Caroline Planning/Ideas for GVSPost Repo Organization:

Separate folders for different types of code

Devices: code for sparky and the TTS (could create sub-folders if desired)

SHOT: code for processing SHOT data (could create sub-folders for data processing/checking and then metrics/plotting)

VerbalReports: code for processing subject verbal reporting (could have separate subfolders for processing/checking and metrics/plotting)

IMU: code for processing IMU data

Stats: runs statistics and/or combines metrics across different data types

All other files are in the main GVSPost directory (functions common across code types, documentation, a limited amount of data or plots)

To implement this I would start in a different branch – move all the files to the folders and then go through and make sure all of the code will work (there will definitely be some initial issues for running the common functions that will no longer be in the same directory as the main code – so we will have to add a change directory command in the code, add a soft link for the function, or copy the function into all of the scripts that use it.)

After verifying that all of the code still works we would move it into the main branch

At this point we would also protect the main branch and prevent people from directly editing and pushing to the main branch

To make modifications to any file (even if they are the only person working on it) the person would have to create a new branch, make their edits on that branch, commit their edits and push them on that branch to the repo and initiate a pull request to merge the code with main – I’m a little fuzzy on how the details of how a pull request works, but especially if other changes have been made to a file someone should make sure the code still works properly with those changes (this is where it would be helpful to have some sample data to compare results with). Functions that serve the same role (ie. right now we have multiple gain functions) should be developed with the same name/based off of each other, but in different branches to help keep documentation consistent and allow proper merging into main.

To just run the code, users can check out the main branch (or to check another user’s work they can check out their branch) small changes to input variables (like subject numbers or datatype) should be discarded rather than trying to push them.

The data/results will not be tracked by github, but is stored on the communal sharepoint (where the code touches) and backed up on the NAS server(which the code should not touch). Therefore, users should be careful when running code that produces results that others might use and alert other users of potential data changes if changes are made to the script that produces them/a non-main version of the code is used to produce the results.

Ideal process for code editing:

1. Open github desktop app before starting to code
2. Change to your appropriate development branch to work on your development code
3. Add lots of comments to your code as you write it
4. At the end of any coding session (but possibly throughout), review the changes you’ve made to the code and commit any intentional/major changes you’ve made.
   1. make separate commits for each file, unless the changes in each file are related to each other.
   2. Add as much detail as necessary to describe what and why you made your changes
   3. Do not commit or push .asv files
5. Push commits to the repo development branch as appropriate
6. Restore any unwanted changes
7. If you made a new file or deleted a file update the “read me”/ whatever code running order document we make (and indicate that the script is currently in development)
8. Close code editor and github desktop or git bash

If/when you are ready to add your changes to main

1. I would like to implement some sort of testing protocol to make sure the code works (especially if there are changes to a file that produces data that is used downstream by other existing files)
2. Notify code owner(s) – at a minimum Caroline, but potentially other code users- when you create your merge/pull request.
3. Unclear whether we want to require other users to review before approving the merge request – but maybe give the other users adequate time to review the code before the merge request is approved.
4. After merging the code – I would potentially want to implement some other sort of testing protocol.

Ideal process for running code

1. Open github desktop app before running the code
2. Change to the appropriate branch (likely main)
3. Modify any inputs as necessary to produce your desired results
4. Run the code
   1. If you overwrite the “normal” version of a data file (ie. create an aggregate file or plot without a subject that is normally included) notify other users, or re-run the “normal” version of the code
5. Restore/discard any minor changes
6. Close code editor and github desktop/git bash

Things to do:

* Create/update the file that has the script summaries and run order (I think Lanna made one previously)
* Move files into sub folders
  + Make new branch for this
  + Check to make sure all files are still working
    - Modify files/add links if they are not working
  + Check again to make sure all files are working
  + Merge into main
    - Possibly delete files in main if they are duplicates – but make sure that the file history is captured before doing so
* Make/update this file so that it has pictorial instructions on how to do execute the “ideal” processes
  + Alternatively make a video or powerpoint