Foundations of Data Science

Master's in Data Science

2022 / 2023

Setup

Conda environment

- CONDA
  - Package management and environment management
  - Install, run and update packages and dependencies
  - Create, save, load and switch between environments

$ conda env create -f ds_env.yml

$ conda activate ds

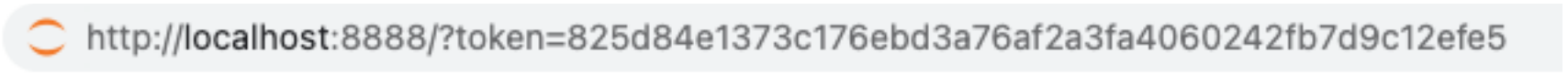
(ds) $

```yaml
name: ds
channels:
- anaconda
- defaults
- conda-forge
dependencies:
- python
- pandas
- seaborn
- pyspark
- scikit-learn
- tensorflow
- keras
- networkx
- nltk
- jupyterlab
```

Jupyter Lab

- Web-based interactive development environment for Jupyter notebooks, code, and data
- Highly popular with data scientists

$ jupyter lab



Warm-Up: Python

Hello world!

```
% python
Python 3.7.6 (default, Jan  8 2020, 13:42:34)
>>> print(“Hello world!”)
Hello world!
>>>
```

Interactive shell

Hello world!

```
% vim hello.py
% cat hello.py
print("Hello world!")
% python hello.py
Hello world!
```

Executing a script

A simple program

```
import sys
from time import localtime, strftime

def echo(greeting):
    print(greeting)

def main():
    current_time = localtime()
    greeting = "Hello, " + sys.argv[1] \
        + "\nThe time now is " \
        + strftime("%d %b %Y %H:%M:%S", current_time)
    echo(greeting)

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

Imports

Function definition

Assigning a value to a variable

Using command line arguments

Calling a function

Program entry point

% python greeting.py Sérgio

Hello, Sérgio

The time now is 15 Sep 2021 10:27:39

Data types

- Numeric: *int*, *float*, *complex*
  n = 3; x = 5.76; w = 3 + 5j
- Text: *str*
  s = "Hello!”
- Data Structures: *list*, *tuple*, *dict, set*
  l = [0, 4, 8, 12, 16]
  t = (3, 5, ‘e’)
  d = {‘a’: 1, ‘b’:2, ‘c’:3}
  s = {5, 7, 11, 12}
- Nulls: NoneType
  - m = None

Conditions

```
import sys
from time import localtime, strftime

def echo(greeting):
    print(greeting)

def main():
    current_time = localtime()
    if current_time.tm_hour >= 6 and current_time.tm_hour < 12:
        greeting = "Good morning, "
    elif current_time.tm_hour >= 12 and current_time.tm_hour < 20:
        greeting = "Good afternoon, "
    else:
        greeting = "Goog evening, "

    greeting += sys.argv[1] \
        + "\nThe time now is " \
        + strftime("%d %b %Y %H:%M:%S", current_time)
    echo(greeting)

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

Loops

```
u = input("What's your name? ")
n = 0
for c in u.lower():
    if c >= 'a' and c <= 'z': n += 1
    if n > 1: break
```

We wouldn’t actually do this!

```
while True:
    u = input("What's your name? ")
    if len(re.sub(r"[^a-z]+", '', u.lower()))>1: break
```

Regular expressions

```
import re

def main():

    dob_re = re.compile("^(\d{2})[/-](\d{2})[/-](\d{4})$")

    while True:
        dob = input('Date of birth: ')
        res = dob_re.match(dob)
        if not res:
            print("Incorrect date format, use DD/MM/YYYY or DD-MM-YYYY.\n")
            continue

        dd = res.group(1); mm = res.group(2); yy = res.group(3)
        print("You were born on the {}th day of the {}th month in the year {}" \
            .format(dd, mm, yy))

        break

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

Regular expressions: named groups

```
import re

def main():

    dob_re = re.compile("^(?P<dd>(\d{2}))[/-](?P<mm>(\d{2}))[/-](?P<yy>(\d{4}))$")

    while True:
        dob = input('Date of birth: ')
        res = dob_re.match(dob)
        if not res:
            print("Incorrect date format, use DD/MM/YYYY or DD-MM-YYYY.\n")
            continue

        print("You were born on the {}th day of the {}th month in the year {}" \
            .format(res.group('dd'), res.group('mm'), res.group('yy')))

        break

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

I/O

```
import os

def main():
    while True:
        fname = input('File to read: ')
        if not os.path.isfile(fname):
            print('File {} does not exist.\n'.format(fname))
            continue

        f = open(fname)
        for i, line in enumerate(f):
            print('{line_number}: {text}'.format(line_number=i, text=line))


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

I/O

```
import os

def main():
    while True:
        fname = input('File to read: ')
        if not os.path.isfile(fname):
            print('File {} does not exist.\n'.format(fname))
            continue

        with open(fname) as f:
            for i, line in enumerate(f):
                print('{line_number}: {text}'.format(line_number=i, text=lin


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

Handling exceptions

```
import os

def main():
    while True:
        fname = input('File to read: ')
        try:
            f = open(fname)
            for i, line in enumerate(f):
                print('{line_number}: {text}'.format(line_number=i, text=lin
        except:
            print('File {} does not exist.\n'.format(fname))


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

Handling exceptions

```
import os

def main():
    while True:
        fname = input('File to read: ')
        try:
            f = open(fname)
        except Exception as e:
            print(e)
        else:
            for i, line in enumerate(f):
                print('{line_number}: {text}'.format(line_number=i, text=line))


