

INFORME PARCIAL 2

Estudiante	Código
Dylan Torres	12103021

URL REPOSITORIO CON CODIGO FUENTE PYTEST Y ARCHIVO README

<https://github.com/dylan9538/ParcialDos>

PASOS PREVIOS PARA DESARROLLAR EL PARCIAL

Iniciamos la maquina con el usuario root: Previamente nos aseguramos de tener los puertos necesarios, como otras configuraciones anteriores que puedan ser de utilidad. IPTABLES o VISUDO.

SEGUIMOS CON LOS SIGUIENTES PASOS

Clono el repositorio que necesito

En este repositorio añadiremos los archivos que se manejen.

```
mkdir repositories
cd repositories
git clone https://github.com/dylan9538/ParcialDos.git
cd ParcialDos
```

```
git config remote.origin.url
"https://token@github.com/dylan9538/ParcialDos.git"
```

En el campo token añadido el token generado en github.

Creo un directorio y el ambiente

```
cd ~/
$ mkdir ambientes
```

```
$ cd ambientes
$ virtualenv ambientePar2
```

Lo activo:

```
cd ~/ambiente
. ambientePar2/bin/activate
```

Instalo El Flask en el ambiente

```
Pip install Flask
```

PUNTO QUE SE CORRE CON PYTEST DESDE LA CONSOLA DE COMANDOS

Creo un directorio para el ejercicio dentro del repositorio clonado

```
$ cd ~/
$ mkdir -p ejercicios/ejercicioPy
$ cd ejercicios/ejercicioPy
```

Creo el archivo comandos_para_prueba.py que contiene el siguiente código

```
from subprocess import Popen, PIPE

def get_all_files():
    grep_process = Popen(["ls"], stdout=PIPE, stderr=PIPE)
    file_list = Popen(["awk","-F","/","'{print $1}'"],
        stdin=grep_process.stdout,
        stdout=PIPE,stderr=PIPE).communicate()[0].split('\n')
    return filter(None,file_list)

def add_file(filename,content):
    add_process = open(filename+'.txt','a')
    add_process.write(content+'\n')
    add_process.close()
    return "Se creo el archivo" , 201

def remove_file(filename):
    vip = ["ambientes","repositories","comandos","comandosTest"]
    if filename in vip:
        return True
    else:
        remove_process = Popen(["rm","-r",filename], stdout=PIPE,
            stderr=PIPE)
        remove_process.wait()
        return False if filename in get_all_files() else True
```

Estos métodos contienen código para dar todos los archivos, agregar un archivo y eliminar un archivo. Todos desarrollados en el parcial 1.

Ubicado en el repositorio dado en la siguiente URL:

```
https://github.com/dylan9538/parcialUno
```

Creo el archivo test_comandos.py que contiene el siguiente código

```
from comandos_para_prueba import *

def test_add():

    fileNameUno = "fileAddTest"
    fileNameDos = "fileAddTestDos"
    contenido1 = "soy el archivo de add test"
    contenido2 = "soy el archivo de add test 2"

    archivosIniciales = get_all_files()
    numFilesUno = len(archivosIniciales)

    add_file(fileNameUno,contenido1)
    add_file(fileNameDos,contenido2)
    archivosNuevos = get_all_files()
    numFilesDos = len(archivosNuevos)

    numEsperado = numFilesUno + 2

    assert numEsperado == numFilesDos,"El numero de archivos actual
    deberia ser igual al numFilesUno mas dos"

def test_delete():
    fileNameUno = "fileAddTest.txt"
    fileNameDos = "fileAddTestDos.txt"

    archivosActuales = get_all_files()
    numFilesUno = len(archivosActuales)

    remove_file(fileNameUno)
    remove_file(fileNameDos)
    archivosNuevos = get_all_files()
    numFilesDos = len(archivosNuevos)

    numEsperado = numFilesUno - 2

    assert numEsperado == numFilesDos,"El numero de archivos deberia
    ser igual a numFilesUno menos dos"

def test_getAll():

    primeraCantidad = len(get_all_files());

    add_file("fileOne","soy el archivo one");
    add_file("fileTwo","soy el archivo dos");
    add_file("fileThree","soy el archivo tres");
    segundaCantidad = len(get_all_files());

    remove_file("fileOne.txt");
    remove_file("fileTwo.txt");
    remove_file("fileThree.txt");
    tercerCantidad = len (get_all_files());

    assert primeraCantidad == tercerCantidad and primeraCantidad + 3 ==
    segundaCantidad
```

