Introduction to python for AI

Jan Carbonell

July 10, 2019

Academy AI

Github 101

We are knowledge Workers

We create and edit **files** (text, images, etc.)

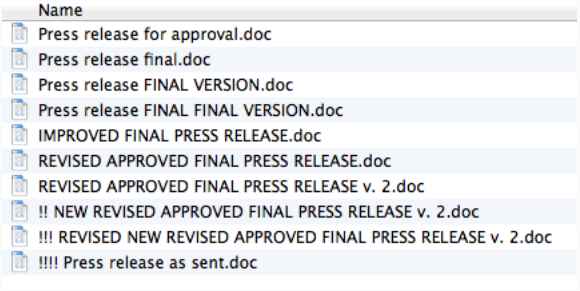
Typical Workflow

1. Create a file
2. Save it
3. Edit it
4. Save it again
5. and repeat the loop again...

Finding, saving, editing the wrong file, editing the right file takes **time**...

Manual version Control

Or also, how most people keep track of different versions of a file.



Name
Press release for approval.doc
Press release final.doc
Press release FINAL VERSION.doc
Press release FINAL FINAL VERSION.doc
IMPROVED FINAL PRESS RELEASE.doc
REVISED APPROVED FINAL PRESS RELEASE.doc
REVISED APPROVED FINAL PRESS RELEASE v. 2.doc
!! NEW REVISED APPROVED FINAL PRESS RELEASE v. 2.doc
!!! REVISED NEW REVISED APPROVED FINAL PRESS RELEASE v. 2.doc
!!!! Press release as sent.doc

Figure 1: Does this sound familiar?

Can this be automated / improved?

Yess!! For each document version, we would need to know:

1. **When** the file was modified
2. **What** changed
3. **Why** it was modified

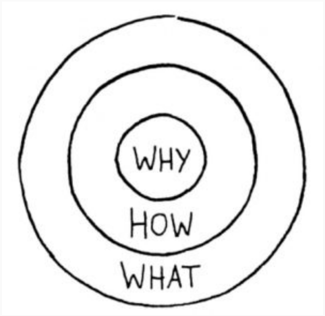


Figure 2: Side knowledge

Even more complicated: Shared work

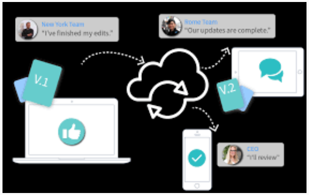


Figure 3: University memories for someone?

Which gives us the **following theorem**: *The more people you add to a project, the less productive it is and the odds of you getting mad increase exponentially*

Automation of the gdocs era:

For each document version, we would need to know:

1. **When** the file was modified
2. **What** changed
3. **Why** it was modified
4. **Who** did the change

In summary

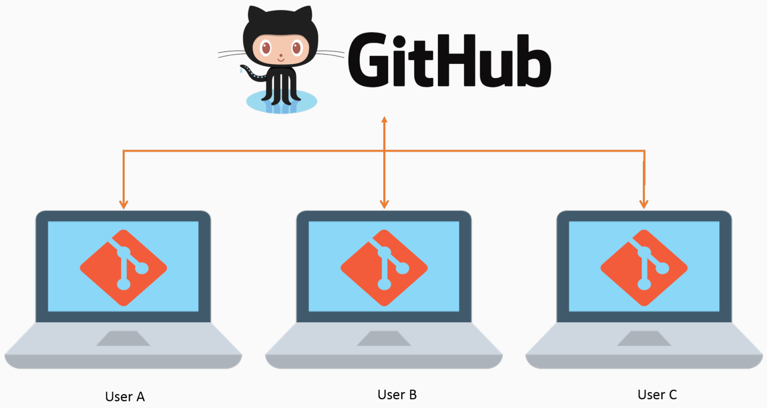
We want a tool in which we can:

1. track document versions
2. keep an history of document changes
3. foster teamwork

