



Go from 0 to 60: GitHub for Beginners

Workshop Taught by Jami Jackson Mulgrave

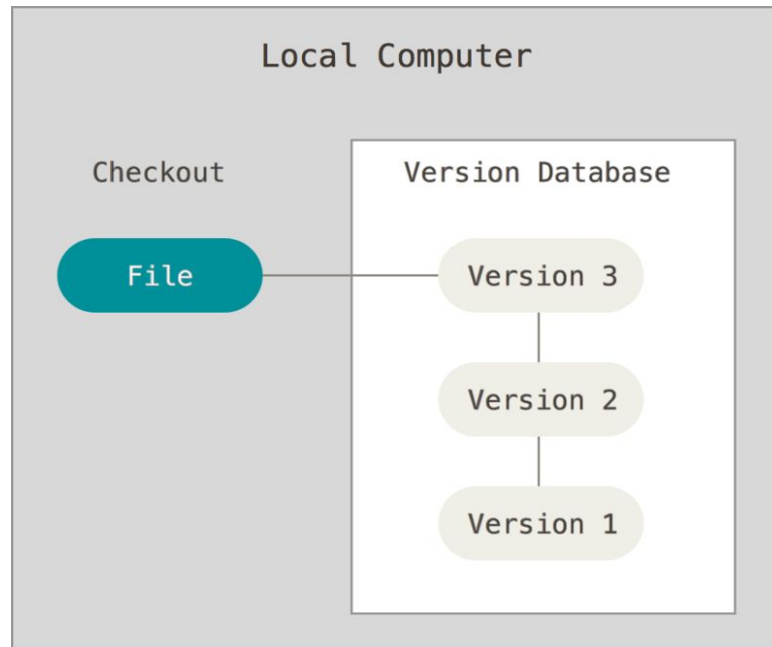


Outline

- What is Version Control?
- What is Git and GitHub?
- Why use Git and Github?
- What do people use Git and GitHub for?
- Housekeeping
- Introduction to Git
- A Little about GitHub

Local Version Control Systems

- Many people's version-control method of choice is to copy files into another directory.
- This approach is simple, but it is also error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.
- To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.
- When a file is “**checked out**” by a user, others can read that file, but no one else may change that file until that user “checks in” the updated version (or cancels the checkout).



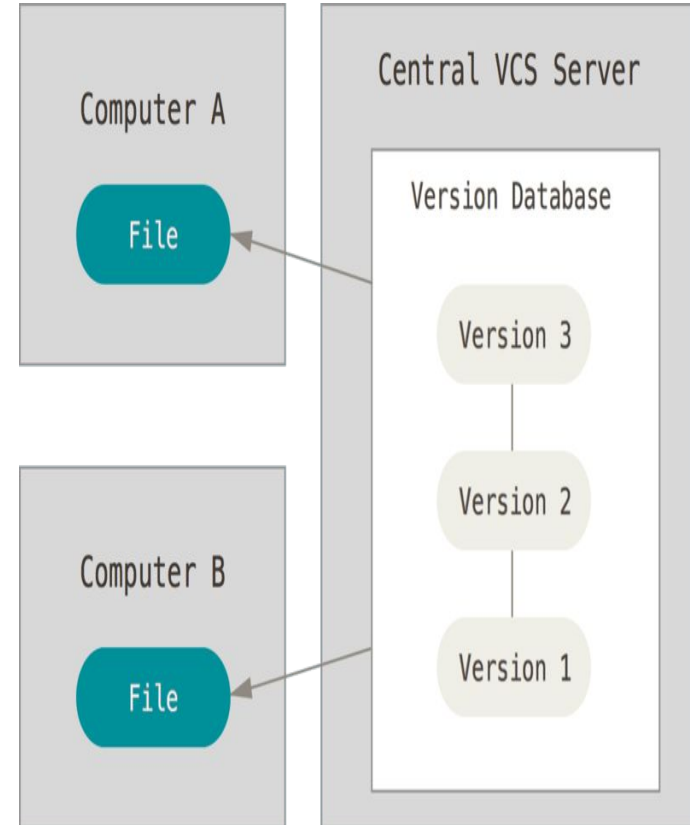


What is Version Control?

- Version control is a system (VCS) that records changes to a file or set of files over time so that you can recall specific versions later.
- It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.
- Using a VCS also means that if you mess things up or lose files, you can easily recover with very little overhead.

