Tests en .NET, Précis et Concis

Table des matières

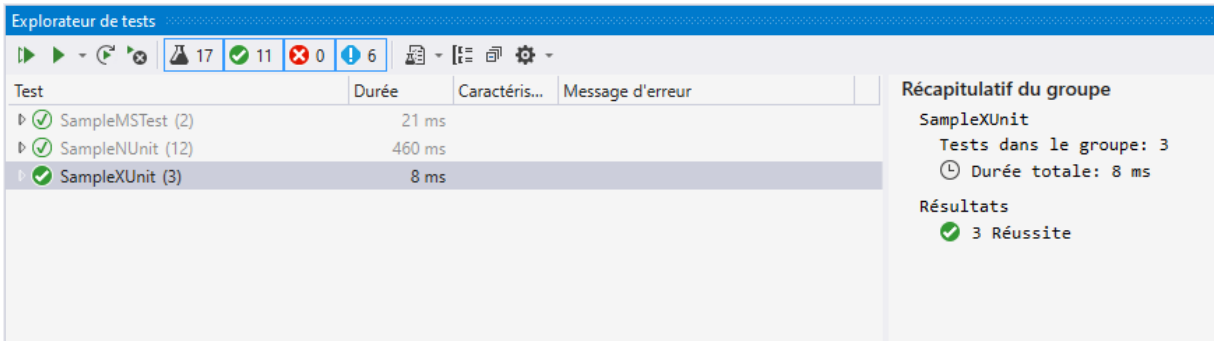
Types de test :

- Unit test (test uniquement partie par partie des classes, etc.)
- Integration test (valide un fonctionnement/ complet)
- User Interface test

Arrange => Act => Assert

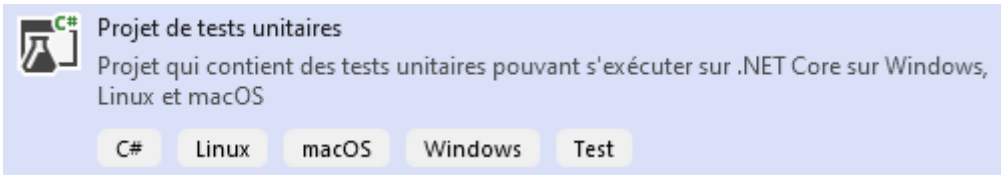
Explorateur de tests

Pour l'afficher menu « Test » … « Explorateur de tests ». On peut lancer tous les tests ou seulement sélectionner, de même pour le débogage



MS Tests

Créer un projet MS Tests



Et référencer le projet à tester

Exemple de test

```
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Sample
{
    [TestClass]
    public class CalculatorMSTests
    {
        [TestMethod]
        public void Sum_TwoInt_GetCorrectResult()
        {
            // Arrange
            Calculator calculator = new Calculator();

            // Act
            int result = calculator.Sum(2, 3);

            // Assert
            Assert.AreEqual(5, result);
```

```
        }
    }
}
```

Test case

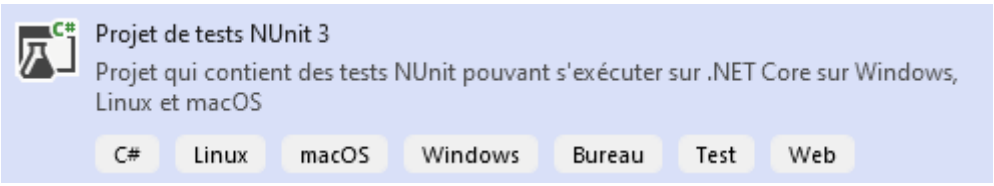
```
[TestMethod]
[DataRow(1, 1)]
[DataRow(2, 2)]
public void Multiply_TwoInt_GetCorrectResult(int a, int b)
{
    // Arrange
    Calculator calculator = new Calculator();
    int expected = a * b;

    // Act
    int result = calculator.Multiply(a, b);

    // Assert
    Assert.AreEqual(expected, result);
}
```

Variante avec le résultat attendu

```
[TestMethod]
[DataRow(1, 1, 1)]
[DataRow(2, 2, 4)]
public void Multiply_TwoInt_GetCorrectResult(int a, int b, int expected)
{
    // Arrange
    Calculator calculator = new Calculator();

    // Act
    int result = calculator.Multiply(a, b);

    // Assert
    Assert.AreEqual(expected, result);
}
```

