Git Tutorial

A Source Control Management (SCM), i.e. a tool, which allows you to manage and track changes to files over time, making it possible to revert to a file.

1. Installation: https://git-scm.com/downloads
    a. Windows user: git is not automatically installed.
    b. Mac user: git is typically already installed.

    Note: when git is installed, git Bash is automatically installed.

2. Open Git Bash (or any terminal of your choice).
3. Navigate to the folder (directory) that you want to set as your local repository.

    Ex. user/documents/winter2023/gitRepos

4. **Create a repository:**
    a. **$git init** //the command with no parameters initializes the current directory (i.e. folder) as the repository.
    b. git init *nameOfRepo* //the command with parameter (fileName) creates and initializes a new repository (i.e. subdirectory/subfolder) in the current directory. You can name it whatever you please.

5. **Execute the $git config commands below:**
    a. **$git config --global init.defaultBranch *main*** //rename the name of the default branch in our repository to 'main' (a typical default name).
    b. **$git branch -m *main*** //set the name of the master branch; make sure you are in the subdirectory if you used $git init with parameters. (Notice how the name "(master)" gets changed to "(main)" on the right the first time you do this).
    c. **$git config --global user.name "*nameHere*"** //you need this for when you commit later
    d. **$git config --global user.email "*emailHere*"** //note, executing this command resets your email if you already have one.
    e. $ git config –global user.email *"your-username@users.noreply.github.com"* //specify a private email (this can also be changed directly through GitHub desktop)

    You can verify your name and email by entering:
    $ git config --global user.name
    $ git config --global user.email

6. Check that your repository has the **.git** directory in it (this is what makes the directory a fit repository and what tracks the changes of the files within the target directory). You can check by navigating to the directory/folder through your terminal or file system (file explorer for windows or Finder for Mac); you might to need to unselect hidden files in your system to see it.

   $ls -a //type this in your terminal to see directory
   Or use your file explorer (windows) or Finder (Mac) to view directory from there

7. Quick Helper tip:
   a. $git *command* -h //displays help information about the config command in the terminal.

   b. $git help *command* //opens up a manual (a more detailed description) that is hosted on your computer.

8. **$git status** //to show current branch and commits. It also allows you to see tracked vs. untracked files. It is a very useful command!

Activity 1: Create a git repo.

1. Create a directory you want to set as your repository in a location:
   $cd ./documents
   mkdir myFirstRepo
   cd myFirstRepo #you should be in ~documents/myFirstRepo
   #note: or ~documents/…/myFirstRepo if you specified subdirectories

2. Initialize the directory as a repository:
   $git init
   $ls -a #check that git exists
   $git config --global init.defaultBranch *main*
   $git branch -m *main*

3. Use config to add your name and email:
   $git config --global user.name "*nameHere*"
   $git config --global user.email "*emailHere*"

9. **Create a new file in your repository:**
    a. Directly in the terminal
        i. $ vim *myFirstFile.py*
        ii. $ nano *myFirstFile.py*
        iii. $ vi *myFirstFile.py*

    or use a text editor of your choice and save it in your git repository (Avoid using Microsoft word!):
        iv. Visual Studios
        v. Notepad
        vi. Notepad++

Basic Vim Commands	**Basic Nano Commands**
enter INSERT mode = click the letter **I** get out of INSERT mode = click **esc** Save & quit = click **esc** then type **:wq** and click **Enter** Exit = click **esc** then type **:x** and click **Enter**	Save = click **ctrl** and the letter **O** at the same time, then click enter. Exit = click **ctrl** then type **x**

    b. Create a python script.
    c. Enter "conda activate cs124" so it will know to use the python version we have in our environment.
    d. Run the script in terminal: **$python *scriptName***
    e. Save python script in your git repo.

Activity 2: Create a python script (ex. helloWorld.py) that prints out "HelloWorld!" and run it in the terminal.

1. Create a python script in your directory:
   nano myFirstFile.py

2. Inside the file, write code to print the text "Hello World!"
   Print("Hello World!")

3. Save the file:
   Ctrl + O then click enter

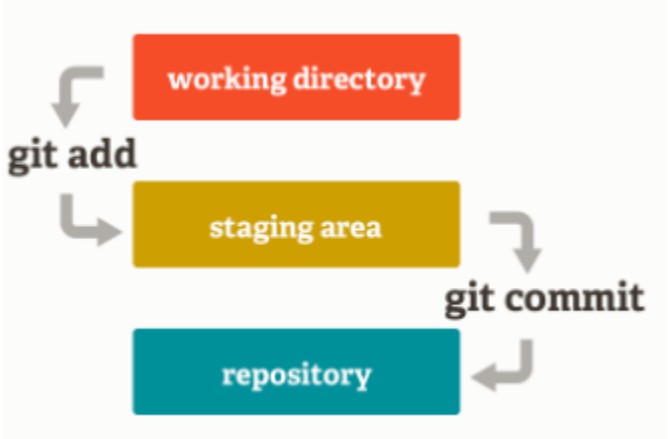
4. Exit the file:
   Ctrl + x

5. Run python file in terminal:
   $python myFirstFile.py

10. Add/remove the file to/from the repository:

- **working directory** = Untracked files (red); working directory - you can make changes in file and don't have to commit those changes; git won't care about these files.
- **staging area** = Tracked files (green). Changes that are ready to be committed, meaning git will keep track of the changes in these files.
- **commit** = Officially saving the repository at a point in time (i.e. saving the current state of all the tracked files in our git record).

