# Github Instructions/Info

Stephen J D Kay
stephen.kay@uregina.ca
University of Regina
22/01/2021

## 1 Introduction

This document outlines the steps required to setup the relevant repositories for the analysis of the PionLT and KaonLT experimental data. I will outline the steps you should follow and try to provide some extra contextual info where applicable.

## 2 Repositories

Are variety of repositories are utilised in the analysis of our data. The relevant repositories are -

- hallc_replay_lt - https://github.com/JeffersonLab/hallc_replay_lt
  - This is the main repository which contains replay scripts, calibrations, standard.kinematics and so on
- UTIL_BATCH - https://github.com/JeffersonLab/UTIL_BATCH
  - This repository contains various shell scripts.
- UTIL_PION - https://github.com/JeffersonLab/UTIL_PION
  - This contains the relevant scripts for the analysis of the pion data.
- UTIL_KAONLT - https://github.com/JeffersonLab/UTIL_KAONLT
  - This contains the relevant scripts for the analysis of the kaon data.
- Consult the relevant README/documentation within each repo for more info

You should "fork" each of these repositories so that you have your own copy that you can update and play around with. From the links above, simply click the fork button in the mid/upper right (See Fig 1).

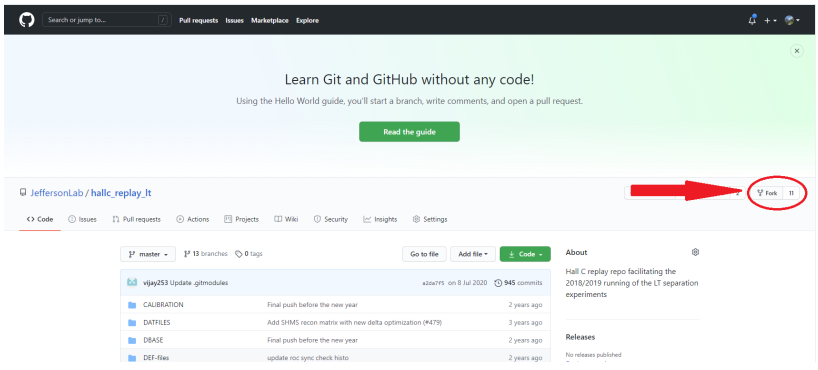


Figure 1: Fork button as it should appear on github

Once forked, you should have your own copy of the repository, for example - https://github.com/sjdkay/hallc_replay_lt is my personal fork of the hallc_replay_lt repository.

# 3 Setting Up Repositories on the iFarm

Once you have all of the repositories you want to use forked, you should clone them on the iFarm (or your local machine). For the following, I will outline instructions as though you are setting things up in the c-kaonlt group area. For the pionlt area, the idea is the same just with different pathing. If you have not done so already, make a directory for yourself in the c-kaonlt area -

- cd /group/c-kaonlt/USERS/
- mkdir $USERNAME && cd $USERNAME
- git clone https://github.com/GITHUB_USERNAME/hallc_replay_lt
  - This will clone the repository, it may take a little bit of time as it downloads everything
- cd hallc_replay_lt
  - Once it finishes cloning of course
- You should then clone the UTIL repositories you want into this directory
- We aren't ready to go yet, unfortunately we have a little more setup to do

Once you have cloned all of the repositories you want, we now need to switch to the correct working branch for each repository and set up the "upstream" remote. We need to set this up so that we can merge in changes to the main JeffersonLab repositories with your own fork. Below, I will outline the general procedure for switching branch and the upstream remote. I will then list the relevant working branches for each repository. I will also include some commands for you to check things are set up correctly.

- git checkout $BRANCH
  - This will checkout the specified branch, you should do this from the directory of the repository you want to switch branch in
  - By specifying git checkout -b $BRANCH, you can create and switch to an entirely new branch
- git remote add --track $BRANCH upstream https://github.com/JeffersonLab/REPOSITORY
  - This adds the specified branch of the specified JLab repo as a new remote (called upstream)
  - Remote repositories are versions of your project that are hosted on the Internet or network somewhere, see https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes for some more info
  - You can check the remotes you currently have set via "git remote -v"
- In all of the above $BRANCH and REPOSITORY are just whatever branch or repository you're trying to sort out
- The relevant working branches for each repository are (as of 22/01/21) as follows -
  - hallc_replay_lt - Offline_Pass1
  - UTIL_BATCH - develop
  - UTIL_PION - develop
  - UTIL_KAONLT - offline

So as an example set of commands, to set up your hallc_replay_lt repo, you would do -

- git checkout Offline_Pass1

- git remote add --track Offline_Pass1 upstream https://github.com/GITHUB_USERNAME/hallc_replay_lt
- git remote -v
  - Just to check it's working

### 3.1 General Setup Reminders for hallc_replay_lt

Remember that to get your replays to work, you will need various sym links in place for data analysis. For the UTIL_PION and UTIL_KAONLT repositories, you will also need to set up some sym links in places, carefully check the scripts you are running.
For hallc_replay_lt, you will need -

