



# PyBuilder

---

UNIVERSIDAD DE ALMERÍA

HMIS

---

Juan Antonio Pérez Clemente

# Índice

---

|   |    |
|---|----|
| 1. Introducción .....                             | 2  |
| 2. Por qué otra herramienta de construcción ..... | 2  |
| 3. Conceptos .....                                | 2  |
| 4. Construcción de tareas .....                   | 3  |
| 5. Inyección de dependencias .....                | 3  |
| 6. Configuración del proyecto .....               | 4  |
| 6.1 Inicializadores .....                         | 4  |
| 6.2 Atributos del proyecto .....                  | 4  |
| 6.3 Propiedades del proyecto .....                | 4  |
| 7. Ejemplo Práctico .....                         | 5  |
| 7.1 Instalación .....                             | 5  |
| 7.2 Agregando archivos de código .....            | 6  |
| 7.3 Agregando test unitarios .....                | 8  |
| 7.4 Medir la cobertura del código .....           | 12 |

## 1. Introducción

---

PyBuilder es una herramienta de gestión y construcción de software comúnmente destinada a proyectos Python.

Algunas de sus capacidades son:

- Ejecución automática de pruebas unitarias y de integración
- Análisis automático de la cobertura del código
- Ejecución automática e interpretación de resultados de herramientas de análisis, como flake8

La idea general es que todo lo que hagas en tu cadena de integración continua, también lo hagas localmente antes de revisar tu trabajo.

## 2. Por qué otra herramienta de construcción

---

Existían muchas herramientas de construcción y pruebas para Python, pero no eran integrables entre sí. Había que estar realizando script de construcción para cada una de las herramientas, algo tedioso y repetitivo.

PyBuilder nace con el propósito de crear una herramienta que permita hacer las cosas simples de manera simple, las cosas difíciles tan simple como sea posible, utilizando cualquier herramienta que se desee integrar, etc. Todo ello escribiendo los archivos de compilación en Python.

## 3. Conceptos

---

PyBuilder ejecuta la lógica de construcción que se organiza en tareas y acciones.

**Tareas:** Son los principales bloques de construcción de la lógica de construcción. Una tarea es una pieza cerrada que se ejecutará como una sola unidad. Cada tarea puede llamar a un conjunto de otras tareas de las que dependa. Una tarea se ejecutará solo cuando se hayan ejecutado todas sus dependencias.

**Acciones:** Son piezas de la lógica de construcción más pequeñas que las tareas que solo son ejecutadas si son nombradas en una tarea.

## 4. Construcción de tareas

---

Escribir una tarea es fácil, simplemente crea una función y decórala con el decorador `@task`.

```
from pybuilder.core import task

@task
def say_hello ():
    print "Hello, PyBuilder"
```

Hemos creado una tarea llamada `say_hello`. Puede verificar las tareas con el comando `pyb -t`.

## 5. Inyección de dependencias

---

PyBuilder admite la inyección de dependencias para tareas basadas en nombres de parámetros, algunos de los parámetros aceptados para recibir componentes son:

**logger:** Se puede utilizar para emitir mensajes al usuario.

**Project:** Se puede usar para pasar una instancia de un proyecto que se esté construyendo actualmente.

```
from pythonbuilder.core import task

@task
def say_hello (logger):
    logger.info("Hello, PyBuilder")
```

## 6. Configuración del proyecto

---

### 6.1 Inicializadores

Un inicializador es una función de Python que permite inicializar un proyecto pasándoselo por parámetro.

```
from pybuilder.core import init
@init
def initialize(project):
    pass
```

### 6.2 Atributos del proyecto

Son valores que describen al proyecto, como pueden ser la versión o la licencia del mismo. Estos atributos se pueden configurar desde dentro del inicializador.

```
@init
def initialize(project):
    project.version = "0.1.14"
```

### 6.3 Propiedades del proyecto

Se utilizan para configurar los complementos del proyecto. Estos complementos suelen establecer un valor por defecto que puede ser sobrescrito. Por ejemplo, para que PyBuilder sepa cuando un archivo de Python contiene pruebas unitarias, dicho archivo tiene que tener la terminación `_tests`. Esto puede ser sobrescrito utilizando el método `set_property` dentro de un inicializador de la siguiente manera:

```
@init
def initialize(project):
    project.set_property('unittest_module_glob', '*_unittest')
```

Una referencia completa de las propiedades disponibles se incluye en el siguiente enlace:  
<http://pybuilder.github.io/documentation/plugins.html>

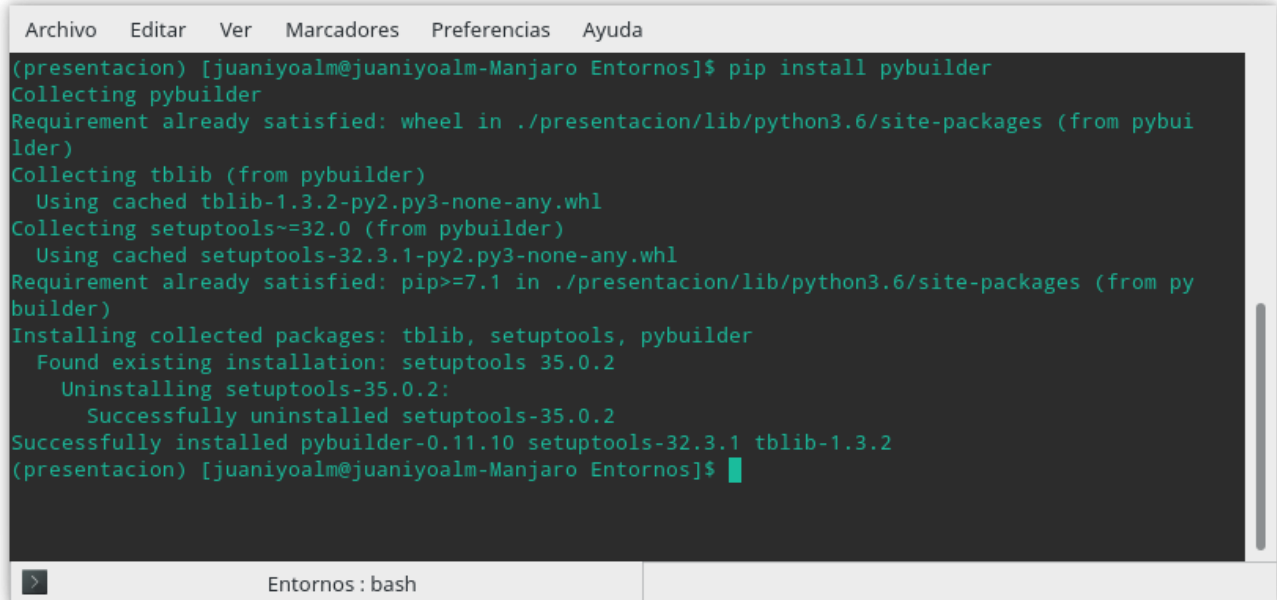
## 7. Ejemplo Práctico

### 7.1 Instalación

PyBuilder es configurado mediante un archivo Python llamado *build.py*. Para realizar esta demostración de funcionamiento se ha creado una clase, *Complejos.py*, con la que trabajaremos. Además, se ha creado un archivo de pruebas unitarias con diferentes métodos de prueba.