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

List comprehension & Generators

- List comprehensions

```
[(n, n**2) for n in range(100) if n % 3]
```

- Generators

```
(n for n in range(100) if n % 2)
```

```
def gen():
    for n in range(12):
        yield n ** 2
```

lambda, map, filter

```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, items))


def multiply(x):
    return (x*x)
def add(x):
    return (x+x)

funcs = [multiply, add]
for i in range(5):
    value = list(map(lambda x: x(i), funcs))


number_list = range(-5, 5)
less_than_zero = list(filter(lambda x: x < 0, number_list))
```

Function annotations

```
>>> def f(ham: str, eggs: str = 'eggs') -> str:
...     print("Annotations:", f.__annotations__)
...     print("Arguments:", ham, eggs)
...     return ham + ' and ' + eggs
...
>>> f('spam')
Annotations: {'ham': <class 'str'>, 'return': <class 'str'>, 'eggs': <class 'str'>}
Arguments: spam eggs
'spam and eggs'
```

Decorators

- Wrap a function and modify its behaviour in some way

```
from functools import wraps

def a_new_decorator(a_func):
    @wraps(a_func)
    def wrapTheFunction():
        print("I am doing some boring work before executing a_func()")
        a_func()
        print("I am doing some boring work after executing a_func()")
    return wrapTheFunction

@a_new_decorator
def a_function_requiring_decoration():
    """Hey yo! Decorate me!"""
    print("I am the function which needs some decoration to "
          "remove my foul smell")

print(a_function_requiring_decoration.__name__)
# Output: a_function_requiring_decoration
```

https://book.pythontips.com/en/latest/decorators.html

PEP 8 -- Style Guide for Python Code

https://www.python.org/dev/peps/pep-0008/

- Code Lay-out
  - Indentation / Tabs or Spaces?
  - Blank Lines
  - Source File Encoding
- String Quotes
- Whitespace in Expressions and Statements
- When to Use Trailing Commas
- Comments
  - Documentation Strings
- Naming Conventions
- Programming Recommendations
  - Function Annotations
  - Variable Annotations

Warm-Up: Numpy, Scipy

Numpy, Scipy

- Numpy
  - Multidimensional arrays
  - Linear algebra

- Scipy
  - Linear algebra
  - Statistics, inc. distributions
  - Optimization, Interpolation, …

Numpy data types

- ndarray: n-dimensional array
  - Fixed size, same data type

- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float16, float32, float64
- complex64, complex128

```
x = np.array([3, 6, 2])

np.arange(3, dtype=np.uint8)
```

1-dimensional arrays

```
import numpy as np

a1 = np.array([1.87,  1.87, 1.82, 1.91, 1.90, 1.85])
a2 = np.array([81.65, 97.52, 95.25, 92.98, 86.18, 88.45])

np.mean(a1)

# 1.87

np.mean(a2)

# 90.33833333333335

np.corrcoef(a1, a2)

# array([[ 1.        , -0.23077148],
#        [-0.23077148,  1.        ]])
```

N-dimensional arrays

```
import numpy as np

x = np.arange(27).reshape((3,3,3))

# array([[[ 0,  1,  2],
#         [ 3,  4,  5],
#         [ 6,  7,  8]],
#
#        [[ 9, 10, 11],
#         [12, 13, 14],
#         [15, 16, 17]],
#
#        [[18, 19, 20],
#         [21, 22, 23],
#         [24, 25, 26]]])

x.sum(axis=0)

# array([[27, 30, 33],
#        [36, 39, 42],
#        [45, 48, 51]])
```

Useful methods

```
np.zeros((3, 4))

np.linspace(-3, 3, 20)

np.random.random((3, 4))

np.diag(range(4))
```

Indexing and slicing

```
x = np.arange(30)
x[-2:]
```

```
array([28, 29])
```

```
x = x.reshape(3, 10)
x
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]])
```

```
x[1]
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
x[2, :5]
```

```
array([20, 21, 22, 23, 24])
```

```
x[:, 1::3]
```

```
array([[ 1,  4,  7],
       [11, 14, 17],
       [21, 24, 27]])
```

Array operations

```
x1 = np.arange(9).reshape(3,3)
x2 = x1 + 2
x2
```

```
array([[ 2,  3,  4],
       [ 5,  6,  7],
       [ 8,  9, 10]])
```

```
x1 + x2                    # element-wise
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

```
x1 * x2                    # also element-wise
```

```
array([[ 0,  3,  8],
       [15, 24, 35],
       [48, 63, 80]])
```

```
np.dot(x1, x2)             # matrix multiplication
```

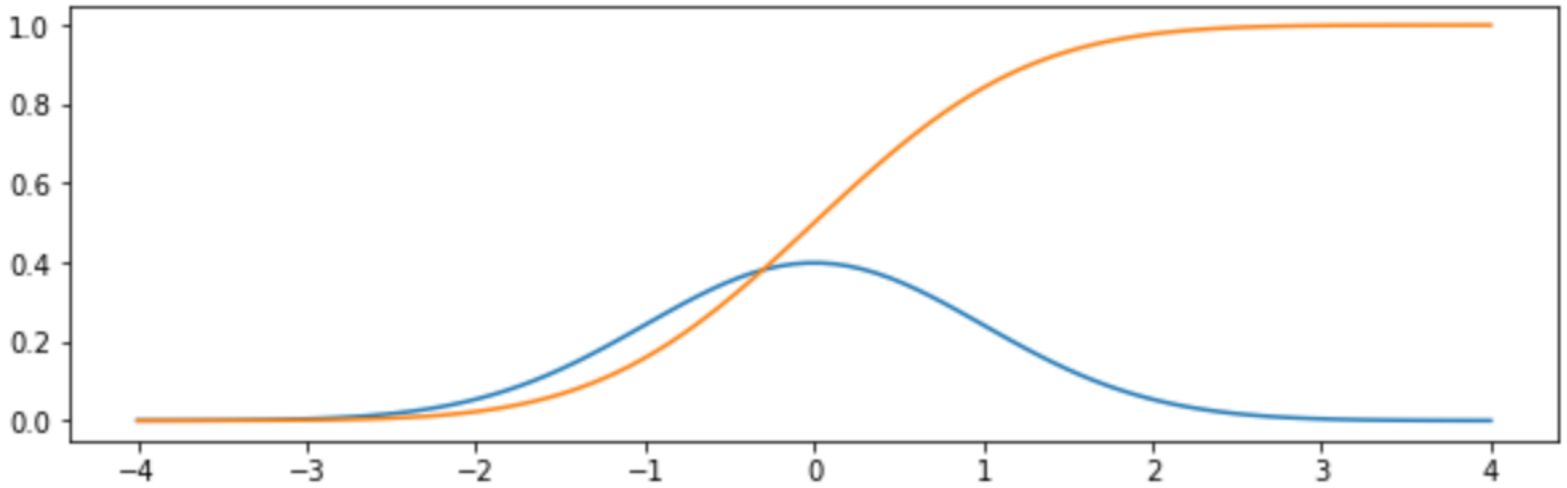
```
array([[ 21,  24,  27],
       [ 66,  78,  90],
       [111, 132, 153]])
```

Scipy: Scientific Python