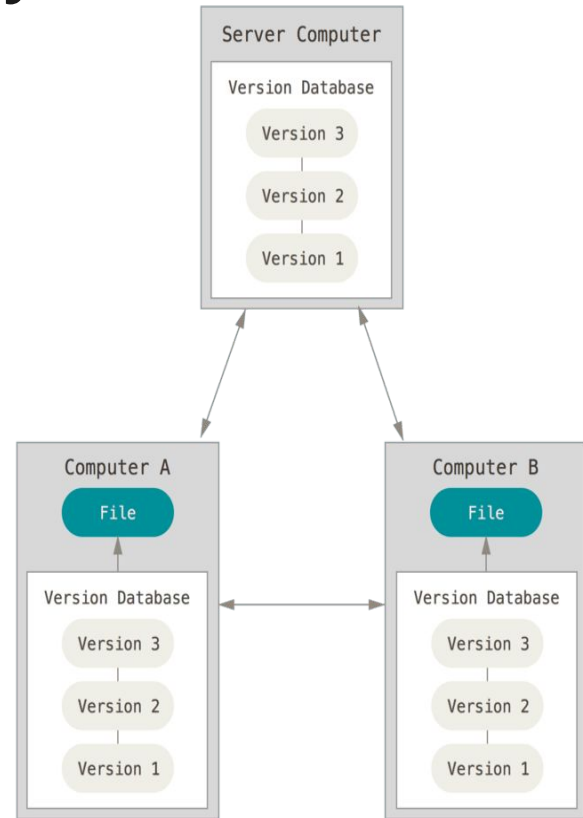
Centralized Version Control Systems

- In order to collaborate with others, Centralized Version Control Systems (CVCSs) were developed.
- These systems have a single server that contains all the versioned files, and a number of clients that check out files from that central place.
- However, if the server goes down, then nobody can collaborate or save versioned changes. If the hard disk the server is on becomes corrupted, and proper backups haven't been kept, you lose everything.
- Whenever you have the entire history of the project in a single place, you risk losing everything.



Distributed Version Control Systems

- With Distributed Version Control Systems, such as Git, clients don't just check out the latest snapshot of the files: they fully mirror the repository.
- Thus if any server dies, and these systems were collaborating via it, any of the client repositories can be copied back up to the server to restore it.
- Every clone is a full backup of all the data.
- Furthermore, these systems deal well with having several remote repositories they can work with, so you can collaborate with different groups of people in different ways simultaneously within the same project.





What is Git?

- Git is a distributed version control system.
- It manages all the versions of all the documents.
- It never throws anything away, and you can access any prior version at any time.
- It's an open source project that anyone can use and run locally.
- It requires knowledge of command line syntax.
- It doesn't have any visual or GUI interface.



What is GitHub?

- GitHub uses the power of Git behind the scenes.
- It manages information in an easier way.
- It provides you with a powerful interface for collaboration.
- Is a central place to store and share your repositories.



Why use Git and GitHub?

- **History.**
 - By preserving the history, and providing an easy way to view it, anyone at any time can see what files have been changed, who changed them, when they were changed, and why.
 - This is great for newcomers to the team who have access to as much information as the rest of the team.
- **Searchable Information.**
 - You can search through a repository's files and issues for any mention of a word or phrase.
 - Much easier than trying to remember where variable names are used, or where certain topics are mentioned.



Why use Git and GitHub?

- **Transparent Collaboration.**
 - Is extremely helpful when working with a distributed team and for cross-team collaboration and input.
- **Shareability.**
 - You have control over who can see your project and who has access to the code. You can unleash it to millions of other users on GitHub for them to use and collaborate with you, or you can keep your project to a small group of friends or just to yourself.
- **URL for everything.**
 - Everything on GitHub is referenced by a URL: comments, requests, issues, comments on lines of code, files, and images.



What do People Use Git and GitHub for?

- Coding Projects
- Travel planning: <https://github.com/dylanegan/travel>
 - Dylan Egan travels the world and crowdsources his journey on GitHub. You can check out where Egan is planning to go, and edit the file to suggest accommodations, food or activities. Afterward, he'll record what he did and post photos.



What do People Use Git and GitHub for?

- Musical Composition: <https://github.com/CMAA/nova-organi-harmonia>
 - Adam Wood, a church music director, put the Gregorian Chant known as Nova Organi Harmonia on GitHub to improve it by putting the chant into different arrangements depending on the musical instruments they had available since the original music is written with an organ in mind.
 - While GitHub does not have a built-in way to edit music, Wood said it's compatible with other editors, and other churches are remixing the chants into different keys.



What do People Use Git and GitHub for?

- **Remixing Recipes:** <http://forkthecookbook.com/>
 - Fork the Cookbook is a recipe site based on GitHub that lets you make changes to recipes to make them better, and then lets other people see the adjustments you've made.
- **Open Source Font Editing:** <https://github.com/theleagueof>
 - The League of Movable Type is an open-source type community that wants collaborators around the world to help it create the most aesthetically pleasing typefaces possible—for free.
 - You can't edit type on GitHub, but you can fork one of the font files to copy it into your own editor.



What do People Use Git and GitHub for?

