# Git is super exciting!

I <3 Google Presentation Themes

# OMG slide titles

What is Git?

- Git is a *version control system*.
- It allows people to work together on projects at the same time without unintentionally deleting each other's work.
- It allows users to work *locally*, only making changes to shared data and documents when ready.

# I really like these headers.

What is GitHub?

- GH is a place to host repositories, or projects.
- It is the most popular open source code repository in the whole world.
- It's widely considered to be a "resume for developers" — most tech employers will ask for your GH username.

# Flow, baby, flow

How is GitHub used?

- There are two main workflows for folks using GH.
- One is the *fork and pull* methodology: this lets anyone "fork" an existing repository and push changes to their personal version without sending them back to the source. Changes must then be pulled to the source repo intentionally.

# Flow, baby, flow (cont'd)

- The other is the *shared repository model*, where everyone working on a project is granted access to push to a single shared repository, and branching is utilized to isolate changes.
- Which are we going to use? I don't know! I've mostly worked with fork and pull in the past, but it's going to be up to Paul.

# Repos? Branches? What?

It all starts with a repository.

- Repositories, or repos, are collections of files and directories that make up your project, as well as a list of changes to those files that are made over the life of your project.
- Every repo has a main branch called maser, which is the master copy with the current state of the project.

# Repos? Branches? What? (cont'd)

- Other branches can be created for development or other purposes. This is optional, but useful.
- Most projects using the *shared repository model* have at least one development branch that runs alongside master to capture changes in development.
- Non-master branches can be pulled back into the master branch.

# Local vs. remote

Most GH users work on local branches.

- A local branch exists (yeah you guessed it) on your local machine. This can be created on your desktop or cloned from an existing repo.
- This allows you to do work on your own computer whenever — no internet connection required!

# Local vs. remote (cont'd)

Remote branches have to exist, too.

- The remote branch is the one that lives on GitHub (or on another server if you're not using GH, but it's not on your local machine).
- Having a remote branch/repo is smart: if you lose your work, it's on the server!
- The remote branch is referred to as the *origin*.

# The Three States

Git has three main states of existence:

- *Committed*: the data is safely stored in your local database.
- *Modified*: you have changed a file, but not yet committed it to the database.
- *Staged*: you have marked a file in its current version to go into your next commit.

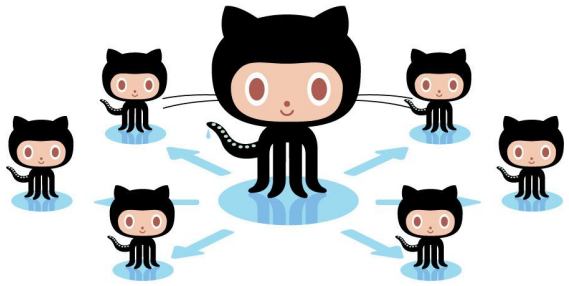
# The Three States (cont'd)

These three states correspond to three main sections of a git project:

- *Git directory (repo)*: This directory stores the metadata and object database for your project; it is what is copied when you clone or initiate a project.
- *Working directory*: a single working copy of one version of your project, pulled out of git directory and placed on disk for you to use/modify.

# The Three Stages (cont'd)

- *Staging area*: a simple file, generally contained in the git directory, that stores info about what will go in your next commit.

