Lisa Wade

Notes on Git/GitHub training

* Git is a version control technology
* GitHub is a company with a website based on Git
* Git is distributed version control – meaning that the local drive can have more versions than the server.
* Version identifier is automatically assigned by Git.
* You should write good commit messages so that you can figure out later what was done. This is essential to maximizing the functionality of Git.
* You should use Markdown syntax (text files that have a few extra symbols to do formatting, like a “LaTex light”. For details, see <https://guides.github.com/features/mastering-markdown/>
* You start with the master branch – your baseline repository. Then you start making branches and then you merge them back into the master, or not, depending on the situation.
* Question for Steve – is there a way in the automation process to force TSTool or StateDMI to use a particular version of Hydrobase?
  + Only if you are using scripts, can’t do it in a command file. Only can have a comment that the command file depends on a particular version
  + Steve recommends including what version of Hydrobase should be used when writing commits. (well, that is fine for the people on GitHub, but I think it needs to be in the files as well.
* Challenges = “imposes a level of rigor for version control that has not been used before in CDSS, which requires a paradigm shift” (and people hate paradigm shifts)
* Steve recommends writing ReadMe.md for your repositories.
  + Need to figure out why Erin dislikes ReadMe…
* Not sure what Eclipse is about…
* Other suggestions for Steve
  + What is a PATH? Maybe offer a couple of sentence explanation
  + What is a CRLF? Maybe offer a couple of sentence explanation
* Overview of setting up a new repository – always do it on GIT first, and then clone it to your local drive so that the correct settings are initialized.
* Handy commands in the GIT BASH command shell
  + pwd (print working directory) aka where am I?
  + pwd –W (print working directory windows style)
  + cd (get to your home directory).
* Steve – I am not getting the same configuration settings as you, specifically my
  + merge.conflictstyle=diff3
  + push.default=upstream
  + are just not listed. Is this a problem?
* Steve – from the link <https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>, the piece that was most helpful was the instructions on getting Notepad++ set up as the default editor. This means when I try to do a commit, Notepad++ pops up and I fill in my commit message there.
  + Testing with TextPad. Appears to work just fine if you use the following command (point the path to where ever your executable is)
  + $ git config --global core.editor "'C:/Program Files/TextPad 8/TextPad.exe'"
  + Need to be in your “Home” directory to make the change
* The lesson 5 formatting is different than the other lessons.
  + Not sure if I need to worry right now about how to best create a repository, since I think Steve will make one and we will all work out of it?
  + Steve, I don’t know what to do about licensing. I either need more or less information
  + The paragraph under “Clone the Repository to a Local Computer” about “fork” seems misplaced. Or if not, I am missing something.
  + Focus, Project, Repository.
  + Try not to need to rename repositories. If this happens, use GitHub and then clone to your computer.
  + Repositories are cheap, essentially just a folder
  + Backing up a Repository. This seems redundant.
    - Also, my testing shows that it does not work well to have the repository on your local drive be under Google Drive. That creates lots of confusion for Google Drive.
* Lesson 6
  + Probably need to have more of a conversation about what should be “ignored”.
  + Seems like this end of line issue is really important. How do we make sure we get that right for ArkDSS?
    - Seems like we need to add all the “\*.b43” and other similar StateMod binary output to the binary list? Or does it go on the “ignore” list? Probably go on the “ignore” list, but all the text file output can get saved to GitHub no problem. Question is does it make sense to keep the file output on GitHub. That is something we will have to figure out as we are working on this, I guess.
* Lesson 7a – using Git
  + Only tested and validated changes should be committed to the master.
    - But we probably don’t want branches hanging out for too long in ArkDSS development, right?
  + Git workflow
    - Step 1 – make sure your local copy matches the master copy on GitHub
    - Can run “git status”
    - If you are out of date, run “git pull” to get the most recent version of the master copy.
    - Not sure what this does, but apparently “git pull” combines the commands “git fetch” and “git merge”.
    - Steve suggests using branches to temporarily isolate new work, so it can be tested before it gets merged.
      * How do I make a branch on GitHub? I can do that easily on my local copy, but what about the cloud version?
    - Create and Checkout a Branch
      * Suggests giving the branch a very clear name and use dashes to separate “parts” of the name. Recommended naming convention is “IssueNumber-Type-Topic”, where IssueNumber is from the Issues tab, type is a category such as bug, feature, test, or whatever else. And topic is some short of phrase describing what you are doing.
    - “git branch *branchname”* = git will create a new branch and label it with *branchname.*
    - “git checkout *branchname*” = changes the current working files to the new branch.
    - “git branch” = tells you what branch you are on. This is important because you need to know where you are making your changes.
  + Steve, I keep getting warnings in Git Bash about “desktop.ini” file when I am running through the commands in the training. (This is a Google Drive issue.)
  + Important note about how Git works: There is your local repository, GitHub and then there is staging area between the two. To get files from your local repository to GitHub, you first have to send them to the staging area using the command “git add *filename*” (looks like “git add –A” adds all new files or all files with changes? Not clear).
  + Can use “git reset” to clear out the staging area
  + After you have moved files to the staging area, you can commit them. Now your local repository and the branch are the same. Use “git commit” and your default text editor will pop up and prompt you for a commit message. Once you have saved and closed your text editor, the commit is finished.
