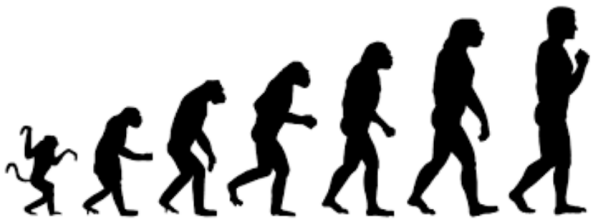
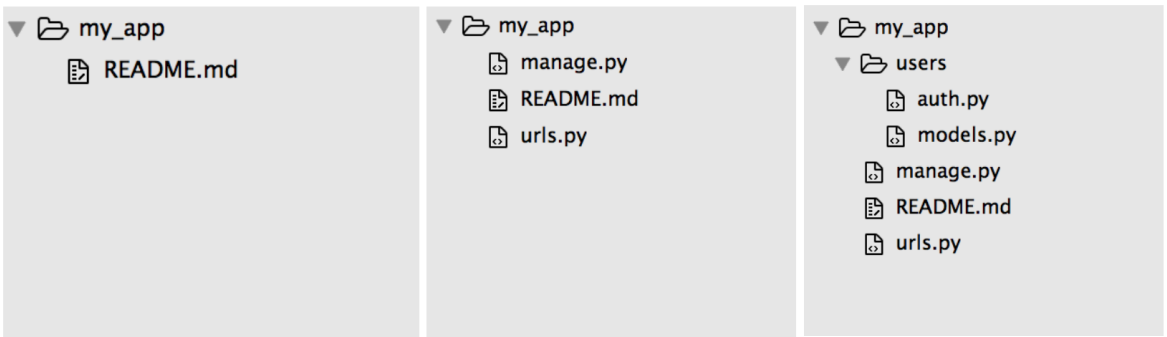
# Software Development Class - Session 2

## Version Control using Git

### The case for version control




- Variety is in nature of things.
- Be it a story, a drawing, an aeroplane or a piece of software, there would be multiple versions of the same.
- All creative processes involve iterating and switching between various versions
- ... and in case of software design it involves high collaboration too
- Softwares and the underlying code goes through many versions before the finality, if ever there is
- Tools are required to easily switch between versions and gather different pieces of code to make one software

### A version control system enables

- storing multiple versions of a code project
- switching between multiple versions

- viewing differences between multiple versions
- combining pieces from individual team members

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- https://git-scm.com/

How does `git` work?
- In any folder where our project lies, we ask git to track changes
- After every few set of changes, we ask git to store these changes for us, we give it a message to remember what changes we made.
- `git` then stores these changes, gives a unique ID to refer to this set of changes latter.
- If we would like to go to a previous version, we ask `git` to take us to that version. It then updates our working files to that version state.

**Git Basics**

Setup

```
git --version
git config --global user.name "Sravan"
git config --global user.email "sravfeyn@gmail.com"
git config --global core.editor vim
git config --list
```

```
git init  # makes a folder into a repostory
git status  # can use anytime, shows what's going on with the repository
git add <path to files>  # adds to staging area
git commit -m "Implemented a user login page"
git status
```

**Brancing**

```
git branch
git branch login_bug
git checkout login_bug
```

**Git for collaboration**

A git server
- Hosts git repositories
- Allows multiple users to push/pull changes to the repository
- Access permission for users
- Additionally, servers like github.com (bitbucket.org, gitlab) provide better UI for code reviews, pull requests etc

```
git remote add origin git@github.com:sravfeyn/ps2.git
git push -u origin master
git pull origin friends_feature
```

**git terminology**

- Repository: The folder containing current working version and all other versions of a code base.
- Commit: The incremental set of changes that we ask `git` to store
- Staging area
- Branch
- Remote