- In general, the sym links you need are specified by the replay script you are using, carefully check where it is looking for files and where it is trying to output them
- A sym link called "raw" - This needs to point to the directory where the raw .dat files are stored
  - You may also wish to add a sym link called "cache" which points to the raw .dat files on the cache
- A sym link called ROOTfiles - Needs to point to wherever your rootfiles will be saved
  - Again, this is highly dependent upon your script. Some may save things to more specific sym links, e.g. ROOTfilesHGC and so on
  - In general, you should utilise /volatile for any files you want to save but that you do not want to keep permanently
  - For permanent storage, files should be saved to a directory in /cache
  - Do not overwrite files in /cache! If you need to re-run a file, make a NEW /cache directory before re-running the file
  - This is due to how the file will be copied to the tape storage silo
- A sym link called OUTPUT - Again, some directory to store output, script dependent
- A sym link called REPORT_OUTPUT - Some directory to store report output, script dependent, you may need subfolders within this such as COIN and so on, check your script

For the UTIL repos, you should also check the scripts, in general, ROOTfiles will be stored somewhere on /volatile or /cache and you will need a sym link somewhere, likewise, OUTPUT is not stored in /group (unless it is very small) so you will also likely need sym links for output too. Finally, don't forget to do

- source setup.csh

from your hallc_replay_lt folder. You should also add this to your environment setup.

# 4 Keeping Your Repositories Up To Date

Once you have all of the repositories and branches you want set, you're probably then going to start playing around with the code and writing your own. As you write and edit your code, you should try to keep it backed up. We can use git for this purpose (in addition to the fact that /group directories are snapshotted). You should update your personal github fork of the repository frequently. To do this, we need to prepare a "commit".

- At any point as you're working, you can use the command "git status" to check what has changed in the repository
  - As you execute this command, it will list new, changed, renamed and deleted files in the repository
  - Note that some files you may have edited may NOT be on this list, this is likely because they fall under something that git has been told to ignore
  - You can check what git is ignoring via - "emacs .gitignore" (from the first directory of the repository, also, feel free to use vim or whatever of course)
  - You can add to this list as you see fit, in general, avoid adding any large files or analysed data to the git repo
- When you ran "git status", all of the items were probably shown in red, this means that they are currently not staged for a commit
- To add them to a commit, execute "git add -A"
  - The "-A" flag adds all items listed to the commit
  - You may only wish to add specific items, you can do this via "git add $PATH"
- Once added, commit the changes with "git commit"
  - I highly recommend executing - git commit -m "Message here!" - instead
  - The -m flag adds a message to the commit (in the speech marks where "Message here" is), this should be a short, but descriptive, message of what has changed in the commit
- Finally, push the changes with "git push"
  - You will be prompted for your github username and password
  - The first time you do this, it may also complain about some git setup related stuff, follow its instructions
- Once this is done, your "remote" will now be updated, you should see the changes if you check the github page for your repo

Again, I highly recommend that you carry out this process frequently and keep your repositories up to date.

# 5 Updating Your Repositories

In addition to your own changes, the main JLab repo may be updated with some code that you need. In that case, you will need to update your own fork of the repository. This is why we went to the trouble of adding the JLab remote earlier. To update your repository -

- git fetch upstream
  - This fetches the latest version of the upstream repository
- git merge upstream/$BRANCH
  - This attempts to merge in changes from the main JLab repo with your own

When you try this, it is highly likely that you will get some conflicts (situations where you have edited a file that has also been changed upstream). We need to resolve these conflicts before we can push the changes to our personal fork. You can do this manually for all files which conflict (it will tell you) OR do the following to just overwrite the file in question with their (the upstream) version or your own via -

- git checkout --theirs PATH_TO_FILE
  - This will overwrite your version of the file with the one from upstream
- git checkout --ours PATH_TO_FILE
  - This overwrites the upstream version of the file with your own
  - Typically you want to do this if you know your work supersedes whatever has changed upstream

Manually editing the files is a little annoying but may be necessary. See, for example, https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts on how to do this if you need to. When you have resolved all of the conflicts, simply execute "git push". Note, you should do this even if there were no conflicts so that your repository is actually updated.

# 6 Contributing Back

When you've finished up some code that needs to be used more widely you will need to add it to the main JLab repo. You can do this in one of two ways -

1. Execute "git push upstream" to push your changes on your current branch to the upstream remote you have set
   - You should be very sure you want to push the code to the JLab repo when you do this, please take care that you are not overwriting anything vital
   - Currently, I believe most of you should *not* have permission to do this and you will likely be bounced back if you try this
   - Once you grow more familiar with git and are more confident about working with it, I can update permissions
2. Submit a pull request on github
   - When you do this, you can select the branches you wish to merge together
   - This is probably the safest bet as you (and I) can then carefully review all of the changes before merging things
   - Note, github will often prompt you to delete the merged in branch - obviously, only do this if you actually want to!