```
a1 = np.array([1.87,  1.87, 1.82, 1.91, 1.90, 1.85])
scipy.stats.norm.cdf(a1)
```

```
array([0.96925809, 0.96925809, 0.9656205 , 0.97193339, 0.97128344,
       0.96784323])
```

```
x  = np.linspace(-4, 4, 100)
plt.plot(x, scipy.stats.norm.pdf(x,0,1))
plt.plot(x, scipy.stats.norm.cdf(x,0,1))
plt.show()
```



Extra: Git

(Slides from SMSC641)

What is Git

- Git is a version control system
- Developed as a repository system for both local and remote changes
- Allows teammates to work simultaneously on a project
- Tracks each commit, allowing for a detailed documentation of the project along every step
- Allows for advanced merging and branching operations

Git Basics

- Snapshots, not changes
- A picture of what all your files look like at that moment
- If a file has not changed, store a reference
- Nearly every operation is local
- Browsing the history of project
- See changes between two versions

Initialization of a git repository

```
C:\> mkdir CoolProject
C:\> cd CoolProject
C:\CoolProject > git init
Initialized empty Git repository in
C:/CoolProject/.git
C:\CoolProject > notepad README.txt
C:\CoolProject > git add .
C:\CoolProject > git commit -m 'my first commit'
[master (root-commit) 7106a52] my first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

Git Basics I

The three (or four) states of a file:

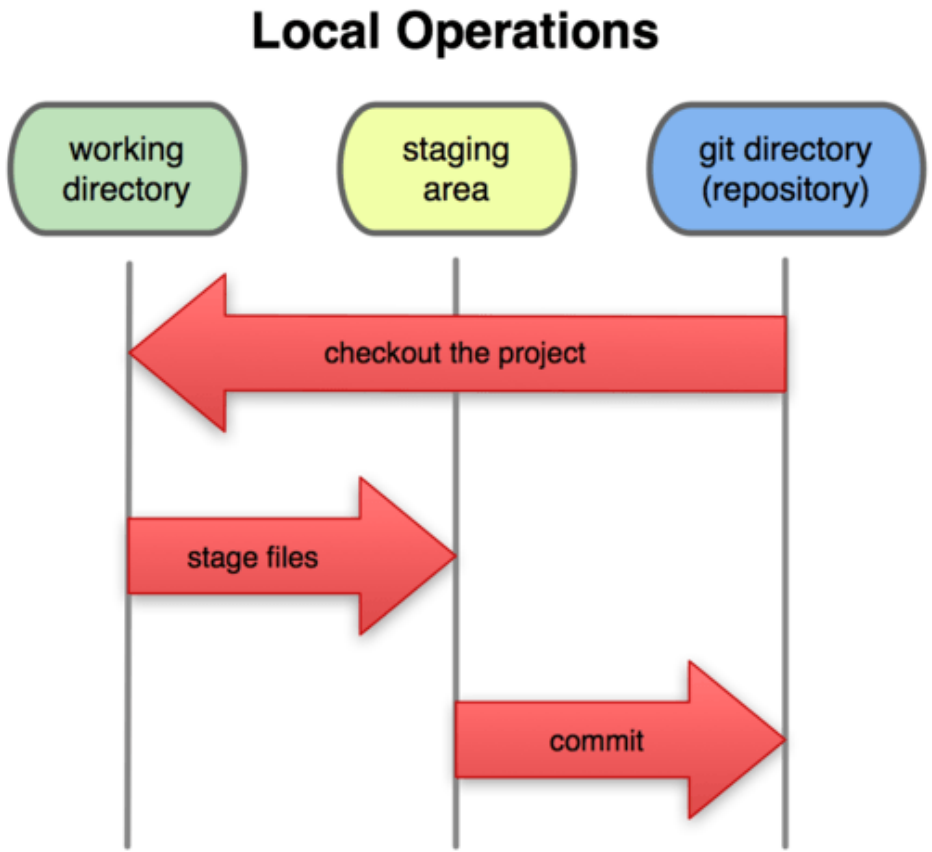
- Modified:
  - File has changed but not committed
- Staged:
  - Marked to go to next commit snapshot
- Committed:
  - Safely stored in local database
- Untracked!
  - Newly added or removed files

