Introduction to github and git commands:

- Log into your lab machines:
    - Centos OI: You should log out and restart if you are in this OS
    - Use Ubuntu

- Go to Github, and create an account
- Create a new repository **with a license** named gitdemo
- Clone your new repo:
    - $ git clone https://github.com/YOUR_ID/gitdemo.git
- Change into the cloned directory:
    - $ cd gitdemo

- Create a readme file from command line:
    - $ echo “# gitdemo” >> readme.md
- Confirm that the readme has your content:
    - $ cat readme.md
- Stage the readme for commit:
    - $ git add readme.md
- Commit the readme:
    - $ git commit readme.md -m “add readme file”
- Push your committed change to your remote (called "origin"):
    - $ git push origin master

    - Getting help on subcommand (like commit):
    - $ man git-commit

- Look at git's config:
    - $ git config --list
- Tell git your name:
    - $ git config -—global user.name "Firstname Lastname"
- ...and email:
    - $ git config —-global user.email "your@email.com"

- To remove files you created before you have staged them:
    - Create garbage file:
        - $ echo “test” >> temp.txt
    - Remove the file normally:
        - $ rm temp.txt
    - OR use git (BE CAREFUL - dry run with -n):
        - $ git clean -fd

- To remove files after you've staged them:
    - Create new garbage file:
        - $ echo “test” >> temp.txt
    - Stage the file:
        - $ git add temp.txt
    - Clear your stage:
        - $ git reset temp.txt
    - OR just remove one file from your stage:
        - $ git reset temp.txt

- Show list of commits:
    - $ git log
- OR for different output
    - Graph including branches: $ git log --graph
    - GUI tool with graph: $ gitk –all

- Do two commits and push
    - create file
    - $ git add …
    - $ git commit...
    - create file 2
    - $ git add …
    - $ git commit...
    - $ git push origin master

- Make a file elsewhere
    - Within github, create a file.
    - Commit the file

- Back in the terminal, pull it in
    - $ git pull origin master

- Make a conflict and resolve it:
    - Create another file within github and commit.
    - Create a file **with the same name** in terminal, and commit.
        - create file
        - $ git add …
        - $ git commit...
    - Try to push (get rejected)
        - $ git push origin master
    - Pull first (get a conflict)
        - $ git pull origin master
    - Resolve the conflict:
        - Edit the file locally, save it
        - Stage the file
            - $ git add …
        - Commit with a message indicating conflict resolution
            - $ git commit -m “resolved…”
        - Push your resolution:
            - $ git push
        - Check the online repo, verify it’s the same as local commits:
            - $ git log

- Conclusion: conflicts are a hassle. Before starting work:
    - $ git pull
- ...then start working, then add, commit and push, etc. Do those often.

- Ignoring some files/foldersfrom being tracked with a .gitignore file:
    - $ mkdir data
    - $ touch data/a.txt data/b.txt
    - $ echo “data/*.txt” >> .gitignore
- Notice the difference in status
    - $ git status
- Stage, commit and push the new .gitignore file.

- Jump to previous commit with git checkout:
    - $ git checkout <sha_of_an_older_commit>
    - first 4 letters are enough
- Note that HEAD is detached (**Don’t change the code in this state!**)
    - $ git status
- To edit old code safely, use branches. First reattach HEAD:
    - $ git checkout master

- Recall: commits are snapshots of the project. Switching between them is so easy and fast.
- Branches allows us to work from a previous version, and then merge those changes back in if/when we want. A branch essentially says "I want to include the work of this commit and all parent commits.
    - Don’t be scared - branch early, and branch often
    - master is the common name for the default branch. It doesn’t need to exist, but it often does.
- HEAD
    - HEAD is the symbolic name for the currently checked out commit(always points to the most recent commit which)
    - HEAD can be thought of as a variable pointing to a specific commit
    - It can change and isn’t related to a branch.
- Remotes store copies of all pushed commits and branches. They are a remote copy of the repository. See them as URLs:
        - $git remote -v
- Origin is the default alias for your remote repo

- List your local branches:
    - $ git branch
- List local and remote branches:
    - git branch -a
- Create new branch (**without** checking it out):
    - $ git branch integrate-database
- Check out new branch:
    - $ git checkout integrate-database
- Create a new file called “db.txt”, add it and commit.
    - $ touch db.txt
    - $ git add db.txt
    - $ git commit -m “add a database”
- Push the new branch to origin
    - $ git push origin integrate-database
- Examine the created file in github, within the new branch.
- Create a new file called “db2.txt”, add it, commit it, and push again
    - $ touch db2.txt
    - $ git add db2.txt
    - $ git commit -m “add another database”
    - $ git push origin integrate-database
- Notice that master is not affected:
    - $ git checkout master
    - Look at file manager, run $ git status, etc

- While on master add a file, add it, commit it, and push
    - $ touch functionality1.txt
    - $ git add functionality1.txt
    - $ git commit -m “add new functionality”
    - $ git push origin master

- Note that our branches have diverged:
    - $ gitk --all
    - OR look in github: https://github.com/YOUR_ID/gitdemo/network
- Add files to either branch and see how they diverge.
- Resolve with a local merge:
    - $ git checkout master
    - $ git merge integrate-database
- OR with a remote merge using github' pull request feature