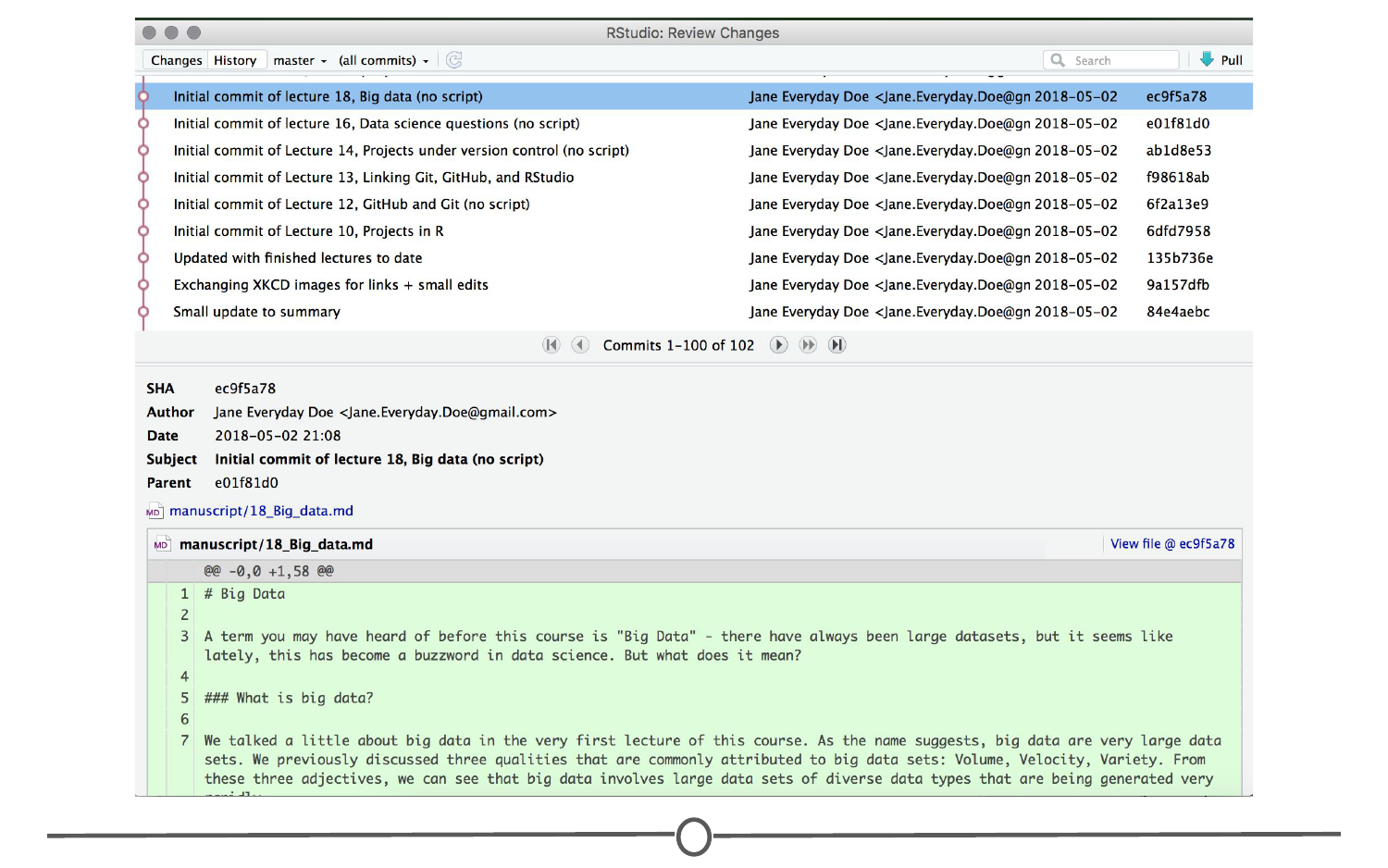
# Online repositories-Version control

Version control is a system that records changes that are made to a file or a set of files over time. As you make edits, the version control system takes snapshots of your files and the changes, and then saves those snapshots so you can refer or revert back to previous versions later if need be.

