IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

LAB01 – Git / Gestão do Código Fonte

Licenciatura em Engenharia Informática

Objetivos:

1. Compreender a necessidade de utilizar este tipo de ferramentas;
2. Familiarização com os principais comandos Git;
3. Perceber o conceito de *Branch* e quando utilizá-lo;
4. Software disponível para fazer essa gestão de uma forma mais "amigável".

Requisitos:

- Visual Studio 2017 com ***.Net Core***
  https://www.visualstudio.com/pt-br/downloads/
- Git BASH
  https://git-for-windows.github.io/
- GitHub Desktop
  https://desktop.github.com/

GIT

Hoje em dia o Git é o sistema de controlo de versões de código fonte mais utilizado mundialmente. É uma ferramenta *open source* muito madura e em constante atualização, destaca-se de outras como o SVN devido à sua performance, segurança e flexibilidade.

Pode ler mais sobre o Git aqui: https://www.atlassian.com/git/tutorials/what-is-git

GitHub

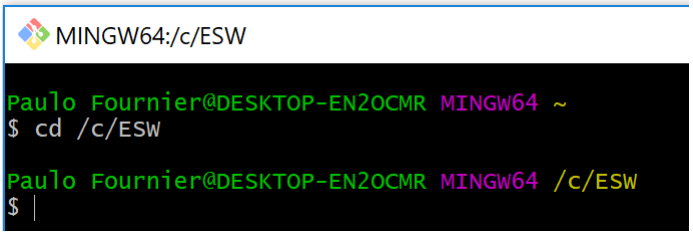
O GitHub é uma plataforma online com o claro foco no desenvolvimento de software e iremos utilizá-lo como ferramenta de gestão do código bem como o controlo de versões.

1. Crie uma conta na plataforma, caso ainda não possua:
   https://github.com/
2. Crie um novo repositório chamado "Hello World"

3. Escolha o tipo de Licença que mais se adequa ao seu projeto (ou escolha none): https://choosealicense.com/

Git BASH

1. Navegue para o diretório onde irá guardar o projeto



2. Faça o clone do repositório para uma pasta

   git clone [repositório] [destino]

```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW
$ git clone https://github.com/pfournier-est/hw.git hw
Cloning into 'hw'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 2), reused 7 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
```

3. Com o explorador do windows, navegue até ao respositório clonado e crie um novo ficheiro com o nome README sem qualquer extensão e verifique o estado do respositório.

   git status

```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README

nothing added to commit but untracked files present (use "git add" to track)
```

   Podemos ver que existe um novo ficheiro README marcado como ***Untracked.***

4. Adicione o ficheiro criado ao repositório e verifique novamente o estado

   git add [file_name]

```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git add README

Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README
```

5. Edite o ficheiro e escreva o seu número de aluno. Guarde as alterações e efetue commit.

   git commit -m "[comentário aqui]"

```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git commit -m "Adicionado ficheiro README"
[master 3f55835] Adicionado ficheiro README
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README
```

6. Neste momento o commit do projeto foi efetuado apenas localmente. Se veriricar o repositório no GitHub pode constantar que as alterações ainda não foram publicadas. Isso pode ser feito a qualquer momento utilizando o comando ***push*** mesmo após vários ***commits* locais**. Aliás, esta é uma das grandes vantagens a nível de performance pois podemos efetuar enumeros ***commits* locais** poupando assim os tempos de espera de comunicação com o servidor externo. Pode efetuar o ***push*** apenas no final do dia de trabalho.

   git push [destination] [branch]

