This tutorial is meant to focus on the essential features that will enable us to easily and effectively collaborate and co-develop. For more details, you can find several comprehensive tutorials online, e.g. http://git-scm.com/documentation.

1. set up ssh key
   If you don't have the ssh key on your machine:
   $> cd ~/.ssh
   $> ssh-keygen -t rsa
   This will generate the public ssh key ~/.ssh/id_rsa.pub. Log into your github account and copy your public key into Account Settings -> SSH Keys -> Add SSH Keys. Now you can pull and push without entering your password. Note that this will be applied to your account and can be used for all the repositories.

2. clone
   To get a copy of the "abq_dev" repository from github into the local directory mytest, use clone:
   $> git clone git@github.com:shoeybi/abq_dev.git ./mytest
   This will create the directory mytest and copy the content of abq_dev repository into mytest.
   $> cd mytest
   $> ls
   If the local directory name (here mytest) is not provided, then git will put the content of the repository in the abq_dev directory.

3. pull
   To update your current branch (default is set to master) with any changes that have been made on the github server use
   $> git pull
   In some cases you will want to check that you have not made any modifications to your local files that this pull will try and merge with:
   $> git status

4. branch
   Branching is a useful feature especially when we have a stable release and we want to have incremental developments. As soon as our main code is in place, please do work in branches other than master.
   To create the local branch mydev to do some work:
   $> git checkout -b mydev
   To check which branch you are in:
   $> git branch
   To switch between existing branches, use checkout:
   $> git checkout master
   $> git checkout mydev
   Now do some work in mydev and commit it:
   $> git status
   $> git commit -a -m "some mydev work"
   Note that the -a automatically adds all modified tracked files. At this point, you will want to push the branch up to github, so you can test the modifications on different systems before

merging them into master:
$> git push origin mydev
Now on other systems, checkout the branch and do some testing:
$> git checkout -b mydev origin/mydev
When you return to the original system (the one where you first branched), you will not be able to simply pull to get any modifications pushed from the other system(s). You can either:
$> git pull origin mydev
or set tracking information for this branch and issue a default pull:
$> git branch --set-upstream mydev origin/mydev
$> git pull

5. merge and push
   Once all your developments are tested n your branch (mydev here), you will want to merge all your mods into the master:
   $> git checkout master
   $> git pull
   $> git merge mydev
   $> git push
   This will merge the mods from mydev into the master branch. Note that this workflow can be a bit dangerous, because the git pull may bring down mods that have not been tested with your own work in mydev. Also, for branches that persist for a while, you may want to do the opposite merge to keep the rest of your code up to date with the stable master branch:
   $> git checkout master
   $> git pull
   $> git checkout mydev
   $> git merge master
   You can now push your changes back to the origin (github) using:
   $> git push
   You can delete a merged branch using:
   $> git branch -d dev
   It is recommended that you set the push.default configuration variable to current. This will prevent you from unexpectedly pushing any incomplete work in other branches to github:
   $> git config --global push.default current