**Benefits of using version control.**

* Without version control, you might be keeping multiple, very similar copies of a file, you might start editing the wrong version, not recognizing that the document labelled “FINAL” has been further edited to “FINAL2” - and now all your new changes have been applied to the wrong file!
* Keep a single, updated version of each file, with a record of all previous versions AND a record of exactly what changed between the versions.
* keeps track of who, when, and why those specific changes were made.
* when working with a group of people on the same set of files, version control is helpful for ensuring that you aren’t making changes to files that conflict with other changes.
* Version control allows multiple people to work on the same file and then helps merge all of the versions of the file and all of their edits into one cohesive file.



GitHub

GitHub is an online interface for Git. Git is software used locally on your computer to record changes. GitHub is a host for your files and the records of the changes made. You can sort of think of it as being similar to DropBox.

Version control vocabulary

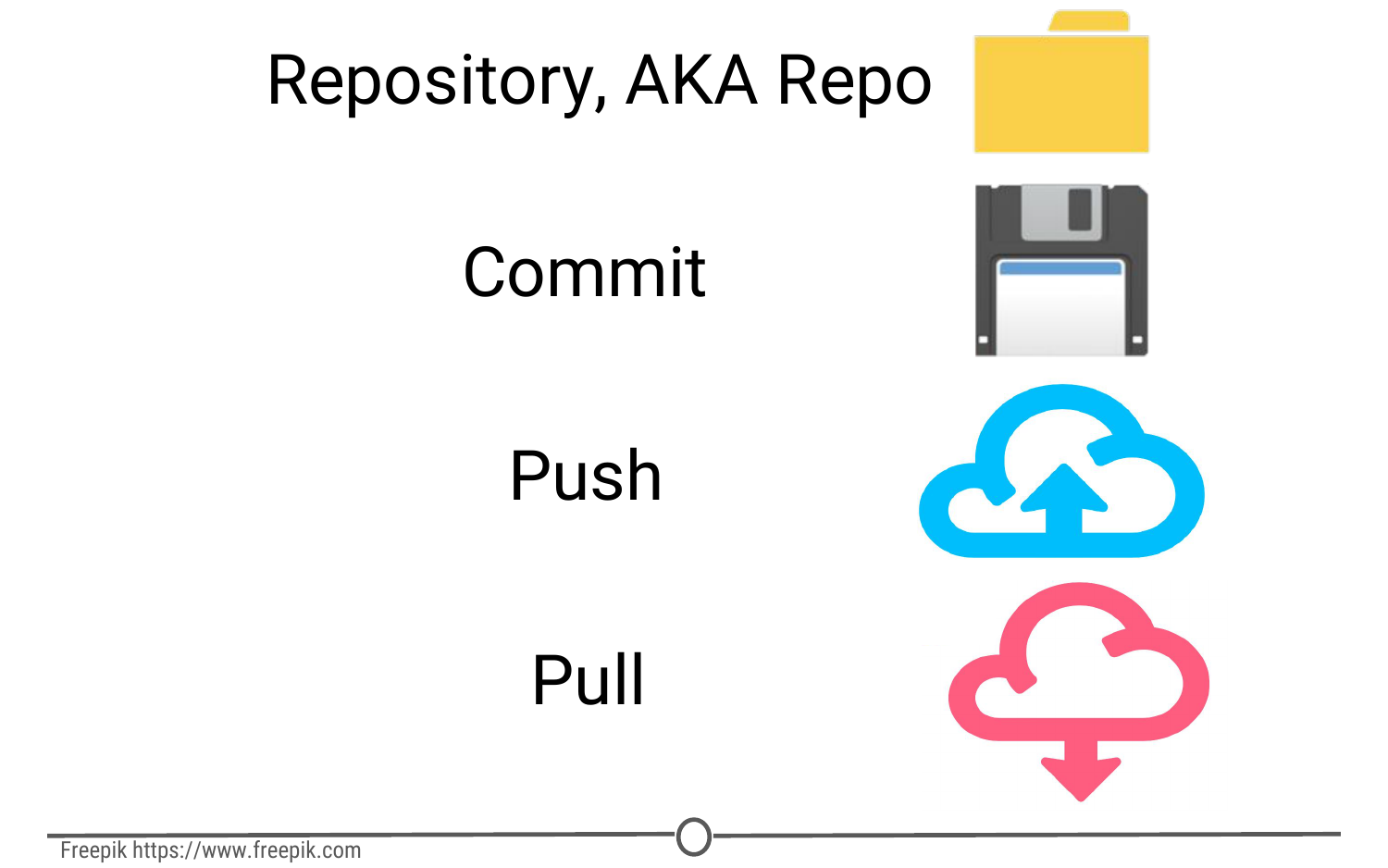
**Repository:** Equivalent to the project’s folder/directory - all of your version controlled files (and the recorded changes) are located in a repository. This is often shortened to **repo**. Repositories are what are hosted on GitHub and through this interface you can either keep your repositories private and share them with select collaborators, or you can make them public - anybody can see your files and their history.

