AUTOMATE THE PLANET

NUnit Unit Testing Framework Cheat Sheet

Installation

```
Install-Package NUnit
Install-Package NUnit.TestAdapter
Install-Package Microsoft.NET.Test.Sdk
```

Test Execution Workflow

```
using NUnit.Framework;
namespace NUnitUnitTests
{
   // A class that contains NUnit unit tests. (Required)
   [TestFixture]
   public class NonBellatrixTests
   {
      [OneTimeSetUp]
      public void ClassInit()
      {
         // Executes once for the test class. (Optional)
      }
      [SetUp]
      public void TestInit()
      {
         // Runs before each test. (Optional)
      }
      [Test]
      public void TestMethod()
      {
      }
      [TearDown]
      public void TestCleanup()
      {
         // Runs after each test. (Optional)
      }
      [OneTimeTearDown]
      public void ClassCleanup()
      {
         // Runs once after all tests in this class are
executed. (Optional)
         // Not guaranteed that it executes instantly after all
tests from the class.
      }
   }
}
// A SetUpFixture outside of any namespace provides
SetUp and TearDown for the entire assembly.
[SetUpFixture]
public class MySetUpClass
{
  [OneTimeSetUp]
  public void RunBeforeAnyTests()
  {
     // Executes once before the test run. (Optional)
  }
  [OneTimeTearDown]
  public void RunAfterAnyTests()
  {
     // Executes once after the test run. (Optional)
  }
}
```

Attributes

NUnit 3.x	MSTest v2.x.	xUnit.net 2.x	Comments
[Test]	[TestMethod]	[Fact]	Marks a test method.
[TestFixture]	[TestClass]	n/a	Marks a test class.
[SetUp]	[TestInitialize]	Constructor	Triggered before every test case.
[TearDown]	[TestCleanup]	IDisposable.Dispose	Triggered after every test case.
[OneTimeSetUp]	[ClassInitialize]	IClassFixture<T>	One-time triggered method before test cases start.
[OneTimeTearDown]	[ClassCleanup]	IClassFixture<T>	One-time triggered method after test cases end.
[Ignore("reason")]	[Ignore]	[Fact(Skip="reason")]	Ignores a test case.
[Property]	[TestProperty]	[Trait]	Sets arbitrary metadata on a test.
[Theory]	[DataRow]	[Theory]	Configures a data-driven test.
[Category("")]	[TestCategory("")]	[Trait("Category", "")]	Categorizes the test cases or classes.

Assertions- Classic Model

```
Assert.AreEqual(28, _actualFuel); // Tests whether the specified values are equal.
Assert.AreNotEqual(28, _actualFuel); // Tests whether the specified values are unequal. Same as AreEqual for numeric values.
Assert.AreSame(_expectedRocket, _actualRocket); // Tests whether the specified objects both refer to the same object
Assert.AreNotSame(_expectedRocket, _actualRocket); // Tests whether the specified objects refer to different objects
Assert.IsTrue(_isThereEnoughFuel); // Tests whether the specified condition is true
Assert.IsFalse(_isThereEnoughFuel); // Tests whether the specified condition is false
Assert.IsNull(_actualRocket); // Tests whether the specified object is null
Assert.IsNotNull(_actualRocket); // Tests whether the specified object is non-null
Assert.IsInstanceOf(_actualRocket, typeof(Falcon9Rocket)); // Tests whether the specified object is an instance of the expected type
Assert.IsNotInstanceOf(_actualRocket, typeof(Falcon9Rocket)); // Tests whether the specified object is not an instance of type
StringAssert.AreEqualIgnoringCase(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified strings are equal ignoring their casing
StringAssert.Contains(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string contains the specified substring
StringAssert.DoesNotContain(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string doesn't contain the specified substring
StringAssert.StartsWith(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string begins with the specified substring
StringAssert.StartsWith(_expectedBellatrixTitle, "Bellatrix"); // Tests whether the specified string begins with the specified substring
StringAssert.IsMatch("(281)388-0388", @"(?d{3})?-? *d{3}-? *-?d{4}"); // Tests whether the specified string matches a regular expression
StringAssert.DoesNotMatch("281)388-0388", @"(?d{3})?-? *d{3}-? *-?d{4}"); // Tests whether the specified string does not match a regular expression
CollectionAssert.AreEqual(_expectedRockets, _actualRockets); // Tests whether the specified collections have the same elements in the same order and quantity.
CollectionAssert.AreNotEqual(_expectedRockets, _actualRockets); // Tests whether the specified collections does not have the same elements or the elements are in a different order and quantity.
CollectionAssert.AreEquivalent(_expectedRockets, _actualRockets); // Tests whether two collections contain the same elements.
CollectionAssert.AreNotEquivalent(_expectedRockets, _actualRockets); // Tests whether two collections contain different elements.
CollectionAssert.AllItemsAreInstancesOfType(_expectedRockets, _actualRockets); // Tests whether all elements in the specified collection are instances of the expected type
CollectionAssert.AllItemsAreNotNull(_expectedRockets); // Tests whether all items in the specified collection are non-null
CollectionAssert.AllItemsAreUnique(_expectedRockets); // Tests whether all items in the specified collection are unique
CollectionAssert.Contains(_actualRockets, falcon9); // Tests whether the specified collection contains the specified element
CollectionAssert.DoesNotContain(_actualRockets, falcon9); // Tests whether the specified collection does not contain the specified element
CollectionAssert.IsSubsetOf(_expectedRockets, _actualRockets); // Tests whether one collection is a subset of another collection
CollectionAssert.IsNotSubsetOf(_expectedRockets, _actualRockets); // Tests whether one collection is not a subset of another collection
Assert.Throws<ArgumentNullException>(() => new Regex(null)); // Tests whether the code specified by delegate throws exact given exception of type T
```