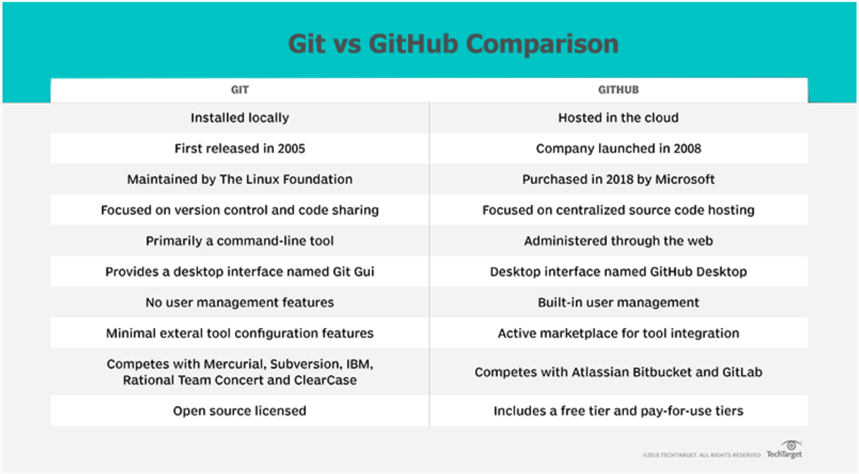
And that is...

git

So... what is github?



What is the difference?



Git vs GitHub Comparison

GIT	GITHUB
Installed locally	Hosted in the cloud
First released in 2005	Company launched in 2008
Maintained by The Linux Foundation	Purchased in 2018 by Microsoft
Focused on version control and code sharing	Focused on centralized source code hosting
Primarily a command-line tool	Administered through the web
Provides a desktop interface named Git Gui	Desktop interface named GitHub Desktop
No user management features	Built-in user management
Minimal exteral tool configuration features	Active marketplace for tool integration
Competes with Mercurial, Subversion, IBM, Rational Team Concert and ClearCase	Competes with Atlassian Bitbucket and GitLab
Open source licensed	Includes a free tier and pay-for-use tiers

Git Basic Commands

Starting

```
# From existing repository (on GitHub for instance)
git clone <github_ssh_clone_url>

# Or from scratch
mkdir new_project
cd new_project
git init
```

Status

git can tell you if your folder has some modified files (dirty)

```
git status
```

If you installed **OhMyZsh** as explained in the https://github.com/akademyai/setu
you will get some additional help from your terminal :)

Commit

A commit (a snapshot of the folder) is a 3-steps job.

```
# First check which files have been modified
git status

# Then, add the ones you want to the staging area.
git add <file_1_which_has_been_modified>
git add <file_2_which_has_been_modified>

# You can review your staging area
git status

# Take a snapshot of what is in the staging area.
git commit --message "A meaningful message about this change"
```

Diff

If git status tells you something changed, you can inspect exactly what changed:

```
git diff
git diff <a_specific_file_or_folder>
```

Log

Show commit history with:

```
git log

# More fancy command in your ~/.gitconfig
git lg
```

Live-code: git init

Let's create a project and start tracking it.

```
#before that, this is a good practice
~/code/\$GITHUB_USERNAME/
```

```
mkdir git-101
cd git-101
git init
ls -a # it has created a .git hidden folder
```

Live-code: first commit

Let's create an index.html file and code some basic HTML content

```
touch testing.py
```

Time to commit our work

```
git status #file not staged
git add testing.py
git status #file staged, ready to commit
git commit -m "Basic HTML content for home page"
git status
```

Live-code: second commit

Let's add another file

```
----code inside testing.py changed----
more_testing.py
```

Time to commit our work

```
git status
git diff index.html # what has changed?
git add testing.py
git add more_testing.py
git status
git commit -m "Coded my first function"
git status
git log # check commits history
```