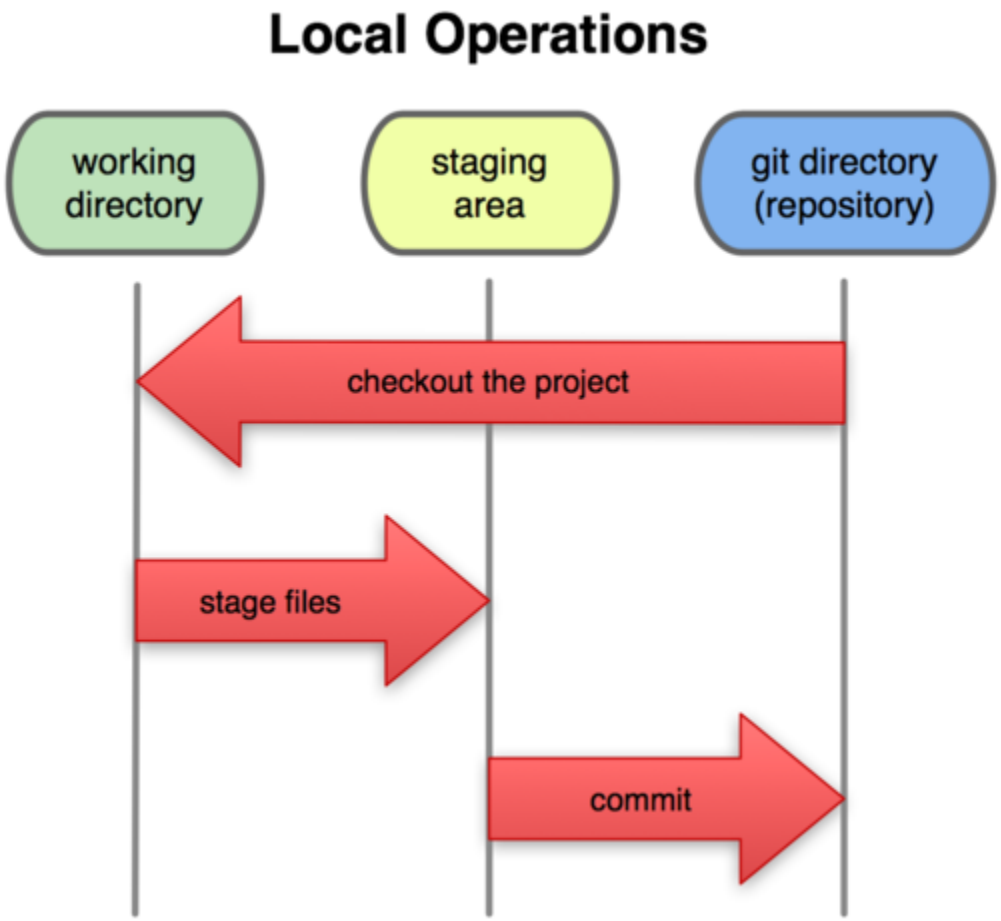


# So, basically...

The basic git workflow looks like this:

- You modify some 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.
- (optionally) You *push* the changes to origin.

# The Three Stages (cont'd)



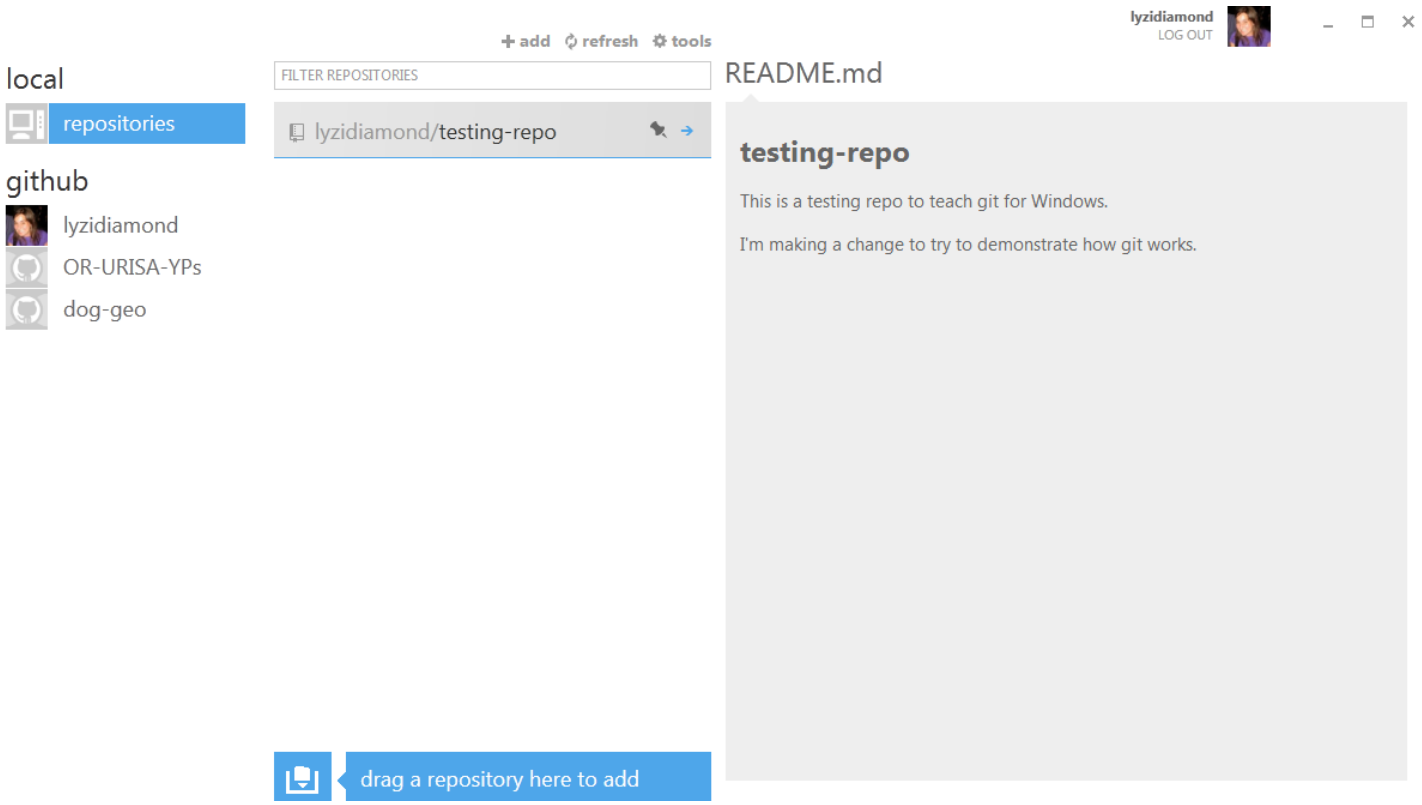
# Okay. So how do I get started?