Para el test_add se agregan dos archivos nuevos y luego comparo la cantidad de archivos luego de agregarlos con la cantidad esperada.

Para el test_delete se borran los dos archivos agregados anteriormente y se compara la cantidad esperada con la cantidad luego de haberlos eliminado.

Para el test_getAll se añaden archivos se pide la cantidad que quedan, luego se borran y se pide la cantidad, luego se compara las cantidades para verificar que está trayendo la cantidad correcta.

Creamos un archivo llamado uri.py que contenga el siguiente código

```
from flask import Flask, abort, request
import json

from files_commands import get_all_files, add_file, remove_file

app = Flask(__name__)

api_url = '/recently_created'

@app.route('/files',methods=['POST'])
def create_file():
    content = request.get_json(silent=True)
    filename = content['filename']
    content = content['content']
    add_file(filename,content)
    return "Se creo",201

@app.route('/files',methods=['GET'])
def read_file():
    list = {}
    list["files"] = get_all_files()
    return json.dumps(list), 200

@app.route('/files',methods=['PUT'])
def update_file():
    return "not found", 404

@app.route('/files',methods=['DELETE'])
def delete_file():
    error = False
    for username in get_all_files():
        if not remove_file(filename):
            error = True

    if error:
        return 'some files were not deleted', 400
    else:
        return 'all files were deleted', 200

if __name__ == "__main__":
    app.run(host='0.0.0.0',port=8088,debug='True')
```

Para la ejecución de las pruebas unitarias es necesario el framework de pytest. Entonces se procede a ejecutar los siguientes comandos:

```
pip install -U pytest
```

Luego se procede a ejecutar el comando para verificar la instalación

```
pytest --version
```

Para la ejecución de las pruebas ejecutamos el siguiente comando, sobre el archivo test_comandos.py

```
pytest -q test_comandos.py
```

Prueba de los test (passed)

Esta prueba fue hecha con la consola y aprueba que los comandos y los test quedaron bien hechos.

```
(ambientePar2) [root@localhost ejercicioPy]# pytest -q test_comandos.py
...
3 passed in 0.07 seconds
(ambientePar2) [root@localhost ejercicioPy]#
```

AHORA PRUEBA CON JENKINS DE LOS SERVICIOS

Para esta etapa se hace uso del testproject al cual se le hizo fork en github. Se copian y pegan los archivos con el código fuente en dicho repositorio para probarlo con jenkins.