//you can read more on the staging area at https://git-scm.com/about/staging-area



$git status //use git status to help you see staged files and unstaged files.

a. **$git add *fileName*** // track one file (move into the staging area)
   $git status

b. $git rm --cached *fileName* //untrack a file (move back into working directory.
$git status

11. Invoke $git commit:

a. **$git commit -m *"messageHere"*** //This command commits everything that is in staging. The current state of the repository is what will be logged. Make sure to add a concise description or summary in the quotation marks of what you changed/added/deleted (i.e. the metadata); this helps other programmers who might look through your commits. If you don't provide a description, then you will get prompted to add one.

b. $git status //enter git status to see what changed after committing.

Activity 3: Change helloWorld.py and run it on the terminal. Then add it, commit it, and do git status and git diff.

a) Add file to staging:

$git status #get used to using this command as it helps to verify which state the files are in

$git add myFirstFile.py

$git status

b) Commit files in staging:

$git commit -m "saving original file"

$git log #to see what our commit looks like

$git status

c) Change, save, and exit out of the python file:

$nano myFirstFile.py

#change the text inside of the file

Print("Hello World! and Stanford!")

#save file

Ctrl + O then click enter

#exit out of file
Ctrl + x

#check file by running it in the terminal
$python myFirstFile.py

$git diff #this will show you the difference between your committed file and the current staged file.

d) Commit it again:

$git commit -m "added more to text"

$git log #to see what our commit looks like

$git status

12. To see changes you make in a **tracked file**:

    * Note: git will only tell you changes that happens to tracked files (ex. modified, deleted ,…)

    a. **$git diff *fileName***

13. Use $git log to see your commits:

a. **$git log** //shows you everything.
b. $git log --oneline //shows just one line

GitHub (Advanced)

A popular cloud (remote) repository where you can host your git repository and collaborate/work together with others (i.e. a social coding website).

14. Create a GitHub account.
    a. Link: https://github.com/

15. Setup a new remote repository (go to the new repo page):
    Notes:
    a. Setting a *description* is helpful and a good practice.
    b. Make it *public* (it can be searched and any can see it) or *private* (it cannot be searched, but you can still share it amongst collaborators).
    c. (optional) add *readME* file and/or a .ignore file. Adding a readMe file is a very good practice, it helps others who are interested in what your repository is all about. These are optional because they can be added at any point. For example, if you decide later that there is a file you don't want to git to track, you can manually create a simple .ignore file.

16. Push an existing repository (through the terminal):
    a. **$git remote add origin *urlLink*** //establishing a remote connection with github and calling it origin (a common name used), with the address we are connecting to.
    b. **$git branch -M main** //sets the target branch to main.
    c. **$git push -u origin main** //push all our contents from our local repository to the cloud.

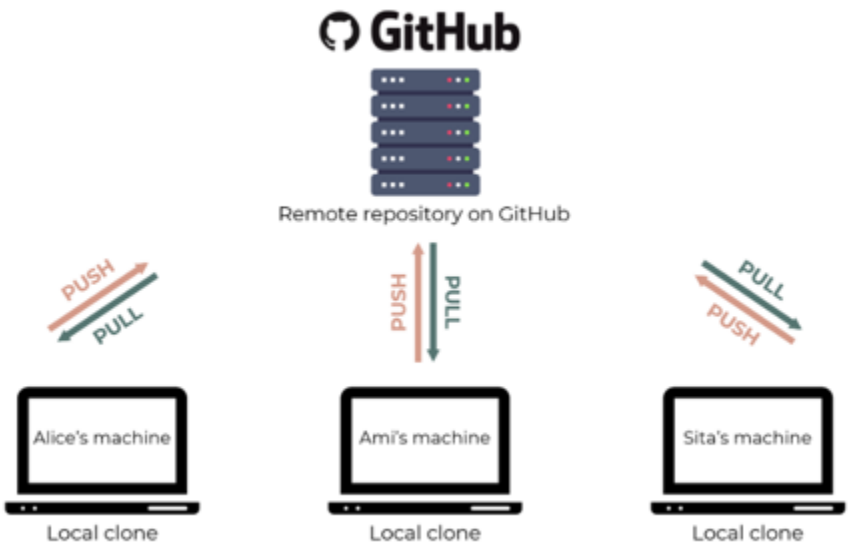
17. Check your pushed repository on GitHub: (it should be synced on GitHub)
    *re-click the name of the repository you created on GitHub and your pushed data should appear.
    a. Review the commits in your GitHub repository
    b. Add a file directly on GitHub
    c. Edit a file directly on GitHub

18. Updating your local repository (through terminal):
    a. **1 command: $git pull** // is used to fetch and download content from a remote repository and immediately update the local repository to match that content. **(recommended)**

b. **2 commands:** **$git fetch** and **$git merge** // git fetch downloads all the history from the remote tracking branches, but we need to execute a git merge after it (on a new line). $git pull = $git fetch + $git merge

19. Adding collaborators
    a. Go under your GitHub settings and you will see on the left side “collaborators” which is how you can add collaborators to your repository.

20. Cloning:
    *The act of copying a remote repository then downloading into our local git repository so we can make commits to it and push it back.
    a. **$git clone *urlLink***



21. To ignore a file in the working directory:
    a. **$git status** //take note of a file you want to hide.
    b. Create a **.ignore** file in your repository
    c. Add enter conditions (or specific files) into your .ignore file in which you do not want to sync. An example of a condition is *.txt which ignores all txt files.
    Hint: to see a comprehensive list of all the different ways you can ignore files, go to https://github.com/github/gitignore
    d. **$git status** //.ignore should appear and files you specified in .ignore should not be visible.