  + Steve then recommends having a final test to make sure your files are good to go before merging with the Master Branch. This is something you would need to do yourself, as it is not something Git does automatically.
  + It is possible to push a branch to GitHub before merging and then someone else can pull the branch. That makes peer review before merging possible. How do I do this?
  + “git merge” – get the changes from the branch integrated into the master repository on GitHub.
    - First, make sure you have finished up the work on your branch, and you have done the “git add” and “git commit” steps.
    - Next, “git checkout master”, so your local repository switches over to the master repository that you cloned earlier.
    - Then “git merge --no-ff” The merge command combines your branch with your master. The --no-ff option means that Git will keep track of the branch merge, not just the individual commits made in the branch. (This was a little confusing, but the advantages of --no-ff are laid out in the training.
  + “git log” shows you what you committed so far. If you want a pretty picture, try
    - git log --oneline --graph --decorate –all
  + Push Files to GitHub – so far, all the changes have only been to the local copies of stuff. Now we want to send all the information to GitHub, use “git push”. Steve recommends pushing files frequently so that nobody’s branch becomes too far off from the master.
  + Once you have finished with your branch, you can delete it using “git branch –d *branchname*”. Want to check that you merged your branch before you delete it?
    - “git branch --merged master”
    - “git branch –no-merged master”
  + Tags represent the status and contents of the repository at a single point in time. Generally used to indicate milestones or release versions. You can checkout the repository as it was at the time of the tag.
    - To write a tag = “git tag –a *tagname* –m ‘*Tag Message*’
    - To read the current tag = “git tag”
    - To write the tag to GitHub = “git push origin –tags
* Other helpful commands
  + If you want to go back to the last committed (or staged, if you have not committed your staging area), you can “git checkout *filename”.* (note the file name must contain any leading folders as well)
  + “git log” shows you all the commit messages. This can be helpful.
* https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
* Example workflow from what I have learned in training.
  + Step 1 – clone the Git Repository from GitHub. You will only need to do this once at the beginning of the project.
    - First, set up a folder on your machine where you want the repository. For ArkDSS, we should coordinate as much as possible so that our path names match and we don’t have to deal with issues of relative paths vs full paths vs changing the path names every time we move something between computers.
    - Second, open a Git Bash session in the folder you want to have your repository.
    - If you have something on your keyboard, you can paste into the Bash window by clicking the scroll wheel on your mouse.
    - Use command
      * git clone *Link\_from\_GitHub*
      * so for this example, I now have the folders. I created Git\_Test and opened my Bash session in that folder.
      * C:\CDSS\Data\Git\_Test\cdss-model-cu-sw-yampa
  + Step 2 – create a local branch so that you can track your work separate from the master branch. Switch your git view to be of the branch, so you can make changes. This is known as your working directory
    - “git branch *branchname”* this creates the new branch. For ArkDSS, not sure if we should come up with a protocal for how to name these, maybe only need that if we are going to be saving branches outside of our local information.
    - “git checkout *branchname*”. Now your git is looking at that branch. Note that you should be able to see in your Bash prompt what branch you are looking at.
  + Step 3 – make whatever changes you want to make – aka do model development work.
    - You can check on which files you have made changes to by using
    - “git status”
  + Step 4 – whenever you get to a point in your development work that you want to save, you can do a “commit”. Basically, you are taking a snapshot of the work completed so far and having git remember that snapshot. Then, if you mess something up and want to revert back to this version, you can. Or you can even just pull a single file from this version. Think of it like in a video game after you have completed a big challenge, you save the game so that no matter how many times you die on the next challenge, you can always start the game over at your last save point. This is all still local on your machine. Even on your local machine, it is a two step process. First you have to assign files that are in your working directory to your staging area, then you have to assign them from your staging area to your repository.
    - “git add *filename*” – assigns the file to the staging area.
    - Alternative use of “add” “git add –A” – assigns all files to the staging area.
    - “git reset *filename*” – unassigns the file from the staging area
    - Use “git status” to check on which files have been assigned to the staging area.
    - When you have assigned the files of interest to the staging area, you are ready to commit
    - “git commit” – then your text editor will pop up and you should write a message describing what state the files are in. Basically need to give yourself enough breadcrumbs. For ArkDSS, we might need to come up with some guidelines for ourselves in terms of length and content. After you have saved and exited your text editor, the commit will proceed.
    - You are now free to keep working on your branch.
  + Step 5 – you have finished the work you wanted to accomplish on this branch, you have tested it, and you are getting ready to share it with everybody else.
    - Time to merge your branch back into the master. First, make sure your branch is 100% up to date (as in, you have committed everything you want to commit).
    - Next, switch your view to your local copy of the master branch using
      * “git checkout master”
    - Now, tell git to merge the branch with the master
      * “git merge --no-ff *branchname*” – this is the preferred merge option that Steve wants us to use, so I think we should go with it, but doesn’t seem like a big deal to me. You will have to write a merge message, which seem like it should be a brief description of what you accomplished on your branch.
    - At this point, Steve suggests doing whatever final check needs to happen that your files are ready to be shared with the team. For ArkDSS, we should probably come up with something…
    - Note, you can also go in the other direction. Say something has been updated on your master branch that has not yet been updated on one of your branches. Checkout the branch and then “git merge master”.
  + Step 6 – You are ready to share your changes with the team. Time to “push” your changes back to GitHub – aka get them off your local machine and into the cloud.
    - “git push” – scary simple?