NUnit

Créer un projet de tests NUnit



Et référencer le projet à tester

A partir d'une bibliothèque de classes « vide »

Packages à installer

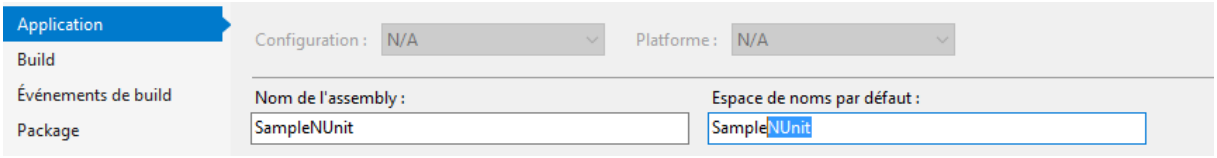
```
INSTALL-PACKAGE NUnit
INSTALL-PACKAGE NUnit3TestAdapter
INSTALL-PACKAGE Microsoft.NET.Test.Sdk
```

Ou ajouter les PackageReferences ua csproj

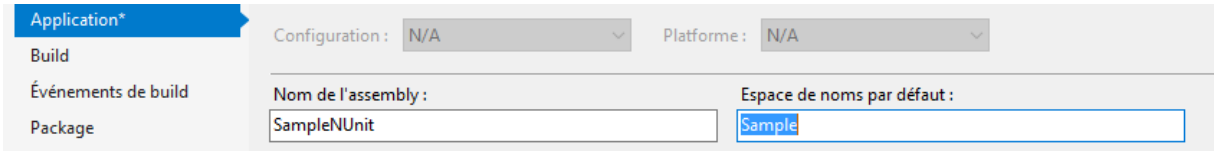
```
<PackageReference Include="NUnit" Version="3.12.0" />
<PackageReference Include="NUnit3TestAdapter" Version="3.16.1" />
<PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.5.0"/>
```

Changer le namespace par défaut du projet de test

Permet d'avoir une structure similaire au projet testé



…



Exemple de test

```
using NUnit.Framework;

namespace Sample
{
    [TestFixture]
    public class CalculatorNUnitTests
    {
        [Test]
        public void Sum_TwoInt_GetCorrectResult()
        {
            // Arrange
            Calculator calculator = new Calculator();

            // Act
            int result = calculator.Sum(2, 3);

            // Assert
            Assert.AreEqual(5, result);
        }
    }
}
```

Test case

Exemple : 2 tests seront exécutés

On passe les paramètres à la méthode de test

```
        [Test]
        [TestCase(1,1)]
        [TestCase(2, 2)]
        public void Multiply_TwoInt_GetCorrectResult(int a , int b)
        {
            // Arrange
            Calculator calculator = new Calculator();
            int expected = a * b;
```

```
            // Act
            int result = calculator.Multiply(a, b);

            // Assert
            Assert.AreEqual(expected, result);
        }
```

Ou avec **expected result**

On indique le résultat attendu dans les attributs, on retourne la méthode testée et affecte le type de résultat à la function de test

```
        [Test]
        [TestCase(1, 1, ExpectedResult = 1)]
        [TestCase(2, 2, ExpectedResult = 4)]
        public int Multiply_TwoInt_GetCorrectResult(int a, int b)
        {
            Calculator calculator = new Calculator();
            return calculator.Multiply(a, b);
        }
```

« Assert .That » … « Is.. »

Offre des nombreuses méthodes utilitaires

```
Assert.AreEqual(result, "Hello World!");
Assert.That(result, Is.EqualTo("Hello World!"));
```

Assert multiple

```
[Test]
public void GetMessage_ReturnTheCorrectMessage()
{
    // Arrange
    var service = new HelloService();

    // Act
    string result = service.GetMessage("World");

    // Assert
    Assert.Multiple(() =>
    {
        Assert.AreEqual(result, "Hello World!");

        Assert.That(result, Is.EqualTo("Hello World!"));
        Assert.That(result, Does.Contain("world").IgnoreCase);
        Assert.That(result, Does.StartWith("Hello"));
        Assert.That(result, Does.EndWith("!"));
        Assert.That(result, Does.Match("Hello [A-Z]{1}[a-z]+!"));
    });
}
```

