**Learning about git terminology through github**

In this first activity we will learn some basic git terminology by creating repo (short for repositoy) on github.

**What you will need:**

* A web browser
* A github account

**Create a new git repo**

* log into github
* click + sign at top right and select new repo
* choose a name for this repo (suggestion: gitwsrepo1)
* this repo can be public
  + check the the "Initialize with a README" checkbox
  + hit create button

**What you get with each repo**

Once you have a repo in github, there are a number of things that come with the repo. Notice the row of tabs at the top of the screen. The first tab name code is essentially every file your repo tracks. Currently there is a single README.md file. The other tabs are:

* issues
* pull requests
* project
* wiki
* insights

(these have changed over time and may change again in future). A repo in other words can help you organize a project, provide space for feedback, documentation, and some basic analytics. For this first activity we will use only the code tab

**Commiting a new file**

* Make sure you are in the code tab
* Hit the new file button
* give the file a name (suggestion: alphabet.txt)
* Edit the file by adding the following to it:

ABCDEFG

HIJK

QRS

* hit the commit new file button (keep the default Commit directly to the master branch setting in the radio buttons)

This file is now known to git. git is storing a version of this file

**Modify the file**

* click link for the file you created in the last step
* hit the pencil icon in top right to edit the file
* insert the following line after "HIJK"

LMNOP

* then click commit changes button

**Lets look at the history**

* click on the file in code tab
* click the history button (top right)
* notice there are two commits each has a 7 digit hex code associated.
* This 7 digit hexcode is actually the beginning of a much longer hex code that uniquely identifies each commit. These are hash codes.
* notice you can look at the state of each file in each commit
* notice you can see what was changed from the previous commit in the history
* + and green indicate additions
* - and pink indicate deletions

**Branches**

* A key idea that makes git unique is the ease with which you can make and merge branches
* A branch is made when you want to add something but you aren't totally sure if you want to keep it
* By default when you create a repo in git, you create a branch named master \*\* You do not have to keep this branch name...there isn't anything special about it
* notice at the top left there is a dropdown with the word master in it. This is the branch you are looking at.
* more accurately it is the most recent commit within that particular branch
* To create a branch inside github click the dropdown and type a name for your new branch (suggest: alpha1)
* You can click between the two branches by using the dropdown button
* Make sure you are on your new branch (ie NOT master)
* Edit the file by typing: Now I know my ABC's to the bottom of the file
* Commit this file (the default setting should be committing file to new branch

**Look at the history**

* At this point take a quick look at the history
* in the master branch there will be one commit less than the other branch

**Look at Network graph**

* In the insight tab, click graphs
* choose network graph
* The network graph is a visual representation of your branches

**Merge, Compare and pull request**

* Go back into your code tab
* At this point you should have a bar with a green button at top that says "compare and pull request"
* This is because github realizes you have something in a branch that is different from master and is asking if you want to create a request to review and and accept the changes
* Hit this button
* Now, this button does not actually merge the files into master...it creates a pull request. In otherwords it creates a request to ask if you want to merge it back together.
  + You can also make your own pull request even without that button, just go to pull request tab and hit new pull request
  + Notice the two buttons at the top. One is labelled base. The other compare.
  + Base is where you want the changes to go...which branch do you merge to
  + Compare is the changes you want to pull into the Base ... which branchdo the changes come from
* In the pull request tab, you will now see merge pull request button
* Click this button to pull the changes in your second branch back to master
  + Confirm the merge
  + DO NOT DELETE THE BRANCH

**Branching, merging, and conflicts**

* Be in the master branch
* Create a third branch (suggest: alpha2)
* In the **master** version commit (after the line QRS:

TUV

* In the **alpha1** version commit:

Next time won't you sing with me

to the last line

* In the **alpha2** version commit (after the line QRS):

WX

* Now lets go to the pull request tab and start a new pull request
* Use master as the base
* for compare switch between your two branches.... you will notice that alpha1 is followed by green checkmark and the text able to merge. If you switch to alpha2 it is followed by red X and the text can't automatically merge
* Essentially this is github saying that it knows how to put together master and alpha1. It can figure this out on its own. However not master and alpha2 that it needs help with.
* Create a pull request to master from alpha1
* Create a pull request to master from alpha2 (yes even though it can't be done automatically)
* Lets merge in the easy one first, go to the pull request that can be automatically merged and merge it.
* In the other pull request hit the Resolve conflict button
* When you do this, you will open an editor that has what you wrote plus a number of conflict markers like the following: <<<<<<< alpha2
* These conflict markers mark the places where git got confused. It basically can't figure out what do...
  + do you want to keep the version in alpha2 and erase version in master?
  + do you want to keep the version in master and erase version in alph2?
  + do you want to keep both?
  + do you want to switch the order?
* Your job is to figure out which parts of each version you keep and the ordering of the text.
* Edit the file so that it is correct and remove all the lines containing the conflict markers. Hit mark as resolved button
* hit the commit merge button
* finish the pull request now that it can be automatically merged

**Summary**

Here are the ideas and terms we looked at:

* repository
* commit
* branch
* merge
* conflict
* pull request
* history
* uniquely identifying hash codes

# activity 2

Catherine Leung edited this page on Jun 26 · [7 revisions](https://github.com/cathyatseneca/gitworkshop-s17/wiki/activity-2/_history)

### Pages 5

* [**Home**](https://github.com/cathyatseneca/gitworkshop-s17/wiki)
* [**activity 1**](https://github.com/cathyatseneca/gitworkshop-s17/wiki/activity-1)
* [**activity 2**](https://github.com/cathyatseneca/gitworkshop-s17/wiki/activity-2)
* [**activity 3**](https://github.com/cathyatseneca/gitworkshop-s17/wiki/activity-3)
* [**introduction**](https://github.com/cathyatseneca/gitworkshop-s17/wiki/introduction)

##### Clone this wiki locally



[**Clone in Desktop**](https://desktop.github.com/)

This activity will essentially look at how we did everything we did in activity 1 via the github interface using git commands locally. When we write code we aren't going to do this inside the github editor... it doesn't suit our needs. Thus, it is essential that we understand how the same ideas we looked at (repo creation, commits, branch, merge etc.) can be applied to a repo we create locally.