Es aconsejable trabajar con un entorno virtual (*virtualenv*).

Una vez creado y activado el entorno virtual, instalamos PyBuilder mediante el comando *pip install pybuilder*.



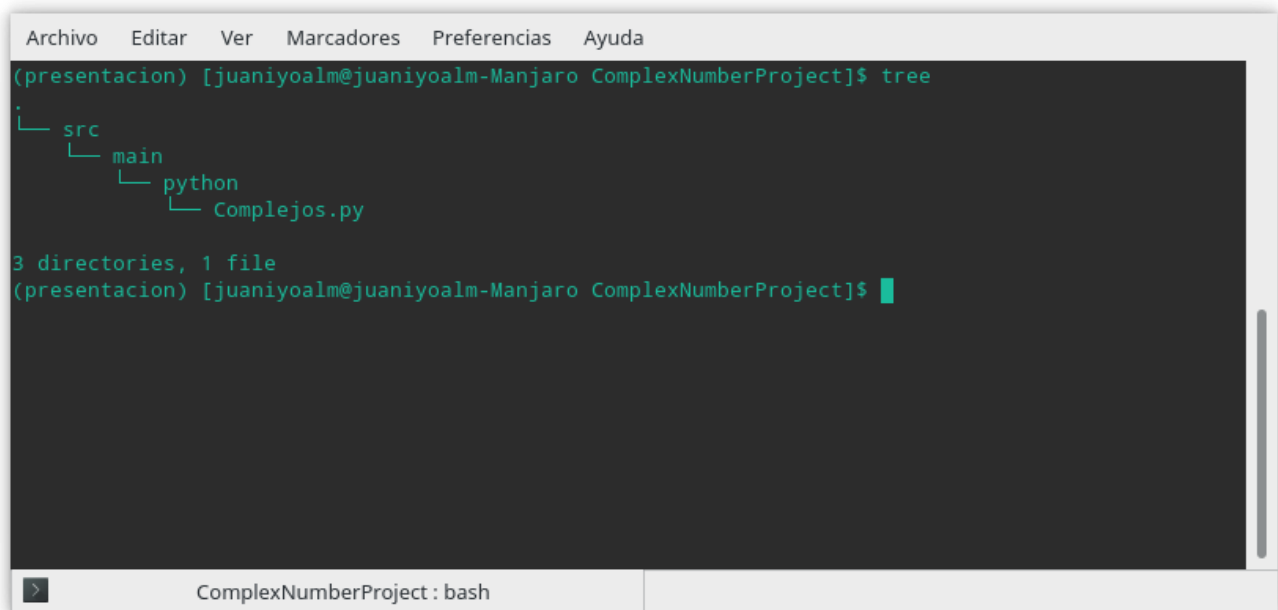
```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro Entornos]$ pip install pybuilder
Collecting pybuilder
Requirement already satisfied: wheel in ./presentacion/lib/python3.6/site-packages (from pybuilder)
Collecting tblib (from pybuilder)
  Using cached tblib-1.3.2-py2.py3-none-any.whl
Collecting setuptools~=32.0 (from pybuilder)
  Using cached setuptools-32.3.1-py2.py3-none-any.whl
Requirement already satisfied: pip>=7.1 in ./presentacion/lib/python3.6/site-packages (from pybuilder)
Installing collected packages: tblib, setuptools, pybuilder
  Found existing installation: setuptools 35.0.2
    Uninstalling setuptools-35.0.2:
      Successfully uninstalled setuptools-35.0.2
Successfully installed pybuilder-0.11.10 setuptools-32.3.1 tblib-1.3.2
(presentacion) [juaniyoalm@juaniyoalm-Manjaro Entornos]$
```

## 7.2 Agregando archivos de código

El siguiente paso es agregar un módulo de Python que contenga nuestro código. PyBuilder separa los archivos de código de los demás en diferentes directorios basados en su significado. La ubicación predeterminada para los códigos principales de Python es