- **Writing And Blogging**
 - GitHub has a secondary service, GitHub Pages, for hosting blogs and other sites.
 - One high profile user was President Obama, whose dev team hosted his campaign blog.
 - You don't need to know any code to use GitHub Pages and you can use GitHub's text editing functions to collaborate with one or more co-writers and editors.
 - One writer, JJ Merelo, is publishing his science fiction novel, Hoborg, <https://github.com/JJ/hoborg>, entirely on GitHub, where you can read it or even suggest edits.



What do People Use Git and GitHub for?

- Legal Documents
 - Lawyer Benjamin Lee posted Twitter's patent agreement, <https://github.com/twitter/innovators-patent-agreement>, to GitHub where it could be analyzed, edited and cloned.
 - Fenwick & West, one of Silicon Valley's top legal firms, posted a 30-page set of legal documents, <https://github.com/seriesseed/equity>, for startups to use when lining up venture funding.

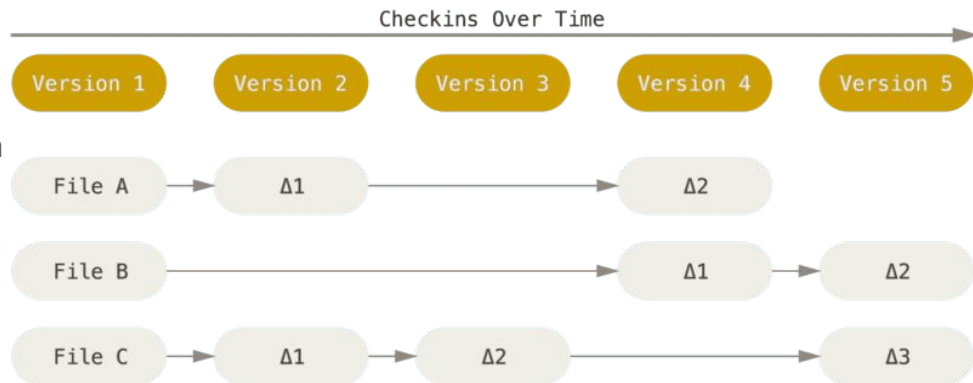


Understanding Git

- There are a lot of different ways to use Git. There are the original command line tools, and there are many graphical user interfaces (GUIs).
- For this workshop, we will begin by using Git on the command line and then we will explore GitHub, a widely used graphical user interface.
- The command line is the only place you can run all Git commands. Most of the GUIs only implement some subset of Git functionality for simplicity.
- If you know how to run the command line version, you can also figure out how to run the GUI version, while the opposite is not necessarily true.
- While your choice of graphical client is a matter of personal taste, all users will have the command-line tools installed and available, which makes collaboration easy.

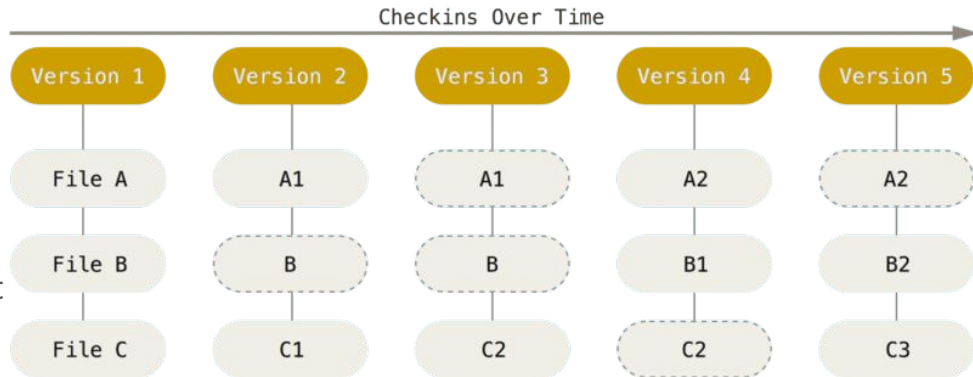
Understanding Git

- The major difference between Git and any other VCS is the way Git thinks about its data. Conceptually, most other systems store information as a **list of file-based changes**.
- These systems think of the information they keep as a set of files and the changes made to each file over time.



Understanding Git

- Git thinks of its data like a set of snapshots of a miniature filesystem. Every time you commit, or save the state of your project in Git, it takes a picture of what all your files look like at that moment and stores a reference to that snapshot.
- To be efficient, if files have not changed, Git doesn't store the file again, just a link to the previous identical file it has already stored.
- Git thinks about its data as a **stream of snapshots**.





Understanding Git

- Most operations in Git only need local files and resources to operate – generally no information is needed from another computer on your network.
- That means you can still work if you're offline or off VPN, and you can wait until you get to a network connection to upload the changes.
- Everything in Git is check-summed before it is stored and is then referred to by that checksum. This means it's impossible to change the contents of any file or directory without Git knowing about it.
- A checksum is a piece of information derived from a block of digital data for the purpose of detecting errors.