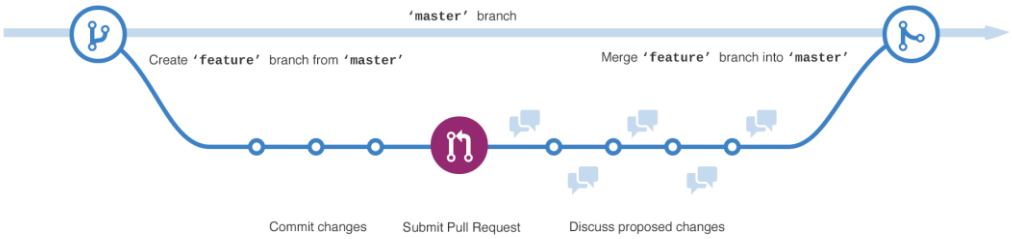
```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 335 bytes | 335.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/pfournier-est/hw.git
   f22eb22..3f55835  master -> master
```

   Neste caso fizémos o push do ***branch master*** para o destino ***origin*** que é o respositório no GitHub.

   As alterações foram publicadas no servidor.

7. Se reparar, este repositório tem apenas um *branch* sendo que estamos a trabalhar sobre o *branch* ***master***, o que não é boa prática.

   O Git permite ramificar o repositório em diferentes *branches* para que o código desenvolvido não seja aplicado diretamente sobre o *branch master*, que normalmente contém o código de produção. Assim, sempre que se desenvolve novo código, deverá ser num novo *branch* e depois, no caso dos testes correrem conforme o esperado, é efetuado o ***marge*** para o *branch* principal.



Crie um novo *branch* chamado "**dev**" e mude (***checkout***) para este *branch*.

git branch [nome_do_branch] [branch_alvo]

git checkout [branch]

git push [destino] [branch]

```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git branch dev master

Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git checkout dev
Switched to branch 'dev'
M       README

Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (dev)
$ git push origin dev
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/pfournier-est/hw.git
 * [new branch]      dev -> dev
```

Note que agora está a trabalhar sobre o branch (dev)

8. Altere o ficheiro README e acrescente uma nova linha com o seu nome e faça o *commit* do mesmo.

   git commit -am "[comentário aqui]"

9. Mude para o *branch master* e verifique o conteudo do ficheiro README.

   Neste momento as alterações efetuadas no *branch* dev não estão presentes na *branch* maser.

   Efetue o *merge* do *branch* dev no *master* e envie as alerações para o servidor.

   git merge [branch]

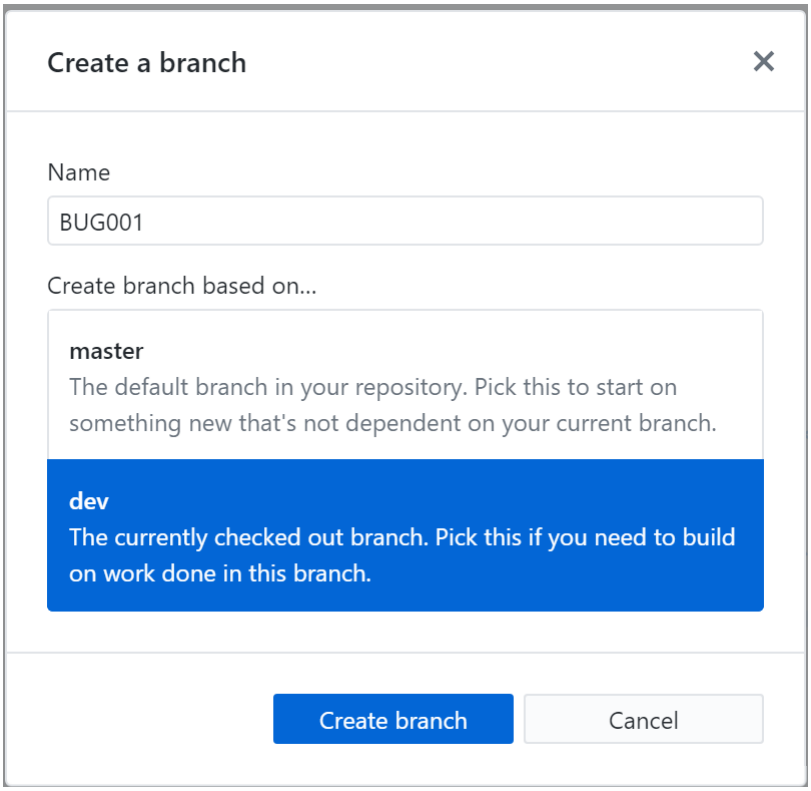
```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git merge dev
Updating 3f55835..ac602f2
Fast-forward
 README | 2 ++
 1 file changed, 2 insertions(+)
```

```
Paulo Fournier@DESKTOP-EN2OCMR MINGW64 /c/ESW/hw (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 347 bytes | 347.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/pfournier-est/hw.git
   3f55835..ac602f2  master -> master
```