Equivalent et méthodes utiles de test pour string

Test de collections

```
[Test]
public void Collection_IsEquivalent()
{
    var result = new List<string> { "A", "B" };
```

```
    var comparison = new List<string> { "A", "B" };

    Assert.That(result, Is.EquivalentTo(comparison));

    Assert.That(result, Does.Contain("B"));
    Assert.That(result, Is.Not.Empty);
    Assert.That(result.Count, Is.EqualTo(2));
    Assert.That(result, Has.No.Member("C"));
    Assert.That(result, Is.Ordered);
    Assert.That(result, Is.Unique);
}
```

« IsInRange »

```
Assert.That(15, Is.InRange(10,20));
```

Exception

```
[Test]
public void GetMessage_WithoutName_ThrowException()
{
    var service = new HelloService();

    var exception = Assert.Throws<ArgumentException>(() => service.GetMessage(""));

    Assert.AreEqual("A name is required", exception.Message);

    // with That
    Assert.That(() => service.GetMessage(""),
Throws.ArgumentException.With.Message.EqualTo("A name is required"));
}
```

Méthode « SetUp »

Permet d'initialiser des éléments utilisés par la classe de tests. Exemple

```
[TestFixture]
public class HelloServiceNUnitTests
{
    HelloService _helloService;

    [SetUp]
    public void Setup()
    {
        _helloService = new HelloService();
    }

    [Test]
    public void GetMessage_WithSetup_ReturnTheCorrectMessage()
    {
        string result = _helloService.GetMessage("World");
        Assert.AreEqual(result, "Hello World!");
    }
}
```

Test Type

```
var cat = new Cat();
Assert.That(cat, Is.TypeOf<Cat>());
```

Moq (marche avec tous les frameworks, exemples avec Nunit)

Installation

```
INSTALL-PACKAGE Moq
```

Ou PackageReference

```
<PackageReference Include="Moq" Version="4.16.1" />
```

Exemple : on remplace un service (logger)

```
var mock = new Mock<ILogger>();
mock.Setup(x => x.Log(""));

var service = new HelloService(mock.Object);

string result = service.GetMessage("World");
Assert.AreEqual(result, "Hello World!");
```

```
public class HelloService : IHelloService
{
    private readonly ILogger logger;

    public HelloService(ILogger logger)
    {
        this.logger = logger;
    }

    public HelloService()
        : this(new ConsoleLogger())
    {

    }

    public string GetMessage(string name)
    {
        if (string.IsNullOrWhiteSpace(name))
            throw new System.ArgumentException("A name is required");

        logger.Log($"GetMessage {name}");
        return $"Hello {name}!";
    }
}
```

Return value

```
[Test]
public void GetMessage_WithMoq_GetMockMessage()
{
    var mock = new Mock<IHelloService>();
    mock.Setup(x => x.GetMessage("")).Returns("Empty!");
    mock.Setup(x => x.GetMessage("world")).Returns("World!");

    Assert.AreEqual(mock.Object.GetMessage(""), "Empty!");
    Assert.AreEqual(mock.Object.GetMessage("world"), "World!");
}
```

Retourner un résultat pour toutes les valeurs passées

```
[Test]
public void GetMessage_WithMoq_GetMockMessage()
{
    var mock = new Mock<IHelloService>();
    mock.Setup(x => x.GetMessage(It.IsAny<string>())).Returns("value");
    Assert.AreEqual(mock.Object.GetMessage(""), "value");
```

```
    Assert.AreEqual(mock.Object.GetMessage("world"), "value");
}
```

Fonction de retour

```
[Test]
public void GetMessage_WithMoq_GetMockMessage()
{
    var mock = new Mock<IHelloService>();
    mock.Setup(x => x.GetMessage(It.IsAny<string>())).Returns((string x) =>
x.ToUpper());

    Assert.AreEqual(mock.Object.GetMessage("world"), "WORLD");
}
```

