ICS Spring 2018

Lab Session 5

Agenda

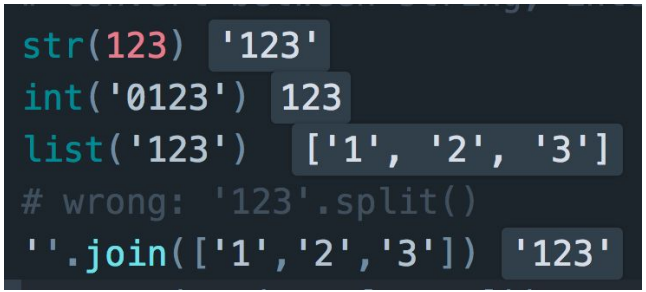
- **Quiz 1 recap**
- **Recursion**
- **Github**
- **Pickling(Serialization)**
- **OOP**
- **In-class exercises**

Quiz 1 recap

Quiz 1, Black hole problem

Convert between str, int and list:

str(), int(), join()



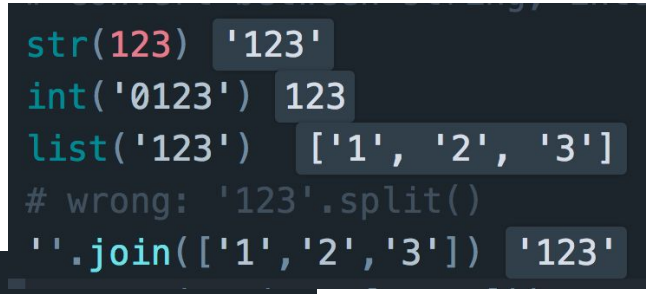
Quiz 1, Black hole problem

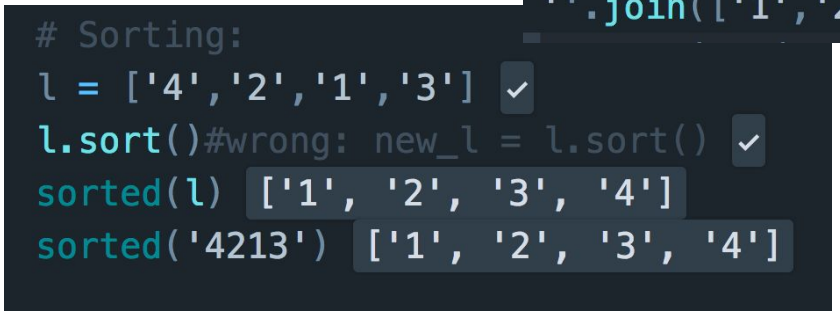
Convert between str, int and list:

str(), int(), join()

Sorting:

sort(), sorted()





Quiz 1, Black hole problem

Convert between str, int and list:

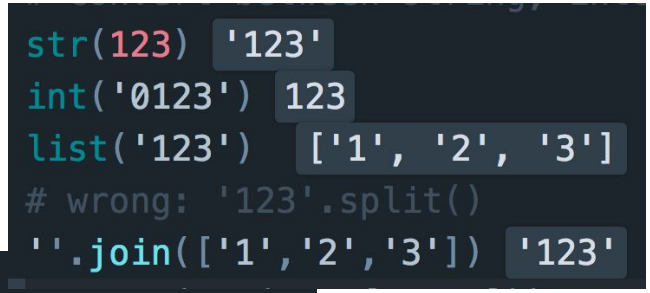
str(), int(), join()

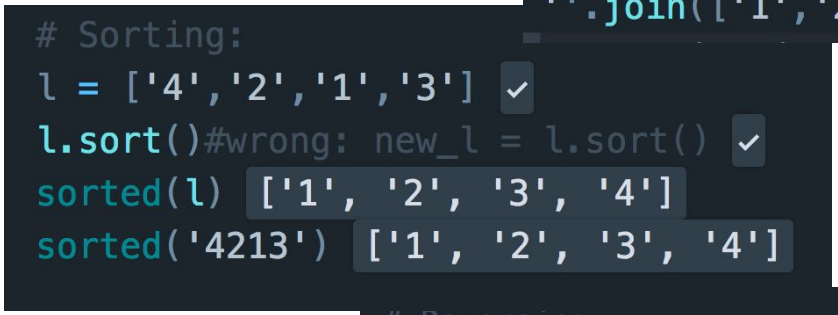
Sorting:

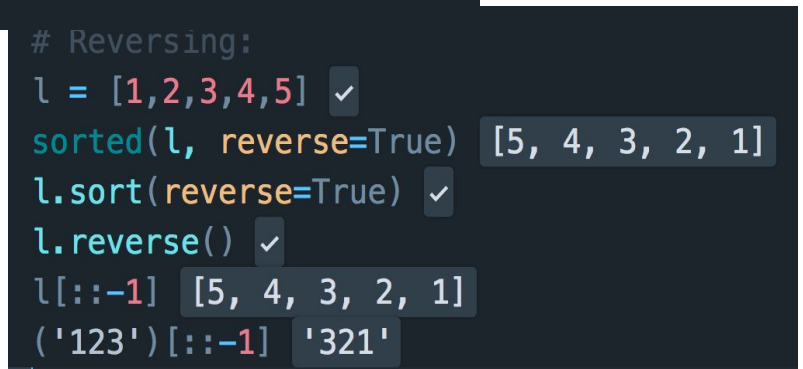
sort(), sorted()

Reversing:

sort(reverse=True),
sorted(reverse=True),
reverse(),
slicing[::-1]







Recursion

Recursion

- Concept: A recursive function is a function that calls itself.
- This is a recursive function which runs forever
  - (unless you interrupt with Ctrl+C).

```
def main():
    message()

def message():
    print("This is a recursive function.")
    message()

main()
```

Recursion

•Similar to a loop, a recursive function can be controlled by some decision structure.

```
def main():
    message(5)

def message(times):
    if times > 0:
        print("This is a recursive function.")
        message(times - 1)

main()
```

Recursion Examples

- Fibonacci Series:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Recursion Examples

- Mergesort:

```
def merge_sort(m):
    print(m)

    if len(m) <= 1:
        return m

    middle = len(m) // 2
    left = m[:middle]
    right = m[middle:]

    left = merge_sort(left)
    right = merge_sort(right)
    return merge(left, right)
```

Comprehension

```
def powerset_comb(l):
    pset = []
    total_items = len(l)
    for i in range(2 ** total_items):
        subset = []
        code = bin(i).split('b')[-1]
        code = code[::-1]
        for j in range(len(code)):
            if code[j] == '1':
                subset.append(l[j])
        pset.append(subset)
    return pset
```

```
def powerset_comb_list_comprehension(l):
    pset = []
    total_items = len(l)
    for i in range(2 ** total_items):
        code = bin(i).split('b')[-1]
        code = code[::-1]
        subset = [l[j] for j in range(len(code)) if code[j] == '1']
        pset.append(subset)
    return pset
```

Recursive Powerset with List Comprehension

```
def powerset_recursion(l):
    if not l:
        return [[]]
    subset = []
    for x in powerset_recursion(l[1:]):
        subset.append([l[0]] + x)
    return powerset_recursion(l[1:]) + subset
```

Use debugger to step into the code!

- Set breakpoint
- Execute/continue
- Print variables….

Recursive Powerset with List Comprehension

```
def powerset_recursion(l):
    if not l:
        return [[]]
    subset = []
    for x in powerset_recursion(l[1:]):
        subset.append([l[0]] + x)
    return powerset_recursion(l[1:]) + subset


def powerset_recursion_comp(l):
    # Base case: the empty set
    if not l:
        return [[]]
    # The recursive relation
    # Do a powerset call for l[1:]
    # Add lists of all combinations of the 1st element (l[0]) with other elements' powersets
    return powerset_recursion_comp(l[1:]) + [[l[0]] + x for x in powerset_recursion_comp(l[1:])]
```

Recursion Exercise

Time to work on Problem 1 in today's in-class exercise!