*src/main/Python*

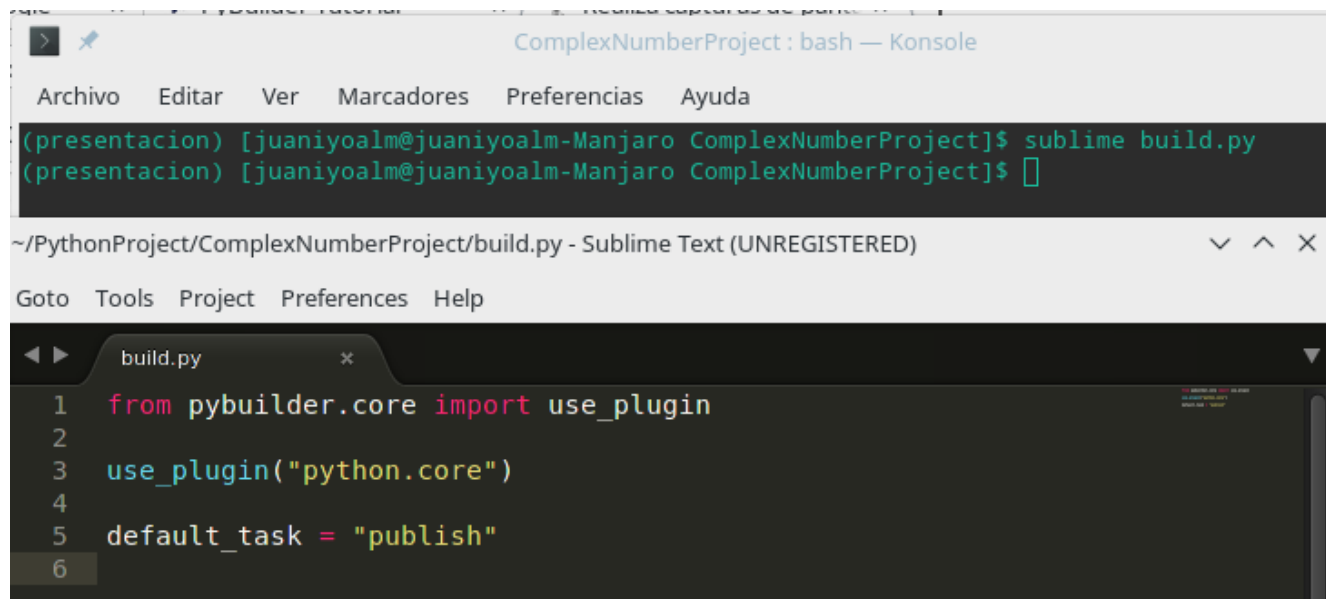
Por lo tanto, crearemos un directorio ComplexNumber y guardaremos la clase complejos en la ruta descrita.



```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ tree
.
├── src
│   └── main
│       └── python
│           └── Complejos.py
3 directories, 1 file
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

ComplexNumberProject : bash

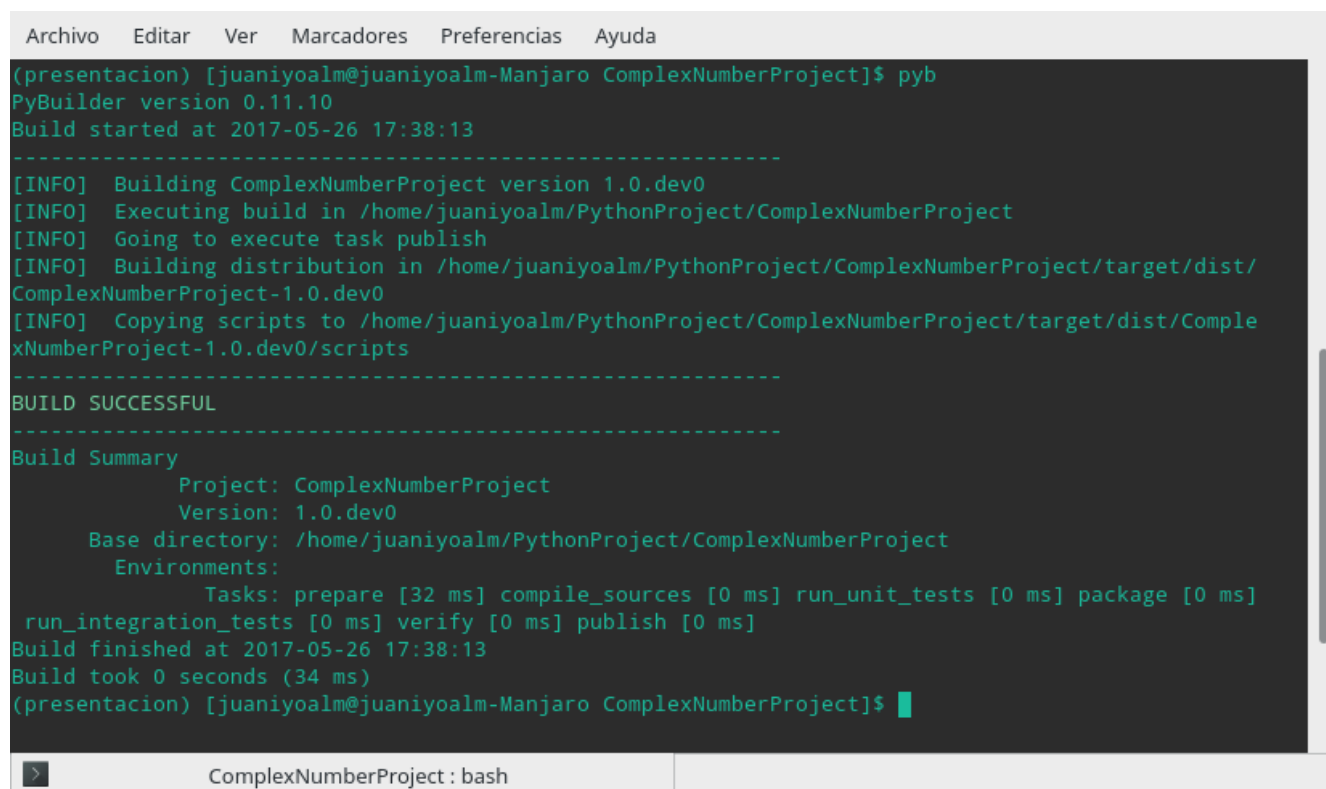
Ahora tendremos que decirle a PyBuilder que queremos construir un proyecto Python. Para ello creamos en la carpeta del proyecto el archivo *build.py* y agregamos lo siguiente



The screenshot shows a terminal window titled "ComplexNumberProject : bash — Konsole". The terminal displays the command `sublime build.py` being executed. Below the terminal, a Sublime Text editor window titled "ComplexNumberProject/build.py - Sublime Text (UNREGISTERED)" is open, showing the contents of `build.py`:

```
1 from pybuilder.core import use_plugin
2
3 use_plugin("python.core")
4
5 default_task = "publish"
6
```

Ahora si ejecutamos el comando *pyb* obtenemos los siguiente:



The screenshot shows a terminal window titled "ComplexNumberProject : bash". The terminal displays the output of the `pyb` command:

```
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ pyb
PyBuilder version 0.11.10
Build started at 2017-05-26 17:38:13
-----
[INFO] Building ComplexNumberProject version 1.0.dev0
[INFO] Executing build in /home/juaniyoalm/PythonProject/ComplexNumberProject
[INFO] Going to execute task publish
[INFO] Building distribution in /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0
[INFO] Copying scripts to /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0/scripts
-----
BUILD SUCCESSFUL
-----
Build Summary
  Project: ComplexNumberProject
  Version: 1.0.dev0
  Base directory: /home/juaniyoalm/PythonProject/ComplexNumberProject
  Environments:
    Tasks: prepare [32 ms] compile_sources [0 ms] run_unit_tests [0 ms] package [0 ms]
           run_integration_tests [0 ms] verify [0 ms] publish [0 ms]
Build finished at 2017-05-26 17:38:13
Build took 0 seconds (34 ms)
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

