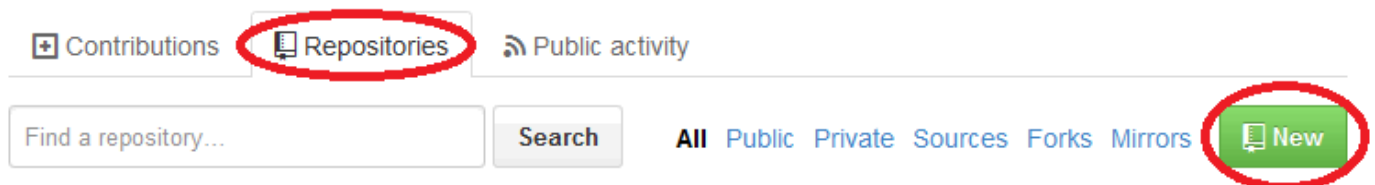



از نگارش 2012 ویژوال استودیو، امکان کار با مخازن Git، به صورت یکپارچه و توکار و بدون نیاز به ابزارهای جانبی، توسط آن فراهم شده‌است. در ادامه قصد داریم به کمک این ویژگی توکار، نحوه‌ی ارسال یک پروژه‌ی از پیش موجود VS.NET را برای اولین بار به GitHub بررسی کنیم.

تنظیمات مقدماتی GitHub

در ابتدا نیاز است یک مخزن کد خالی را در GitHub ایجاد کنید. برای این منظور به برگه‌ی Repositories در اکانت GitHub خود مراجعه کرده و بر روی دکمه‌ی New کلیک کنید:



سپس در صفحه‌ی بعدی، نام پروژه را به همراه توضیحاتی وارد نمائید و بر روی دکمه‌ی Create repository کلیک کنید. در اینجا سایر گزینه‌ها را انتخاب نکنید. نیازی به انتخاب گزینه‌ی READ ME و یا انتخاب مجوز و غیره نیست. تمام این کارها را در سمت پروژه‌ی اصلی می‌توان انجام داد و یا VS.NET فایل‌های ignore را به صورت خودکار ایجاد می‌کند. در اینجا صرفاً هدف، ایجاد یک مخزن کد خالی است.

Owner  VahidN / **Repository name**

Great repository names are short and memorable. Need inspiration? How about [turnt-octo-bugfixes](#).

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.


☐ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately. Skip this step if you have already run `git init` locally.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

از اطلاعات صفحه‌ی بعدی، تنها به آدرس مخصوص GitHub آن نیاز داریم. از این آدرس در VS.NET برای ارسال اطلاعات به سرور استفاده خواهیم کرد:

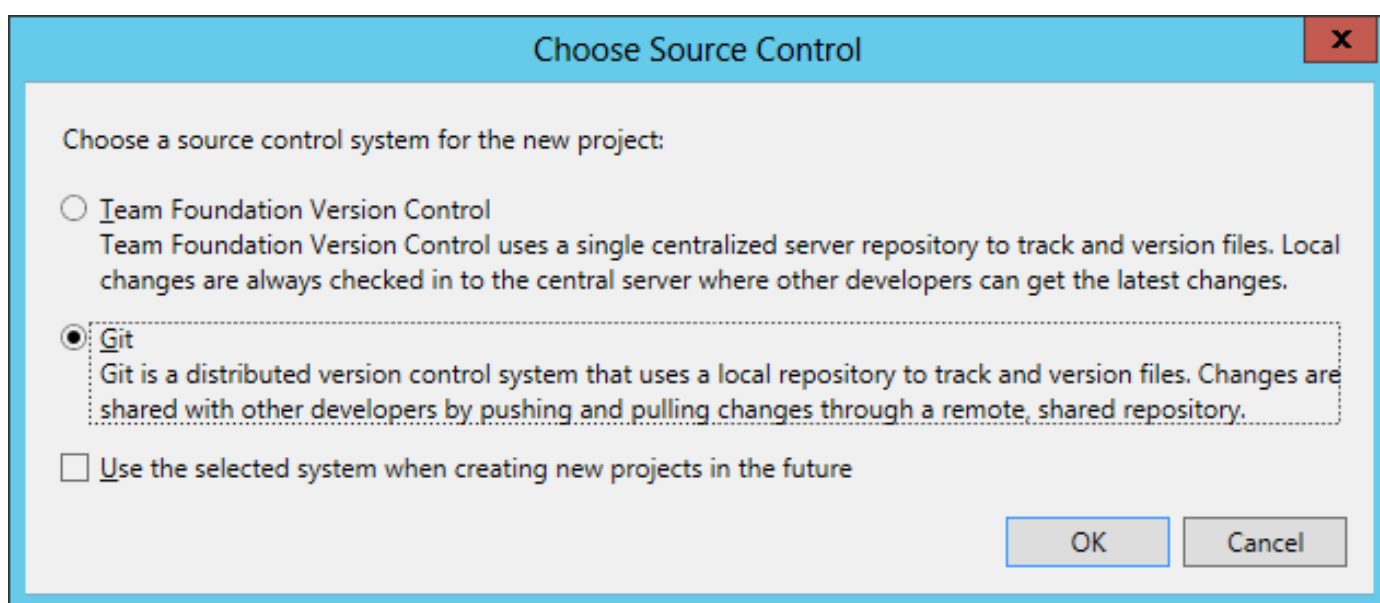
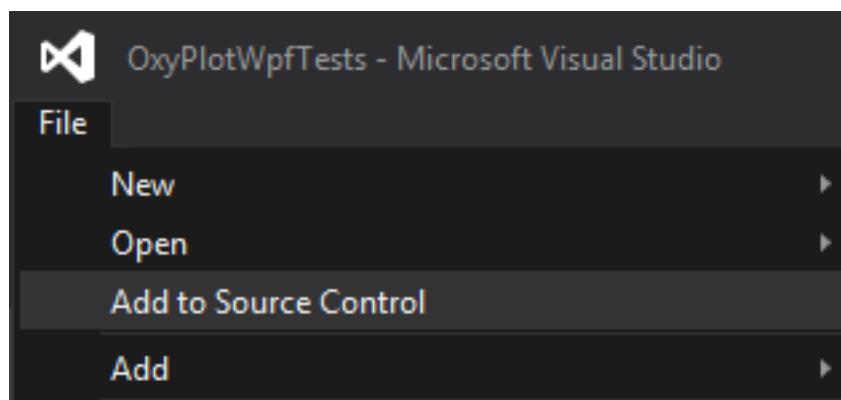
Quick setup — if you've done this kind of thing before