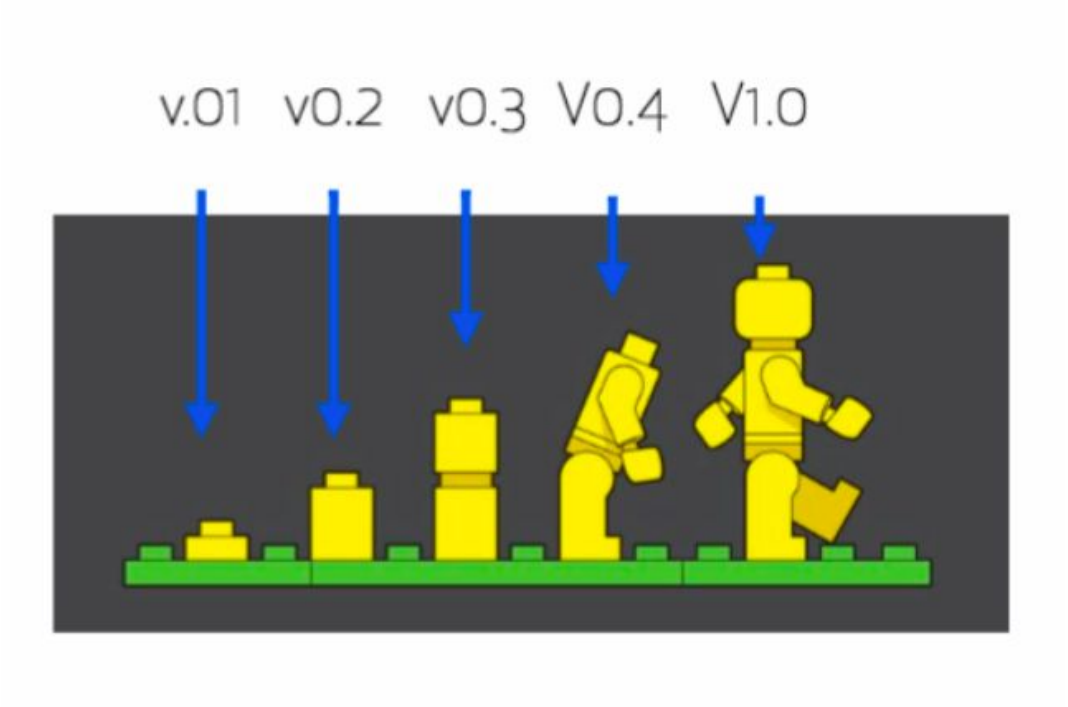
Github

Some materials gathered from Github campus advisors

Git is a version control system

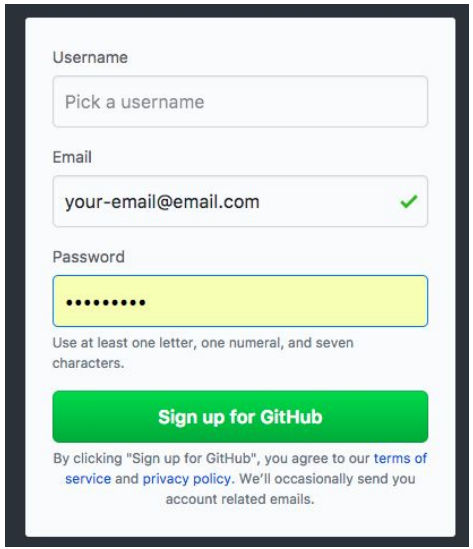
Track your changes and progress overtime.

Go back to previous version when desired.