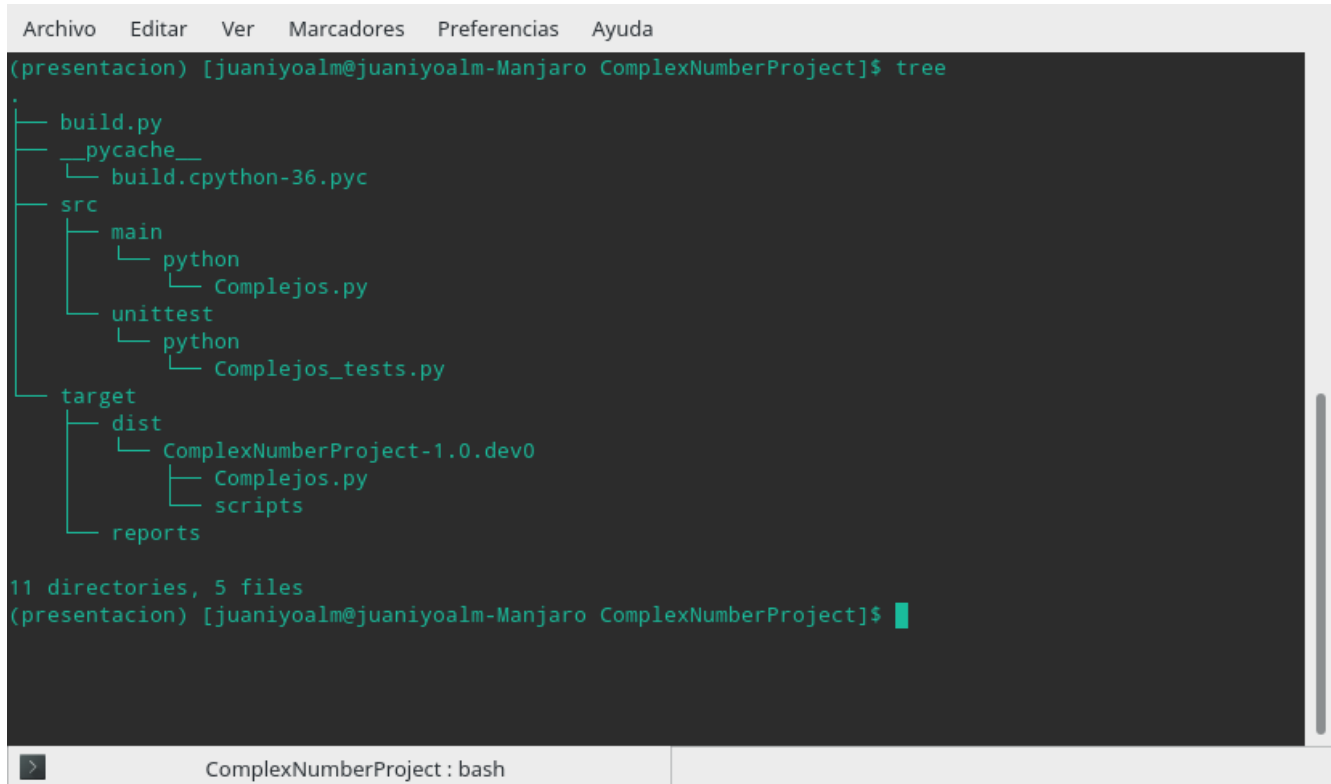


### 7.3 Agregando test unitarios

El directorio que PyBuilder toma por defecto para los test unitarios es el siguiente:

*src/unittest/Python*

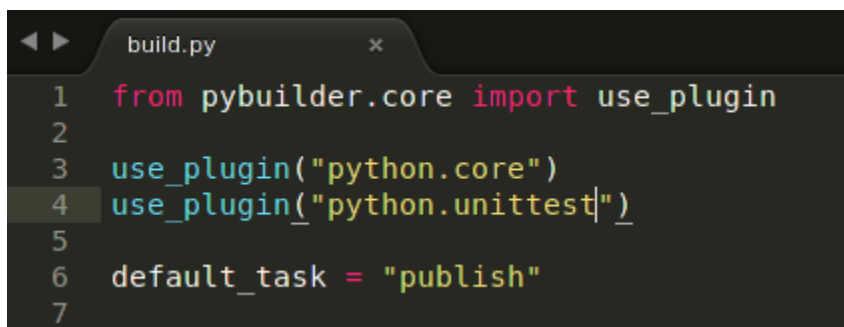
Hay que tener en cuenta que para que PyBuilder reconozca los test de prueba, estos deben tener la terminación `_tests`, aunque como hemos visto anteriormente este parámetro es modificable.



```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ tree
.
├── build.py
├── __pycache__
│   └── build.cpython-36.pyc
├── src
│   ├── main
│   │   └── python
│   │       └── Complejos.py
│   └── unittest
│       └── python
│           └── Complejos_tests.py
└── target
    ├── dist
    │   └── ComplexNumberProject-1.0.dev0
    │       ├── Complejos.py
    │       └── scripts
    └── reports

11 directories, 5 files
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

También debemos agregar al archivo *build.py* el plugin necesario para ejecutar test.



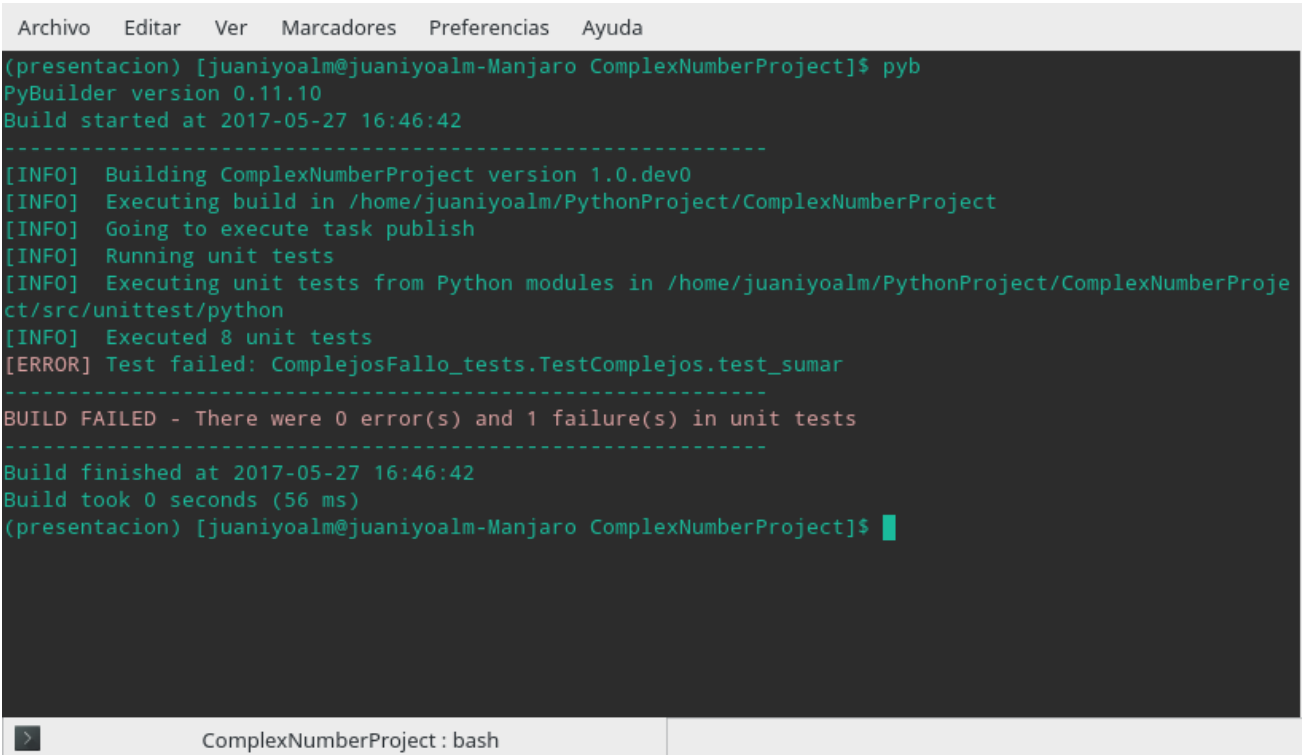
```
build.py
1  from pybuilder.core import use_plugin
2
3  use_plugin("python.core")
4  use_plugin("python.unittest")
5
6  default_task = "publish"
7
```

Si ejecutamos ahora el comando `pyb` obtendremos lo siguiente:

```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ pyb
PyBuilder version 0.11.10
Build started at 2017-05-27 16:40:52
-----
[INFO] Building ComplexNumberProject version 1.0.dev0
[INFO] Executing build in /home/juaniyoalm/PythonProject/ComplexNumberProject
[INFO] Going to execute task publish
[INFO] Installing plugin dependency unittest-xml-reporting
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python
[INFO] Executed 4 unit tests
[INFO] All unit tests passed.
[INFO] Building distribution in /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0
[INFO] Copying scripts to /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0/scripts
-----
BUILD SUCCESSFUL
-----
Build Summary
      Project: ComplexNumberProject
      Version: 1.0.dev0
Base directory: /home/juaniyoalm/PythonProject/ComplexNumberProject
Environments:
      Tasks: prepare [686 ms] compile_sources [0 ms] run_unit_tests [21 ms] package [1 ms]
run_integration_tests [0 ms] verify [0 ms] publish [0 ms]
Build finished at 2017-05-27 16:40:53
Build took 0 seconds (718 ms)
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

ComplexNumberProject : bash

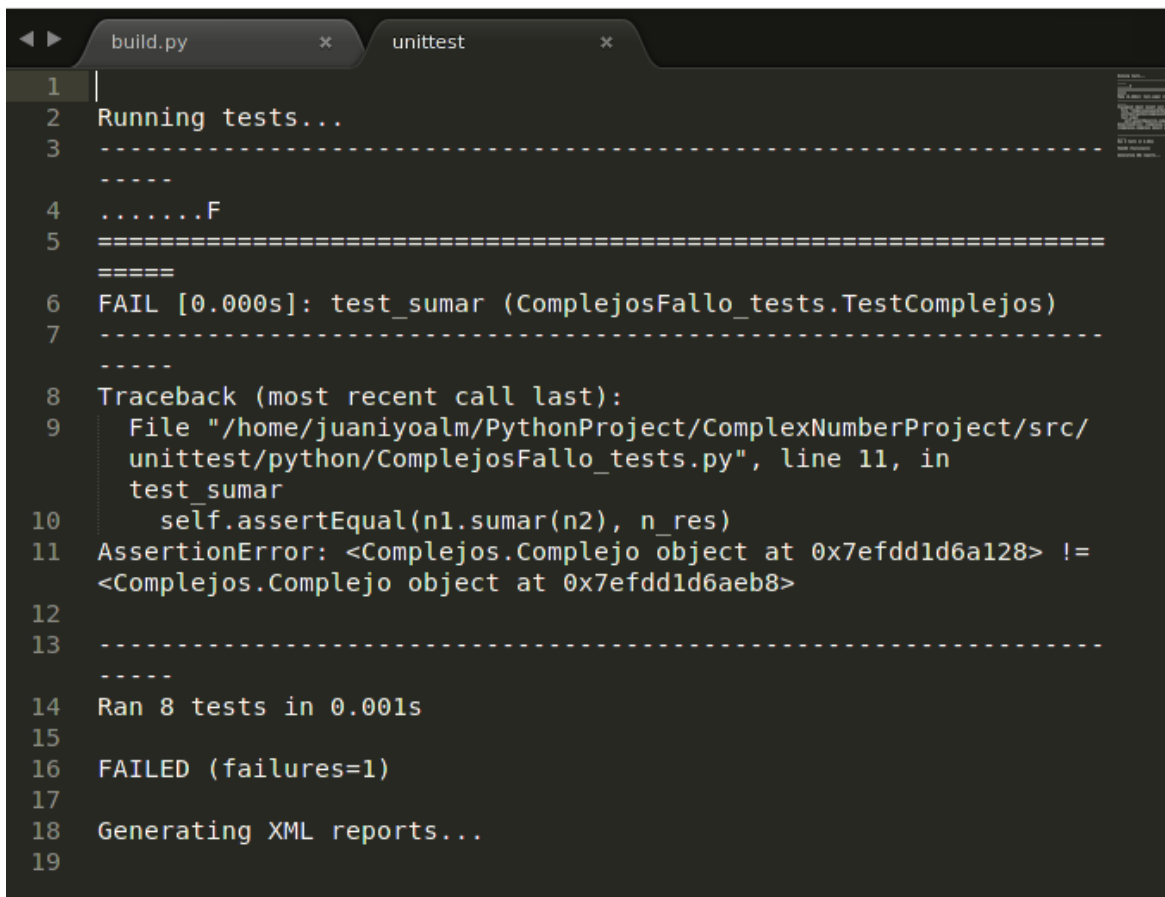
¿Qué ocurre si el test falla? Vamos a comprobarlo, modifiquemos el test para que falle. Hemos duplicado el archivo de test, pero esta vez con fallos y el resultado es el siguiente:



```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ pyb
PyBuilder version 0.11.10
Build started at 2017-05-27 16:46:42

-----
[INFO] Building ComplexNumberProject version 1.0.dev0
[INFO] Executing build in /home/juaniyoalm/PythonProject/ComplexNumberProject
[INFO] Going to execute task publish
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python
[INFO] Executed 8 unit tests
[ERROR] Test failed: ComplejosFallo_tests.TestComplejos.test_sumar
-----
BUILD FAILED - There were 0 error(s) and 1 failure(s) in unit tests
-----
Build finished at 2017-05-27 16:46:42