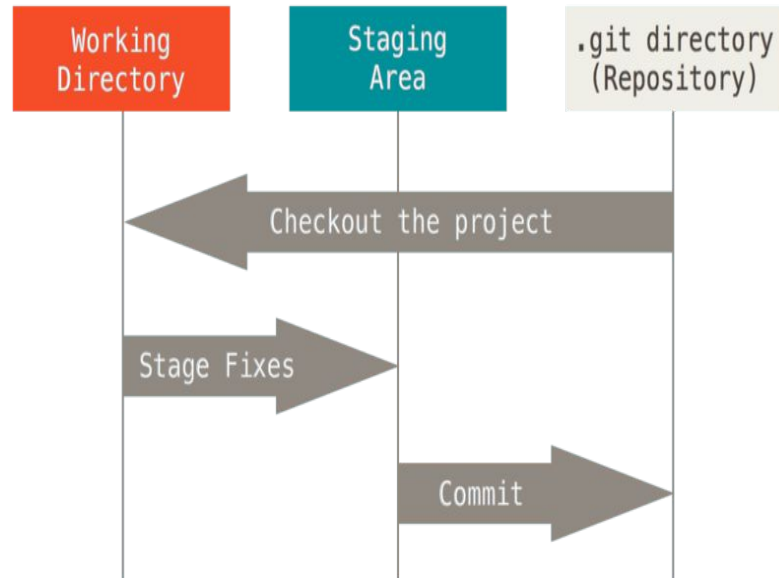


Understanding Git

- Git has three main states that your files can reside in: **committed**, **staged**, and **modified**.
- **Committed** means that the data is safely stored in your local database.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Modified** means that you have changed the file but have not committed it to your database yet.
- This leads us to the three main sections of a Git project: the Git directory, the working directory, and the staging area.

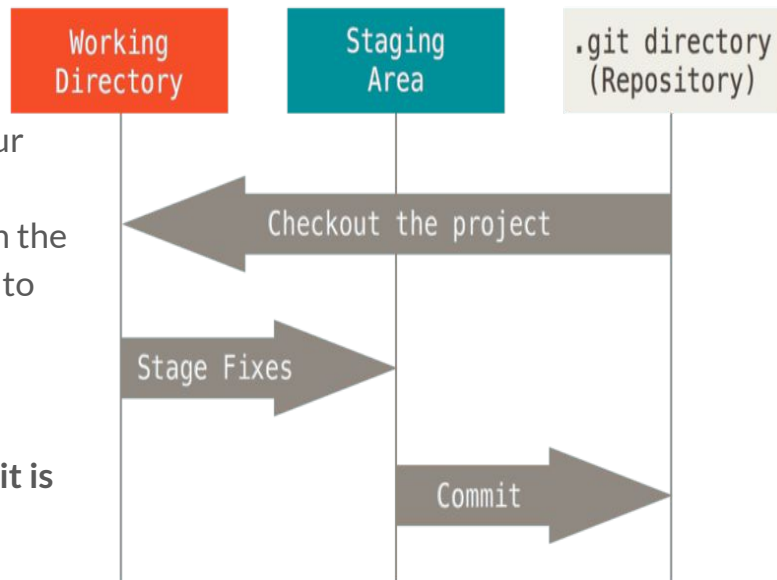
Understanding Git

- The **Git directory** is where Git stores the metadata and object database for your project. This is the most important part of Git.
- The **working directory** is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.
- The **staging area** is a file, generally contained in your Git directory, that stores information about what will go into your next commit.



Understanding Git

- The basic Git workflow goes something like this:
 - You modify files in your working directory.
 - You stage the files, adding snapshots of them to your staging area.
 - You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.
- If a particular version of a file is in the Git directory, **it is committed**.
- If it has been modified and was added to the staging area, **it is staged**.
- And if it was changed since it was checked out but has not been staged, **it is modified**.





Housekeeping: Create a GitHub Account

- GitHub <https://github.com/> offers free accounts to work on public and open source projects, as well as paid accounts that offer unlimited private repositories.
- NCSU GitHub <https://github.ncsu.edu/> is self-contained and self-managed by NC State University. It gives users the opportunity to create an unlimited number of repositories – both public and private.
- NCSU GitHub is specifically designed to not compete with Github.com for public offerings. “Public” inside of NCSU GitHub is public to the institution, not the world.
- If a given project is truly open source, then it may be more appropriate to host it at GitHub.com with a personal account.
- Any faculty, staff or student of NC State University is allowed a free account with NCSU GitHub.