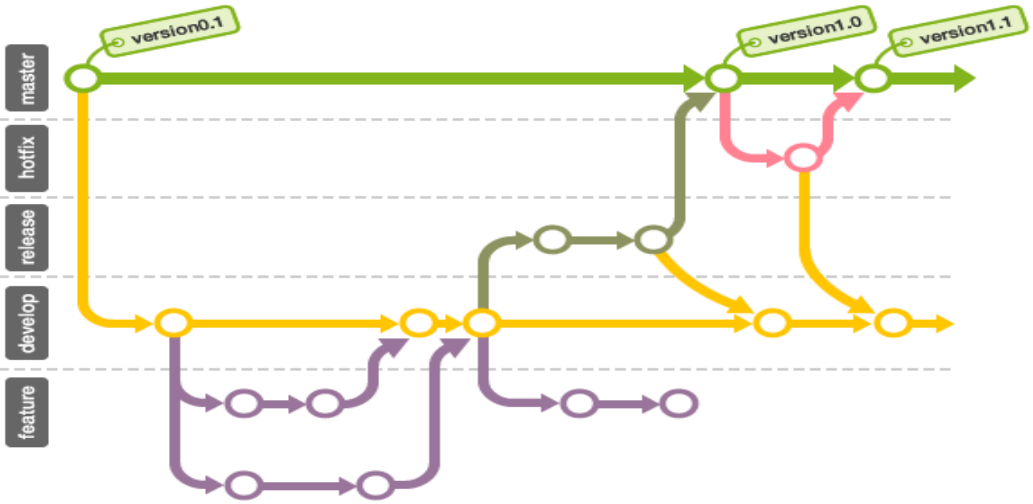
GitHub Desktop

O GitHub Desktop é uma ferramenta desenvolvida para facilitar a gestão do seu código através de janelas em vez de utilizar a linha de comandos. Existem várias ferramentas para este fim sendo que o TortoiseGit em conjunto com esta serão as mais utilizadas no mercado.

1. Faça o clone do repositório “Hello World”.
2. Mude para o *branch* **dev** e crie um novo *branch* sobre este chamado **BUG001** e faça o *publish* do mesmo.



Foi criada uma nova ramificação em cima da anterior. Na verdade podemos criar tantas quantas as necessárias.



3. Altere o ficheiro README e acrescente o seu endereço de e-mail.
4. Faça o commit dessas alterações e efetue o *merge* sucessivamente até ao *branch master*.
5. Publique as alterações efetuadas.