 **Set up in Desktop** or **HTTPS** **SSH**

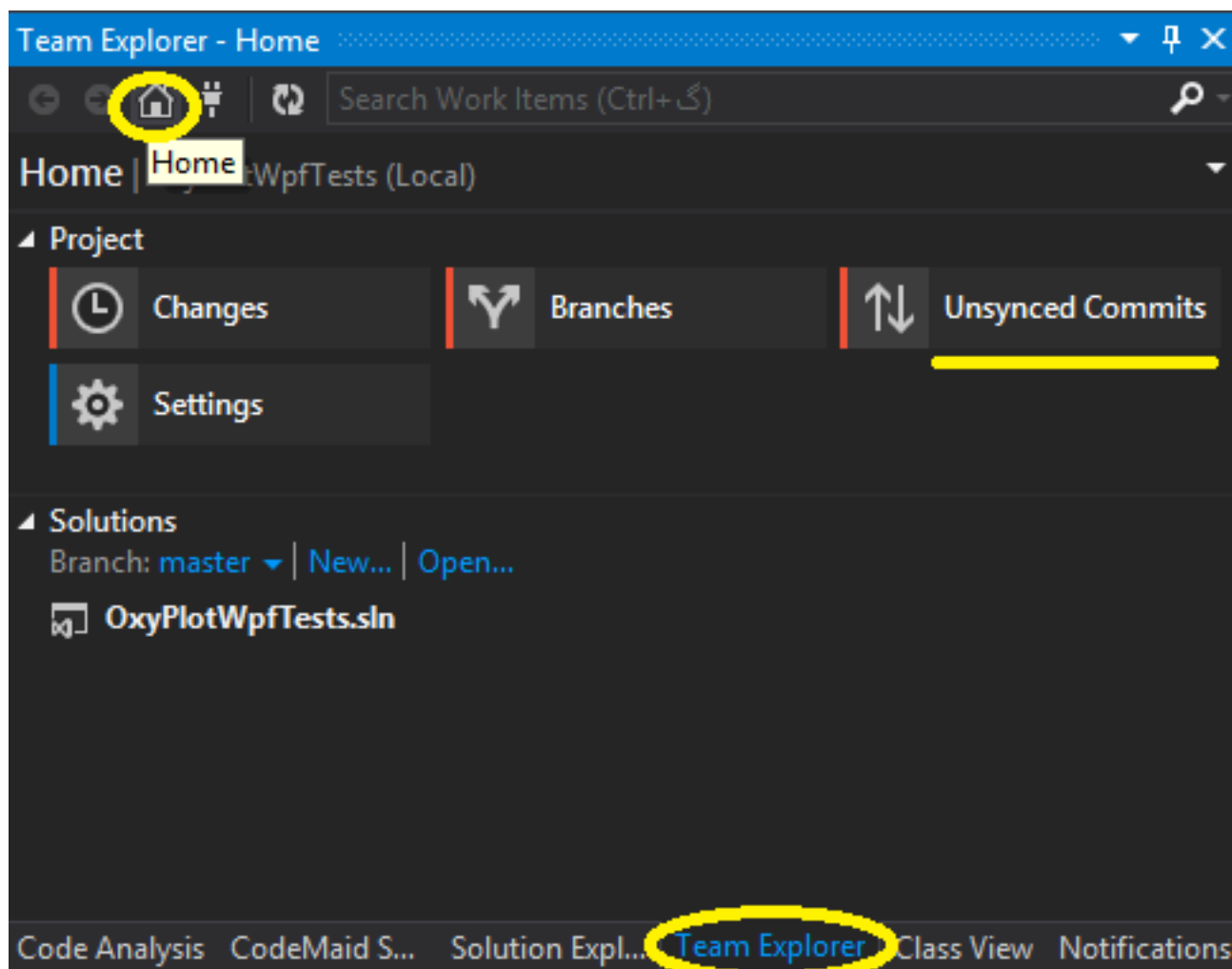
We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