Mock de property

```
var mock = new Mock<Cat>();
mock.Object.Cry = "Miaou";
```

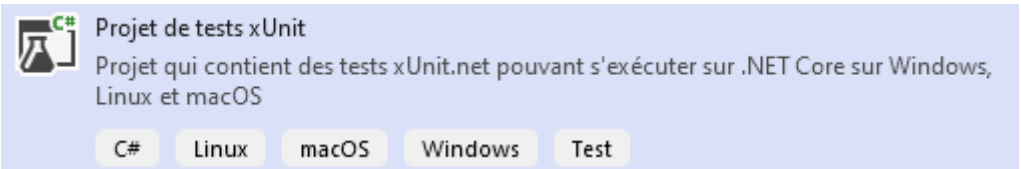
Mock verify

```
mock.Verify(x=> x.GetMessage(It.IsAny<string>()), Times.Exactly(1));
```

XUnit

Compariason de XUnit avec les autres frameworks de test
https://xunit.net/docs/comparisons

Création de projet XUnit



A partir d'une bibliothèque de classes « vide »

```
Install-Package Microsoft.NET.Test.Sdk
Install-Package xunit
Install-Package xunit.runner.visualstudio
Install-Package coverlet.collector
```

Exemple

```
using Xunit;

namespace Sample
{
    public class CalculatorXUnitTests
```

```
    {
        [Fact]
        public void Sum_TwoInt_GetCorrectResult()
        {
            // Arrange
            Calculator calculator = new Calculator();

            // Act
            int result = calculator.Sum(2, 3);

            // Assert
            Assert.Equal(5, result);
        }
}
```

« Test case »

« Theaory » à la place de « Fact » et « InlineData » pour passer les valeurs

```
[Theory]
[InlineData(1, 1, 1)]
[InlineData(2, 2, 4)]
public void Multiply_TwoInt_GetCorrectResult(int a, int b, int expected)
{
    // Arrange
    Calculator calculator = new Calculator();

    // Act
    int result = calculator.Multiply(a, b);

    // Assert
    Assert.Equal(expected, result);
}
```