Build took 0 seconds (56 ms)
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

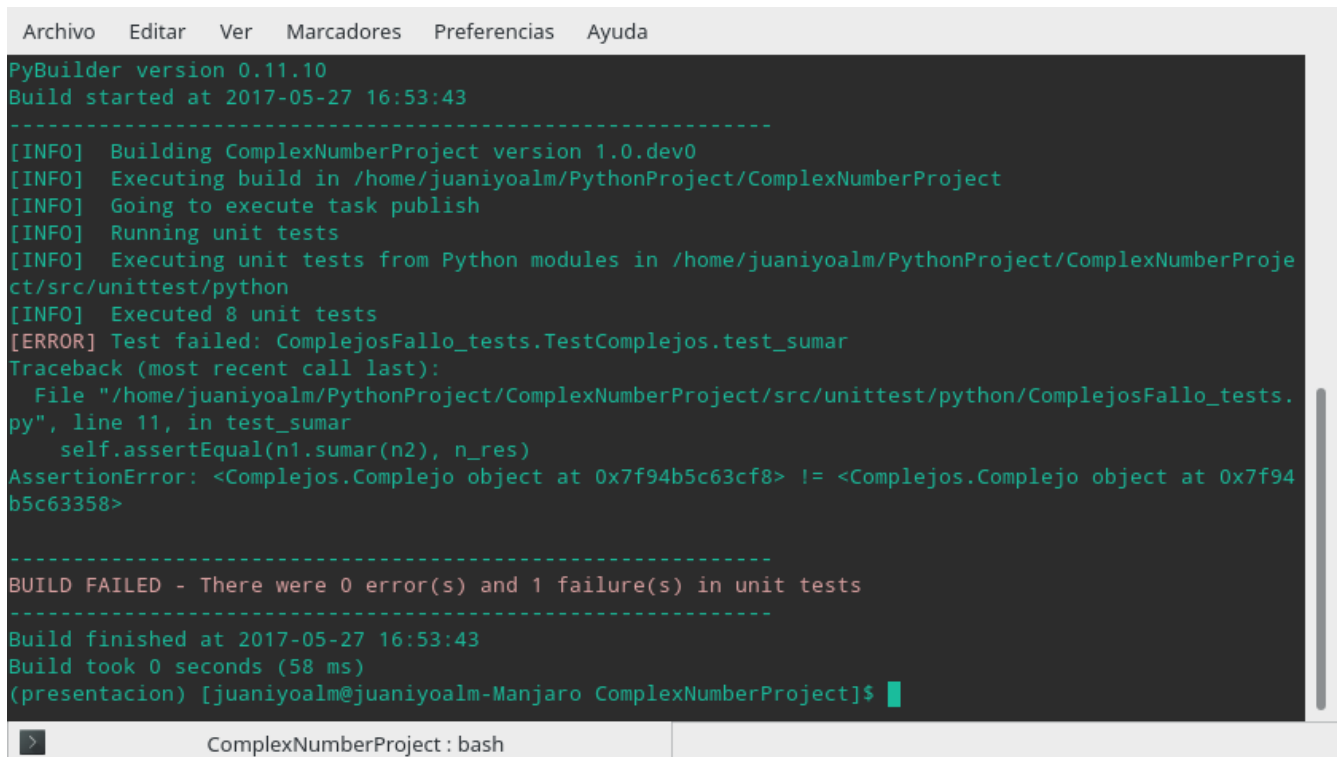
¿Cómo podemos ver que ha fallado concretamente? Hay 2 posibilidades, una de ellas es ejecutar el comando con verbose (`pyb -v`), la otra es dirigirse a la ruta `target/reports/unittest`. Al abrir el archivo comprobaremos que ha fallado.



```

1
2 Running tests...
3 -----
4 .....F
5 =====
6 FAIL [0.000s]: test_sumar (ComplejosFallo_tests.TestComplejos)
7 -----
8 Traceback (most recent call last):
9   File "/home/juaniyoalm/PythonProject/ComplexNumberProject/src/
10     unittest/python/ComplejosFallo_tests.py", line 11, in
11     test_sumar
12     self.assertEqual(n1.sumar(n2), n_res)
13 AssertionError: <Complejos.Complejo object at 0x7efdd1d6a128> !=
14 <Complejos.Complejo object at 0x7efdd1d6aeb8>
15 -----
16 Ran 8 tests in 0.001s
17
18 FAILED (failures=1)
19
20 Generating XML reports...
21

```



```

Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
PyBuilder version 0.11.10
Build started at 2017-05-27 16:53:43
-----
[INFO] Building ComplexNumberProject version 1.0.dev0
[INFO] Executing build in /home/juaniyoalm/PythonProject/ComplexNumberProject
[INFO] Going to execute task publish
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProje
ct/src/unittest/python
[INFO] Executed 8 unit tests
[ERROR] Test failed: ComplejosFallo_tests.TestComplejos.test_sumar
Traceback (most recent call last):
  File "/home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python/ComplejosFallo_tests.
py", line 11, in test_sumar
    self.assertEqual(n1.sumar(n2), n_res)
AssertionError: <Complejos.Complejo object at 0x7f94b5c63cf8> != <Complejos.Complejo object at 0x7f94
b5c63358>
-----
BUILD FAILED - There were 0 error(s) and 1 failure(s) in unit tests
-----
Build finished at 2017-05-27 16:53:43
Build took 0 seconds (58 ms)
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$