Well, this kind of depends. We can either use the GUI, or use git through the command line. Both are useful, so I'll instruct on both, and we can decide later.

*For the command line:* First, we need to download git. This is easy. Go to git-scm.com/download/win and download it. When installing, use all the default settings.

# Okay. So how do I get started?

*For the GUI:* Download the GUI. Head to windows.github.com and download it.



# I am not good at presentations.

So, this is how it works.

- The basic workflow for a git project is to either initialize a repo or clone and fork a repo.
- Both of these start on github.com. Either click "New Repository" on your github page or click "Fork" on the page of the repo you want to use.

# I am trying to get better.

- Now that we have a repo in our github, we need to get it on our local machine. This is called cloning.
- On the command line, the way to clone a repo is by typing the following:
  - *git clone https://github.com/lyzidiamond/test.git*
- In the GUI, you simply click on the repo you want to clone and press the Clone button.

# These slides have too much text.

- Let's say I forked the *test* repo from Paul. If he changes anything, I want to know about it! I can do so by creating another remote (besides *origin*) called *upstream*.
- First, I want to navigate to my new folder that was created when I cloned *test.*
  - *cd test*
- Now I want to add Paul's version of *test* as my upstream.
  - *git remote add upstream https://github.com/paulferro/test.git*

# I am breaking all the rules.

- When I want to get Paul's changes, I can do so by *fetching* data from the upstream. This will not affect any changes I've made.
  - *git fetch upstream*
- Awesome! Now you can get to work making and adding files to your repo. So long as you save a file in the *test* folder, you can commit the changes later.

# ALL THE BULLET POINTS

- When you're ready to commit, you go through the three step process we went through before. Remember to fetch changes first!
  - *git fetch upstream*
- First, add the files that you wish to commit.
  - *git add filename1.py*
- Then, make a commit.
  - *git commit -m "This is my first commit"*
- Then, push the commit back to GH.
  - *git push origin master*

# Some things we can talk about later.

There's more to this than I've gone over. There are many tools that can be useful for git and github, but this is a good list of the basics. Other stuff I didn't talk about:

- Branching and merging
- Snapshotting, tracking, and updating (except fetch)
- Rebasing
- GH for social purposes.

# FUN EXERCISE TIME!

Let's talk it through!

Also, here is octocat as a ninja.

# An example, using fork and pull.

Let's say Paul and Matt are working on a project together. Paul created the repository, which he called ArcGIS-scripts. Matt has forked this project, so now he has a copy of the same repository. Matt cloned the project to his machine. He wrote a new script called script1.py. He added it to the folder on his machine called ArcGIS-scripts that was created when he cloned the repo.

# An example, using fork and pull.

Matt has modified files, now he wants to update the remote repo (living on github.com/matt/arcgis-scripts). He adds the new file to the staging area by typing *git add script1.py* into the command line. He then commits the staged changes by typing *git commit -m "Matt's changes."* The part in quotes after -m is the comment that will appear next to his commit.

# An example, using fork and pull.

Matt then needs to push to his remote repository. He does so by typing *git push origin master* into the command line. His changes are then pushed to the master branch of his remote repository. (It was called origin, remember?) Let's say that Paul and Matt have decided to reconcile their changes once per week. That day rolls around, and Matt wants to add script1.py to the main repo that Paul maintains.

# An example, using fork and pull.

Matt initiates a pull request with Paul. He does so by heading to Paul's version of the ArcGIS-scripts repository (github.com/paul/arcgis-scripts) and clicking on the *Pull Request* button. Matt fills out the little form and sends the pull request to Paul. Paul sees the pull request and looks over the proposed change (the addition of script1.py). He deems them satisfactory.

# An example, using fork and pull.

Paul is then tasked with adding the changes from the remote branch (Matt's master) to the master branch (Paul's master). Paul adds a new remote for Matt's repo by typing *git remote add Matt git://github.com/matt/arcgis-scripts.git*. He pulls in all the commits from Matt's repo by typing *git fetch Matt*. He merges those commits with his branch by typing *git merge Matt*. He pushes it all back to GH with *git push origin master.*