DS4-L: Hands-on training of Git and GitHub – Lab **5/19/16**
Michael Dolan, Ph.D.

Tutorial 1

Objectives: Download and install Git, configure Git, initialize a new repository, examine .git directory, create file in working directory, add it to the staging index, commit the file, check to see where the HEAD pointer is pointing, modify file and make a second commit, check HEAD pointer, create a new branch

Exercise 1 – Install Git

1. Go to the following link and download the latest version of Git:

   https://git-scm.com/

2. Click 'OK' or 'Next' for all options, then finish (you don't have to review notes).

3. Go to:

   Programs > Git > Git Bash

~ If a review of UNIX commands is needed, we will spend some time doing so.~

4. Which directory are we in?

   pwd

5. Change to Desktop directory

   cd Desktop
   pwd

6. To see if git is installed correctly and to see git options, type:

   git

7. To see which version of git is installed, type:

   git --version

   Note: Make sure to use TWO DASHES!

8. Configure git by typing:

git config --global user.name “yourGitUserName”

where *yourGitUserName* is a name that you would like to use with Git (and later when signing up on GitHub)

9. Configure git by typing:

git config --global user.email “your@email.com”

where *your@email.com* is the email that you would like to use with Git (and later when signing up on GitHub)

10. Check your configs:

git config --list

11. Help!

git --help or just git

git --help log ← will open a new browser

Exercise 2 - Initializing a new repo

12. Make and change into a new directory which will contain our new repo:

```
mkdir new_repo
cd new_repo
pwd
```

13. Initialize a new repo:

git init

A new, empty Git repository is created. Well….not *quite* empty.

14. List contents:

ls

Nothing is seen. Use options to see “hidden files:”

```
ls –al
```

The **.git** directory is seen. A .git directory is essentially everything that is needed to define a local Git repo.

15. Change into and view contents of .git directory:

```
cd .git
pwd
ls
```

Exercise 3 - Add and commit file(s) to new repo

16. Go back to new_repo directory:

```
cd ..
```

17. Check status:

```
git status
```

18. Create a file with text:

```
echo hello > file1.txt
```

19. Check to see if file is present and contents of file:

```
ls
cat file1.txt
```

20. Add the file to the staging index:

```
git add file1.txt
```

Note: In Unix systems the end of a line is represented with a line feed (LF). In Windows, a line is represented with a carriage return (CR) and a line feed (LF) hence "CRLF."

21. Check status:

```
git status
```

22. Commit the file to the local repo:

```
git commit –m “Commit first file”
```

Note: The SHA1 hash is a cryptographic hash function, but used here to ensure data integrity and tracking.

23. Check status:

```
git status
```

Exercise 4 - Follow HEAD pointer

24. Let’s check the log to see what was done:

```
git log
```

25. Check branches:

```
git branch
```

26. Look at HEAD pointer:

```
cd .git
ls
cat HEAD
```

27. Follow the path:

```
cd refs/heads/
```

28. Look in ‘master’ file:

```
cat master
```

Note: The HEAD pointer points to the last commit on the master branch.

29. Get back to working directory:

```
cd ../../..
pwd
```

30. List files in working directory:

```
ls
```

31. Make a change to file1.txt:

    echo NIEHS >> file1.txt

    Note: Use a double redirect (>>)

32. Check contents of file1.txt:

    cat file1.txt

33. Compare differences between file1.txt in the repo and what is in your working directory:

    git diff

34. Add and commit changes:

    git add file1.txt
    git commit –m "Modify file1.txt contents"

35. Let's check the log to see what was done:

    git log

    Note: Two SHAs are seen with the latest commit on top.

36. Take another look at where HEAD is pointing:

    cat .git/refs/heads/master

    Note: HEAD now points to the last commit in the current branch which is master.

Exercise 5 - Create and merge a new branch while resolving merge conflicts

37. Look at current branch:

    git branch

38. Create a new branch:

    git branch new_feature

39. Verify branch creation:

```
git branch
```

40. Take a look at .git folder changes:

```
ls –al .git/refs/heads
cat .git/refs/heads/new_feature
cat .git/refs/heads/master
```

Note: Each branch points to same commit.

41. Switch to new branch:

```
git checkout new_feature
```

42. Confirm switch:

```
git branch
```

43. Modify file1.txt:

```
echo “New feature” > file1.txt
```

44. Add and commit at same time:

```
git commit –am “Introduce new feature”
```

45. Check log using a more compressed view:

```
git log --oneline
```

46. Switch back to master branch, verify, check log:

```
git checkout master
git branch
git log --oneline
```

47. Make a change to file1.txt in the master branch and commit:

```
echo banana > file1.txt
git commit –am “Introduce banana”
```

48. Compare the two branches:

```
git diff master..new_feature
```

Note: Two dots!

49. Merge changes in new branch into master branch. From master branch:

    git merge new_feature

    Note: No merge took place. There is a conflict that must be resolved.

50. Check status:

    git status

51. Look at file using notepad:

    notepad file1.txt

52. Fix the conflict manually and save the file.

53. Commit to resolve merge:

    git commit

    Note: No message needed when in the middle of a merge. A standard merge message is used instead.

54. Type the following to see a nice graphical representation of what we just did:

    git log --graph --oneline --all --decorate

END