**Commit:** To save your edits and the changes made. A commit is like a snapshot of your files: Git compares the previous version of all of your files in the repo to the current version and identifies those that have changed since then. When you commit a file, typically you accompany that file change with a little note about what you changed and why.

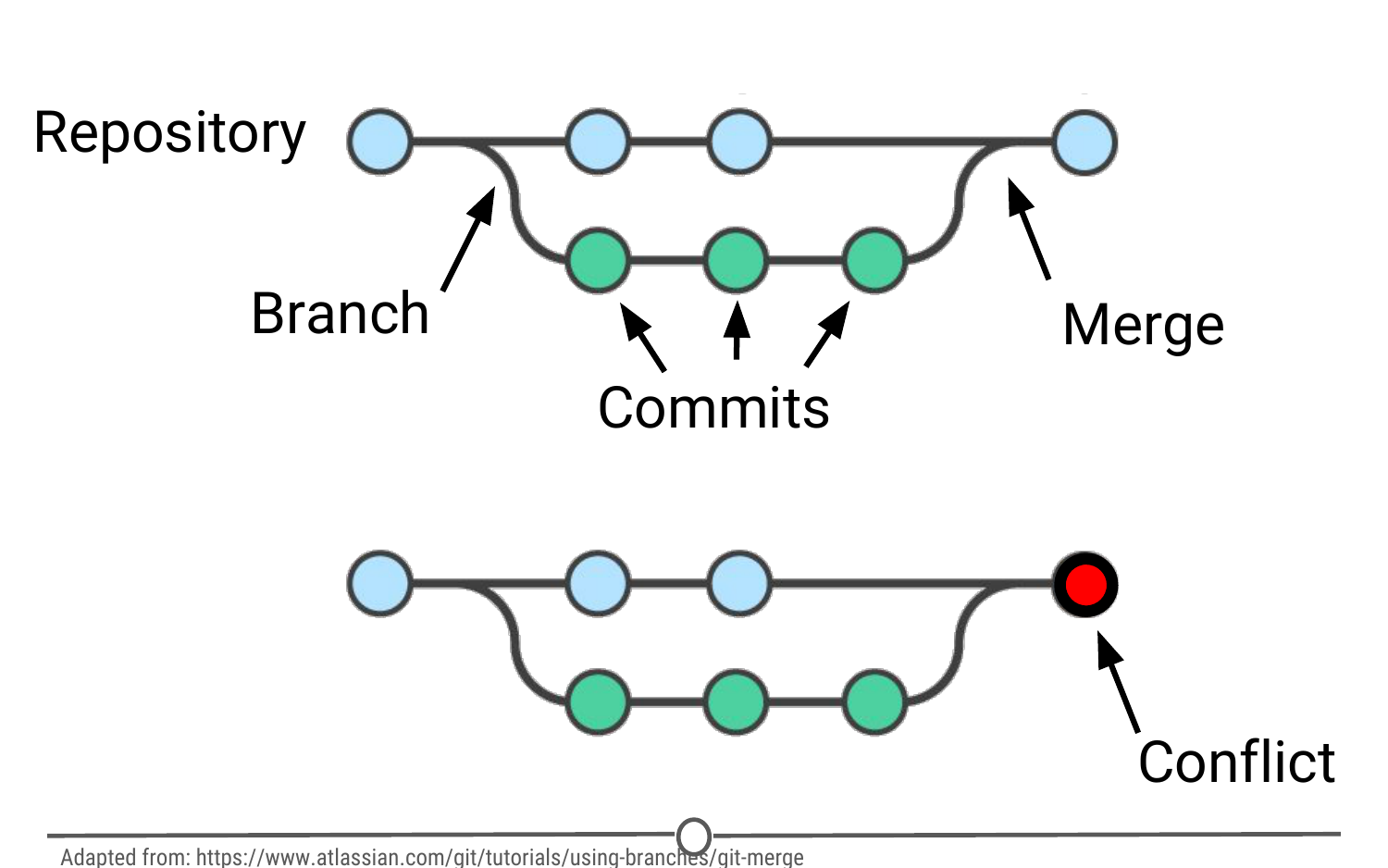
When we talk about version control systems, commits are at the heart of them. If you find a mistake, you revert your files to a previous commit. If you want to see what has changed in a file over time, you compare the commits and look at the messages to see why and who.