Git Basics II

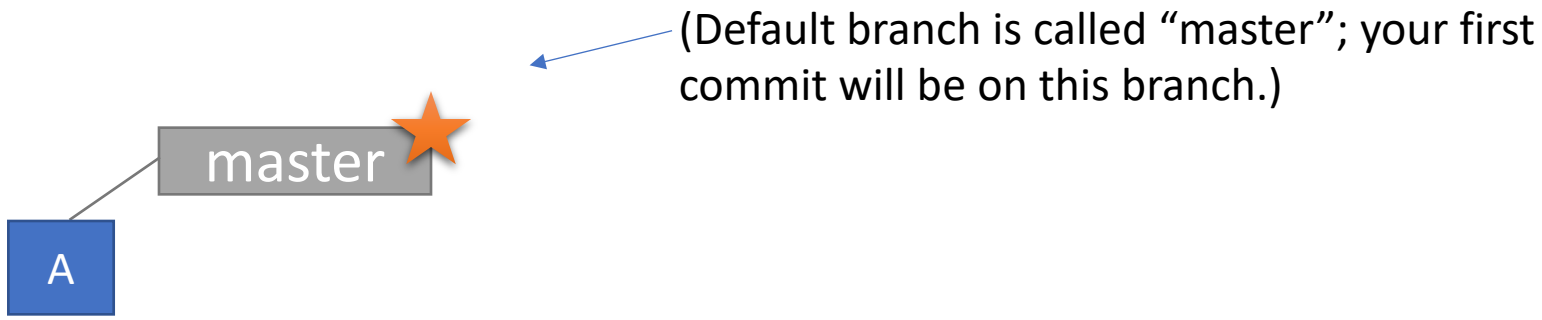
- Three main areas of a git project:
- Working directory
  - Single checkout of one version of the project.
- Staging area
  - Simple file storing information about what will go into your next commit
- Git directory
  - What is copied when cloning a repository

Git Basics III

- Three main areas of a git project:

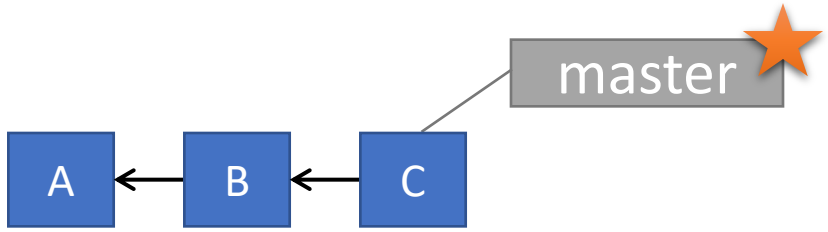


Branches Illustrated



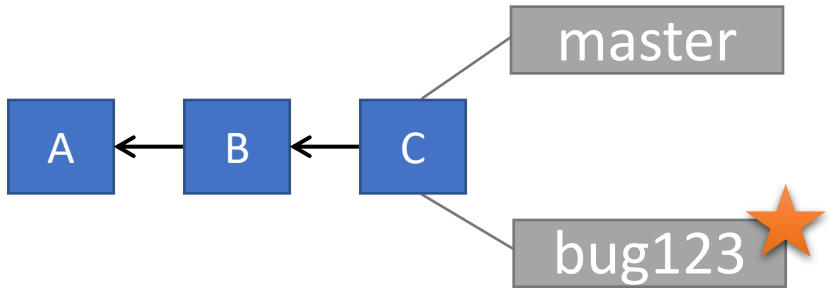

```
> git commit –m ‘my first commit’
```