URL: <https://github.com/dylan9538/testproject>

Modificamos el código de test_comandos.py por el siguiente código

```
from comandos_para_prueba import *

@pytest.fixture
def client(request):
    client = usermgmt.app.test_client()
    return client

def test_add():

    fileNameUno = "fileAddTest"
    fileNameDos = "fileAddTestDos"
```

```

contenido1 = "soy el archivo de add test"
contenido2 = "soy el archivo de add test 2"

jsonUno={ 'filename' : filenameUno,
          'content': contenido1}
jsonDos={ 'filename' : filenameDos,
          'content': contenido2}

archivosIniciales = client.get('/files',follow_redirects=True)
numFilesUno = len(archivosIniciales)

client.post('/files', data = json.dumps(jsonUno), content_type =
'application/json')
client.post('/files', data = json.dumps(jsonDos), content_type =
'application/json')
archivosNuevos = client.get('/files',follow_redirects=True)
numFilesDos = len(archivosNuevos)

numEsperado = numFilesUno + 2

assert numEsperado == numFilesDos,"El numero de archivos actual
deberia ser igual al numFilesUno mas dos"

def test_delete():
    fileNameUno = "fileAddTest.txt"
    fileNameDos = "fileAddTestDos.txt"

    archivosActuales = client.get('/files',follow_redirects=True)
    numFilesUno = len(archivosActuales)

    remove_file(fileNameUno)
    remove_file(fileNameDos)
    archivosNuevos = client.get('/files',follow_redirects=True)
    numFilesDos = len(archivosNuevos)

    numEsperado = numFilesUno - 2

    assert numEsperado == numFilesDos,"El numero de archivos deberia
ser igual a numFilesUno menos dos"

def test_getAll():

    jsonU={ 'filename' : 'fileOne',
            'content': 'soy el archivo one'}

    jsonA={ 'filename' : 'fileTwo',
            'content': 'soy el archivo dos'}

    jsonB={ 'filename' : 'fileThree',
            'content': 'soy el archivo tres'}

    primeraCantidad = len(client.get('/files',follow_redirects=True));

    client.post('/files', data = json.dumps(jsonU), content_type =
'application/json')

```

```

    client.post('/files', data = json.dumps(jsonA), content_type =
'application/json')
    client.post('/files', data = json.dumps(jsonB), content_type =
'application/json')

    segundaCantidad = len(client.get('/files',follow_redirects=True));

    remove_file("fileOne.txt");
    remove_file("fileTwo.txt");
    remove_file("fileThree.txt");
    tercerCantidad = len (client.get('/files',follow_redirects=True));

    assert primeraCantidad == tercerCantidad and primeraCantidad + 3 ==
segundaCantidad

```

Al archivo que tiene las pruebas se le añade el client:

```

@pytest.fixture
def client(request):
    client = usermgt.app.test_client()
    return client

```

Se realizan los cambios para los llamados a los metodos add y get. Este codigo maneja json.

Para los métodos post (add or delete):

```

client.post('/files', data = json.dumps(jsonU), content_type =
'application/json')

```

Para el método get:

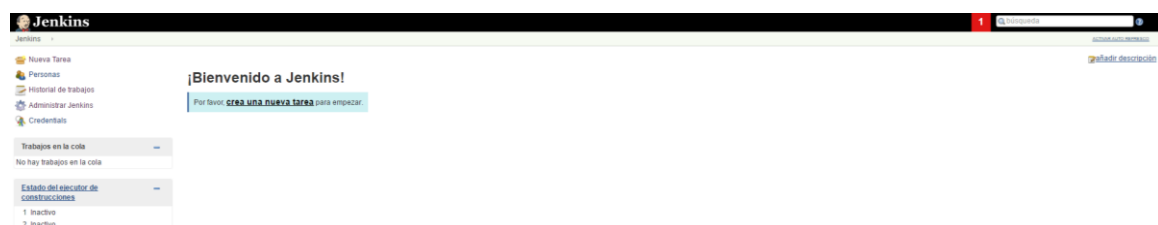
```

client.get('/files',follow_redirects=True)

```

CONTINUAMOS CON LA CREACION DEL PROYECTO

Se crea un nuevo proyecto Iniciamos jenkins con nuestro usuario y password:



Podemos realizar la instalación de plugins si es necesario:

Administrar Jenkins

⚠️ Unsecured Jenkins allows anyone on the network to launch processes on your behalf. Consider at least enabling authentication.



[Configurar el Sistema](#)

Configurar variables globales y rutas.



[Configuración global de la seguridad](#)

Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización).



[Configure Credentials](#)

Configure the credential providers and types



[Global Tool Configuration](#)

Configure tools, their locations and automatic installers.



