OOP Python Semester 1
Dr. Bianca Schoen-Phelan
02 December 2020
Tutorial Walkthrough: Git Revision and Testing Python Code using unittest

Git Revision

1. What is the most basic form I could use git?
    - Do everything locally
    - Have git installed (click)
    - Have one folder designated where you do all your work in. This can have files in it already or not, it doesn't matter. In this demo there's nothing in it yet. Open a gitbash (Windows) or the general terminal (called console on Windows) to this folder. And declare this folder as your git location.

```
$ git status
fatal: not a git repository (or any of the parent directories): .git
(base) Live Long and Prosper$ git init
Initialized empty Git repository in
/Users/bianca.schoenphelan/Documents/Tutorials/.git/
```

- Python code file v1 called `my_code.py`:

```
# Python code file to demo usage of git
# author: B. Schoen-Phelan
# date: Dec 2020

# define a class
class MyDemo:
    pass


# create an instance
demo = MyDemo()
```

- Add and commit this file to your git repository:

```
$ git add my_code.py
(base) Live Long and Prosper$ git commit -m "first version" my_code.py
[master (root-commit) 91996c1] first version
 Committer: Bianca SchoenPhelan <bianca.schoenphelan@soc-mbp13-bsp.lan>
 1 file changed, 11 insertions(+)
 create mode 100644 my_code.py
(base) Live Long and Prosper$ git status
On branch master
```

```
nothing to commit, working tree clean
```

- For working on this file we will create a new branch:

```
(base) Live Long and Prosper$ git checkout -b feature_1
Switched to a new branch 'feature_1'
(base) Live Long and Prosper$ git branch
* feature_1
  master
```

- We now work with this file and make a change in the branch feature_1. We can check if there was a change with git status, and we can also check what the actual difference is between different branches, too. If we are happy with the change we can add and commit the change. We can also check the progress in this branch with git log --oneline.

```
class MyDemo:
    def __init__(self):
        print("Hello Demo!")


# create an instance
demo = MyDemo()
```

```
(base) Live Long and Prosper$ git status
On branch feature_1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   my_code.py

no changes added to commit (use "git add" and/or "git commit -a")
(base) Live Long and Prosper$ git diff master my_code.py
diff --git a/my_code.py b/my_code.py
index 1242883..2b437c9 100644
--- a/my_code.py
+++ b/my_code.py
@@ -4,7 +4,8 @@

 # define a class
 class MyDemo:
-    pass
+    def __init__(self):
+        print("Hello Demo!")


 # create an instance
```

```
(base) Live Long and Prosper$ git add my_code.py
(base) Live Long and Prosper$ git commit -m "my first feature" my_code.py
[feature_1 8831ef4] my first feature
 Committer: Bianca SchoenPhelan <bianca.schoenphelan@soc-mbp13-bsp.lan>
 1 file changed, 2 insertions(+), 1 deletion(-)
(base) Live Long and Prosper$ git log --oneline
8831ef4 (HEAD -> feature_1) my first feature
91996c1 (master) first version
```