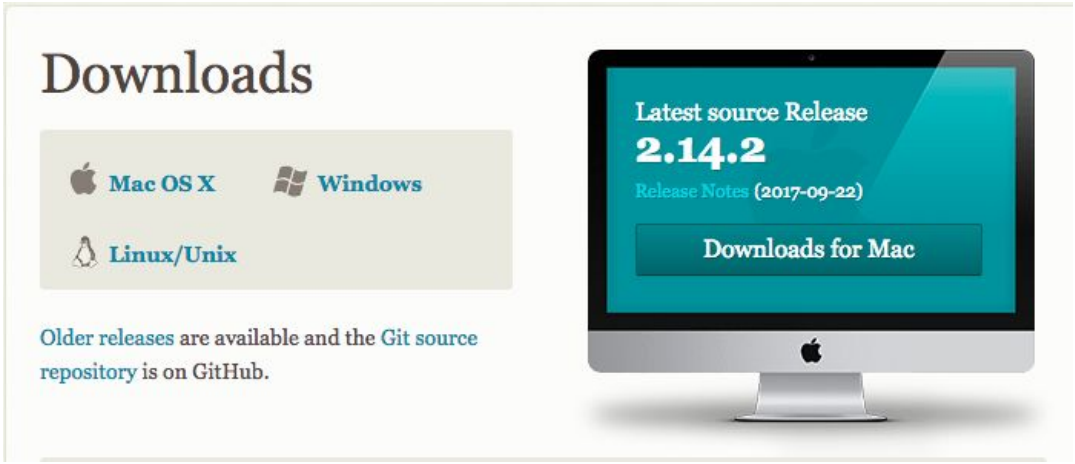


Register github & install Git bash

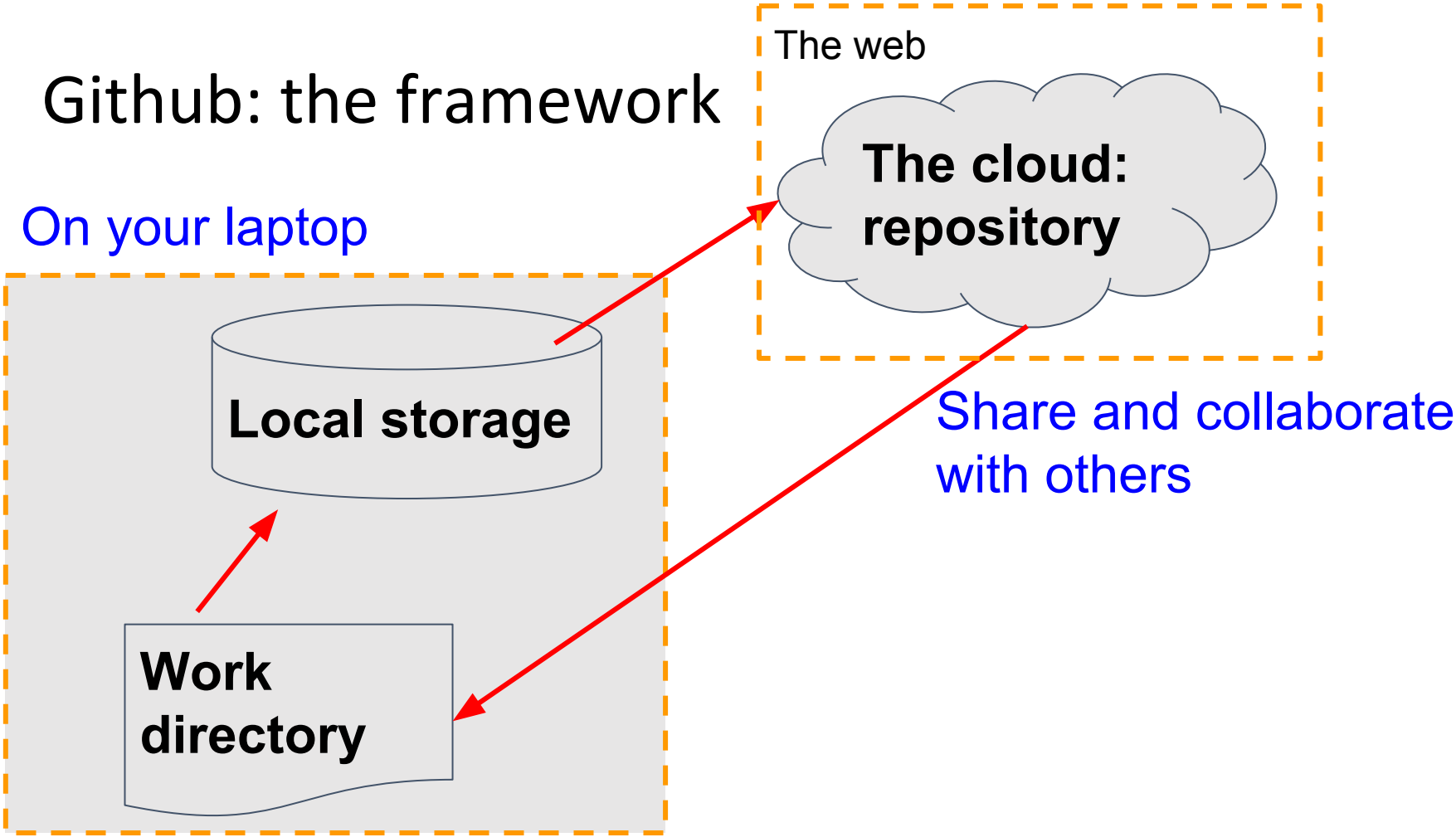
Sign up: https://github.com/



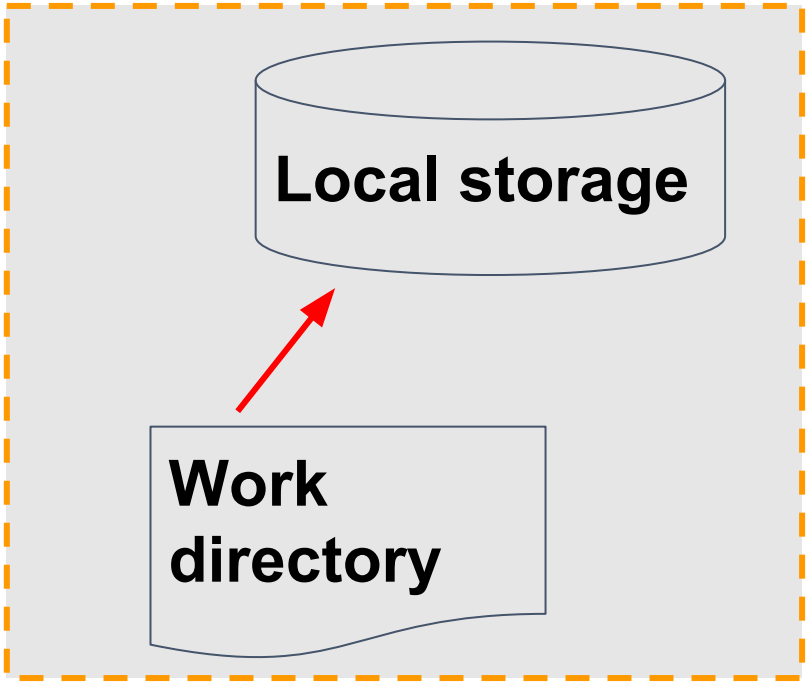
Install Git bash : https://git-scm.com/downloads



Github: the framework



Github: the framework



On your laptop

How does the version control work exactly?