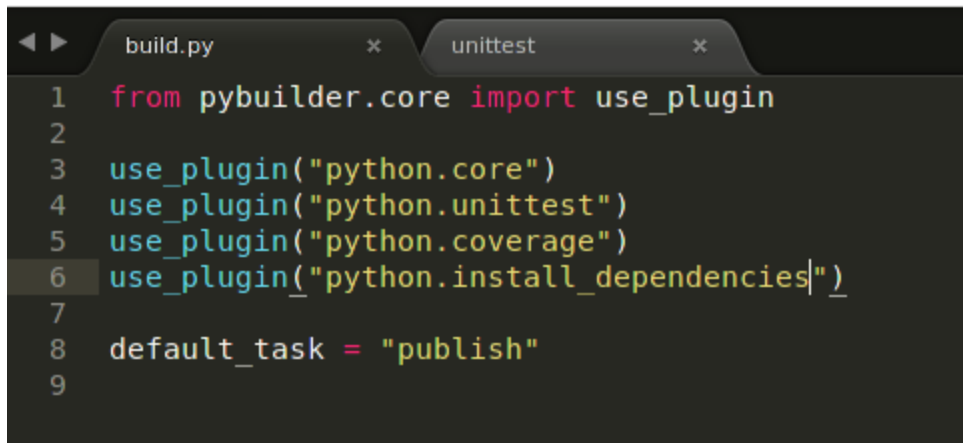
```

## 7.4 Medir la cobertura del código

Para medir la cobertura de código que ha sido probada usaremos *coverage*.

El complemento `python.coverage` utiliza Ned Blatchers `coverage.py` para determinar la cobertura de la prueba. Asegúrese de que se instalará en su entorno virtual ejecutando `pyb install_dependencies` de nuevo. Después de ejecutar `pyb` de nuevo.

Para agregar este plugin simplemente lo incluimos en el archivo `build.py`. Además, debemos instalar la dependencia si no se encuentra, es por ello que también agregaremos la línea de `install_dependencies`.

A screenshot of a code editor with two tabs: 'build.py' and 'unittest'. The 'build.py' tab is active, showing a Python script. The script imports 'use\_plugin' from 'pybuilder.core' and then calls 'use\_plugin' for 'python.core', 'python.unittest', 'python.coverage', and 'python.install\_dependencies'. The last line is 'default\_task = "publish"'.

```
1 from pybuilder.core import use_plugin
2
3 use_plugin("python.core")
4 use_plugin("python.unittest")
5 use_plugin("python.coverage")
6 use_plugin("python.install_dependencies")
7
8 default_task = "publish"
9
```

Tras ello ejecutamos de nuevo `pyb`:

```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ pyb
PyBuilder version 0.11.10
Build started at 2017-05-27 17:09:24
-----
[INFO] Building ComplexNumberProject version 1.0.dev0
[INFO] Executing build in /home/juaniyoalm/PythonProject/ComplexNumberProject
[INFO] Going to execute task publish
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python
[INFO] Executed 4 unit tests
[INFO] All unit tests passed.
[INFO] Building distribution in /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0
[INFO] Copying scripts to /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0/scripts
[INFO] Collecting coverage information
[WARN] coverage_branch_threshold_warn is 0 and branch coverage will not be checked
[WARN] coverage_branch_partial_threshold_warn is 0 and partial branch coverage will not be checked
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python
[INFO] Executed 4 unit tests
[INFO] All unit tests passed.
[INFO] Overall coverage is 92%
[INFO] Overall coverage branch coverage is 100%
[INFO] Overall coverage partial branch coverage is 100%
-----
BUILD SUCCESSFUL
-----
Build Summary
      Project: ComplexNumberProject
      Version: 1.0.dev0
  Base directory: /home/juaniyoalm/PythonProject/ComplexNumberProject
    Environments:
      Tasks: prepare [35 ms] compile_sources [0 ms] run_unit_tests [14 ms] package [1 ms] run_integration_tests [0 ms] verify [130 ms] publish [0 ms]
Build finished at 2017-05-27 17:09:24
Build took 0 seconds (191 ms)
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

ComplexNumberProject : bash

Como se puede apreciar comprobamos los resultados.

También podemos ver gráficamente los resultados comprobando los archivos que se encuentran en `src/reports/coverage`.

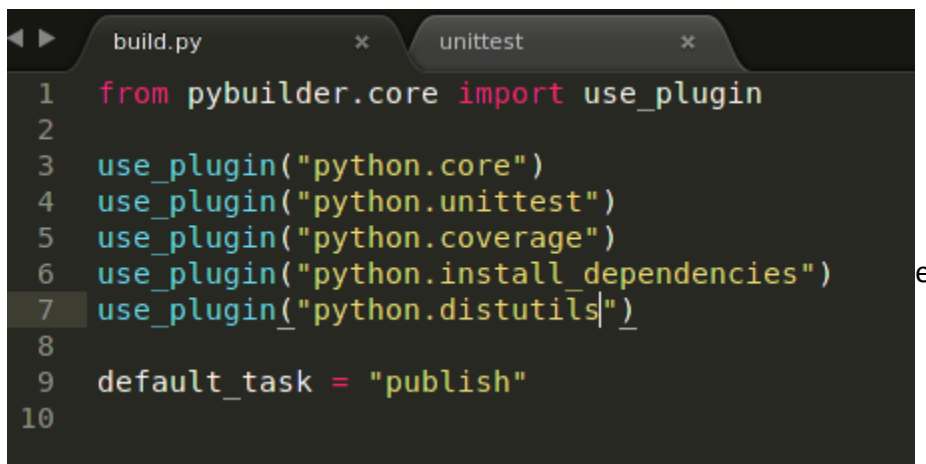
Este complemento además corta la compilación si el porcentaje de código es menor del 70% para así asegurar que el código esta mínimamente comprobado.

Antes de que su código esté disponible para el público, debe pensar en cómo los usuarios pueden instalar el software en sus equipos. En el mundo Python, una forma estándar de hacerlo es usar distutils que se entrega con la distribución estándar de Python.

Usar distutils significa básicamente proporcionar un setup.py que se puede usar para instalar el software.