Branches Illustrated



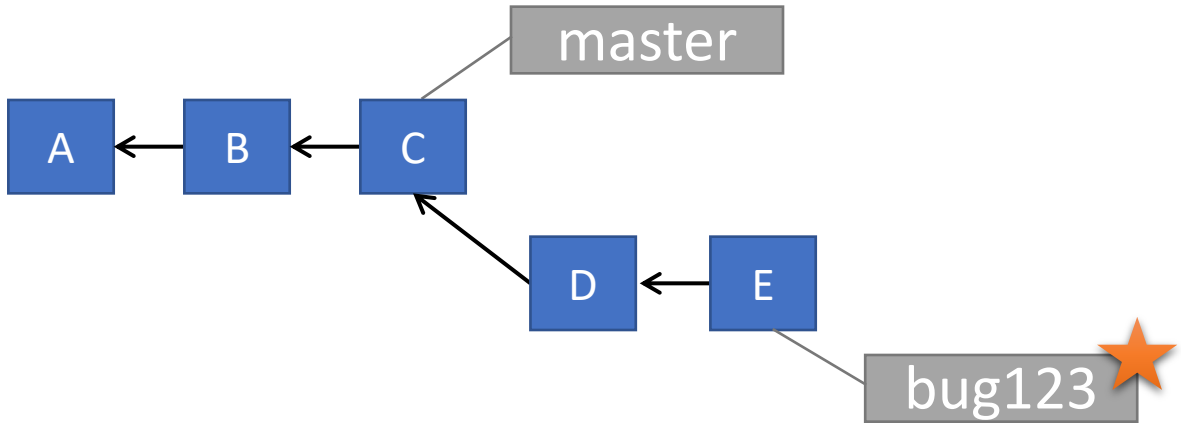

```
> git commit (x2)
```

Branches Illustrated



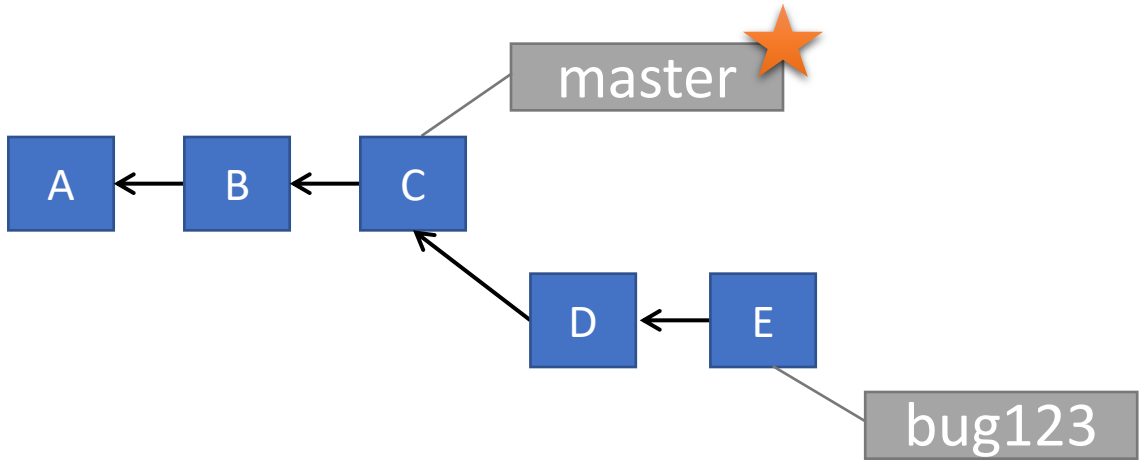

```
> git checkout -b bug123
```

Branches Illustrated



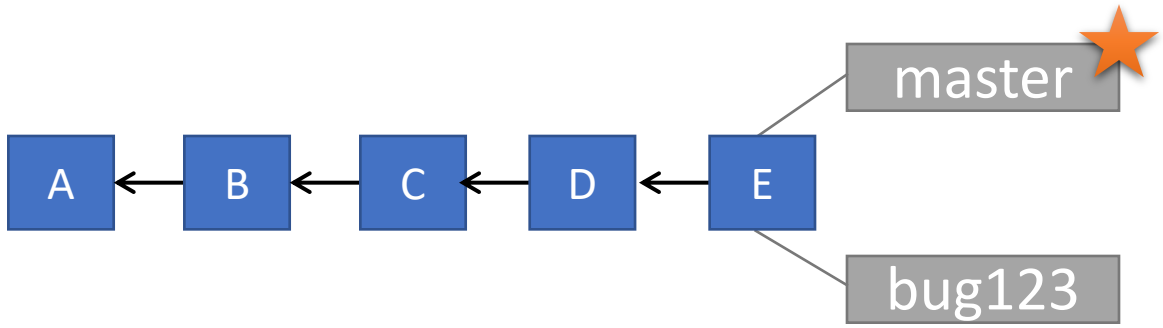

```
> git commit (x2)
```

Branches Illustrated



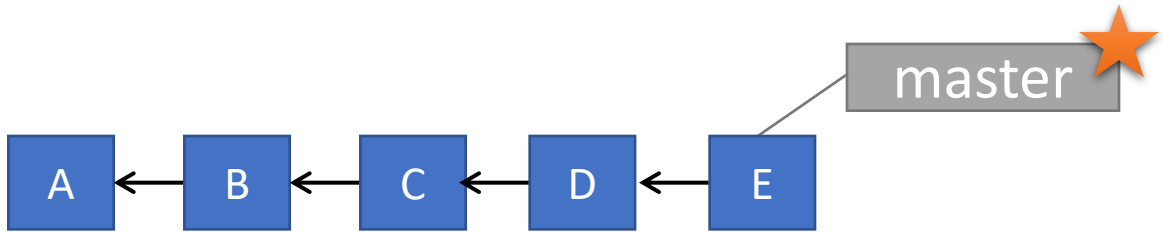

```
> git checkout master
```