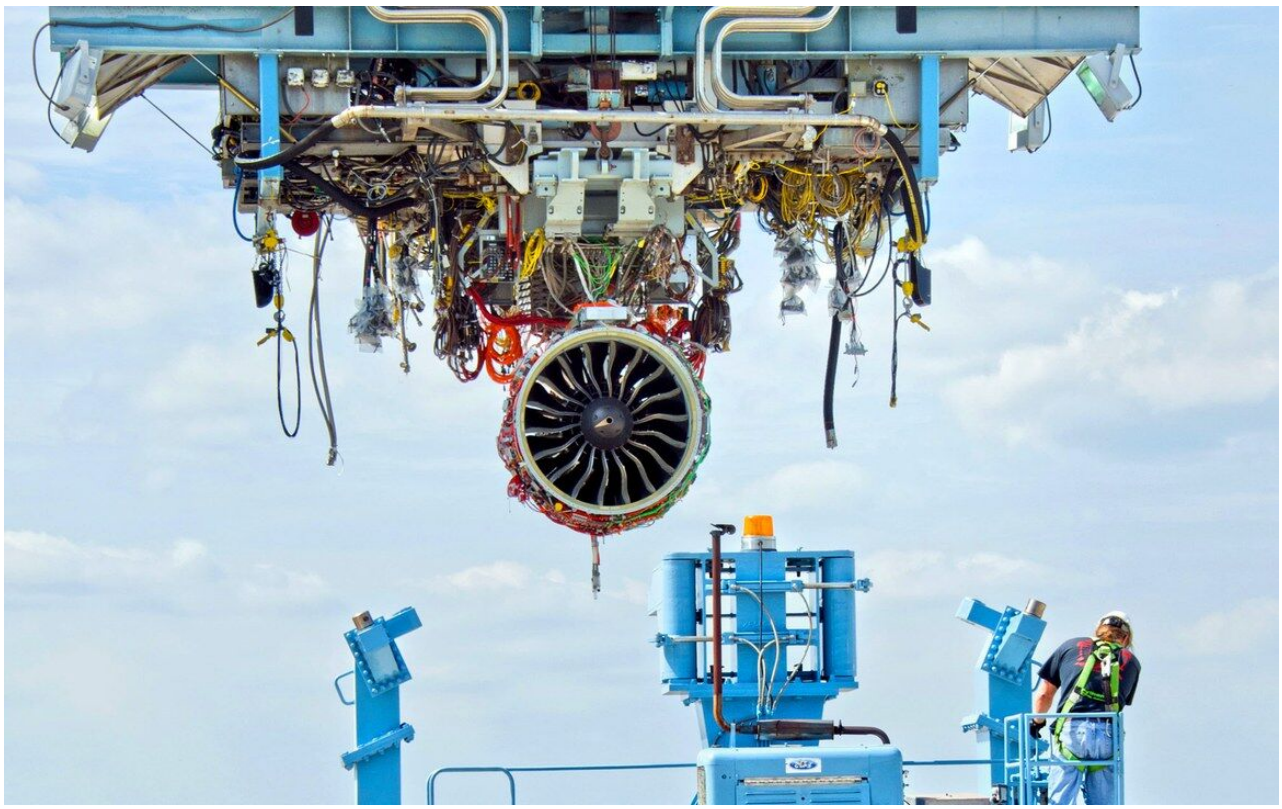
**Git reference and tutorials**

- https://git-scm.com/doc
- https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup
- https://help.github.com/articles/set-up-git/
- https://help.github.com/articles/adding-an-existing-project-to-github-using-the-command-line/
- https://try.github.io/

## Testing

### The case for testing

A facility of General Electric for jet engine testing.



- Make sure what we make works
- ... works in almost all scenarios

- Helps identify components that are buggy
- Trust a piece of code without reviewing it a lot

**Testing types**

- User Interface testing
- Functional testing
- Code testing
  - Manual testing
  - Unit tests
  - Integration tests
  - Test driven development

**Writing tests**

A very basic way of writing

- Write a test in a file called test.py
- Run `python test.py`

```
def test_max_function():
    actual = max_element([1, 4, 5, 3, 2])
    expected = 5
    if actual == expected:
        pass
    else:
        raise Exception("Incorrect answer"
    # Repeat above for each test case

test_max_function()
```

A little better way

```
def test_max_function():
    actual = max_element([1, 4, 5, 3, 2])
    expected = 5
    assert actual == expected
    # Repeat for each test case

test_max_function()
```

```
def test_user_age():
    user = create_user(dob=datetime.date(1990, 7, 7)  # expensive opera
tion
```

```
    actual_age = get_user_age(user)
    assert actual_age == 27
    # Repeat for various date-0f-births

def test_max_function():
    actual = max_element([1, 4, 5, 3, 2])
    expected = 5
    assert actual == expected
    # Repeat for each test case

test_max_function()
test_user_age()
```

Case for a testing library

- Running multiple tests with one command
- Pre-setup before tests can be run, deleting data after,
- Debugging tests

**Python unittest library**

https://docs.python.org/3/library/unittest.html

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
```

```
20      unittest.main()
```

More readings
http://pythontesting.net/framework/unittest/unittest-introduction/

## Hangman assignment walkthrough

Using git and testing

## Django Introduction

- How does a website or a mobile app work?
- What all happens when you load google.com on your browser?

The usual components
- An HTTP server
- **URL rules** written in a server specific language
- Server programs that respond to URLs
- Looking up or storing data in **database** using SQL scripts.
- Sending **HTML** back

Django takes care of all these things in a neat way by letting us write all these things in Python
- URL rules are declared in a python program
- SQL queries are done using Django's ORM model in Python without writing SQL
- Lets us use wide variety of template frameworks to easily write HTML
- Implements an MVC pattern
- Provides common utilities to work with like HTML Forms, authentication, admin site etc

https://docs.djangoproject.com/en/2.0/intro/overview/