Lancer les tests automatiquement

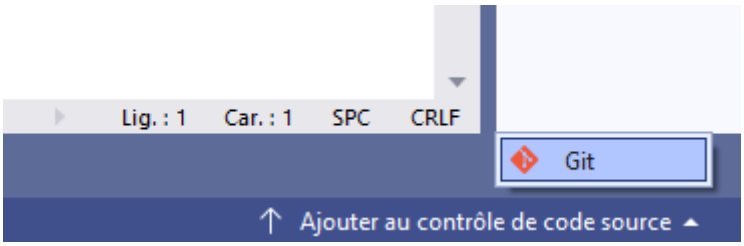
Avec .NET Cli

```
dotnet test
```

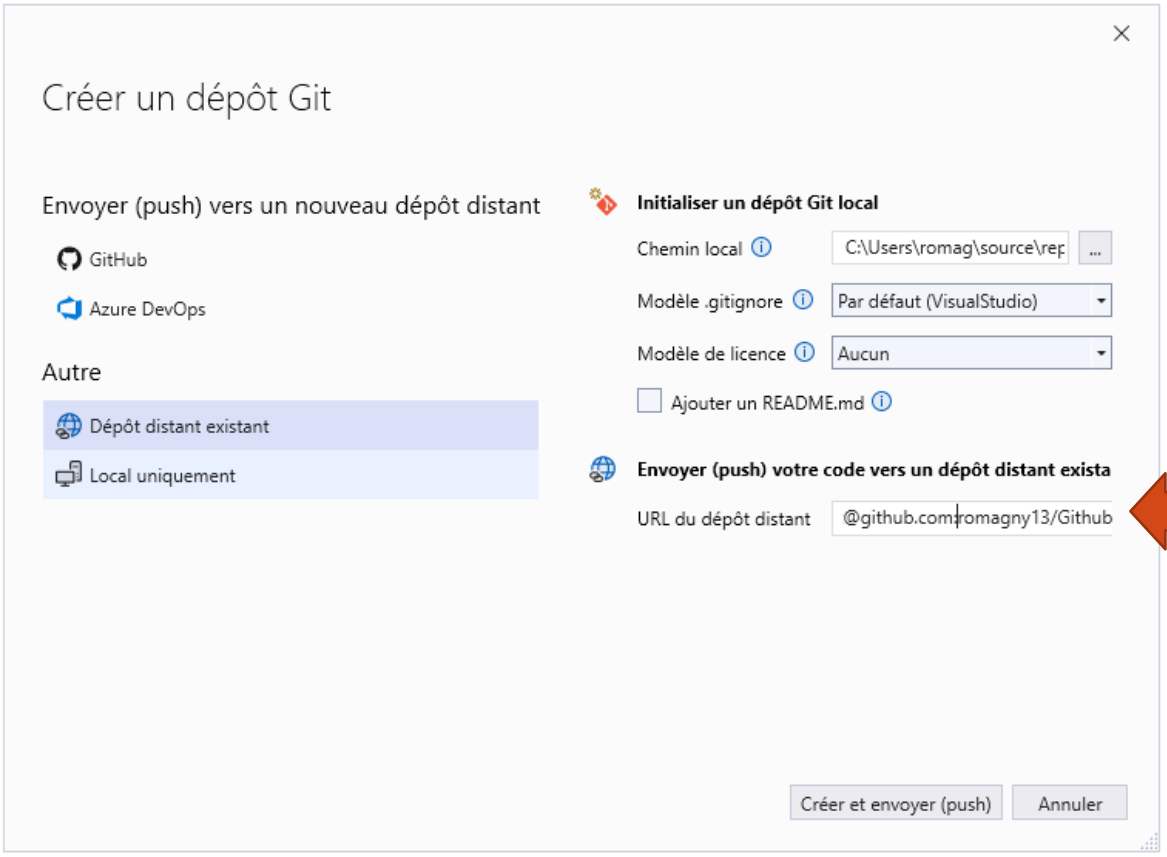
Github Actions

Création d'un **repository sur Gihtub**

Dans Visual Studio : cliquer sur **Ajouter au contrôle de code source .. Git**

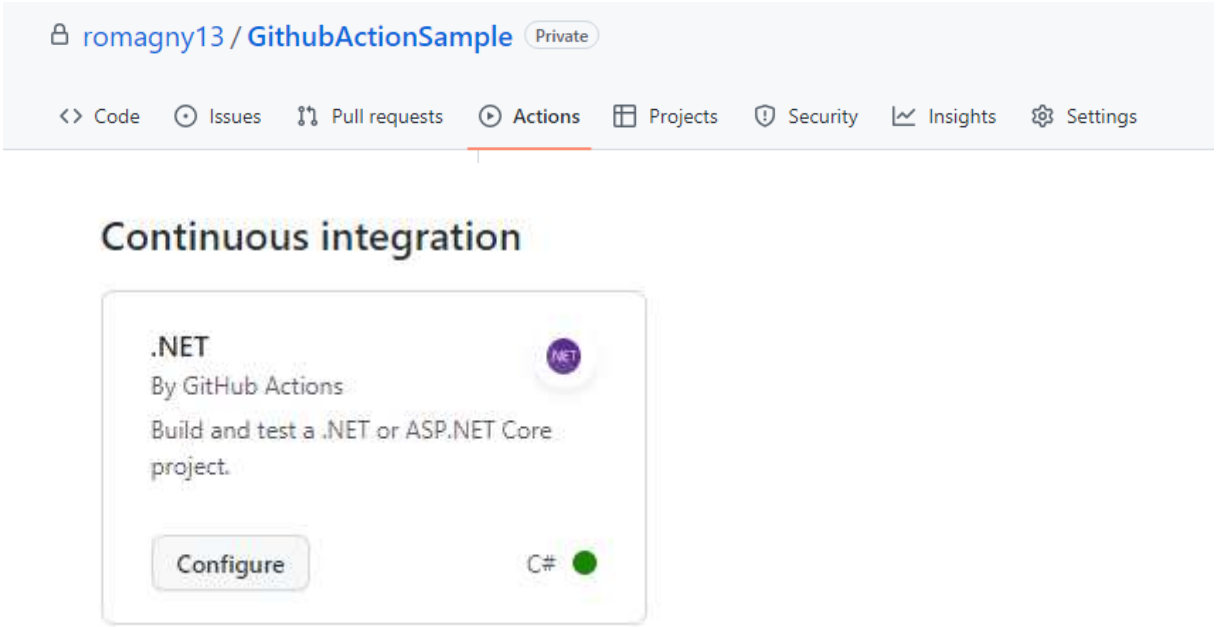


Coller l'url exemple « git@github.com:romagny13/GithubSample.git »