- Let's make another change and inspect the different versions, ultimately, add and commit it:

```python
# define a class
class MyDemo:
    def __init__(self):
        print("Hello Demo!")

    def power(self, base, exponent):
        return base ** exponent


# create an instance
demo = MyDemo()
print(demo.power(6, 2))
print(demo.power(2, 6))
```

```
(base) Live Long and Prosper$ git status
On branch feature_1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   my_code.py

no changes added to commit (use "git add" and/or "git commit -a")
(base) Live Long and Prosper$ git diff master my_code.py
diff --git a/my_code.py b/my_code.py
index 1242883..2dd3963 100644
--- a/my_code.py
+++ b/my_code.py
@@ -4,8 +4,14 @@

 # define a class
```

```
 class MyDemo:
-    pass
+    def __init__(self):
+        print("Hello Demo!")
+
+    def power(self, base, exponent):
+        return base ** exponent


 # create an instance
 demo = MyDemo()
+print(demo.power(6, 2))
+print(demo.power(2, 6))
(base) Live Long and Prosper$ git diff 91996c1:my_code.py my_code.py
diff --git a/my_code.py b/my_code.py
index 1242883..2dd3963 100644
--- a/my_code.py
+++ b/my_code.py
@@ -4,8 +4,14 @@

 # define a class
 class MyDemo:
-    pass
+    def __init__(self):
+        print("Hello Demo!")
+
+    def power(self, base, exponent):
+        return base ** exponent


 # create an instance
 demo = MyDemo()
+print(demo.power(6, 2))
+print(demo.power(2, 6))
(base) Live Long and Prosper$ git diff 8831ef4:my_code.py my_code.py
diff --git a/my_code.py b/my_code.py
index 2b437c9..2dd3963 100644
--- a/my_code.py
+++ b/my_code.py
@@ -7,6 +7,11 @@ class MyDemo:
     def __init__(self):
         print("Hello Demo!")

+    def power(self, base, exponent):
+        return base ** exponent
+

 # create an instance
 demo = MyDemo()
+print(demo.power(6, 2))
+print(demo.power(2, 6))
```

```
(base) Live Long and Prosper$ git status
On branch feature_1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   my_code.py

no changes added to commit (use "git add" and/or "git commit -a")
(base) Live Long and Prosper$ git add my_code.py
(base) Live Long and Prosper$ git commit -m "my second feature: power function"
my_code.py
[feature_1 699f38d] my second feature: power function
 Committer: Bianca SchoenPhelan <bianca.schoenphelan@soc-mbp13-bsp.lan>
 1 file changed, 5 insertions(+)
(base) Live Long and Prosper$ git log --oneline
699f38d (HEAD -> feature_1) my second feature: power function
8831ef4 my first feature
91996c1 (master) first version
```

- What if I've changed my mind? I don't want the power anymore?
  You have two choices here, a hard revert that will get rid of all traces of it, or a soft revert which will preserve it locally. Let's do a hard delete and go back to how things were at the first feature:

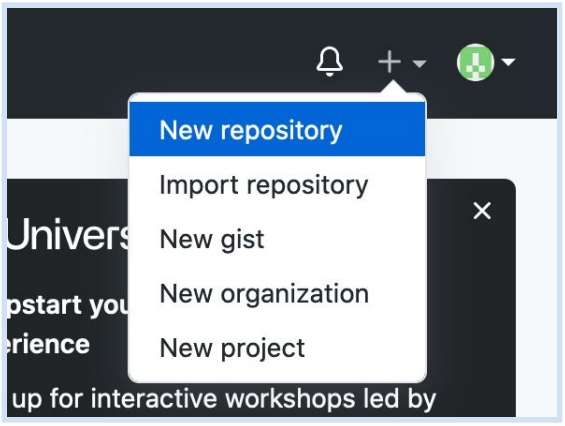
```
(base) Live Long and Prosper$ git reset --hard 8831ef4
HEAD is now at 8831ef4 my first feature
```