**Push:** Updating the repository with your edits. Since Git involves making changes locally, you need to be able to share your changes with the common, online repository. Pushing is sending those committed changes to that repository, so now everybody has access to your edits.

**Pull:** Updating your local version of the repository to the current version, since others may have edited in the meanwhile. Because the shared repository is hosted online and any of your collaborators (or even yourself on a different computer!) could have made changes to the files and then pushed them to the shared repository. The files you have locally on your computer may be outdated, so you pull to check if you are up to date with the main repository.



**Staging:** The act of preparing a file for a commit. For example, if since your last commit you have edited three files for completely different reasons, you don’t want to commit all of the changes in one go; your message on why you are making the commit and what has changed will be complicated since three files have been changed for different reasons. So instead, you can stage just one of the files and prepare it for committing.



To summarize these commonly used terms so far and to test whether you’ve got the hang of this, files are hosted in a **repository** that is shared online with collaborators. You **pull** the repository’s contents so that you have a local copy of the files that you can edit. Once you are happy with your changes to a file, you **stage** the file and then **commit** it. You **push** this commit to the shared repository. This uploads your new file and all of the changes and is accompanied by a message explaining what changed, why and by whom.

**Branch:** When the same file has two simultaneous copies. When you are working locally and editing a file, you have created a branch where your edits are not shared with the main repository (yet) - so there are two versions of the file: the version that everybody has access to on the repository and your local edited version of the file. Until you push your changes and merge them back into the main repository, you are working on a branch. Following a branch point, the version history splits into two and tracks the independent changes made to both the original file in the repository that others may be editing, and tracking your changes on your branch, and then merges the files together.

**Merge:** Independent edits of the same file are incorporated into a single, unified file. Independent edits are identified by Git and are brought together into a single file, with both sets of edits incorporated. But, you can see a potential problem here - if both people made an edit to the same sentence that precludes one of the edits from being possible, we have a problem! Git recognizes this disparity (**conflict**) and asks for user assistance in picking which edit to keep.

**Conflict:** When multiple people make changes to the same file and Git is unable to merge the edits. You are presented with the option to manually try and merge the edits or to keep one edit over the other.

Best practices

* Make purposeful commits. Each commit should only address a single issue. This way if you need to identify when you changed a certain line of code, there is only one place to look to identify the change and you can easily see how to revert the code.
* Make sure you write informative messages on each commit. If each message is precise in what was being changed, anybody can examine the committed file and identify the purpose for your change
* if you are looking for a specific edit you made in the past, you can easily scan through all of your commits to identify those changes related to the desired edit.
* be cognizant of the version of files you are working on. Frequently check that you are up to date with the current repo by frequently pulling.
* If you are done editing a section of code and are planning on moving on to an unrelated problem, you need to share that edit with your collaborators