تنظیمات VS.NET برای ارسال پروژه به مخزن GitHub

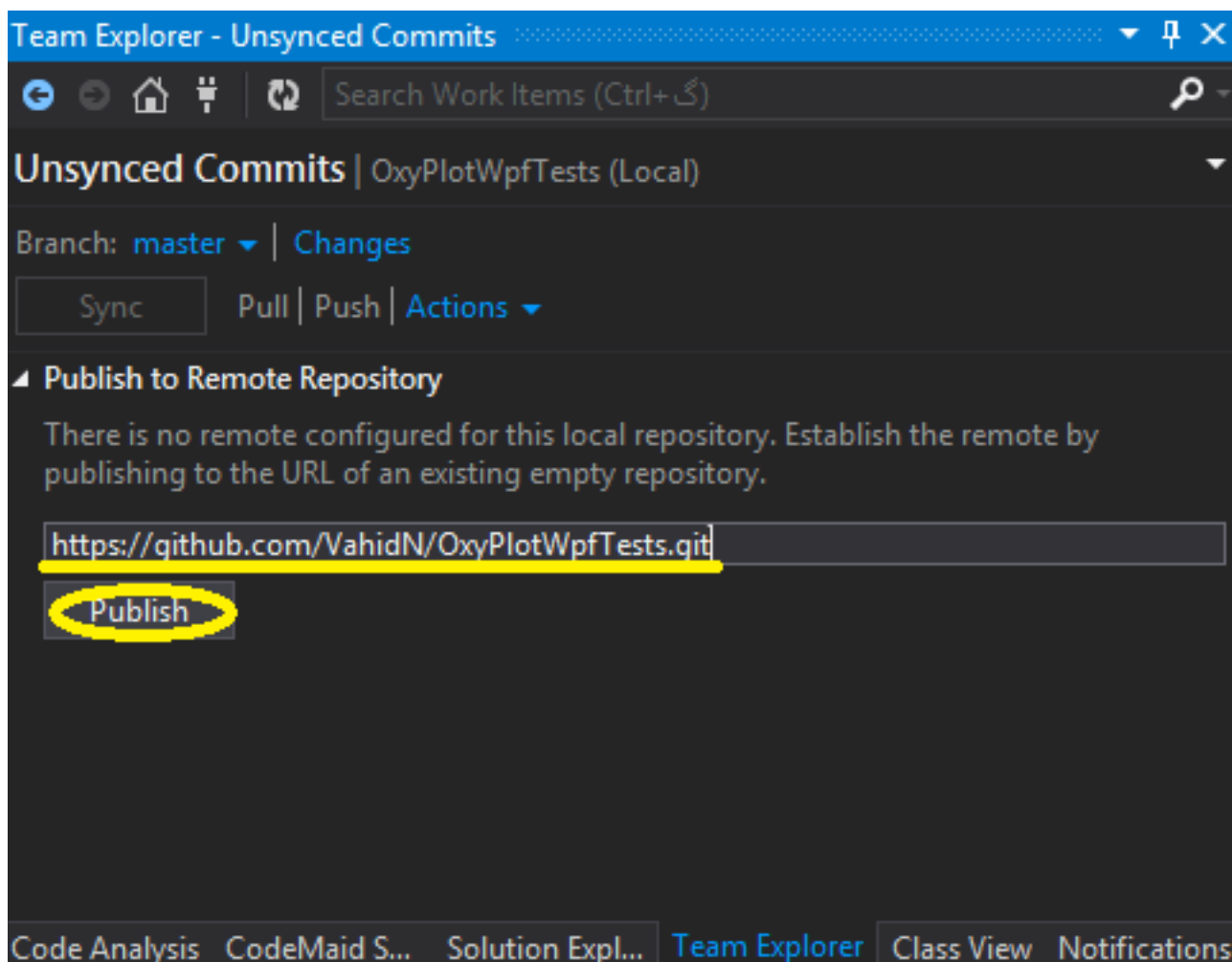
پس از ایجاد یک مخزن کد خالی در GitHub، اکنون می‌توانیم پروژه‌ی خود را به آن ارسال کنیم. برای این منظور از منوی File، گزینه‌ی Add to source control را انتخاب کنید و در صفحه‌ی باز شده، گزینه‌ی Git را انتخاب نمایید:



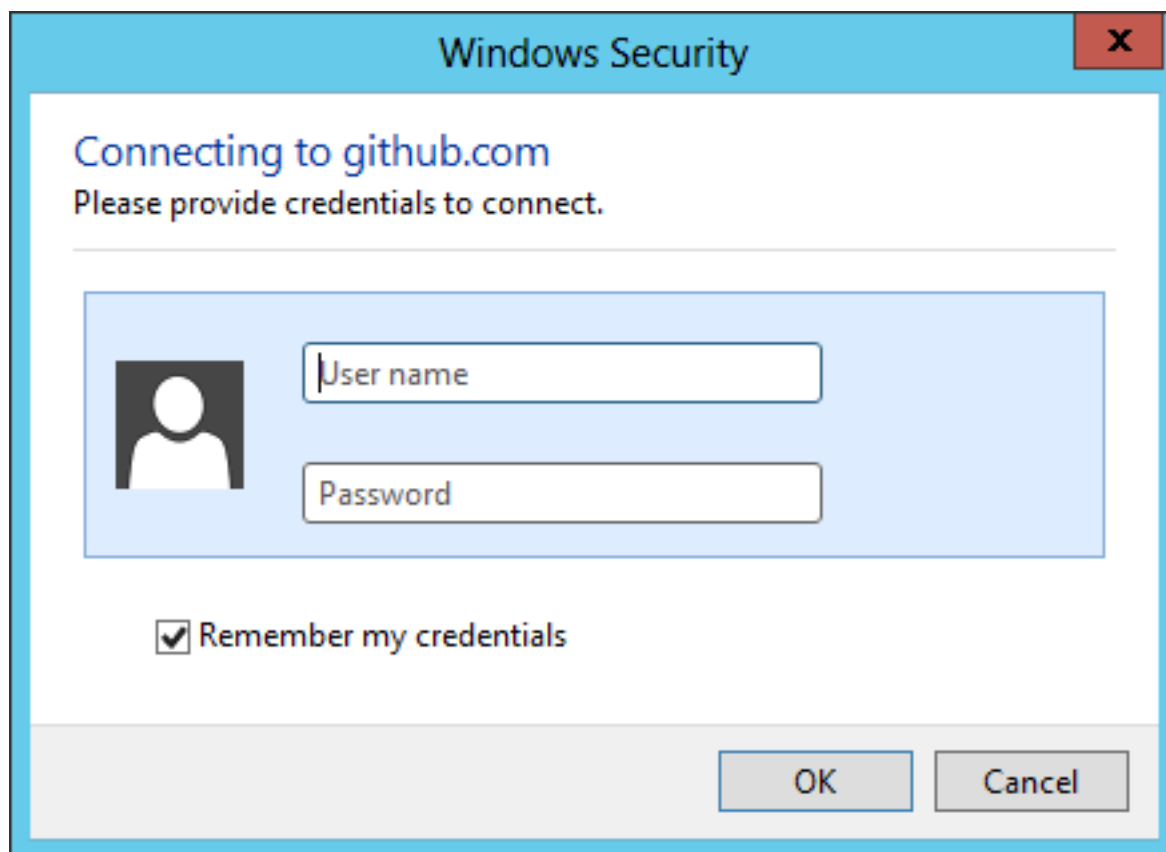
سپس در کنار برگه‌ی Solution Explorer، برگه‌ی Team Explorer را انتخاب کنید. در اینجا بر روی دکمه‌ی Home در نوار ابزار آن کلیک کرده و سپس بر روی دکمه‌ی Unsynced commits کلیک نمایید.



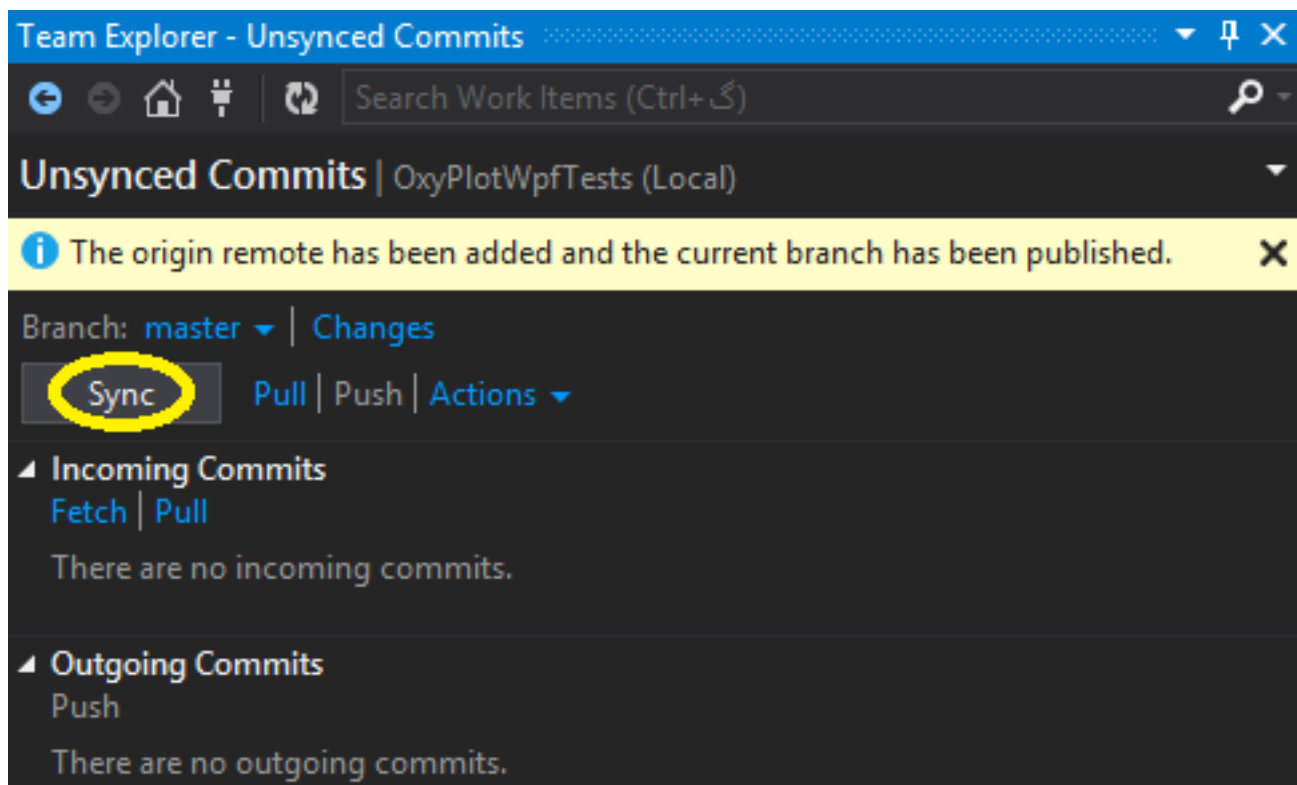
در ادامه در صفحه‌ی باز شده، همان آدرس مخصوص مخزن کد جدید را در GitHub وارد کرده و بر روی دکمه‌ی Publish کلیک کنید:



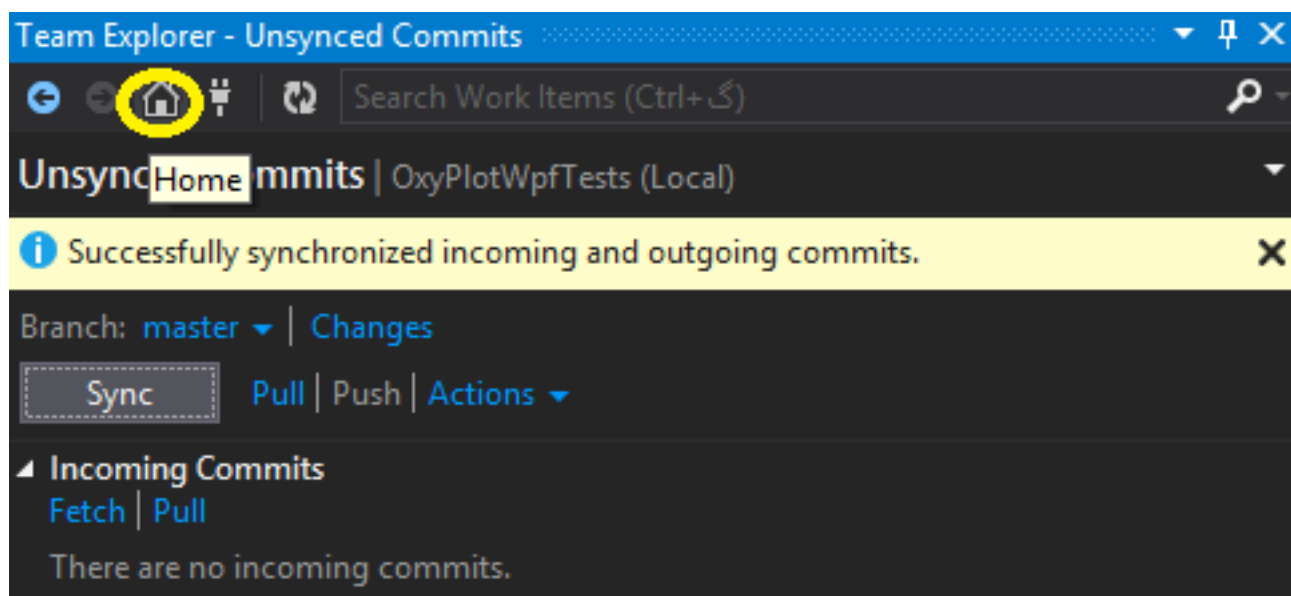
در اینجا بلافاصله صفحه‌ی لاگینی ظاهر می‌شود که باید در آن مشخصات اکانت GitHub خود را وارد نمایید:



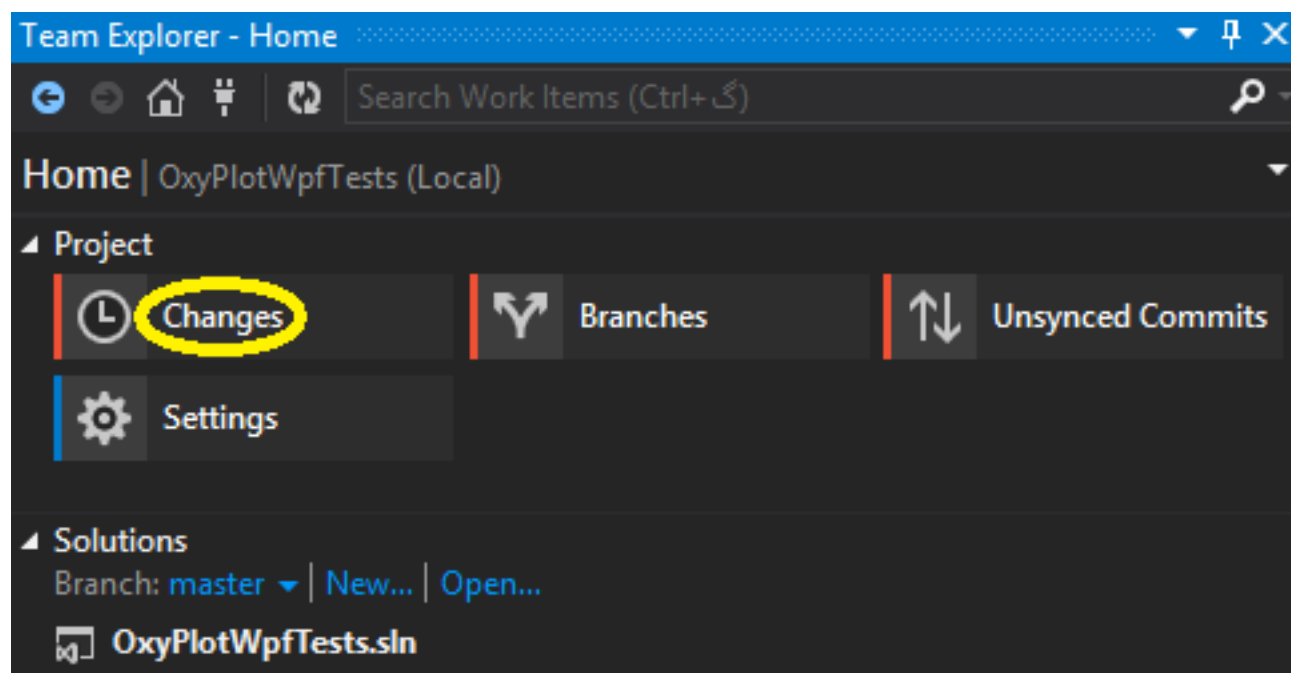
به این ترتیب عملیات Publish اولیه انجام شده و تصویر ذیل نمایان خواهد شد:



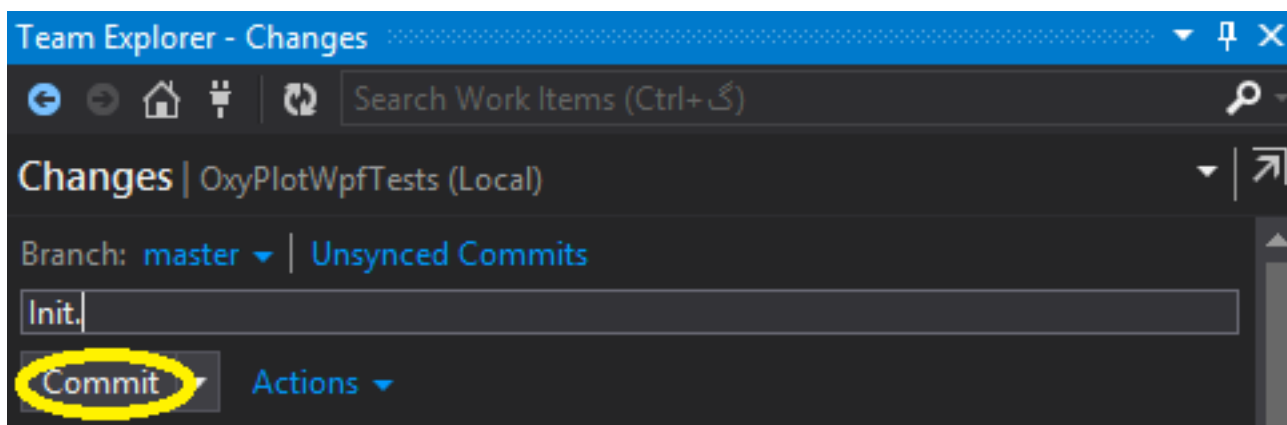
در اینجا بر روی دکمه‌ی Sync کلیک کنید. به این ترتیب مخزن کد GitHub به پروژه‌ی جاری متصل خواهد شد:



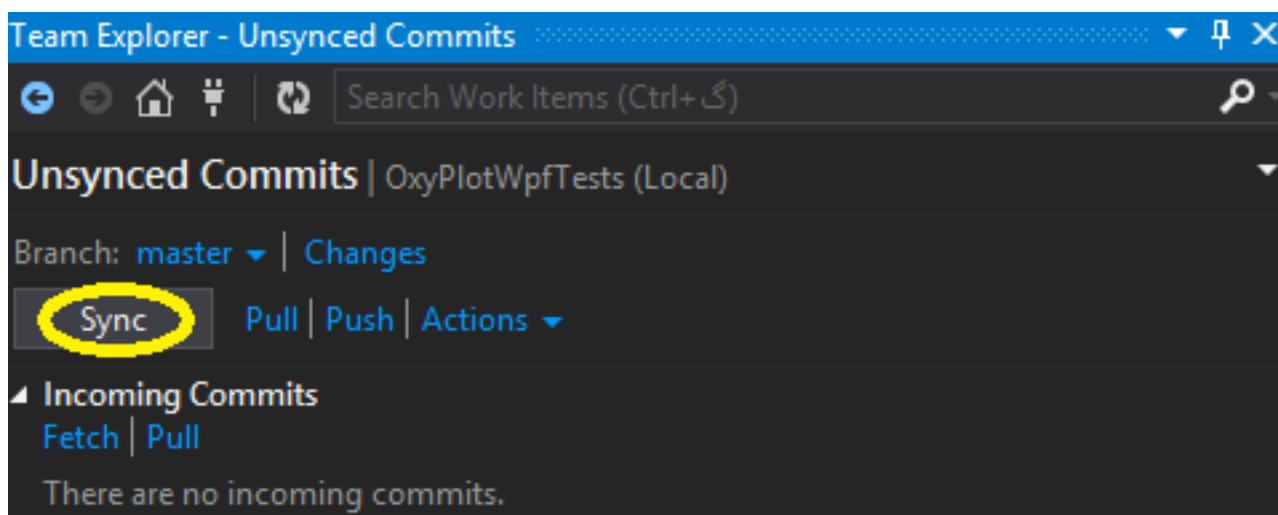
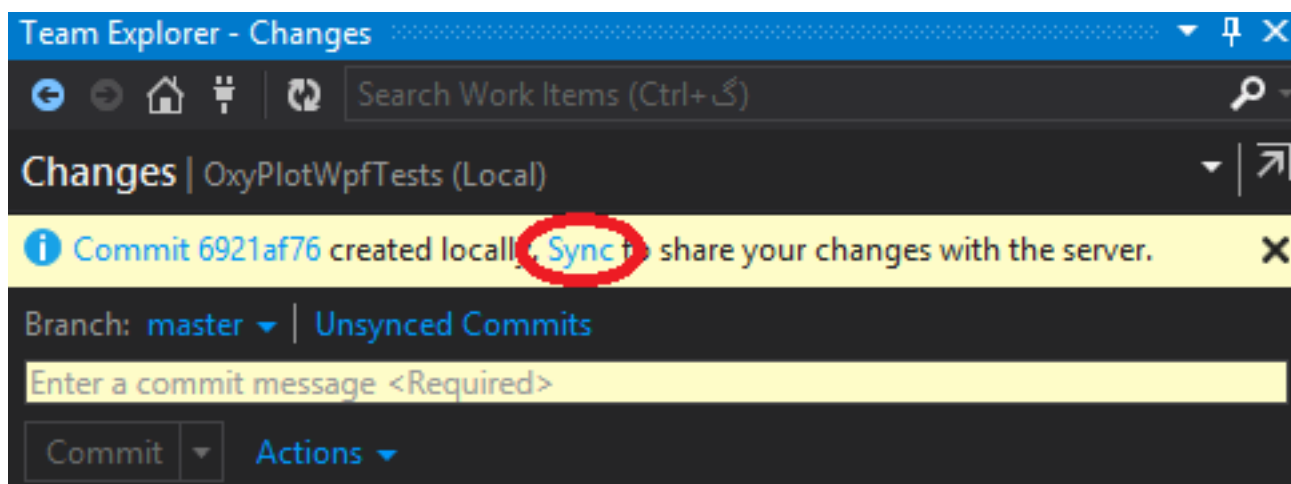
سپس نیاز است فایل‌های موجود را به مخزن کد GitHub ارسال کرد. بنابراین پس از مشاهده‌ی پیام موفقیت آمیز بودن عملیات همگام‌سازی، بر روی دکمه‌ی Home در نوار ابزار کلیک کرده و اینبار گزینه‌ی Changes را انتخاب کنید:



در اینجا پیام اولین ارسال را وارد کرده و سپس بر روی دکمه‌ی Commit کلیک کنید:



پس از مشاهده‌ی پیام موفقیت آمیز بودن commit محلی، نیاز است تا آنرا با سرور نیز هماهنگ کرد. به همین جهت در اینجا بر روی لینک Sync کلیک کرده و در صفحه‌ی بعدی بر روی دکمه‌ی Sync کلیک کنید:



اندکی صبر کنید تا فایل‌ها به سرور ارسال شوند. اکنون اگر به GitHub مراجعه کنید، فایل‌های ارسالی قابل مشاهده هستند:

An OxyPlot Sample — Edit

2 commits 1 branch 0 releases 1 contributor

branch: master OxyPlotWpfTests / +

Init.

VahidN authored 3 minutes ago latest commit 6921af7608

OxyPlotWpfTests	Init.	3 minutes ago
.gitattributes	Initial commit to add default .gitignore and .gitAttribute files.	18 minutes ago
.gitignore	Initial commit to add default .gitignore and .gitAttribute files.	18 minutes ago
OxyPlotWpfTests.sln	Init.	3 minutes ago

We recommend adding a README to this repository to help give people an overview of your project. Add a README

اعمال تغییرات بر روی پروژه‌ی محلی و ارسال به سرور

در ادامه می‌خواهیم دو فایل README.md و LICENSE.md را به پروژه اضافه کنیم. پس از افزودن آن‌ها، یا هر تغییر دیگری در پروژه، اینبار برای ارسال تغییرات به سرور، تنها کافی است به برگه‌ی Team explorer مراجعه کرده و ابتدا بر روی دکمه‌ی Home کلیک کرد تا منوی انتخاب گزینه‌های آن ظاهر شود. در اینجا تنها کافی است گزینه‌ی Changes را انتخاب و دقیقاً همان مراحل عنوان شده‌ی پیشین را تکرار کرد. ابتدا ورود پیام Commit و سپس Commit. در ادامه Sync محلی و سپس Sync با سرور.

نظرات خوانندگان

نویسنده: سیروس
تاریخ: ۱۸:۴۵ ۱۳۹۳/۱۰/۲۵

میخواستم بدونم برای پروژه‌های که نمیخواهیم کد اون در دسترس عموم قرار بگیره مانند پروژه‌های شرکت‌های برنامه نویسی، آیا Github قابل استفاده و اطمینان هست؟ و همینکه مخزن ما بصورت خصوصی باشه، کافیه؟

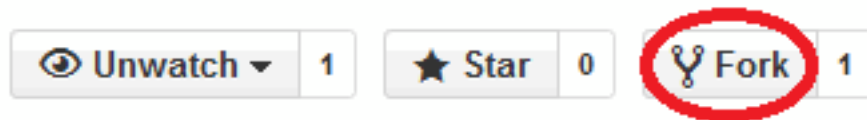
نویسنده: وحید نصیری
تاریخ: ۱۸:۵۷ ۱۳۹۳/۱۰/۲۵

GitHub امکان تهیه مخزن کد خصوصی هم دارد ولی [رایگان نیست](#) . سایت [BitBucket](#) امکان ایجاد مخزن کد خصوصی رایگان را دارد؛ البته با محدودیت حداکثر 5 کاربر تعریف شده‌ی برای کار با یک مخزن.