Remote

Fork and clone

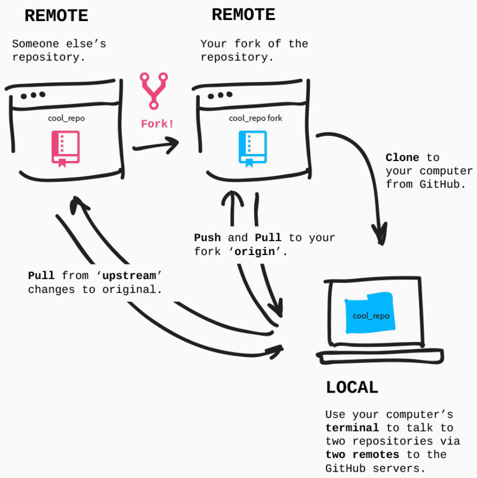

REMOTE
Someone else's repository.
cool_repo
Fork!
REMOTE
Your fork of the repository.
cool_repo fork
Clone to your computer from GitHub.
Push and Pull to your fork 'origin'.
Pull from 'upstream' changes to original.
cool_repo
LOCAL
Use your computer's terminal to talk to two repositories via two remotes to the GitHub servers.

Figure 4: Clone your things, Fork (copy) other repo's

Pushing the changes

Once you've committed your work, push it to Github.

```
# Generic command
git push <remote> <branch>

# What we'll use
git push origin master
```

Wait... where do we push it again?

Live-code: forking and cloning a remote

1. Let's fork the https://github.com/akademyai/setup
2. Then let's clone it with:

```
cd ~/code/$GITHUB_USERNAME$
git clone git@github.com:akademyai/setup.git
git status # it's already tracked by git
```

If you now get something related to ssh, remember to fix it right after this lecture

Live-code: commit and push

Let's make a change, commit and push

```
stt # change the Readme description
git add README.md
git commit -m "adding some custom text"
git status
git push origin master # Pushing on Github
```

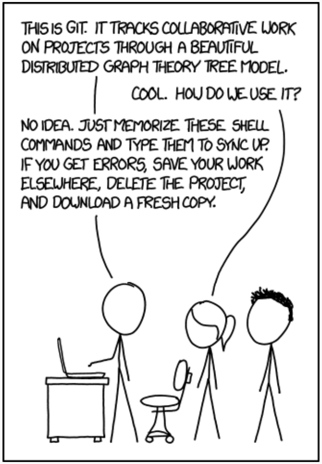
Check that **project was updated on Github.**

Git advanced

In the next final weeks, we'll see how git can help us with

1. Solving conflicts
2. Collaboration (using branches)
3. Production deployment (using multiple remotes)

TLDR;1


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

TLDR;2

COMMENT
DATE
CREATED MAIN LOOP & TIMING CONTROL
14 HOURS AGO
ENABLED CONFIG FILE PARSING
9 HOURS AGO
MISC BUGFIXES
5 HOURS AGO
CODE ADDITIONS/EDITS
4 HOURS AGO
MORE CODE
4 HOURS AGO
HERE HAVE CODE
4 HOURS AGO
AAAAAAAA
3 HOURS AGO
ADKFJSLKDFJSDKLFJ
3 HOURS AGO
MY HANDS ARE TYPING WORDS
2 HOURS AGO
HAAAAAAAANDS
2 HOURS AGO
AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Files

Handling files

Python lets us read, write and append files easily with the `open(filename, mode)` command. Depending on the mode we tell the OS how we want to open the file.

- **r** Reading.
- **r+** Reading and Writing with position at beginning.
- **w** Writing erasing file.
- **w+** Writing and Reading erasing file.
- **a** Appending to the file.
- **a+** Appending and reading the file.

```
with open('filename', 'r+') as file_handler:
    file_handler.readlines()
```

General tip: Don't panic

1. google is your friend
2. worst case scenario, ask for help (literally)

```
help(set)
```

Okay, now how the fuck do I leave help?

```
q
ctrl-z kills python
\textbf{panic method}: close and open the terminal
```

Let's start again: Handling files

1. In Python, there is no need for importing external library to read and write files.
2. Python provides an inbuilt function for creating, writing and reading files.