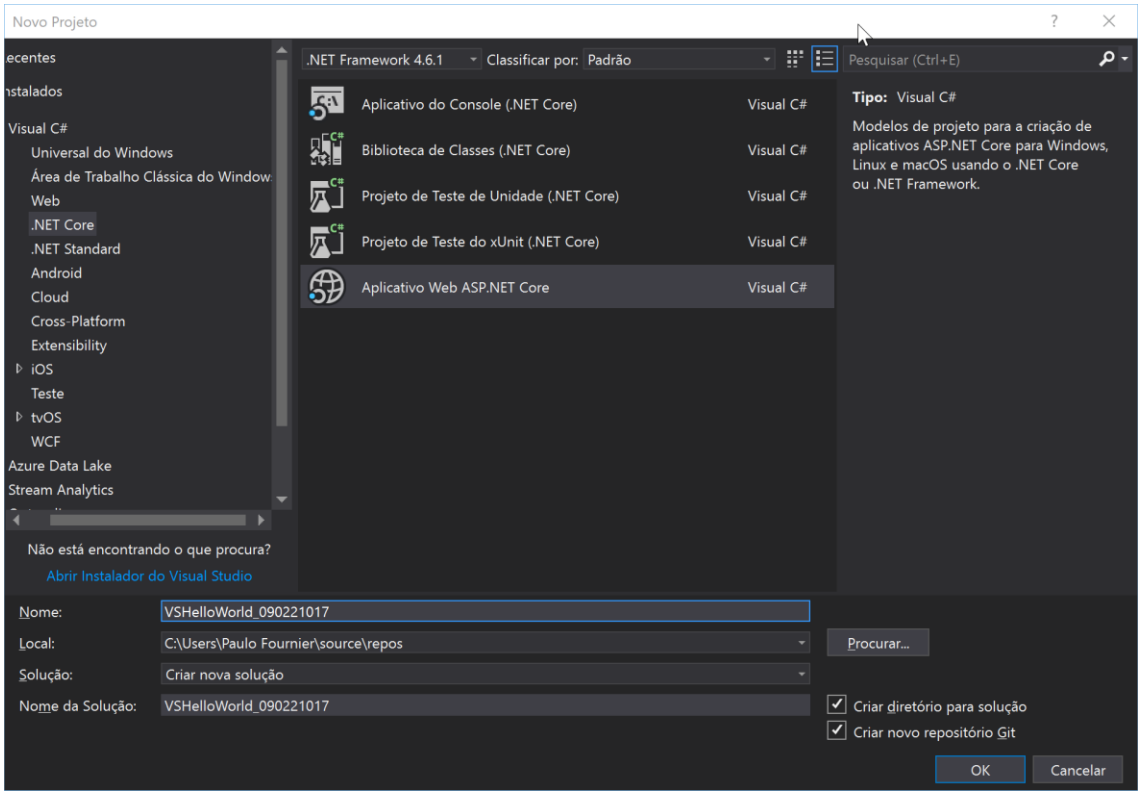
Git no Visual Studio

O Visual Studio contém uma ferramenta chamada "Team Explorer" que permite efetuar a gestão do código fonte diretamente no IDE, sem ter de utilizar linhas de comando ou outro software como o TortoiseGit.

Consulte este tutorial sobre o Git no Visual Studio:

https://docs.microsoft.com/en-us/vsts/git/gitquickstart?tabs=visual-studio

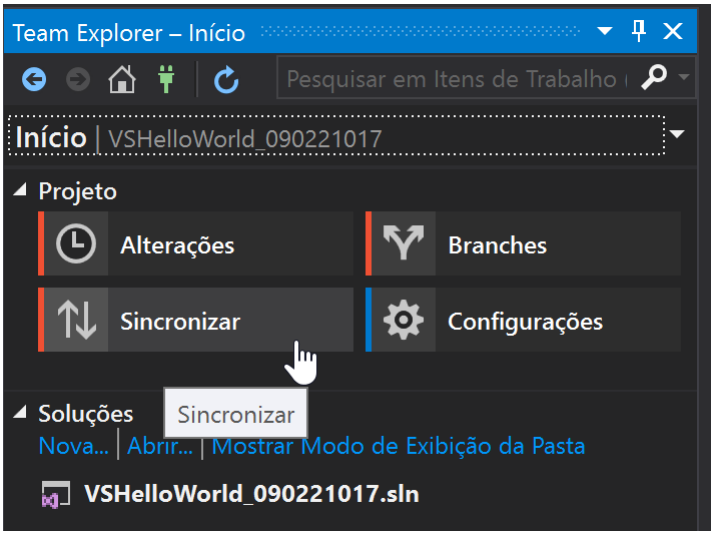
1. Crie um novo projeto **ASP.Net Core** (*API Web*) chamado "**VSHelloWorld_[NUMERO_ALUNO]**".



