#Version Control
Book - git-scm.com -> Documentation -> Book - About Version Control

Distributed
vs
Central (check out files and then check them back into central server)
Central issues - if access is down, or gets corrupted (better hope there's a backup)

Dist - locally, you have every piece of information from last sync to remote repo
every dev has entire backup of the repo

#Installation
git-scm.com -> Downloads

#First Time Setup
git --version *to see if git installed correctly*
add your name to keep track of who's making code changes (when working in a team)
git config --global user.name "Samer Ahmed"
git config --global user.email "hello@samer.co.ke"

git config --list

#Help
config is first git "verb"/"action" that we used
git help <verb> OR git <verb> --help
git config --help
or
git help config / git help add

also:
git config - h (to quickly see options)
press q to exit text editor

#Scenarions
1 - you have an existing project on your local machine that you wanna start tracking using Git
2 - there's an existing project remotely that you wanna start building on

#Scenario 1 -local codebase
git init (from within dir in terminal)
cd then ls -la
no .git
after: .git
tracks everything. if we wanna stop: rm -rf .git

##Git Ignore
first git status
Untracked files
don't wanna track .DS_store
ignore these files

```
touch .gitignore

.DS_store
,project
*.jpg or *.wav (works with wildcard entries)


rerun git status
no longer on there
.gitignore (we want to commit it so we know which files to leave out)

## 3 stages of git *Chapter 2.2 of book -https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository*
Working Area; Staging area & commited files
WD - where we currently have our docs (untracked and modified files here)
*git status to show this*
Staging area - where we put files we want to commit to our repo, allows us to pick and choose what to commit at any one time
If you've worked on multiple files and want to commit several of them at a time, then you can stage each of those in batches with their own commit message
Don't want to make a commit that says "I made a lot of changes in code"

#Staging
git add -A *for all*, don't do yet!
git add .gitignore *individually*

git status shows .gitignore added but not basic-calc.py

now run -A

#Remove from staging area
git reset basic-calc.py
without file removes everything

#Now commit
git add -A
git status
git commit -m "Initial Commit" *messages are important*

git status -> working directory is clean (no untracked/modified files)

git log -> unique hash and info on there

#Cloning - scenario 2 *how most of you will be using it*
git clone <url> <where to clone>

or just git clone in the dir where you want it and use . at the end

let's clone
copy url https://github.com/HanzeUAS-MasterSSE/vc-demo-env.git

now we have files in cloned-repo-example

**Github Personal Access Token**
```

```
*Settings -> Developer Settings -> Personal Access Tokens*
*Generate New -> No Expiry + repo, admin:repo_hook, delete_repo*
*Use in place of password*

git remote -v (info about repo)
git branch -a (all the branches listed)


#Changes
add return x*y to multiply function

*commit locally*
git diff
git status
git add -A
git status *just to show that they're ready to be committed*
git commit -m "Added multiply functional"

*push to repo so other devs have access to them*
git pull origin master *don't forget this, other people may have pushed
changes!*
git push origin master
origin - name of repo
master - branch we want to push it to


#Branches
Common workflow
create branch and work off branch
git branch calc-divide *run "git branch" to show master is still active*
git checkout calc-divide

make changes return x/y
save

status, add -A, commit -m "Added divide function"

no effect on local master

##push branch to server repo
git push -u origin calc-divide
*the -u associates remote repo branch with local one and allows us to
just use git push and git pull in future*

git branch -a
*should see local then remote branches*

branch is useful coz can be tested (unit tests etc) and after confirming
it works well, can merge with master

##merge with master
git checkout master
git pull origin master *always a good check*

git branch --merged *to see branches merged so far: no calc-divide*
```

```
git merge calc-divide
git push origin master

##delete branch
git branch --merged
git branch -d calc-divide *only deleted locally, what about remote repo?*

git branch -a *still on remote*
git push origin --delete calc-divide


#QUICK SUMMARY ON BRANCHES
git branch subtract
git checkout subtract
*make changes to file return x-y*
git status
git add -A
git commit -m "Added Subtract function"
git push -u origin subtract
*after tests*
git checkout master
git pull origin master
git merge subtract
git push origin master

*then you can delete etc*


#Jupyter
Help -> Shortcuts
Rename
2 modes - Command & Edit (get to CMD via ESC key, Click on Cell to Edit)
print ("Hello World") -> Enter just adds space
Cell execution methods + Shortcuts
No need for print("Hello World") -> "Hello World"
Cell numbers when executing (order of execution. Not always top to
bottom)
name = "Samer" (no output)
next cell -> name (gives output)
now next set name = "Ahmed" (go back and run, and now says "Ahmed")
run in order using Cell -> Run All (now order is same)
Run Below (saves a ton on execution time)

#Markdown
Insert cell above
Cell-> Cell Type -> Markdown

#Special commands
! -> bash command (instead of python)
!pip list

##in-built magics
% or %%
% on this line only
%% on the entire cell
```

```
%lsmagic - shows you all the magics

see a few %pwd (workding dir) %ls (files in dir)
%ls -la (long version, just like in bash terminal)

most common %matplotlib inline
allows to display charts within notebook
without running it, try
*matplotlib scatter*
won't complete that's what asterisk is for
Kernel -> Restart & Clear Outputs to kill
now run it then next cell
works

%%HTML
embed video (from YT)

%%timeit *e.g. for list comprehension of squaring numbers*
square_events = [n*n for n in range(0,1000)]

#without magic - inbuilt extras
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(10,5))
df
*shows dataframe*
df.head()
*top 5 rows*

Finally, File -> Download As (if you wanna put it on your blog - HTML)
Download and Open

Open Notebook in Sublime -> Just a JSON file

jupyter gallery
download trapezoid rule (Wiki-> Intro-> Collection)
mv ~/Downloads/Trap..  ./
ls
open it and see how they created the formula using latex
```

[redacted]