Branches Illustrated



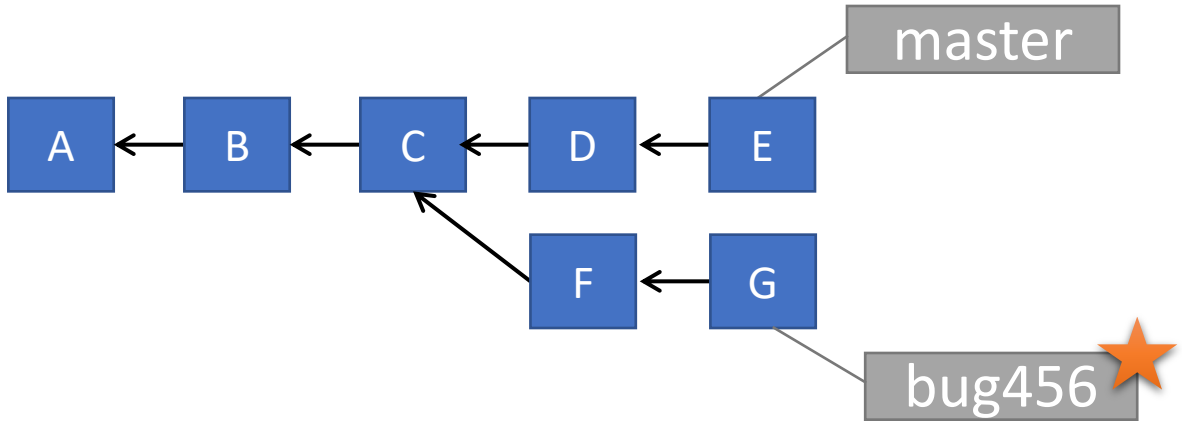

```
> git merge bug123
```

Branches Illustrated

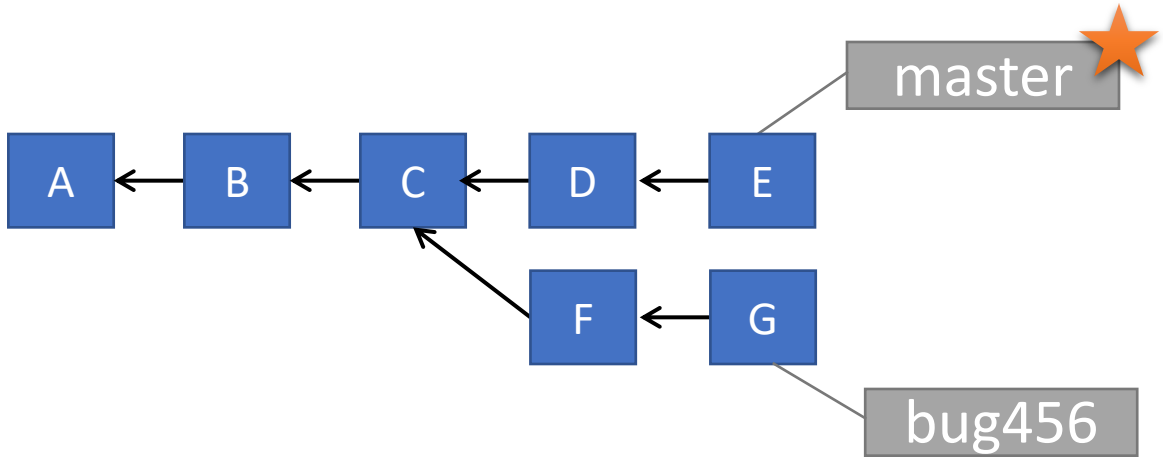



```
> git branch -d bug123
```

Branches Illustrated

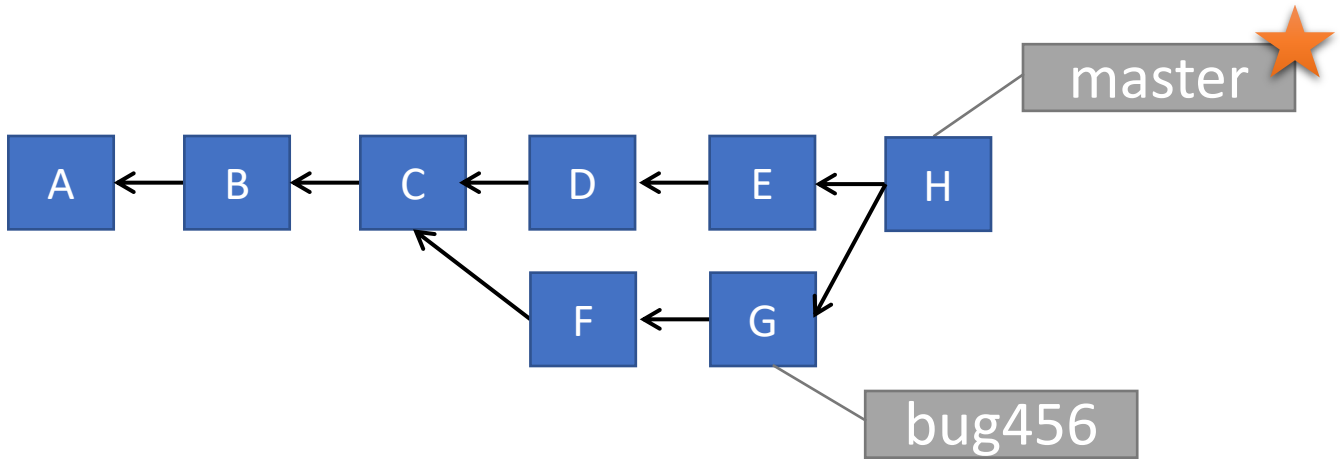


Branches Illustrated



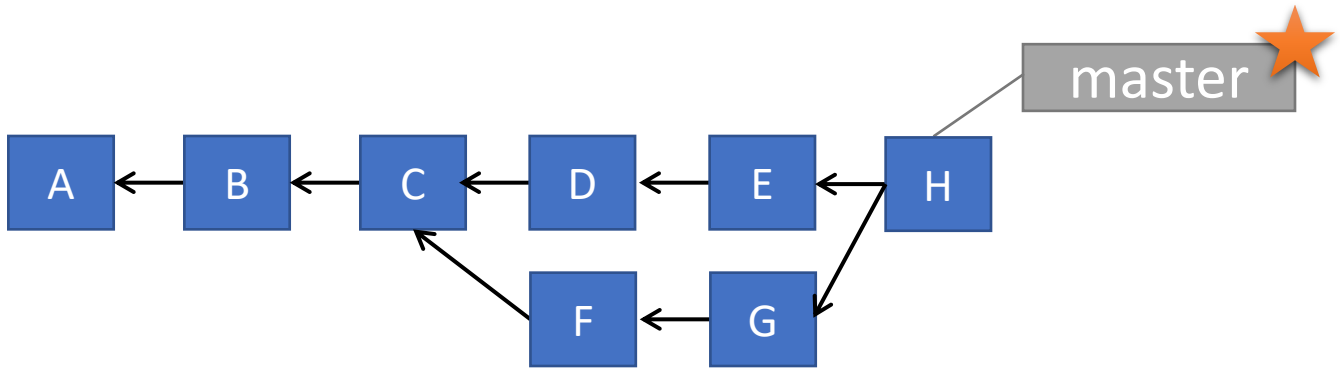

```
> git checkout master
```

Branches Illustrated



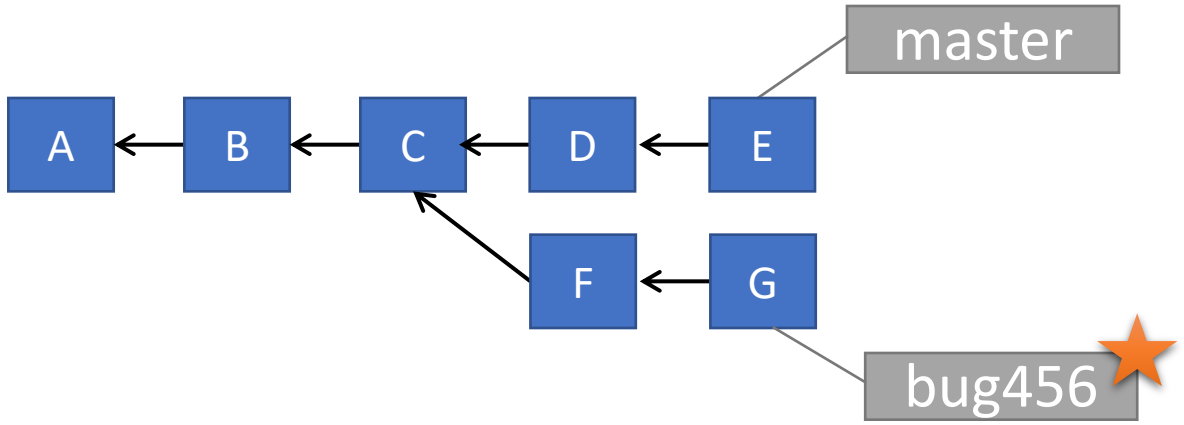