Today we will learn:

1. Why this is needed
2. How to Create a Text File
3. How to Append Data to a File
4. How to Read a File
5. How to Read a File line by line
6. File Modes in Python

Why files are needed

What happens when we execute a python file or we type an operation to the terminal?

1. Having prints is not enough
2. What if we want to store the results of the program but not the process? (It may take a while)
3. Let's look at a somewhat complex program.

Solution: Files! Text, csv, etc.

How to Create a Text File (1)

With Python you can create a .text files (akademy.txt) by coding! Let's see how it is done:

```
f= open("akademyai.txt","w+")
```

What we have done:

1. Variable f opens a file given 2 arguments, the file and the operation
2. We used w to write and the + to create a new file if it does not exist.
3. We could also "r" read or "a" append.

How to Create a Text File (2)

```
for i in range(10):
   f.write("This is line %d\r\n" % (i+1))
```

What we have done:

1. We have a for loop that runs over a range of 10 numbers.
2. Using the write function to enter data into the file.
3. The result is a file that tells us which number of line it is.
4. notice the special characters
   r *carriage return* and
   n *newline*
5. If you don't understand what something does, **remove it** and see what happens

How to Create a Text File (3)

```
f.close()
```

This will close the instance of the *akademy.txt* file (and automatically save it)

How to Create a Text File (4)

Adding all of this together, we have a basic program to write text in files!

```
def main():
    f= open("akademyai.txt","w+")
    for i in range(10):
        f.write("This is line %d\r\n" % (i+1))
    f.close()

if __name__== "__main__":
    main()
```

How to Append Data to a File (1)

Sometimes we don't want to erase what is already in a file. We can then append a new text to the already existing file or the new file.

```
f= open("akademyai.txt","a+")
```

With the + sign it would still create it but in our case we already have the file, so we are not required to create a new file.

How to Append Data to a File (2)

```
for i in range(2):
    f.write("Appended line %d\r\n" % (i+1))
```

This will write data into the file in append mode.

How to Append Data to a File (3)

Adding all of this together, we have a basic program to write text in files!

```
def main():
    f= open("akademyai.txt","a+")
    for i in range(2):
         f.write("Appended line %d\r\n" % (i+1))
    f.close()

if __name__== "__main__":
    main()
```

We can now see that the output in our file has been updated with our new data below

How to Read a File (1)

Not only you can create .txt file from Python but you can also call .txt file in a ”read mode”(r).

```
f= open("akademyai.txt","r")
```

How to Read a File (2)

We use the mode function in the code to check that the file is in open mode. If yes, we proceed ahead

```
if f.mode =='r':
```

How to Read a File (3)

Now we use f.read to read file data and store it in variable content

```
contents =f.read()
```

How to Read a File (4)

Adding all of this together, we have a basic program to write text in files!

```
def main():
    f= open("akademyai.txt","r")
    if f.mode =='r':
        contents =f.read()
        print(contents)
    #No need to close here, but it is still good practice to
                                                           do so.

if __name__== "__main__":
    main()
```

Thinking

1. What if I want to do certain operations in every line? Ex: Web Scrapping? Ex2: Only store something in another file if it contains an A
2. What if I don't want to save / print the entire text? Think about big files like all the wikipedia entries?

Solution: Reading line by line!

How to Read a File Line by Line (1)

Adding all of this together, we have a basic program to write text in files!

```
def main():
    f= open("akademyai.txt","r")
    f1 =f.readlines()
    for line in f1:
        if line ==’\n’:
            print("found an empty line")
        else:
            print(x)

    #No need to close here, but it is still good practice to
                                                    do so.
    f1.close()
if __name__== "__main__":
    main()
```

Modules

Module structure

Every python file creates a module, a module is a set of function or classes that can be imported from other files. This lets us divide our logic into semantically similar functionalities.