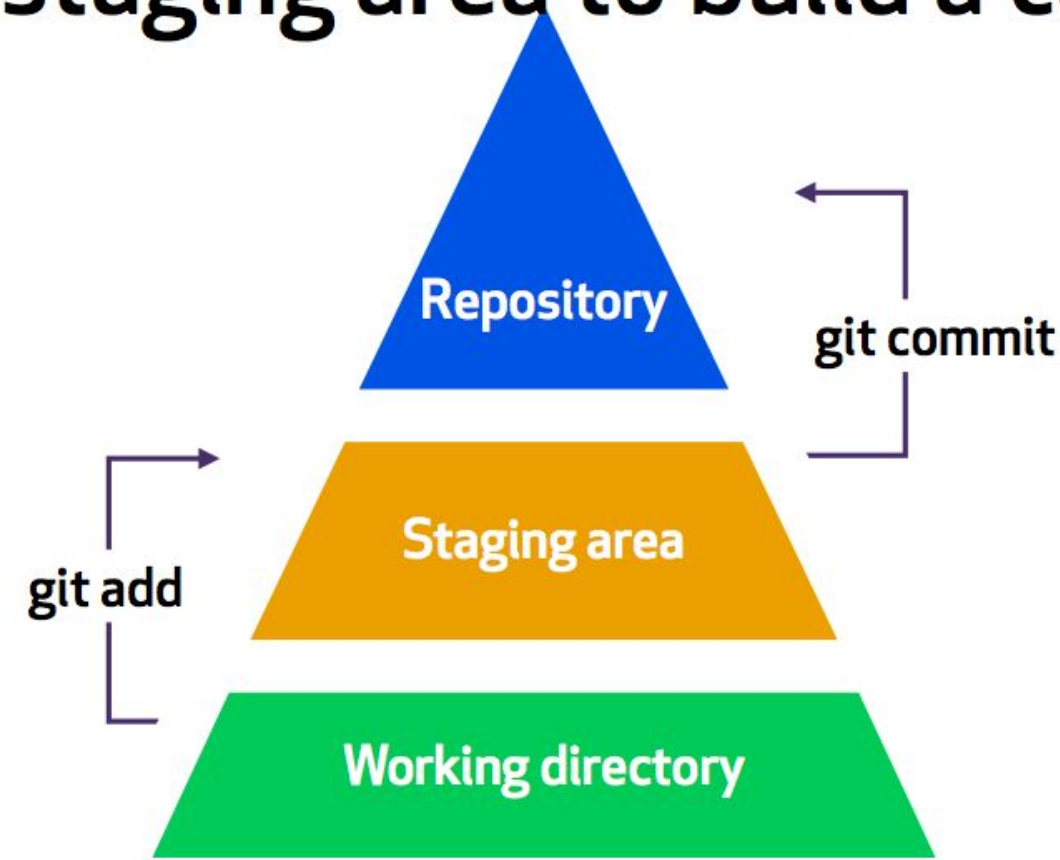
Git takes snapshots

Save snapshots to your history to retrace your steps.

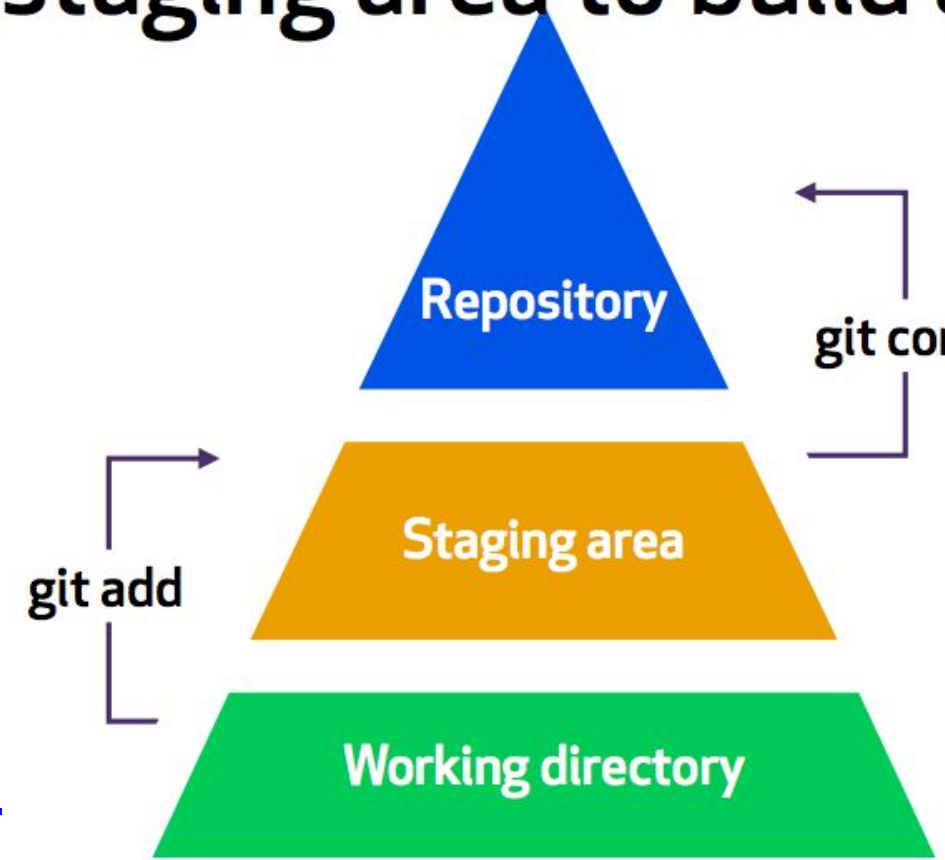
Also keeps others up-to-date with your latest work.

snapshot = “commit” in git vocabulary

Use the staging area to build a commit



Use the staging area to build a commit



A version you definitely want to commit.

A version you might commit.