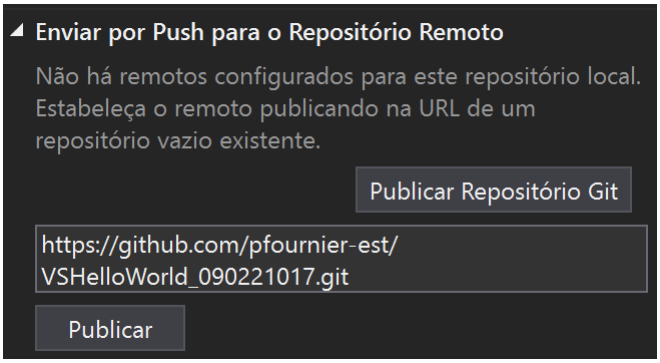
2. Crie um repositório no GitHub com o mesmo nome.

3. Publique este projeto no GitHub seguindo os seguintes passos:
    a. No separador Team Explorer abra o seu projeto.

    b. Entre em Sincronizar.



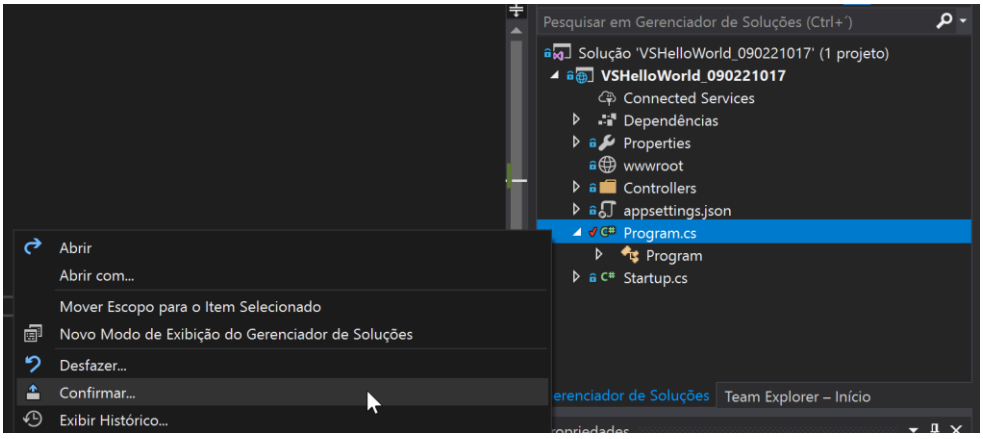
    c. E coloque o endereço do repositório do GitHub e Publique.