```
> git merge bug456
```

Branches Illustrated

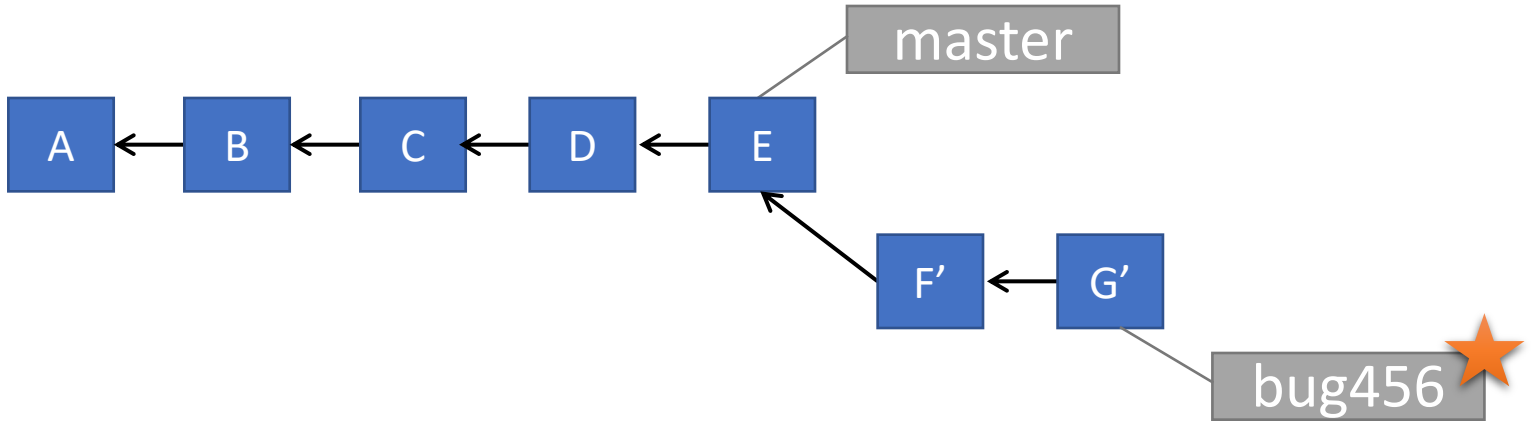



```
> git branch -d bug456
```

Branches Illustrated

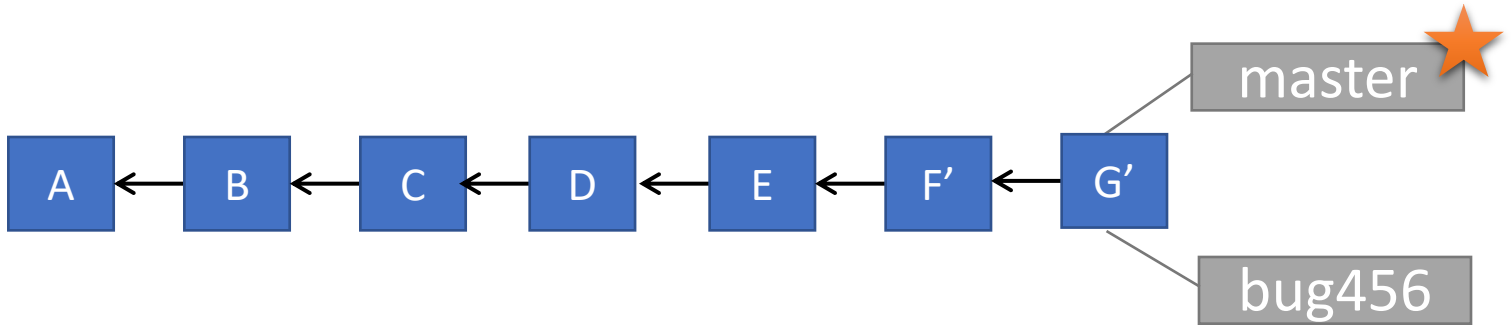


Branches Illustrated




```
> git rebase master
```

Branches Illustrated



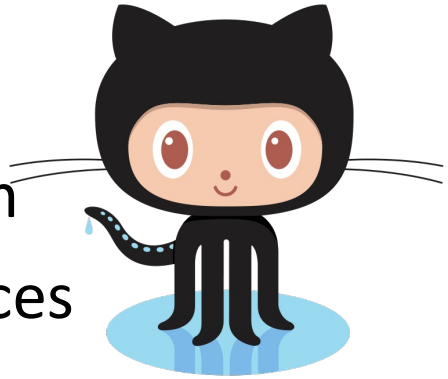

```
> git checkout master
> git merge bug456
```

When to branch?