Assertions- Constraint Model

```
Assert.That(28, Is.EqualTo(_actualFuel)); // Tests whether the specified values are equal.
Assert.That(28, Is.Not.EqualTo(_actualFuel)); // Tests whether the specified values are unequal. Same as AreEqual for numeric values.
Assert.That(_expectedRocket, Is.SameAs(_actualRocket)); // Tests whether the specified objects both refer to the same object
Assert.That(_expectedRocket, Is.Not.SameAs(_actualRocket)); // Tests whether the specified objects refer to different objects
Assert.That(_isThereEnoughFuel, Is.True); // Tests whether the specified condition is true
Assert.That(_isThereEnoughFuel, Is.False); // Tests whether the specified condition is false
Assert.That(_actualRocket, Is.Null); // Tests whether the specified object is null
Assert.That(_actualRocket, Is.Not.Null); // Tests whether the specified object is non-null
Assert.That(_actualRocket, Is.InstanceOf<Falcon9Rocket>()); // Tests whether the specified object is an instance of the expected type
Assert.That(_actualRocket, Is.Not.InstanceOf<Falcon9Rocket>()); // Tests whether the specified object is not an instance of type
Assert.That(_actualFuel, Is.GreaterThan(20)); // Tests whether the specified object greater than the specified value
```

Author Attribute

```
[TestFixture]
[Author("Joro Doev", "joro.doev@bellatrix.solutions")]
public class RocketFuelTests
{
 [Test]
 public void RocketFuelMeassuredCorrectly_When_Landing() { /* ... */ }
 [Test]
 [Author("Ivan Penchev")]
 public void RocketFuelMeassuredCorrectly_When_Flying() { /* ... */ }
}
```

Repeat Attribute

```
[Test]
[Repeat(10)]
public void RocketFuelMeassuredCorrectly_When_Flying() { /* ... */ }
```

Combinatorial Attribute

```
[Test, Combinatorial]
public void CorrectFuelMeassured_When_X_Site([Values(1,2,3)] int x, [Values("A","B")] string s)
{
  ...
}
```

Pairwise Attribute

```
[Test, Pairwise]
public void ValidateLandingSiteOfRover_When_GoingToMars
  ([Values("a", "b", "c")] string a, [Values("+", "-")] string b, [Values("x", "y")] string c)
{
  Debug.WriteLine("{0} {1} {2}", a, b, c);
}
```

Random Attribute

```
[Test]
public void GenerateRandomLandingSiteOnMoon([Values(1,2,3)] int x, [Random(-1.0,
1.0, 5)] double d)
{
  ...
}
```

Range Attribute

```
[Test]
public void CalculateJupiterBaseLandingPoint([Values(1,2,3)] int x, [Range(0.2,0.6)] double y)
{
  //...
}
```

Retry Attribute

```
[Test]
[Retry(3)]
public void CalculateJupiterBaseLandingPoint([Values(1,2,3)] int x, [Range(0.2,0.6)] double y)
{
  //...
}
```

Timeout Attribute

```
[Test, Timeout(2000)]
public void FireRocketToProximaCentauri()
{
  ...
}
```

Execute Tests in Parallel

```
[assembly: Parallelizable(ParallelScope.Fixtures)]
[assembly:LevelOfParallelism(3)]
```

```
[TestFixture]
[Parallelizable(ParallelScope.Fixtures)]
public class TestFalcon9EngineLevels
{
  // ...
}
```