Cliquer sur **créer et envoyer** … le code source est push sur Github

Sur Github, **Onglet « Actions »**



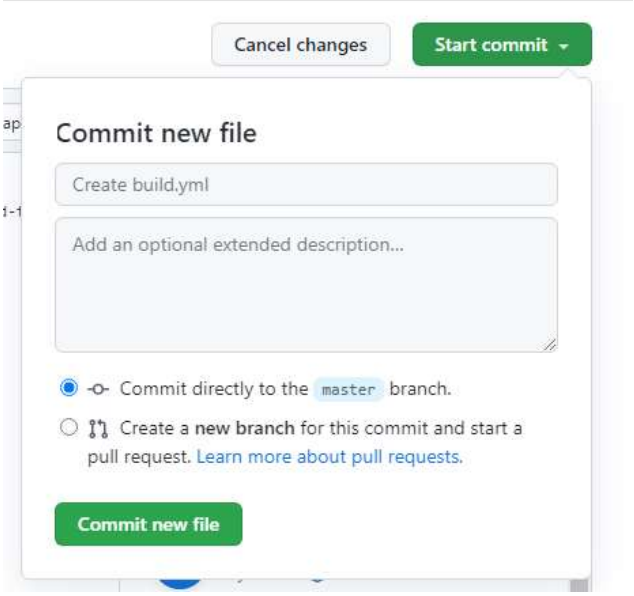
Renommer le fichier en build.yml

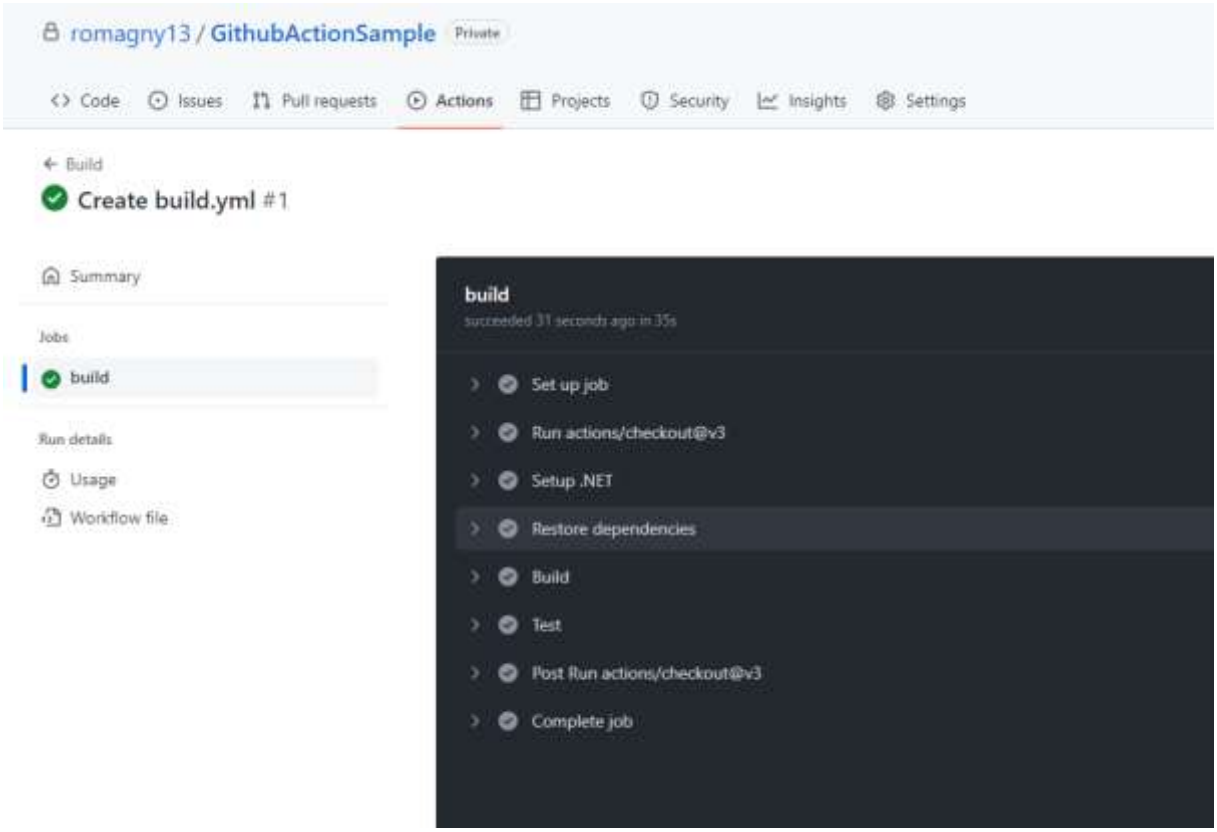
GithubActionSample / .github / workflows / build.yml in master