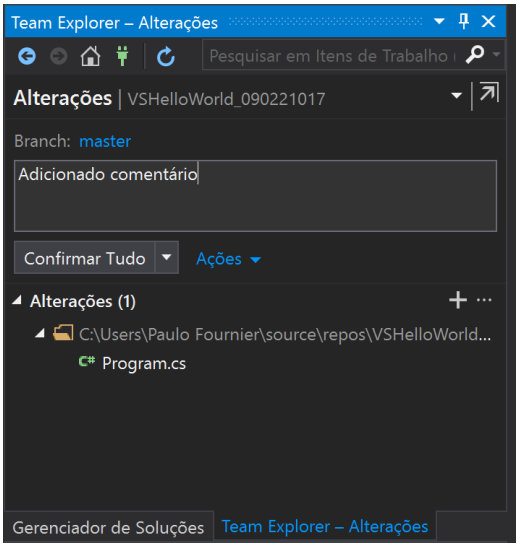


4. Explore o projeto e edite o ficheiro Program.cs colocando um comentário de teste.

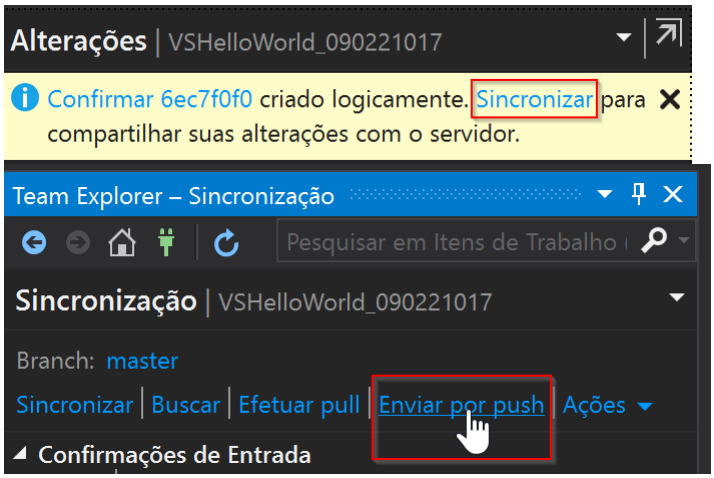
```
namespace VSHelloWorld_090221017
{
    public class Program
    {
        public static void Main(string[] args)
        {
            //Comentário de Teste
            var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                .UseApplicationInsights()
                .Build();

            host.Run();
        }
    }
}
```