Typically we will implement a class or a set of methods per file and then import them from other files.

```
#--------------------file1
def f(a, b):
   return a *b
#--------------------file2
from file1 import f
f(2, 2) # 4
```

The next slides have been generated thanks to **Kaggle External Libraries Tutorial**

The power of Python

One of the best things about Python (especially if you're a data scientist) is the vast number of high-quality custom libraries that have been written for it.

Google: **python external libraries**

Also, this open-source approach is the main reason Python became the main language for Deep Learning (Researchers working together).

The standard library

Some of these libraries are in the ”standard library”, meaning you can find them anywhere you run Python. **It is always good to check if something is already in the standard library before trying to code it yourself**.

Ex: Can we tell what time it is now with Python?

Math module

One of the most famous libraries is the math one (useful for DL ;))

```
import math

print("It’s math! It has type {}".format(type(math)))
```

Math is a module. A module is just a collection of variables (a namespace, if you like) defined by someone else. We can see all the names in math using the built-in function dir(). **This is very useful to know what any module does**

```
print(dir(math))
```

How do we access this variables?

The computer needs to know it comes from the math module. Thus, we need to declare it beforehand with math.function.

```
print("pi to 4 significant digits = {:.4}".format(math.pi))
```

But most of what we'll find in the module are functions, like math.log:

```
math.exp(32, 2)
```

Of course, if we don't know what math.log does, we can call help() on it:

```
help(math.log)
```

We can also call help() on the module itself. This will give us the combined documentation for all the functions and values in the module.

Other import syntax

If we know we'll be using functions in math frequently we can import it under a shorter alias to save some typing (though in this case "math" is already pretty short).

```
import math as mt
mt.pi
```

Now you are thinking... we only save two letters. What about this?

```
import pandas as pd
import numpy as np
import import sklearn.preprocessing.scale as prepro
```

Very often used in ML / DL to the point that some things are a standard (numpy as np, pandas...

Can we optimize more?

Wouldn't it be great if we could refer to all the variables in the math module by themselves? i.e. if we could just refer to pi instead of math.pi or mt.pi? **Good news**: we can do that.

```
from math import *
print(pi, log(32, 2))
```

import * makes all the module's variables directly accessible to you (without any dotted prefix).

Bad news: some purists might grumble at you for doing this.

I still want to optimize...

Even Worse News : they kind of have a point.

```
from math import *
from numpy import *
print(pi, log(32, 2))
```

Error! But it worked before! These kinds of "star imports" can occasionally lead to weird, difficult-to-debug situations. In this case, the math and numpy modules both have functions called log, but they have different semantics.

Because we import from numpy second, its log overwrites (or "shadows") the log variable we imported from math.

Let's compromise

A good compromise is to import only the specific things we'll need from each module:

```python
from math import log, pi
from numpy import asarray
```

Submodules

We've seen that modules contain variables which can refer to functions or values. Something to be aware of is that they can also have variables referring to other modules.

```
import numpy
print("numpy.random is a", type(numpy.random))
print("it contains names such as...",
     dir(numpy.random)[-15:]
    )
```

So if we import numpy as above, then calling a function in the random "submodule" will require two dots.

```
# Roll 10 dice
rolls =numpy.random.randint(low=1, high=6, size=10)
rolls
```

Toolkit for the future

Three tools for understanding strange objects

1. type() (what is this thing?) *Ex: type(rolls)*
2. dir() (what can I do with it?) *Ex: print(dir(rolls))*
3. help() (tell me more) help(rolls.tolist())

SELF EXPLORE OR TOMORROW Random module

https://pynative.com/python-random-module/

SELF EXPLORE OR TOMORROW Time module

SELF EXPLORE OR TOMORROW Execution time

FOR TOMORROW: Classes

Class Definition

Python is an object oriented programming language and therefore has its own form of defining classes with the special keyword `class`.

```
class MyClass(object): # inherits from object (optional)
    multiply_by =2 # Class variable

    def __init__(self, a): # constructor
        self.a =a # instance variable

    def f(self): # instance method
        return self.a *MyClass.multiply_by
```

Function inside classes need to have as first parameter `self`, this means that the function is an instance function and allows the function to access the properties of the object.