PyBuilder viene con un complemento, que puede generar el script setup.py para que no necesite escribirlo por su cuenta. PyBuilder descubre automáticamente sus módulos, paquetes y scripts y escribe la configuración para la secuencia de comandos de configuración en consecuencia.

Agregamos lo siguiente a build.py

A screenshot of a code editor with two tabs: 'build.py' and 'unittest'. The 'build.py' tab is active, showing a Python script with the following code:

```
1 from pybuilder.core import use_plugin
2
3 use_plugin("python.core")
4 use_plugin("python.unittest")
5 use_plugin("python.coverage")
6 use_plugin("python.install_dependencies")
7 use_plugin("python.distutils")
8
9 default_task = "publish"
10
```

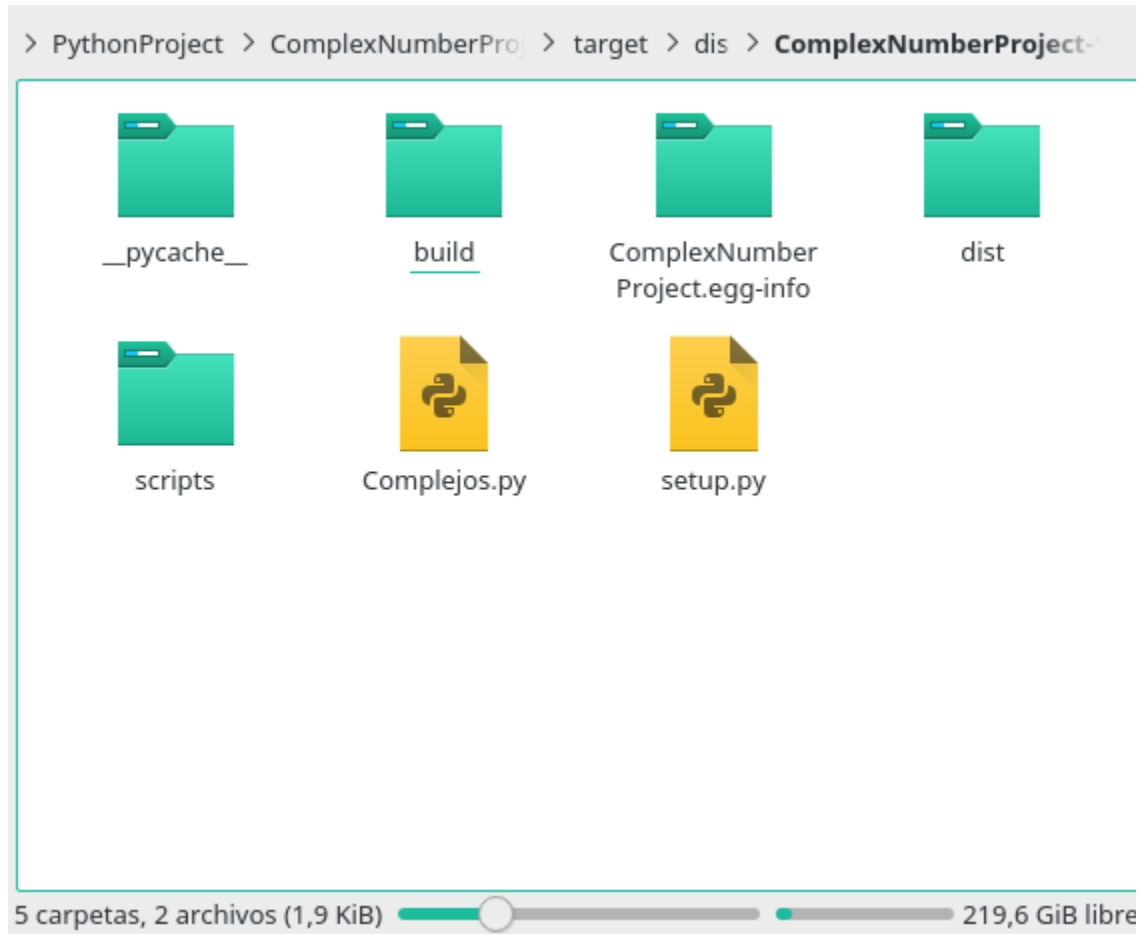
Ejecutamos pyb:

```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$ pyb
PyBuilder version 0.11.10
Build started at 2017-05-27 17:16:35
-----
[INFO] Building ComplexNumberProject version 1.0.dev0
[INFO] Executing build in /home/juaniyoalm/PythonProject/ComplexNumberProject
[INFO] Going to execute task publish
[INFO] Installing plugin dependency py pandoc
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python
[INFO] Executed 4 unit tests
[INFO] All unit tests passed.
[INFO] Building distribution in /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0
[INFO] Copying scripts to /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0/scripts
[INFO] Writing setup.py as /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0/setup.py
[INFO] Collecting coverage information
[WARN] coverage_branch_threshold_warn is 0 and branch coverage will not be checked
[WARN] coverage_branch_partial_threshold_warn is 0 and partial branch coverage will not be checked
[INFO] Running unit tests
[INFO] Executing unit tests from Python modules in /home/juaniyoalm/PythonProject/ComplexNumberProject/src/unittest/python
[INFO] Executed 4 unit tests
[INFO] All unit tests passed.
[INFO] Overall coverage is 92%
[INFO] Overall coverage branch coverage is 100%
[INFO] Overall coverage partial branch coverage is 100%
[INFO] Building binary distribution in /home/juaniyoalm/PythonProject/ComplexNumberProject/target/dist/ComplexNumberProject-1.0.dev0
-----
BUILD SUCCESSFUL
-----
Build Summary
    Project: ComplexNumberProject
    Version: 1.0.dev0
    Base directory: /home/juaniyoalm/PythonProject/ComplexNumberProject
    Environments:
        Tasks: prepare [1398 ms] compile_sources [0 ms] run_unit_tests [18 ms] package [1 ms]
run_integration_tests [0 ms] verify [129 ms] publish [515 ms]
Build finished at 2017-05-27 17:16:37
Build took 2 seconds (2076 ms)
(presentacion) [juaniyoalm@juaniyoalm-Manjaro ComplexNumberProject]$
```

ComplexNumberProject : bash



Te crea una carpeta con lo siguiente



Un segundo directorio es el directorio dist que contiene la distribución. El directorio de distribución contiene los mismos orígenes, pero en un diseño de directorio típico de Python. También puede encontrar el script setup.py allí.