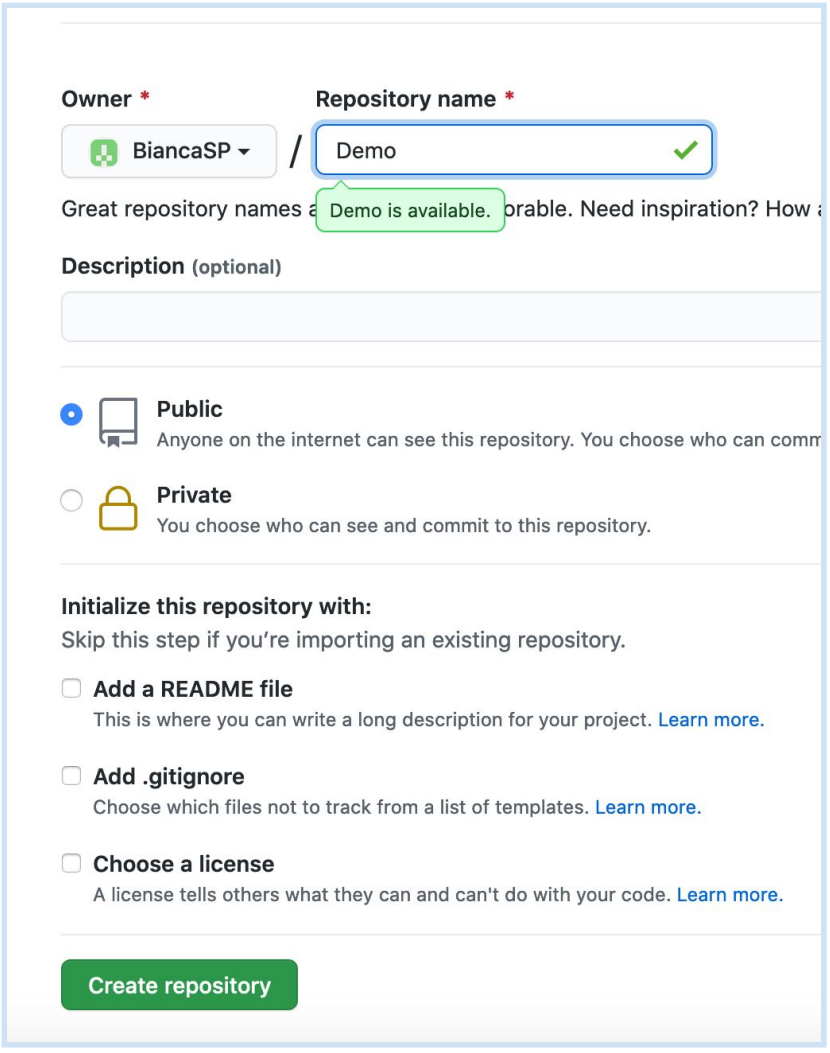
- I'm confident that this version is the one that should go live into production. Merge my branches. We'll be back on master:

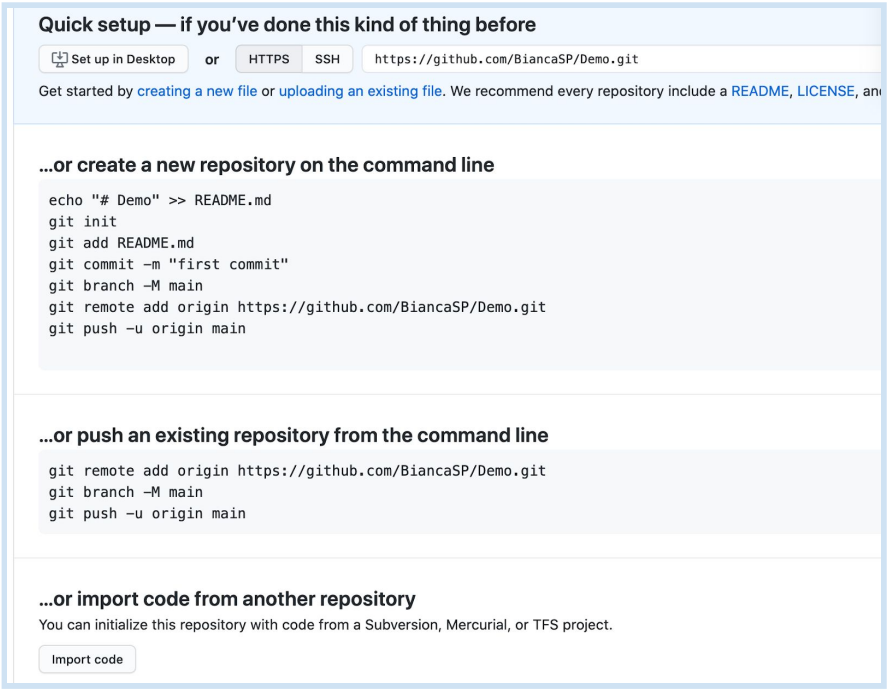
```
(base) Live Long and Prosper$ git checkout master
Switched to branch 'master'
(base) Live Long and Prosper$ git branch
  feature_1
* master
(base) Live Long and Prosper$ git merge feature_1
Updating 91996c1..8831ef4
Fast-forward
 my_code.py | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
(base) Live Long and Prosper$ git branch
  feature_1
* master
(base) Live Long and Prosper$ git branch -d feature_1
Deleted branch feature_1 (was 8831ef4).
(base) Live Long and Prosper$ git branch
* master
```

```
(base) Live Long and Prosper$ git log --oneline
8831ef4 (HEAD -> master) my first feature
91996c1 first version
```