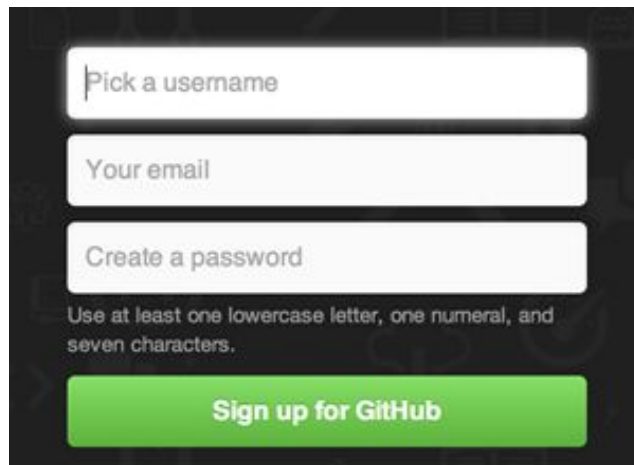
Create a GitHub Account

- For NCSU GitHub account:
 - If you are a student or faculty member, you already have access to GitHub Enterprise. Simply login at <https://github.ncsu.edu/> with your Unity username and password. Your GitHub Enterprise Account will be created instantly after login.
 - “Account Setup and Configuration” is in my GitHub account <https://github.ncsu.edu/injacks3/GitHub-Workshops>



Create a GitHub Account

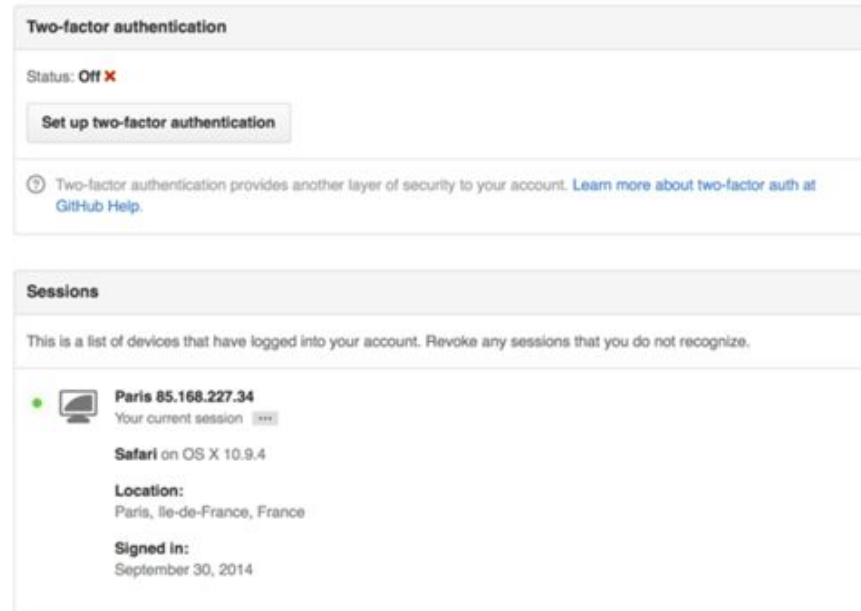
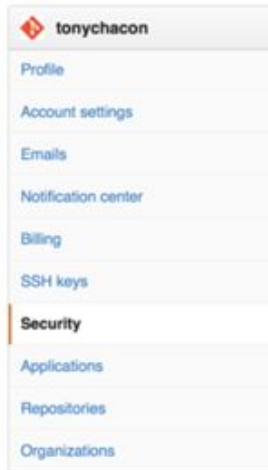
- For Public GitHub.com account:
 - Simply visit <https://github.com>, choose a username that isn't already taken, provide an email address and a password, and click the big green “Sign up for GitHub” button.



A screenshot of the GitHub sign-up form. It features three white input fields on a dark background. The first field is labeled 'Pick a username', the second 'Your email', and the third 'Create a password'. Below the password field, there is a note: 'Use at least one lowercase letter, one numeral, and seven characters.' At the bottom of the form is a large green button with the text 'Sign up for GitHub' in white.

Create a GitHub Account

- Add an Avatar to your profile
- Add your e-mail addresses
- Set up Two-factor Authentication



Housekeeping



- **Sample project for the exercises:**
 - You can use your own projects or download a file from my GitHub account to use as a sample project here: <https://github.ncsu.edu/injacks3/GitHub-Workshops>
 - This account also has all of the exercises, additional resources, and I will post the slides.



Housekeeping: Install Git

- **Make sure Git is installed**
 - See “Installing Git” on my GitHub account
<https://github.ncsu.edu/jnjacks3/GitHub-Workshops>
 - For Mac
 - On Mavericks (10.9+) you can try to run `git` from the Terminal or download at the Git website at <http://git-scm.com/download/mac> or install GitHub for Mac at <http://mac.github.com>
 - For Windows
 - Download at <http://git-scm.com/download/win> or install GitHub for Windows at <http://windows.github.com>
- **Optional: Download a text editor of your choice to work with Git**
 - You can configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your system’s default editor.
 - See “Your Text Editor”.



Command Line

- Git for Windows includes the Git Bash command line.
- The default command line interface for Mac OS X is Terminal, which uses Bash. To open it, go to your Utilities folder. There you should see the icon for Terminal.
- See “Command Line Cheat Sheet” in my GitHub account for a quick reference.