Edit new file | Preview

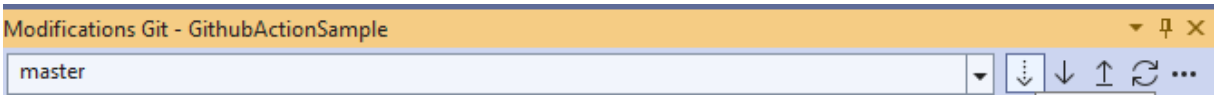
```yaml
# This workflow will build a .NET project
# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-net

name: Build

on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: 6.0.x
    - name: Restore dependencies
      run: dotnet restore
    - name: Build
      run: dotnet build --no-restore
    - name: Test
      run: dotnet test --no-build --verbosity normal

```

… cliquer sur « start commit »





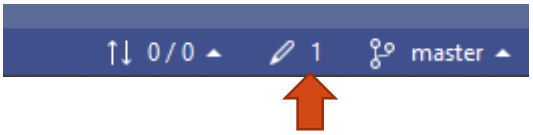
Pour récupérer les modifications faites (ajout fichier build.yml) … cliquer la flèche « récupérer »



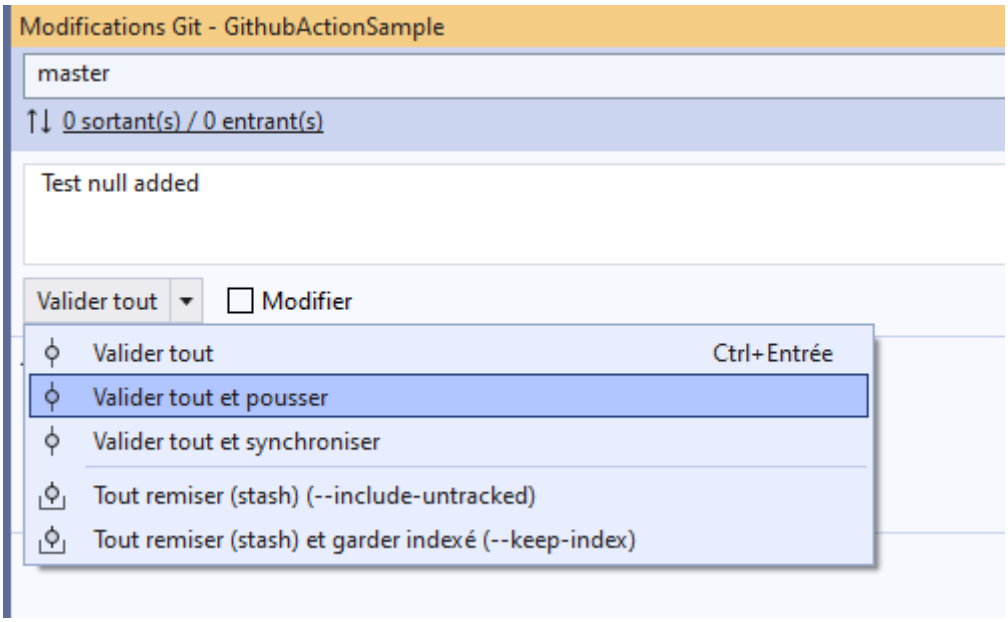
.. Cliquer sur Pull ou « Tirer »

Modification et push

Quand des modifications ont été apportées au projet on peut directement les valider en cliquant sur l'icone crayon



Ajouter un message puis « valider » ou « valider et pousser » directement



Les tests seront relancés automatiquement par les Github Actions au push