## What you will need:

* A text editor (atom on the macs is a good one)
* A terminal (on the macs, use magnifying glass at top right of menu and type terminal)
* if you are using your own machine, you want git bash and some text editor

## Create a new git repo

* Make a folder (put it in a place you can easily cd into from the terminal)
* Open the terminal and cd your way into that folder
* Put the folder under revision control by using the command: git init
* Execute the command: ls -a
* Notice you now have a **.git** hidden folder. This folder is our repository... our data base. DO NOT modify the contents of your .git folder, ignore the fact that it is there
* Type the command: git status and you should see:

On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

* So you see a few things...
  + the branch you are looking at is master
  + The repo has nothing (at Initial Commit)

## Commiting a new file

* Using your text editor create a file containing

ABCDEFG

HIJK

QRS

* save it into the folder you made
* type the command: git status
  + and you should see:

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

alphabet.txt

nothing added to commit but untracked files present (use "git add" to track)

* The above means that git is aware that you have an alphabet.txt file in your folder but that the file is not being tracked at all by the git database.
* To put into the database (ie into version control) you need to execute two commands:

git add alphabet.txt

git commit -m "added 3 lines of the alphabet song"

* Why two commands and not one?
  + git tracks changes in the state of the file system by having a history of commit
  + a commit is a change in one or ***more*** files
  + since a commit can involve multiple files, we need to indicate all the files that were changed for one commit
  + think of add as a "packing" command. You are packing up all the things you want for a commit into one.
  + Once you finish adding it all together, you then commit it.
* Type the command: git status
  + you should now see:

On branch master

nothing to commit, working tree clean

* The above means that git now knows about ever file in the folder. There aren't any changes in any file that git doesn't have a record of

## Modify the file

* edit your file by
* insert the following line after "HIJK"

LMNOP

* save the file
* type the command: git status you will see

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: alphabet.txt

no changes added to commit (use "git add" and/or "git commit -a")

* This indicates that the version of alphabet.txt that git knows about is not the same as the one in your folder
* Use the add and commit commands to commit this change

git add alphabet.txt

git commit -m "added LMNOP"

* Again if you use git status you will see:

On branch master

nothing to commit, working tree clean

## Lets look at the history

* Use the command: git log
* this will show all the commits for the branch along with its hash and commit messages

## Branches

* Lets make a branch called alpha1 git branch alpha1
* Look at where you are atm: git status
  + you should see:

On branch master

nothing to commit, working tree clean

* To switch branches:

git checkout alpha1

* Now take a look at your status again: git status
  + you should see:

On branch alpha1

nothing to commit, working tree clean

* To switch back to master you would use git checkout master
* Switch into your alpha1 branch
* Modify your file by adding: Now I know my ABC's to the bottom of the file
* Commit this file to your alpha1 branch

## Look at the history

* Checkout each of your branches and look at the logs
* Note to see what branches you have: git branch -v

## Look at Network graph

* Use the command: git log --graph

## Merge

* In git you do not really have pull requests... you can pull code in, you can merge it... but the process is different
* To merge what is in alpha1 into master
* Make sure you are sitting in master git checkout master
* Merge alpha1 into master: git merge alpha1

## Branching, merging, and conflicts

* Be in the master branch: git checkout master
* Create a third branch (suggest: alpha2): git branch alpha2
* We should now have 3 branches (master, alpha1 and alpha2). You can see if this is the case by using: git branch -v
* Go into master branch git checkout master
  + In the alphabet.txt add the following(after the line QRS:

TUV

* save file
* add and commit

git add alphabet.txt

git commit -m "added TUV"

* Go into alpha1 version
* Add to the end of alphabet.txt the following line:

Next time won't you sing with me

to the last line

* save file
* commit and add
* Go into alpha2 version
* In the alphabet.txt add the following(after the line QRS:

WX

to the last line

* save file
* commit and add
* Look at the network graph at this point: git log --graph --all --decorate --topo-order --oneline

Now, if you will recall, on github we were able to easily merge alpha1 to master but not alpha2. The same thing will happen here.

* Lets start by merging alpha1 into master:

git checkout master

git merge alpha1

* This should be no problem
* Now lets try to merge alpha2 into master:

git merge alpha2

* this should create a merge conflict
* Using your text editor resolve your conflict in the same manner as you did on github
* save file
* add and commit

## Summary

git commands:

* git init (creates repo)
* git add (add files)
* git status (tells you what branch you are on and what files are known to git and which aren't)
* git log (shows the history of the branch we are in)
* git log --graph --all --decorate --topo-order --oneline (an ascii graph)
* git commit -m "" (puts all things you added into a commit)
* git branch (branch commands)
* git branch -v (list branches you have)
* git checkout (go into a branch)
* git checkout (reverts your file to the version on the tip of branch you are sitting in)

**Git Vs. Github**

git is a revision distributed revision control system. github provides a web interface that makes working together using git easier to do.

**Revision Control systems**

* CVS
* SVN
* git
* mercurial ...

There are many out there. recently git has become very popular

**Git**

* distributed (no central server)
* fast
* think of it as version file system

**Github**

* provides infrastructure for sharing repositories user management
* public vs private repos
* free for open source (don't open source your homework...its against academic policy)
* will provide free micro plan (private repos) for students (get it at education.github.com)