First Time Git Setup

- Now that Git is setup, we will customize your Git environment. You should have to do these things only once on any given computer; they'll stick around between upgrades. You can also change them at any time by running through the commands again.
- Git comes with a tool called `git config` that lets you get and set configuration variables that control all aspects of how Git looks and operates.



First Time Git Setup

- You should first set your user name and email address.
 - This is important because every Git commit uses this information, and it's baked into the commits you start creating.

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

- You need to do this only once if you pass the --global option, because then Git will always use that information for anything you do on that system.
- If you want to override this with a different name or email address for specific projects, you can run the command without the --global option when you're in that project.



Git Basics - Exercise 1: Initializing a Repository in an Existing Directory

- If you want to track an existing project in Git, and you already have the folder/directory created, you need to go to the project's directory:

for Linux:

```
$ cd /home/user/your_repository
```

for Mac:

```
$ cd /Users/user/your_repository
```

for Windows:

```
$ cd /c/user/your_repository
```

- And type: `$ git init`
- This creates a new subdirectory named `.git` that contains all of your repository files. Nothing in your project is tracked yet.



Git Basics - Cloning an Existing Repository

- If you want to get a copy of an existing Git repository – for example, a project you'd like to contribute to – the command you need is **git clone**.
- Instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. Every version of every file for the history of the project is pulled down by default when you run **git clone**.
- When you **git clone** (and other commands, **git fetch**, **git pull**, or **git push**, which we will get to later) to a remote repository using HTTPS URLs on the command line, you'll be asked for your GitHub username and password.
- You can use a **credential helper** to tell Git to remember your GitHub username and password every time it talks to GitHub.



Git Basics - Exercise 2: Cloning an Existing Repository

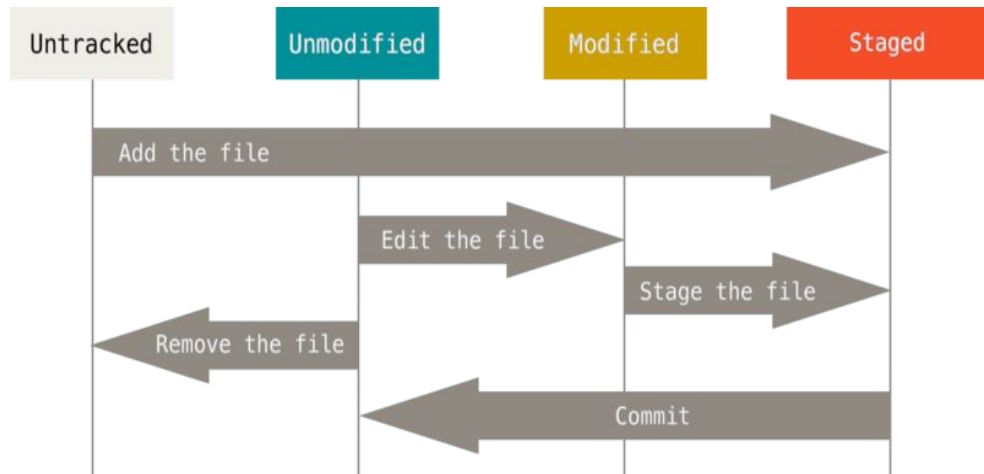
- You clone a repository with `git clone [url]`. For example, to clone my repository, you can do this:
`$ git clone https://github.ncsu.edu/jnjacks3/GitHub-Workshops.git`
- That creates a directory named “**GitHub-Workshops**”, initializes a `.git` directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version.
- If you go into the new directory, you’ll see the project files in there, ready to be worked on or used. If you want to clone the repository into a directory named something other than “**GitHub-Workshops**”, you can specify that as the next command-line option:

```
$ git clone https://github.ncsu.edu/jnjacks3/GitHub-Workshops.git  
myworkshop
```

- Now the target directory is called **myworkshop**.

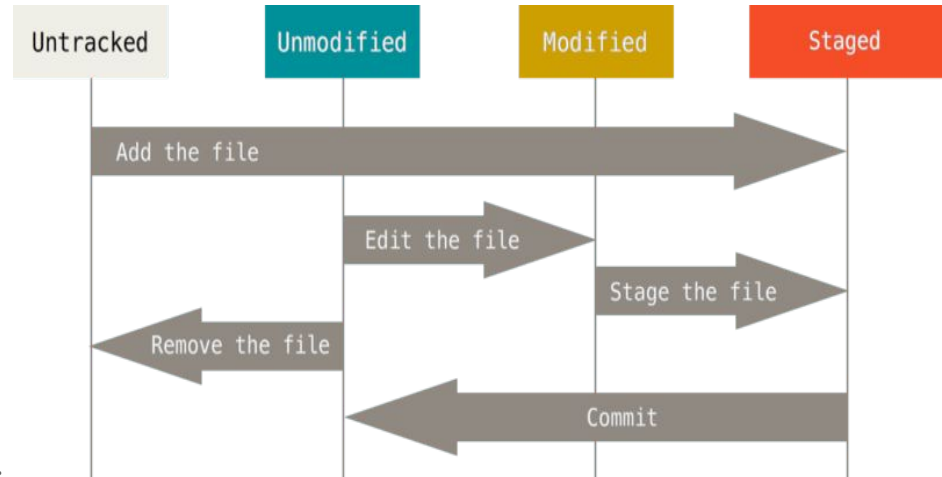
Git Basics - Recording Changes to the Repository