- General rule of thumb:
- Anything in the master branch is always deployable.
- Local branching is very lightweight!
- New feature? Branch!
- Experiment that you won't ever deploy? Branch!
- Good habits:
- Name your branch something descriptive (`add-like-button`, `refactor-jobs`, `create-ai-singularity`)
- Make your commit messages descriptive, too!

So you want somebody else to host this for you …

- Git: general distributed version control system
- GitHub / BitBucket / GitLab / …: hosting services for git repositories
- In general, GitHub is the most popular:
- Lots of big projects (e.g., Python, Bootstrap, Angular, D3, node, Django, Visual Studio)
- Lots of ridiculously awesome projects (e.g., https://github.com/maxbbraun/trump2cash)
- There are reasons to use the competitors (e.g., private repositories, access control)

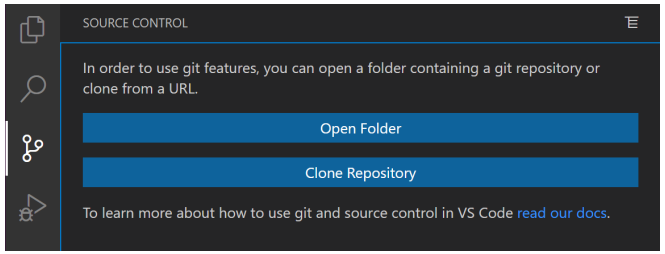


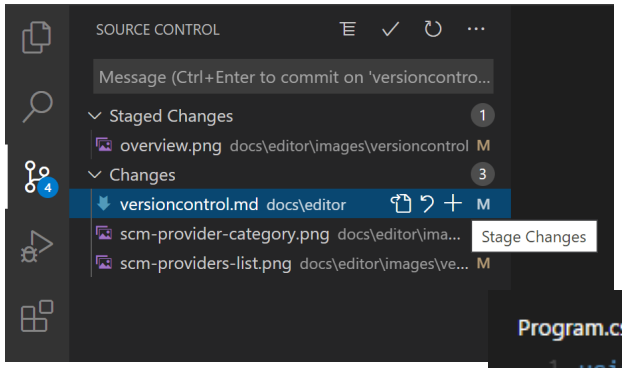
Review: How to Use

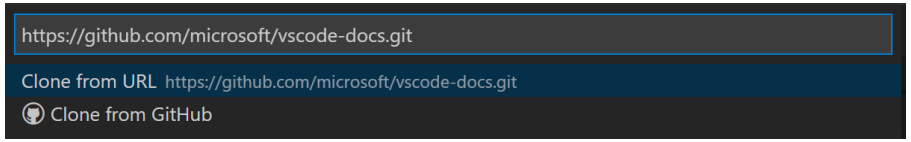
- Git commands for everyday usage are relatively simple
- git pull
  - Get the latest changes to the code
- git add .
  - Add any newly created files to the repository for tracking
- git add –u
  - Remove any deleted files from tracking and the repository
- git commit –m ‘Changes’
  - Make a version of changes you have made
- git push
  - Deploy the latest changes to the central repository
- Make a repo on GitHub and clone it to your machine:
- https://guides.github.com/activities/hello-world/

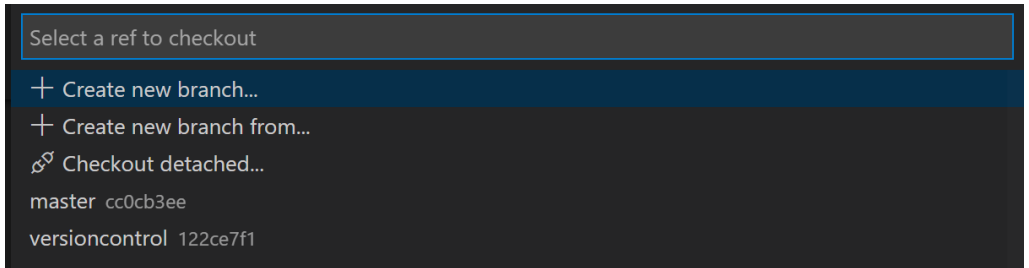
VS Code

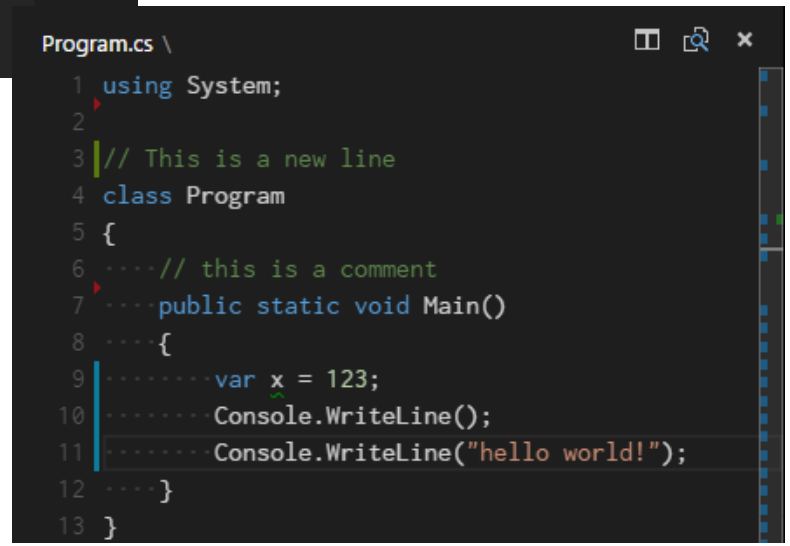
- VS Code version control and git integration
  https://code.visualstudio.com/docs/editor/versioncontrol











VS Code

- VS Code Jupyter integration
  https://code.visualstudio.com/docs/datascience/jupyter-notebooks