Having version control is like having undo button for your project.

GIT + github

Diff builtin tool to find diferences in a file

diff –u file1 file2

To get versions

We can do manually savings

Dropbox

Google Docs

Wikipedia

As a programmer when would you want to have a version of your code saved?

Ar regular intervals.

The user creates checkpoints called commits. Records the version of the content at one point in time requiring a commit message

Git log (to see the commits starting with the most recent) message explains what changes

Git log –-stat (gives more estatistics). To stop seeing the log press q (quit)

Git diff (compares diferent versions of a file withing git) id1 id2

It's usually a good idea to keep commits small. A good rule of thumb is to make *one commit per logical change*.

Many projects contain a file named "README" that gives a general description of what the project does and how to use it. It's often a good idea to read this file before doing anything with the project, so the file is given this name to make users more likely to read it.

Files you want to track together are called a repository (high related files). When commiting updates all the files in a commit.

Git has a command that lets you copy one repository (with its historial) from one computer to another.

Git clone url

**Getting Colored Output**

To get colored diff output, run git config --global color.ui auto

**Should not be doing an octopus**   
Octopus is a strategy Git uses to combine many different versions of code together. This message can appear if you try to use this strategy in an inappropriate situation.

**You are in 'detached HEAD' state**   
HEAD is what Git calls the commit you are currently on. You can “detach” the HEAD by switching to a previous commit, which we’ll see in the next video. Despite what it sounds like, it’s actually not a bad thing to detach the HEAD. Git just warns you so that you’ll realize you’re doing it.

Git checkout + commit id --> reset all your files to a time when the change was made (to a commit)

**Compare two commits, printing each line that is present in one commit but not the other.**

git diff will do this. It takes two arguments - the two commit ids to compare. 

**Make a copy of an entire Git repository, including the history, onto your own computer.**

git clone will do this. It takes one argument - the url of the repository to copy. 

**Temporarily reset all files in a directory to their state at the time of a specific commit.**

git checkout will do this. It takes one argument - the commit ID to restore. 

**Show the commits made in this repository, starting with the most recent.**

git log will do this. It doesn't take any arguments.

git log lists the most recent commit first, as you can verify by checking the commit dates. The middle commit probably contains the code for the mute button, since the commit message indicates that the mute button was added in that commit. The top commit also probably contain the mute button, since that commit is more recent and nothing suggests the mute button has been removed. The bottom commit probably does not contain the mute button, since that commit was created before the commit that added the mute button.

What makes a repository a repository? When you get into the repository you see the same files but inside there are a bunch of metadata of the history of the respository since it was created. It’s in the main directory in hidden .git folder

Making a new repository and commit (without connection)

Git init (initializes a git repository)

When you initialize the repository, Git doesn’t create any commits for you. You’ll have to create the first commit yourself.

Git status (it shows which files had changed since the las commit, so I will run it frequently)

Git add + filename (to add a file into the stagging area, to see what files are going to be commited run git status.

Git reset

You can also specify a commit message via the command line by running git commit -m "Commit message"instead of just git commit. It's still a good idea to get an editor set up, since this will make it easier to write long commit messages that fully describe the change

You should run git checkout master before you commit

git diff (highlight the changes you had made to your files (working directory) in contrast with changes in stagging area.

Git diff --staged (compares the changes in your stagging area and your most recent commit on the repository).

Git reset --hard (discards any changes in either the working directory or staging area). Be careful when you run this command. Be sure to check changes before to run .

Branches work by trying experimental features, testing a/B, etc while continuing to keep the working version. Labels are branches for commits. Master is the main branch in git repositories.

You can combine two commits, its called marging.

Git branch (shows you the current branches). Git branch nameofthenewbranch (creates a new branch).

To change between branches run git checkout branchname and check via git branch.

git log --graph --oneline master coins

(displays a graph) of commits

Remote branch means i hadn’t create it. But behaves = as the branches I created myself.

Each commit knows his parent commit. Reachability (you can not reach any commit)

Git checkout –b new\_branch\_name = running two commits git branch new\_branch\_name and git checkout new\_branch\_name (you do it everytime you try a new feature).

If a branch is deleted and leaves some commits unreachable from existing branches, those commits will continue to be accessible by commit id, until Git’s garbage collection runs. This will happen automatically from time to time, unless you actively turn it off. You can also run this process manually with git gc.

Combine branches

We can do it automatically. Known as merge branch into master.

Git merge master branchName (it propmts a commit to leave a coment)

**Comparing a commit to its parent**

Git show commit\_id (to see the difference between a commit and its parent)

Since I don’t need anymore a branch label I run git branch –d branchName

### A note about git merge

git merge will also include the currently checked-out branch in the merged version. So if you have branch1 checked out, and you rungit merge branch2 branch3, the merged version will combine branch1 as well as branch2 and branch3. That’s because the branch1 label will update after you make the merge commit, so it’s unlikely that you didn’t want the changes from branch1 included in the merge. For this reason, you should always checkout one of the two branches you’re planning on merging before doing the merge. Which one you should check out depends on which branch label you want to point to the new commit.

Since the checked-out branch is always included in the merge, you may have guessed that when you are merging two branches, you don't need to specify both of them as arguments to git merge on the command line. If you want to merge branch2 into branch1, you can simplygit checkout branch1 and then type git merge branch2. The only reason to type git merge branch1 branch2 is if it helps you keep better mental track of which branches you are merging.

Also, since the two branches are merged, the order in which they are typed into the command line does not matter. The key is to remember that git merge always merges all the specified branches into the currently checked out branch, creating a new commit for that branch.

Git merge –abort if there is a conglict

Git log --grap --oneline

Git log –n1

Github

Syncing repository wiht github.