2. How do I connect to an online repository?
    - Why should I do it?
        - As a backup to files locally
        - To share with your team
        - To use as your portfolio for applications
    - How do I do it?
        - Set up an account on github or bitbucket
        - Create a repository there online
        - Follow the instructions to link your existing repository with the online one


New repository
Import repository
New gist
New organization
New project
Univers
pstart you
rience
up for interactive workshops led by

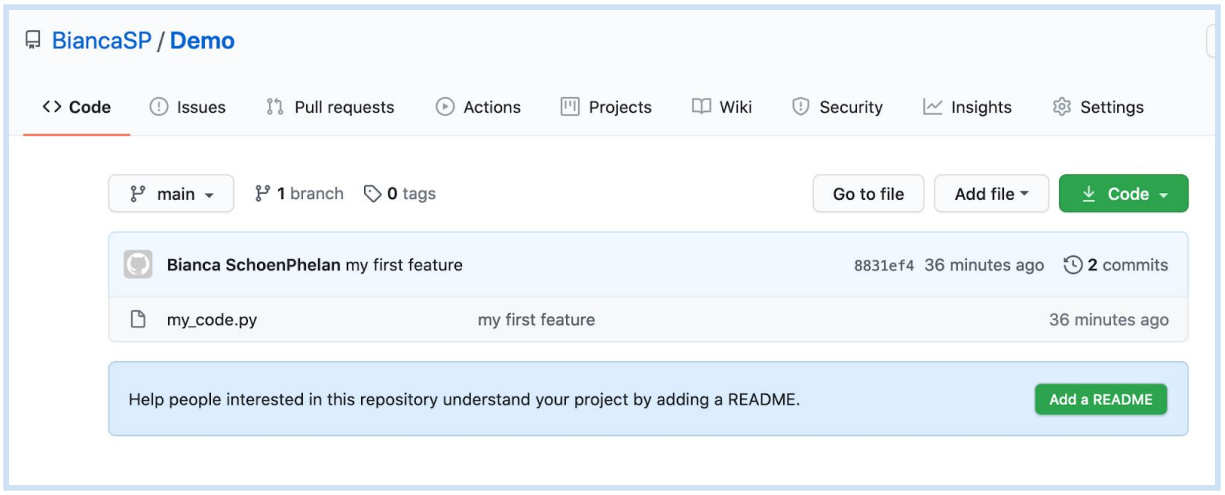

Owner *
Repository name *
BiancaSP
/
Demo
Demo is available.
Great repository names a
orable. Need inspiration? How
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can comm
Private
You choose who can see and commit to this repository.
Initialize this repository with:
Skip this step if you're importing an existing repository.
Add a README file
This is where you can write a long description for your project. Learn more.
Add .gitignore
Choose which files not to track from a list of templates. Learn more.
Choose a license
A license tells others what they can and can't do with your code. Learn more.
Create repository

