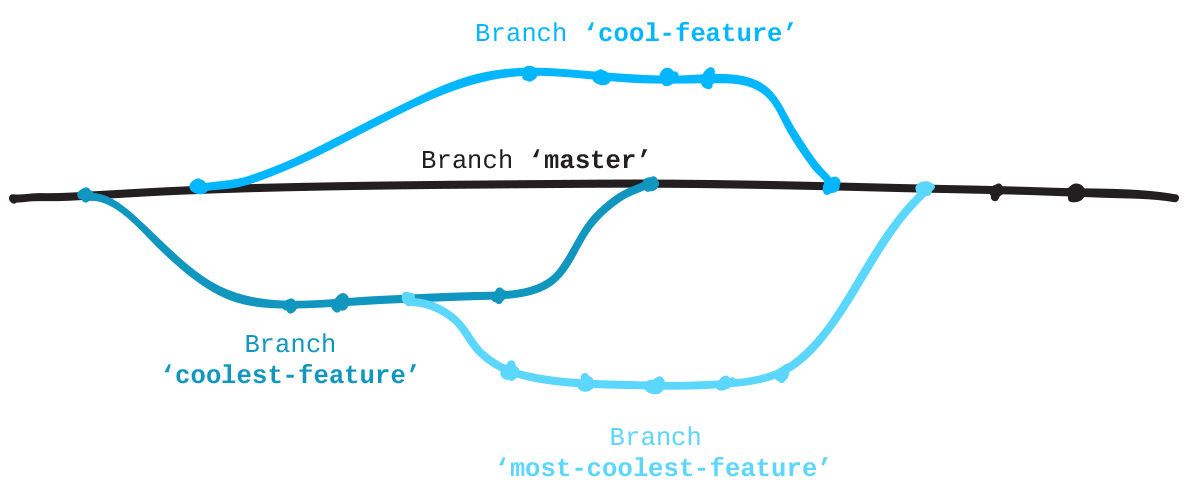
Best Practices in GitHub

* GitHub Enterprise
  + Most of this course's materials are available on General Assembly's GitHub Enterprise, which you can access via cloning, forking, and pulling.
  + It's important to remember that each group or company's GitHub Enterprise (GHE) functions as its own entity.
  + So, for instance, if Google invited you to join its GHE, you would need to create a new account for Google’s Github Enterprise even though you already have a GHE account for General Assembly.
  + Within each GHE, you create an individual account and therefore your own workspace. Other people can view the copies of your repositories but won't be able to contribute to them without your permission.
* Why Developers and Data Scientists Use GitHub:
  + Reason 1: Controlled Collaboration
    - **Controlled collaboration**: With Google Docs, several people can edit the same document at the same time, which is useful. However, a Google Doc collaboration can easily run amok. Changes can be made haphazardly and it's not easy to identify who made what revision and when.
    - For instance Linus, in a burst of inspiration, could overwrite Petunia's brilliant paragraph only to later realize that the text he entered was sloppy and full of errors.
    - With GitHub, collaborators create their own versions of repositories called branches. They can edit these and then merge them back into the master branch.
    - Another collaboration tool is a pull request. This allows administrators to view changes from other collaborators' branches and decide if they should be implemented.
    - And then there is the changelog. This is a record of the changes that individuals make to a repository. The log makes it easy to identify who did what and when.
  + Reason 2: Social Media
    - **Social media:** GitHub is also a portfolio that allows you to showcase your programming skills and knowledge via posting projects, useful code snippets, and gists.
    - Another social function is forking. When you fork, you create a copy of an existing repository. The best part is that, when the original repo is updated, you can fetch that update to your fork.
* Branching and Pull Requests
  + Oftentimes when we work on a group project, collaborators will make changes to the same repo at different times. That's where branches come in.
  + With branches, we can create a branch off the main repo (or "master branch"), perform our work (e.g., additions or alterations), and then merge the changes we've made in our branch back into the master.
  + As you can imagine, when multiple people try to merge their work into the master branch, it can get ugly. Fortunately, we can use pull requests to queue and validate our merges.
  + Rather than having a branch merge automatically, the changes go through an administrator, who reviews them and then approves or denies the pull request.
  + 
  + Pro Tip: Even though we are trying to "push" our information into the master branch, we call it a pull request because we are making a request, asking an administrator to "pull" our branch into the master.
* Branching and Pull Requests in Action
  + Want an example of the kind of work that involves branching and pull requests? You're looking at one! Yup, these lessons were created via GitHub. The GA team started with a master branch, and several subject matter experts were each assigned a single unit.
  + These experts created branches off the master and, within those branches, built out their assigned units. Once they completed the material for a given unit, they submitted a pull request to the repository's administrator, who reviewed their work and either pulled the branch into the master or communicated back additional changes that needed to be made.
* Common GitHub Issue No. 1: Merge Conflict
  + A merge conflict occurs when there are competing changes to a repository. Recall our example:
    - Petunia and Linus create branches at the same point in time off of the master and they both alter the same line of code. Petunia's branch is merged into the master. Linus' branch has a different recollection of the master, so a conflict arises about the change it believed it was going to make during the merge.
  + Fortunately, most merge conflicts can be avoided by keeping your branch up to date with the master. If a merge conflict does arise, GitHub offers the ability to override simple merge conflicts by either accepting or rejecting the conflicting changes.
* Common GitHub Issue No. 2: Clogged Pipeline
  + A clogged pipeline occurs when a user attempts to push a file with a size that exceeds the standard amount of data you can push at a given time (100MB). Note that, throughout this course, you will work with files sizes exceeding this limit.
  + Once again, preventative action is best. You can create a .gitignore file, which will prevent unwanted files from being committed, or utilize the Git Large File Storage package.
  + In the event you have a large file issue and forget to take these steps, you'll need to reset the head of your branch to a commit from before the large file was added. Because you're only resetting the head and not the working directory, none of the changes to your local repository will be erased.
* Resetting a Branch
  + There may come a time when you accidentally erase an entire file or create something you shouldn't have. Don't panic! A branch can easily be reset to the last-known state of the master via a hard reset. Remember, however, that any uncommitted work will be difficult to retrieve after a hard reset.
  + git fetch origin
  + git reset --hard origin/master
* Secure Shell (SSH)
  + While using https as the connection and your local and remote repositories is relatively secure, it's a good idea to include an additional level of security. If you've cloned a repository, you may have noticed an additional option for SSH.
  + SSH, or secure shell, is a common way to add an additional layer of security. Simply put, the SSH key establishes authenticity between a client and a server so that a secure connection, or "tunnel," can be formed. This is useful for secure file sharing or remote application access.
  + How does SSH work? It's fairly simple:
    - The client makes a request to the server.
    - The server responds by asking for authentication.
    - The client provides authentication.
    - If the authentication is correct, a connection is established.