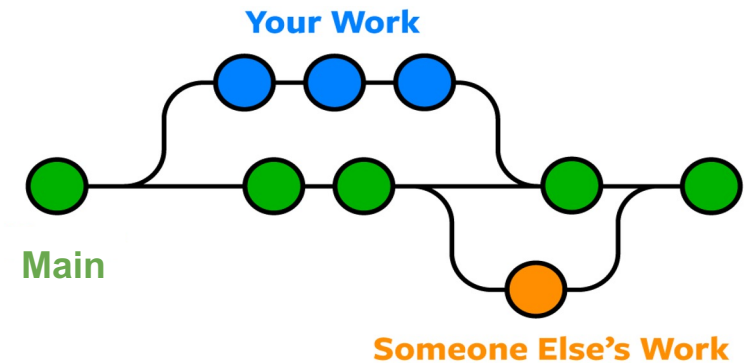
- Your directory is now a repo!
  - Git is now ready to be used
  - Allows us to tell Git what items to watch

# 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

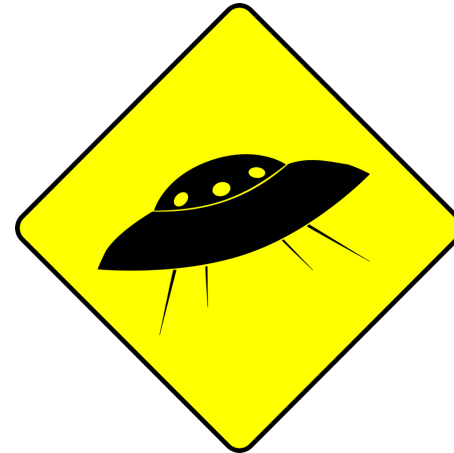
To add a README.md, in “git-tutorial” create and edit the file using nano (or an editor of your choice)

```
$ 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 Workflow

## 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”

## Snapshot Area

- All staged items are captured
- Version of the repo
- Exists locally
- Items are added to this area by using “git commit”

## 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



What if you don't want Git to track something?

# .gitignore

We can add a file named “.gitignore” to our repo

- Specifies what items should never be tracked

Let's create a file to ignore!

```
$ echo "Super secret stuff" > confidential_data.txt
```

Add “.gitignore” to “git-tutorial” and put “confidential\_data.txt” in it

```
$ echo confidential_data.txt > .gitignore
```

Let's add our files to the staging area now!

# Git Add

The git add command **adds a change in the working area to the staging area**

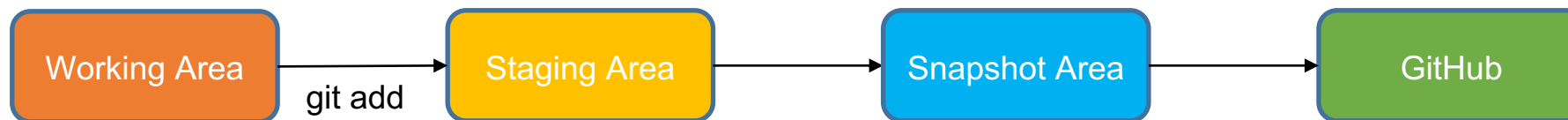
Let's add our README.md to the staging area

```
$ git add README.md
```

or add everything in the current directory

```
$ git add .
```

- Anytime a change is made, you need to do a git add (to track them)



# Git Commit

The git commit command **captures a snapshot of all staged items**

- Commits can be thought of as a version of the repo
- Commits should be accompanied with a brief message

Let's commit our staged item!

```
$ git commit -m 'Create repo, add README.md, add .gitignore'
```

```
$ git status
```



# Common practice – add, commit

- `git add`
  - Can be performed as much as you want
  - Doesn't need to be done after every change
- `git commit`
  - Always include a comment!!
  - Bundle common staged items together
  - Try not to put too many things in a commit



# Git Log

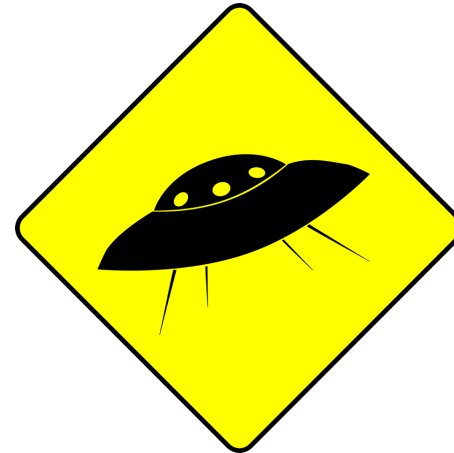
The command git log **lists the commits made in that repository**

- Lists the most recent commits first

\$ git log



All changes and files are only locally stored right now!



# **To GitHub we go!**

# GitHub

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

**Recent Repositories**

 **New**

Find a repository...

# 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](https://github.com/monaghaa/test-repo.git):<user>/test-repo.git

Quick setup — if you've done this kind of thing before



Set up in Desktop

or

HTTPS

SSH

git@github.com:monaghaa/test-repo.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

# 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
```

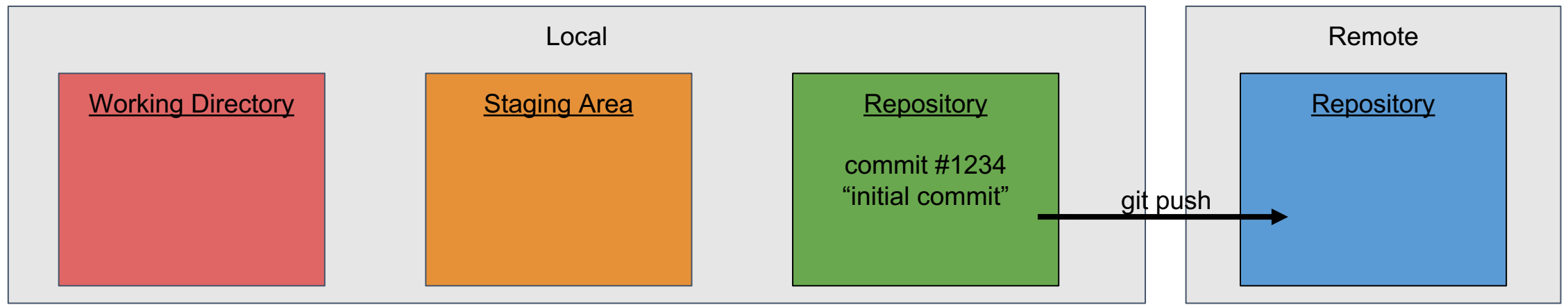


# Git Push

- 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/](http://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](mailto:rc-help@colorado.edu)



# Questions

# Survey and feedback

<http://tinyurl.com/curc-survey18>