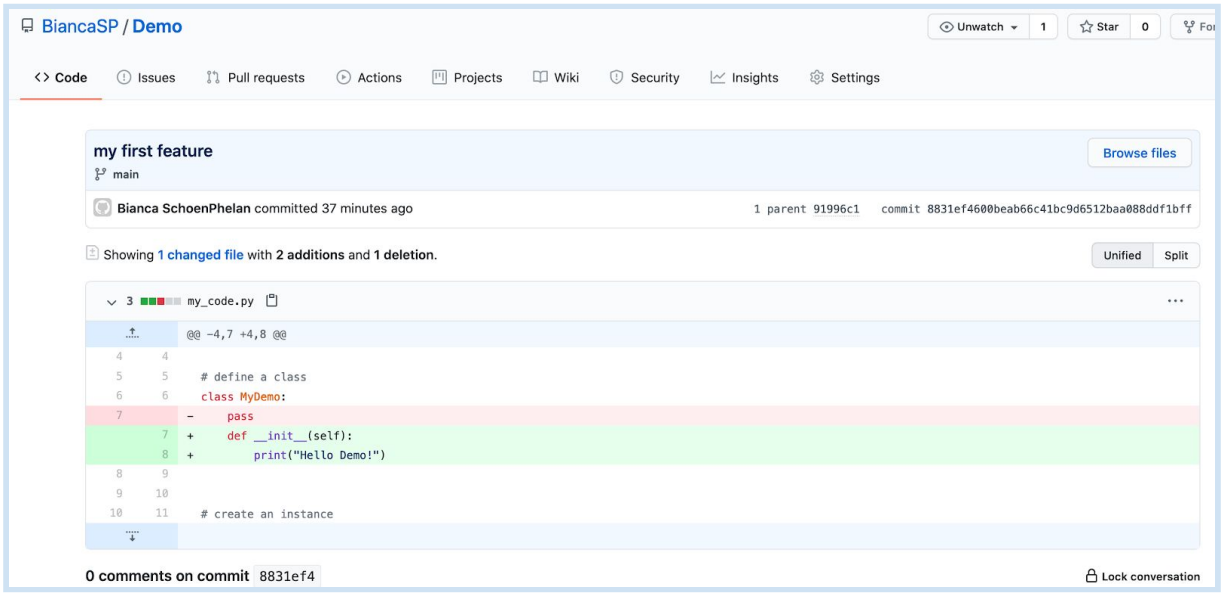


- Let's do option 2 as we have an existing repository

```
(base) Live Long and Prosper$ git remote add origin
https://github.com/BiancaSP/Demo.git
(base) Live Long and Prosper$ git branch -M main
(base) Live Long and Prosper$ git push -u origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 588 bytes | 588.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/BiancaSP/Demo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
(base) Live Long and Prosper$ git remote -v
origin https://github.com/BiancaSP/Demo.git (fetch)
origin https://github.com/BiancaSP/Demo.git (push)
```

- Let's have a look at what is online: It shows the same hex code for commits and clicking on the name of a commit lets you see further info:





- Changes can be made online too. We can fetch changes.

```
(base) Live Long and Prosper$ git remote -v
origin https://github.com/BiancaSP/Demo.git (fetch)
origin https://github.com/BiancaSP/Demo.git (push)
(base) Live Long and Prosper$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/BiancaSP/Demo
   8831ef4..8fbed3e  main       -> origin/main
(base) Live Long and Prosper$ git merge
Updating 8831ef4..8fbed3e
Fast-forward
```

```
my_code.py | 1 +
1 file changed, 1 insertion(+)
```

We don't have to mention anything after fetch or merge because we only have one online location. It's clear what we are taking changes from.

We can now use git push after a local git add and git commit in order to have our online gitHub repository reflect the state of our local repository. I would recommend to do that at the end of a working day.

Testing Python Code with unittest

Now let's do some testing, expand our Python file (remember to git add and commit all our changes - **not shown here**)

```
class MyDemo:
    def __init__(self, a_list):
        # print("Hello Demo!")
        # print("+++++ I made this change online!!!
+++++")
        self.my_list = a_list

    def mean(self):
        return sum(self.my_list) / len(self.my_list)

    def median(self):
        if len(self.my_list) % 2:
            return int(len(self.my_list) / 2)
        else:
            idx = int(len(self.my_list) / 2)
            return (self.my_list[idx] +
self.my_list[idx-1]) / 2


# create an instance
# demo = MyDemo([1, 2, 2, 3, 3, 4, 4])
# # print(demo.mean())
```

```
# print(demo.median())
```

And let's write a test class for it! Create a file `test_my_stuff.py`

```
import unittest
from my_code import MyDemo

class MyFirstTest(unittest.TestCase):
  def setUp(self):
     self.stats = MyDemo([1, 2, 2, 3, 3, 4])

  def test_mean(self):
     self.assertEqual(self.stats.mean(), 2.5)

  def test_median(self):
     self.assertEqual(self.stats.median(), 2.5)
     self.stats.my_list.append(4)
     self.assertEqual(self.stats.median(), 3)


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

These will of course all pass. Change it up, create more test cases. Make them fail. Look for other assertions to use! Put it some prints to see when what is called (for example setUp gets called before every test_*)