5. Efetue o commit da alteração diretamente no explorador.

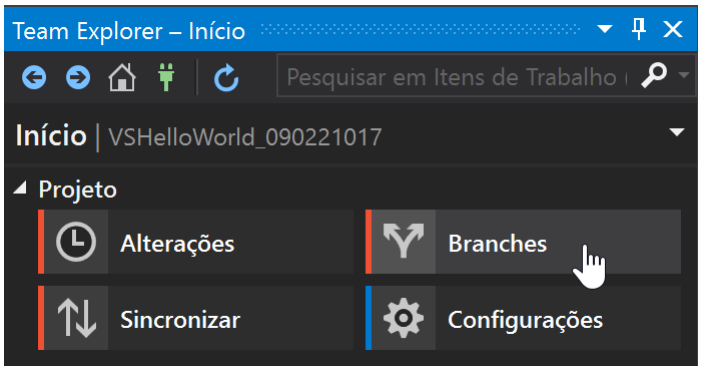


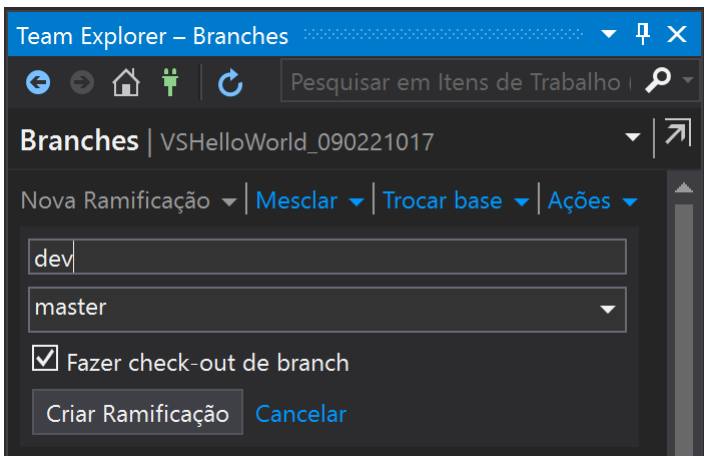


6. Efetue o push, sincronizando o projeto.

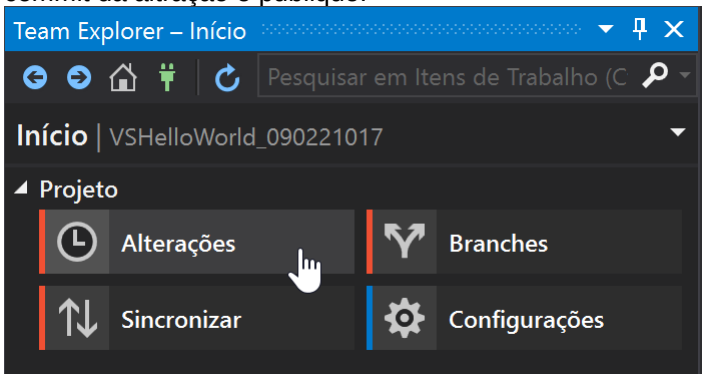


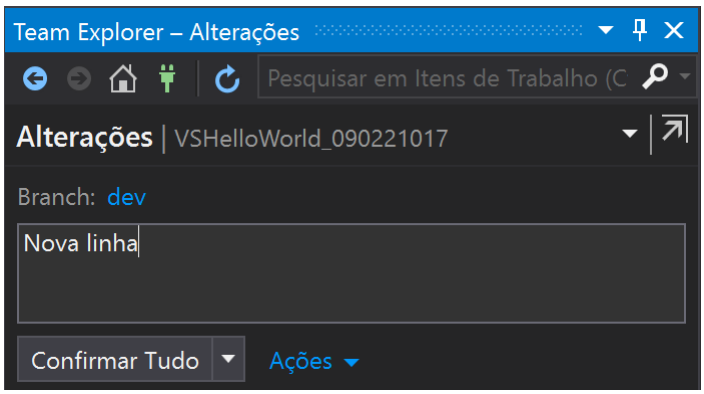
7. Crie um novo *branch* **dev**.

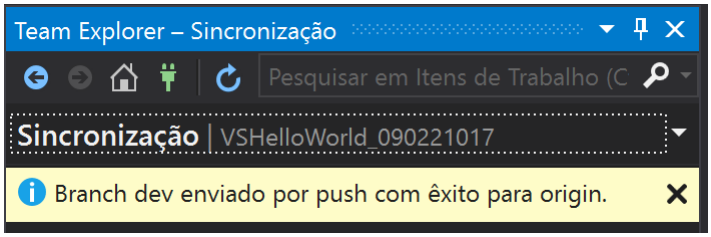




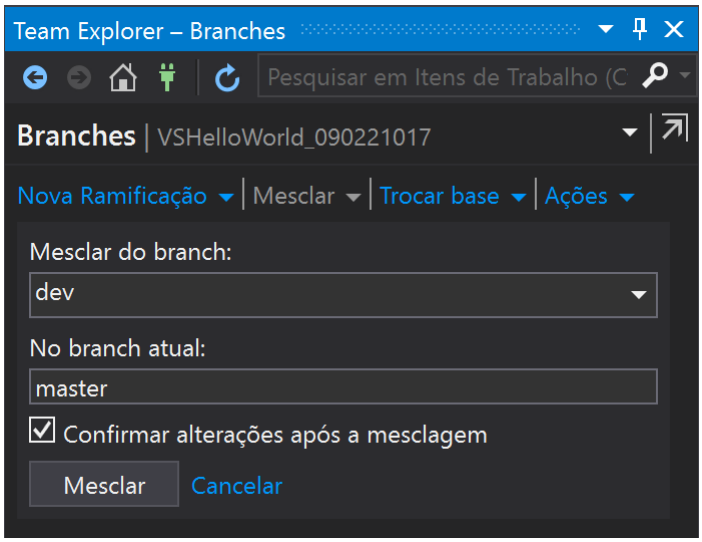
8. Neste novo *branch*, edite o mesmo ficheiro colocando uma nova linha de comentário, faça commit da altração e publique.

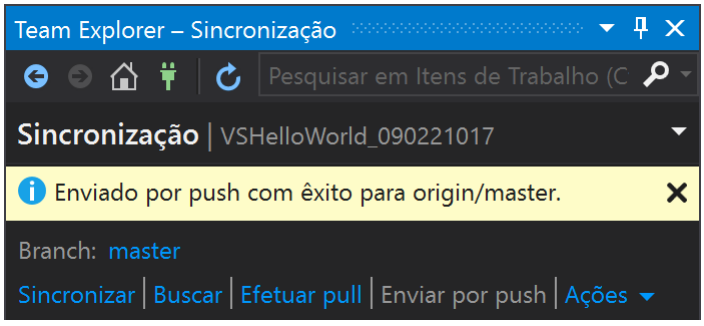






9. Mude para o *branch* master e confirme que o ficheiro voltou ao seu estado original. Efetue o *merge* do *branch dev* e publique a alteração.