Class instantiating

Instantiating a class is straightforward (no need of new):

```
obj =MyClass(2)
```

Function invocation is just as easy

```
obj.f() # 4
```

Exercise

Create a class that holds the information of person (name, surname, born_date and gender).

Exercise

Add to the class created earlier a method that represents the object as a string in the following way:

'Surname, Name - Age - Gender'

Exercise

1. Change the previous method so that the interpreter knows how to represent the object (toString())
2. Create a function that given a list of integers returns a list of the same integers powered to a number given by parameter. (([1, 2], 2) - [1, 4])
3. Create a function that given a string and a character counts the length of each substring divided by the character (('Hello world!', ' ') - [5, 6]).

Special Methods I

Python provides us the opportunity to implement some special methods and add our classes to complex behaviours such as generators, string representations or comparisons. The following are some examples:

1. `__repr__`: Similar to to_string
2. `__gt__` `__lt__` greater than and less than respectively
3. `__ge__` `__le__` greater than or equal and less than or equal respectively
4. `__eq__` `__ne__` equal and not equal respectively-
5. `__add__` add.

Special Methods II

With this behaviour we can do operator overflowing hiding logic transparently and creating a fluent API.

```python
class Point:
    def __init__(self, x, y):
        self.x =x
        self.y =y
    def __add__(self, other):
        x =self.x +other.x
        y =self.y +other.y
        return Point(x,y)
p1 =Point(2,3)
p2 =Point(-1,2)
p1 +p2 # (1,5)
```

Introduction to IPython and Jupyter

IPython

Ipython is an interactive console that adds some functionalities that allows us to develop with ease.

To run it we just need to run `ipython` in the console. A shell will prompt and we will be able to execute commands from there on. Some of the features of IPython are:

1. Tab completion.
2. Run multiline commands.
3. Introspection (*dir*).
4. Run commands from the clipboard with `\%paste`.
5. Run commands from another file `\%run`.
6. Beautiful printing stack traces and outputs.

Magic commands I

The shell brings some utility commands that will allow us to perform some actions such as timing or profiling easily. A list of those can be seen here.

- **%quickref** Display the IPython Quick Reference Card
- **%magic** Display detailed documentation for all of the available magic commands
- **%debug** Enter the interactive debugger at the bottom of the last exception traceback
- **%hist** Print command input (and optionally output) history
- **%pdb** Automatically enter debugger after any exception
- **%paste** Execute pre-formatted Python code from clipboard
- **%cpaste** Open a special prompt for manually pasting Python code to be executed
- **%reset** Delete all variables / names defined in interactive namespace

Magic commands II

- **%page OBJECT** Pretty print the object and display it through a pager
- **%run script.py** Run a Python script inside IPython
- **%prun statement** Execute statement with cProfile and report the profiler output
- **%time statement** Report the execution time of single statement
- **%timeit statement** Run a statement multiple times to compute an emsemble average execution time. Useful for timing code with very short execution time
- **%who, %who_ls, %whos** Display variables defined in interactive namespace, with varying levels of verbosity
- **%xdel variable** Delete a variable and attempt to clear any references to the object in the IPython internals

Running shell commands

This environment let us run shell commands from the OS by using `!` before a statement, so we could for example see the current directory or change it.

```
files =!ls
files # main.py, .git
```

Jupyter notebooks

On top of IPython emerges Jupyter allowing us to run IPython in the browser with rich outputs and a lot of benefits. To span a jupyter process you just have to do the following:

```
$ jupyter notebook
```