Use Case: Track of your ICS codes

Suppose you have a ICS folder with one python file you want to keep track of changes:

Directory: desktop/git_demo

Original version : Downloaded from NYU class as code template.

1.Create a repository: Tell git you want to track this folder

```
cd desktop/git_demo
git init

ls -la # check files
```

Note: Right now, nothing is tracked by git yet

2. Make initial commit

```
git add xxxx.py
git commit -m 'initial commit'
git log # show commit hisotry
```

Why “add”?
Store the new version in staging area.
But you can still make further changes before commit

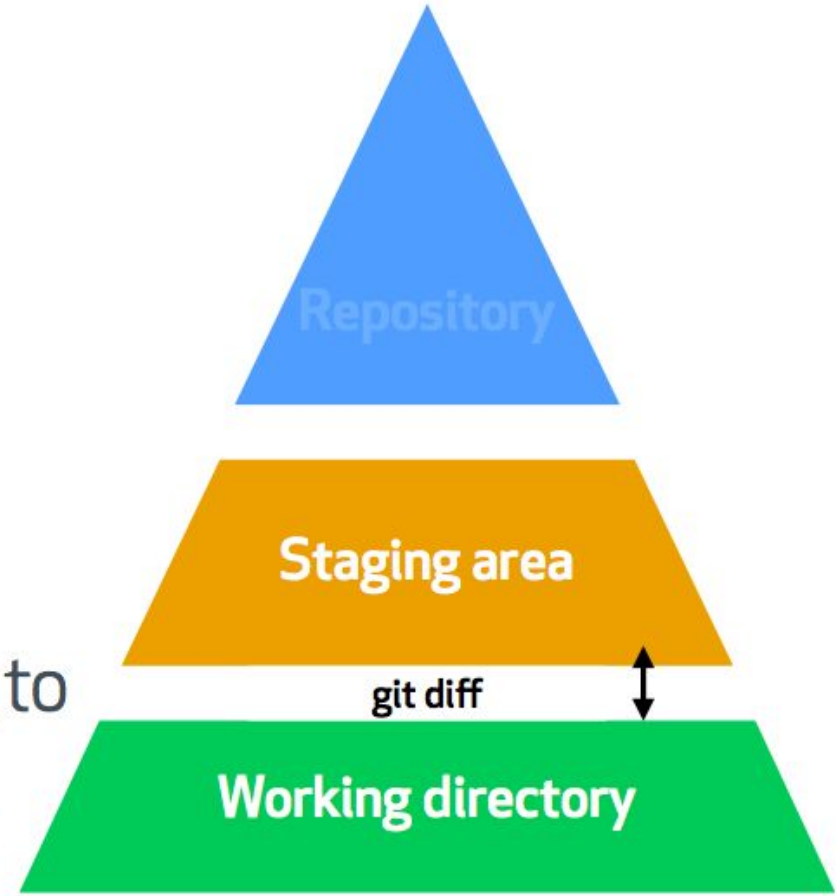
3. Work on the code, then check differences

```
git diff
```

```
git diff --staged
```