[Actualizar configuración desde el disco duro.](#)

Descartar todos los datos cargados en memoria y actualizar todo nuevamente desde los ficheros del sistema. Útil para restaurar la configuración de fábrica.



[Administrar Plugins](#)

Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.

Cuando creamos el proyecto lo llamamos testproject y continuamos con las configuraciones dadas en el video dado en la siguiente URL: <https://www.youtube.com/watch?v=Jy6NfzIVAKg>

General Configurar el origen del código fuente Disparadores de ejecuciones Entorno de ejecución Ejecutar

Acciones para ejecutar después.

Proyecto nombre

testproject

Descripción

[Plain text] [Visualizar](#)

☐ Desechar ejecuciones antiguas

☐ Esta ejecución debe parametrizarse

☒ GitHub project

Project url

<https://github.com/dylan9538/testproject.git>

Avanzado...

☐ Throttle builds

☐ Desactivar la ejecución

☐ Lanzar ejecuciones concurrentes en caso de ser necesario

Avanzado...

Se hacen las configuraciones necesarias

Disparadores de ejecuciones

☐ Build after other projects are built
☒ Ejecutar periódicamente

Programador

Would last have run at Monday, November 7, 2016 12:41:20 PM COT, would next run at Monday, November 7, 2016 12:46:20 PM COT.

☐ Build when a change is pushed to GitHub
☐ Consultar repositorio (SCM)

Ejecutar

☐ Ejecutar línea de comandos (shell)

Comando

[Visualizar la lista de variables de entorno disponibles](#)

[Avanzado...](#)

[Añadir un nuevo paso](#)

Acciones para ejecutar después.

☐ Publicar informes de "Cobertura"

Patrón para encontrar los ficheros xml de "Cobertura":

Es el patrón que se usará para localizar los ficheros 'xml' generados por "Cobertura".
 Por ejemplo, para Maven2 usa `*/target/site/cobertura/coverage.xml`
 La ruta es relativa al directorio raíz del módulo a menos que se haya configurado el origen de software (SCM) para múltiples módulos en cuyo caso la ruta es relativa al espacio de trabajo.
 Para que este plugin funcione debe recolectar la información necesario, para ello "Cobertura" debe estar debidamente configurado para que genere los informes XML.

[Avanzado...](#)

☐ Publicar los resultados de tests JUnit

Ficheros XML con los informes de tests:

[El atributo @includes de la etiqueta <fileset>](#) especifica dónde están los ficheros XML generados, por ejemplo: 'myproject/target/test-reports/*.xml'. El directorio base para la etiqueta 'fileset' es el directorio raíz del proyecto

☐ Guardar la salida estándar y de error aunque sea muy larga.

Health report amplification factor:

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

☐ Allow empty results
☐ Do not fail the build on empty test results

☐ Publish HTML reports

Reports

HTML directory to archive:

Index page[s]:

Report title:

[Publishing options...](#)

[Añadir](#)

Luego de esto ya el proyecto está listo para probarse.


En el menú siguiente del proyecto hacemos click en construir ahora y ejecutamos (Igual por la configuración esta ejecutándose periódicamente).




 Volver al Panel de Control

 **Estado Actual**

 Cambios

 Zona de Trabajo

 Construir ahora

 Borrar Proyecto

 Configurar

 Move

 GitHub

Obtenemos los resultados de las pruebas con jenkins.

Module ↓	statements	missing	excluded	coverage
test_comandos.py	42	0	0	100%
uri.py	23	5	0	84 %
settings/__init__.py	0	0	0	100%
settings/development.py	1	0	0	100%

FIN

CUANDO QUIERA SUBIR ARCHIVOS AL REPOSITORIO EN GITHUB DESDE LA CONSOLA DE COMANDOS

1) Creo el archivo si no existe.

2) Sigo los siguientes comandos: Estos comandos los ejecuto donde se encuentra ubicado el archivo a cargar.

```
git add nombreArchivo
git commit -m "upload README file"
git push origin master
```