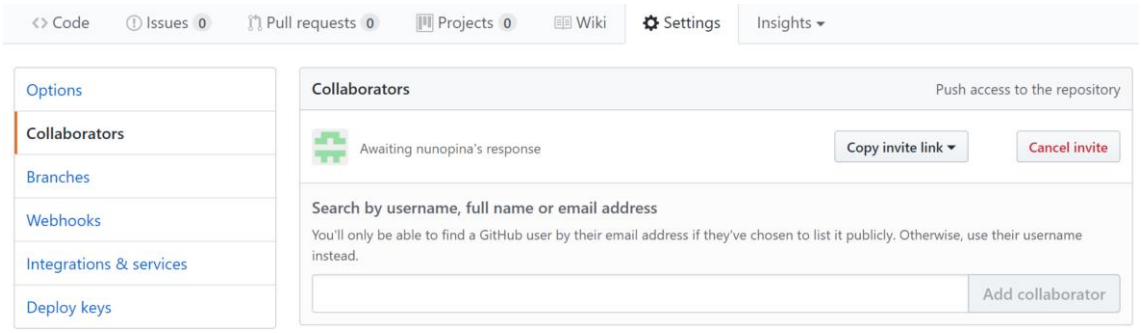


GitHub Collaborators

Até agora vimos exemplos de projetos apenas com um elemento a desenvolver. Escolha um elemento do laboratório para colaborarem em conjunto.

Para que o projeto possa ser editado, o dono do mesmo terá de convidar colaboradores para que possam ter acesso ao respetivo código, editá-lo e publicá-lo.

1. Convide o outro elemento do grupo.
   Este terá de confirmar o convite no seu endereço de e-mail.

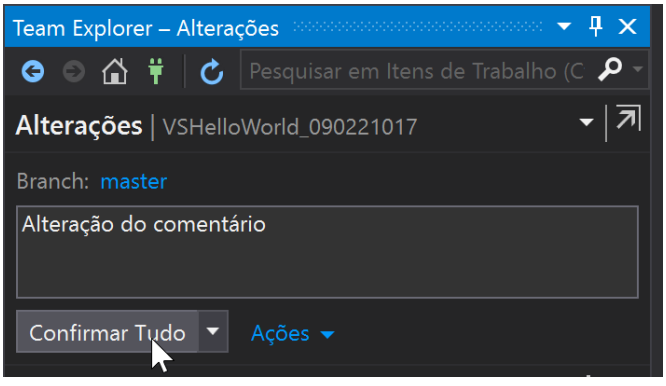


2. O elemento convidado terá de efetuar o clone do projeto (passo 3 do ponto anterior).
3. Selecionem o *branch* master.
4. Abram o ficheiro Program.cs e cada um vai remover os comentários acrescentados anteriormente e adicionar um novo comentário com o seu nome escrito.

```
public static void Main(string[] args)
{
    //Comentário de Teste
    //Nova linha
    var host = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseIISIntegration()
        .UseStartup<Startup>()
        .UseApplicationInsights()
        .Build();

    host.Run();
}
```

```
public static void Main(string[] args)
{
    //Paulo Fournier
    var host = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseIISIntegration()
        .UseStartup<Startup>()
        .UseApplicationInsights()
        .Build();

    host.Run();
}
```

5. Efetuem o *commit* da alteração e de seguida o *push* do mesmo.



6. O que aconteceu? Uma vez que ambos os elementos editaram a mesma linha de código, o Git não sabe qual a que prevalece. Nestas situações terá de recorrer à ferramenta de gestão de conflitos, escolhendo qual o código correto e/ou efetuar as alterações necessários. No final será necessário um novo commit “especial” para corrigir este conflito.