- You now have a Git repository and a checkout or working copy of the files for that project. You need to make some changes and commit snapshots of those changes into your repository each time the project reaches a state you want to record.
- Remember that each file in your working directory can be in one of two states: **tracked** or **untracked**.



Git Basics - Recording Changes to the Repository

- **Tracked files** are files that were in the last snapshot; they can be unmodified, modified, or staged.
- **Untracked files** are everything else – any files in your working directory that were not in your last snapshot and are not in your staging area.
- When you initialize a repository in an existing directory, you will need to track all of the files manually.
- When you first clone a repository, all of those files will automatically be tracked (and unmodified) by Git.





Git Basics - Exercise 3: Recording Changes to the Repository

- In order to track files, you use the command `git add`.
- Go to the “**GitHub-Workshops**” folder on your computer, open the README file, and make a change to it. Now it is a tracked and modified file. If you want these changes to be saved in Git, you need to stage them by running this:

```
$ cd GitHub-Workshops  
$ git add README.md
```

- Once you have added the file via `git add`, it is staged.



Git Basics - Exercise 3: Recording Changes to the Repository

- Now that your files are staged, you can commit your changes. Remember that anything that is still unstaged – any files you have created or modified that you haven't run `git add` on since you edited them – won't go into this commit. They will stay as modified files on your disk.
- Let's say that all of your files are staged, so you're ready to commit your changes. The simplest way to commit is to type `git commit` with a message

```
$ git commit -m "Updated readme"
```

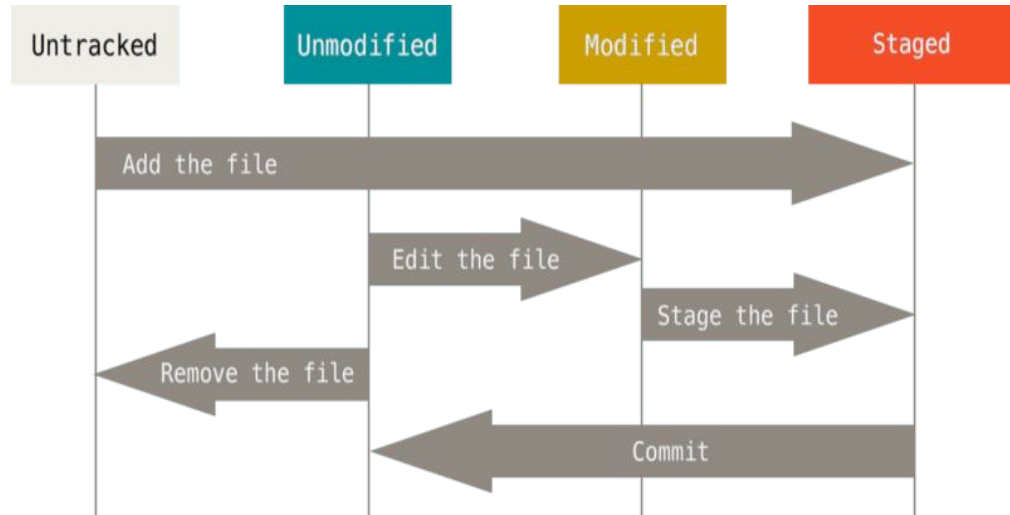
Git Basics - Exercise 4: Recording Changes to the Repository

- Conclusion: As you edit files, Git sees them as **modified**, because you've changed them since your last commit. You **stage** these modified files and then **commit** all your staged changes, and the cycle repeats.
- The main command you use to determine which files are in which state is `git status`

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.  
nothing to commit, working directory clean
```





Git Basics - Exercise 5: Viewing the Commit History

- After you have created several commits, or if you have cloned a repository with an existing commit history, you might want to look back to see what has happened. The most basic and powerful tool to do this is the `git log` command.
- When you run `git log`, you should get output that looks something like this:

```
$ git log
```

```
commit ca82a6dff817ec66f44342007202690a93763949
```

```
Author: XXXX <XXXX@gee-mail.com>
```

```
Date:   XXX XXX XXX 21:52:11 2008 -0700
```

```
    Updated readme
```




Git Basics - Viewing the Commit History

- By default, `git log` lists the commits made in that repository in reverse chronological order – that is, the most recent commits show up first.
- This command lists each commit with its checksum, the author's name and email, the date written, and the commit message.
- A huge number and variety of options to the `git log` command are available to show you exactly what you're looking for.
- You can use time-limiting options such as `--since` and `--until` and you can specify specific dates. You can also filter the list to commits that match some search criteria.
- The `--author` option allows you to filter on a specific author, and the `--grep` option lets you search for keywords in the commit messages.
- Check out <https://git-scm.com/docs/git-log> for more information.



Git Basics - Working With Remotes

- To be able to collaborate on any Git project, you need to know how to manage your remote repositories.
- You can also use remote repositories to store your project in the cloud.
- Remote repositories are versions of your project that are hosted on the Internet or network somewhere.
- These repositories can be accessed on GitHub.



Git Basics - Exercise 6: Working with Remotes

- To see which remote servers you have configured, you can run the `git remote` command. It lists the shortnames of each remote handle you've specified.
- If you've cloned the repository, you should at least see **origin** – that is the default name Git gives to the server you cloned from:

```
$ git remote
```

```
origin
```



Git Basics - Exercise 6: Working with Remotes

- You can also specify `-v`, which shows you the URLs that Git has stored for the shortname to be used when reading and writing to that remote:

```
$ git remote -v
```

```
origin  https://github.ncsu.edu/jnjacks3/GitHub-Workshops.git  (fetch)
```

```
origin  https://github.ncsu.edu/jnjacks3/GitHub-Workshops.git  (push)
```



Git Basics - Exercise 7: Adding Remote Repositories

- We know that the `git clone` command implicitly adds the `origin` remote for you. Here's how to add a new remote explicitly.
- To add a new remote Git repository as a shortname you can reference easily, run `git remote add <shortname> <url>`:

```
$ git remote add test  
https://github.ncsu.edu/jnjacks3/GitHub-Workshops.git
```



Git Basics - Exercise 8: Fetching and Pulling from Your Remotes

- To get data from your remote projects, you can run:

```
$ git fetch [remote-name]
```

- `git fetch origin` fetches any new work that has been pushed to that server since you cloned (or last fetched from) it.
- The command goes out to that remote project and pulls down all the data from that remote project that you don't have yet.



Git Basics - Fetching and Pulling from Your Remotes

- It's important to note that the `git fetch` command only downloads the data to your local repository – it doesn't automatically merge it with any of your work or modify what you're currently working on. You have to merge it manually into your work when you're ready.
- Running `git pull` generally fetches data from the server you originally cloned from and automatically tries to merge it into the code you're currently working on.
- After you run `git fetch`, you should have references to all the branches from that remote, which you can merge in or inspect at any time.
- Learn more about branching and merging at:
<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>



Git Basics - Exercise 9: Pushing to Your Remotes

- When you have your project at a point that you want to share, you have to push it upstream. The command for this is simple: `git push [remote-name] [branch-name]`. If you want to push your master branch to your origin server (again, cloning generally sets up both of those names for you automatically), then you can run this to push any commits you've done back up to the server:

```
$ git push origin master
```

- This command works only if you cloned from a server to which you have write access.



GitHub

- GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects.
- A large percentage of all Git repositories are hosted on GitHub, and many open-source projects use it for Git hosting, issue tracking, code review, and other things.
- So while it's not a direct part of the Git open source project, there's a good chance that you'll want or need to interact with GitHub at some point while using Git professionally.
- GitHub can be used to host your own projects or to collaborate with other projects that are hosted on GitHub.



A Little about GitHub

- If you want to work with Git locally, but don't want to use the command line, you can instead download and install the GitHub Desktop client at <https://desktop.github.com/>. There is a tutorial on [GitHub.com](https://github.com) that walks you through how to get started with GitHub Desktop.
- We will stick with GitHub browser version for the workshop.
- **Forking:** When you “fork” a project, GitHub will make a copy of the project that is entirely yours; it lives in your namespace, and you can push to it.
- **Branching:** You work in a different repository to diverge from the main line of development and continue to do work in that repository without messing with the main line.
- More to come about GitHub tomorrow!



Next Workshop: GitHub for Collaboration

- Creating an Account on GitHub
- The GitHub Flow
- Contributing to a Project in GitHub
- Maintaining a Project in GitHub
- Team Collaboration Tools
- Documenting your Projects on GitHub
- Being Social



Additional Resources

All content is licensed under the [Creative Commons Attribution Non Commercial Share Alike 3.0 license](https://creativecommons.org/licenses/by-nc-sa/3.0/)

Free Tutorials

- Pro Git book: <https://git-scm.com/>
- GitHub Guides: <https://guides.github.com/activities/hello-world/>
- GitHub On Demand Training: <https://services.github.com/on-demand/>
- Command Line Tutorial: <https://www.davidbaumgold.com/tutorials/command-line/>

Documentation

- git Documentation: <https://git-scm.com/documentation>
- GitHub help: <https://help.github.com/>