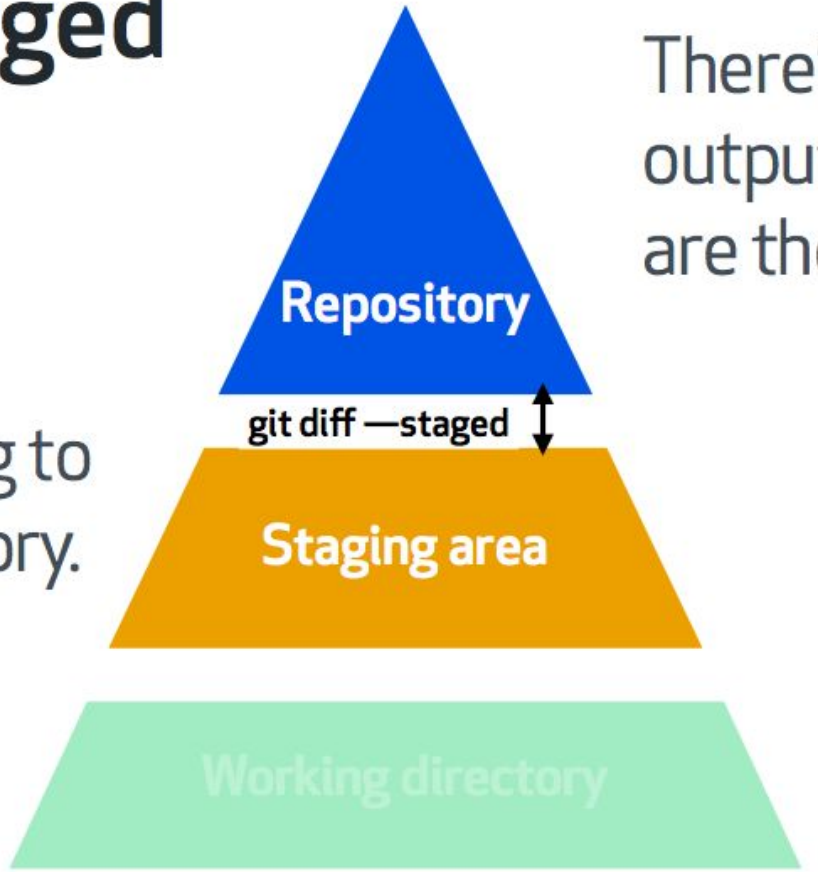
git diff

Compares staging to working directory.



There's no output if they are the same.

git diff --staged



There's no output if they are the same.

Compares staging to repository directory.

4. Commit your changes

```
git add xxxxx.py

git commit -m 'xxxx version 1'

git log
```

After git log, you should see 2 commits, with name, author and time.

5. Go back to initial (or any previous) commit

- If you just want take a look of a past version….
- … and then go back to most recent commit.

```
git checkout <commit name>
git checkout master
```

- If you want to make new changes that are different than most recent commit: (we will discuss that next week)

1. make a branch
2. discard all commit since this previous commit

Pickles & Object Serialization

Pickling

- We can directly work with binary files with the pickle module
- To use: import the pickle module, use the appropriate file mode
  - ‘r’ -> ‘rb’ (read binary)
  - ‘w’ -> ‘wb’
- pickle.**dump**()
  - Writes an object directly to file as its native data type
    - (***not*** a string!)
- pickle.**load**()
  - Reads the first available binary object in the file and returns it

Pickling Analysis

Serialize the data:

```
import pickle
capitals = {"Illinois":"Springfield",
            "California":"Sacramento",
            "New York":"Albany"}
output_file = open("capitals.dat", "wb")
pickle.dump(capitals, output_file)
output_file.close()
```

Pickling Analysis

Load the serialized data

```
import pickle
input_file = open("capitals.dat", "rb")
caps = pickle.load(input_file)
print(caps)
```

Output:

```
{'California': 'Sacramento', 'Illinois': 'Springfield', 'New York': 'Albany'}
```

Object Oriented Programming (OOP)

Elegance

The programmer's conundrum:

Pick any two!

Convenience

Clarity

Example of a Class

the __init__ method
-- define the variables

Mutate method(Setter)

[Coding style (again)!]

Access method(Getter)

program. We will add to it as we go along.

Program 10-1 (Coin class, not a complete program)

```
1  import random
2
3  # The Coin class simulates a coin that can
4  # be flipped.
5
6  class Coin:
7
8      # The __init__ method initializes the
9      # sideup data attribute with 'Heads'.
10
11     def __init__(self):
12         self.sideup = 'Heads'
13
14     # The toss method generates a random number
15     # in the range of 0 through 1. If the number
16     # is 0, then sideup is set to 'Heads'.
17     # Otherwise, sideup is set to 'Tails'.
18
19     def toss(self):
20         if random.randint(0, 1) == 0:
21             self.sideup = 'Heads'
22         else:
23             self.sideup = 'Tails'
24
25     # The get_sideup method returns the value
26     # referenced by sideup.
27
28     def get_sideup(self):
29         return self.sideup
```

Everything in Python is Object!

- Everything is actually a class instance in python
  - And almost ***everything*** has attributes and methods

- Anything which can be used as a value (int, str, float, functions, modules, etc) are implemented as objects.

OOP Exercises

Now, please go ahead working on Problems 2 & 3 in today's in-class exercise!