فرض کنید برای رفع باگی در پروژه‌ای از GitHub، ایده‌ای دارید. روند کاری اعلام آن، روش‌های مختلفی می‌تواند داشته باشند؛ از باز کردن یک Issue جدید تا فرستادن یک فایل zip و غیره. اما روش استاندارد مشارکت در پروژه‌های Git، ارسال یک PR یا Pull Request است. در ادامه نحوه‌ی انجام این کار را به کمک امکانات توکار VS.NET بررسی خواهیم کرد.

ایجاد یک Fork جدید در GitHub

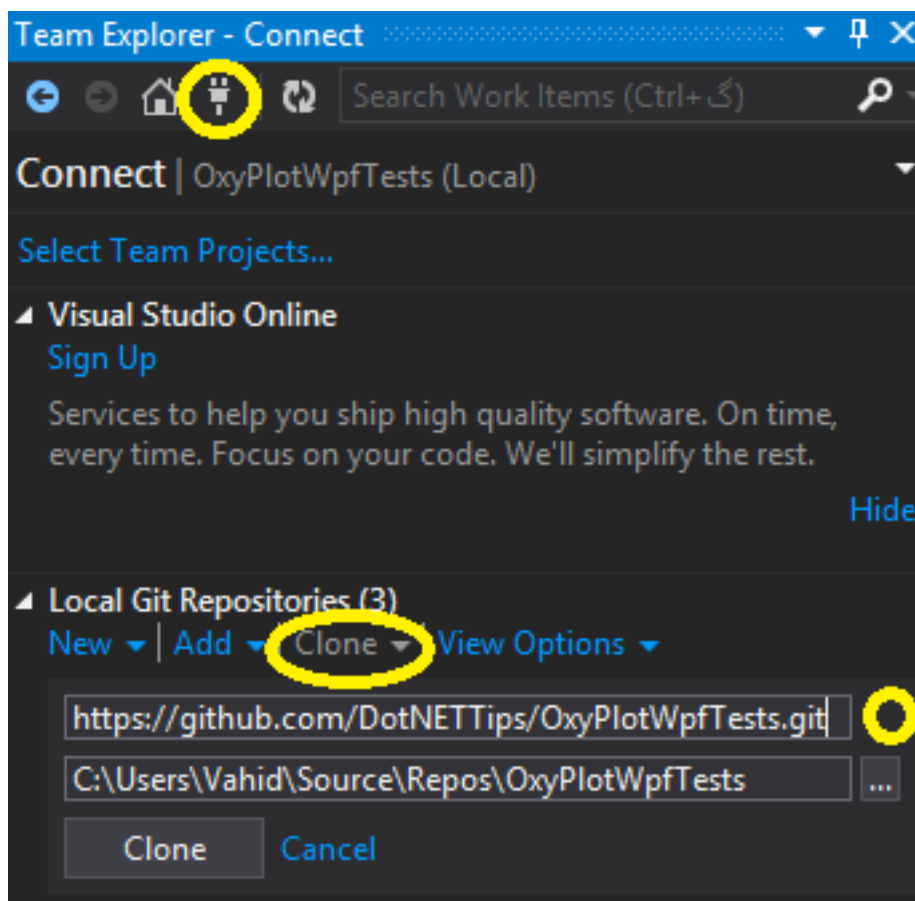
برای ارسال تغییرات انجام شده بر روی یک پروژه، نیاز است به صاحب یا مسئول آن مخزن در GitHub مراجعه و سپس درخواست دسترسی اعمال تغییرات را نمود. در این حالت، احتمال اینکه جواب منفی دریافت کنید، بسیار زیاد است. جهت مدیریت یک چنین مواردی، قابلیت به نام ایجاد یک Fork پیش بینی شده است.



در بالای هر مخزن کد در GitHub، یک دکمه به نام Fork موجود است. بر روی آن که کلیک کنید، یک کپی از آن پروژه را به مجموعه‌ی مخزن‌های کد شما در GitHub اضافه می‌کند. بدیهی است در این حالت، مجوز ارسال تغییرات خود را به GitHub و در اکانت خود خواهید داشت. نحوه‌ی اطلاع رسانی این تغییرات به صاحب اصلی این مخزن کد، ارسال همان PR یا Pull Request است.

دریافت مخزن کد Fork شده از GitHub به کمک Visual Studio

پس از اینکه Fork جدیدی را از پروژه‌ای موجود ایجاد کردیم، نیاز است یک Clone یا کپی مطابق اصل آن را جهت اعمال تغییرات محلی، تهیه کنیم. برای اینکار VS.NET را گشوده و به برگه‌ی Team Explorer آن که در کنار Solution Explorer قرار دارد، مراجعه کنید.



در اینجا بر روی دکمه‌ی Connect در نوار ابزار آن، کلیک کرده و در صفحه‌ی باز شده، بر روی لینک Clone کلیک نمایید. در اینجا می‌توان آدرس مخزن کد Fork شده را جهت تهیه یک Clone مشخص کرد؛ به همراه محلی که قرار است این Clone در آن ذخیره شود.

آدرس HTTPS وارد شده، در کنار تمام مخازن کد GitHub قابل مشاهده هستند:

HTTPS clone URL

`https://github.com/DotNET`

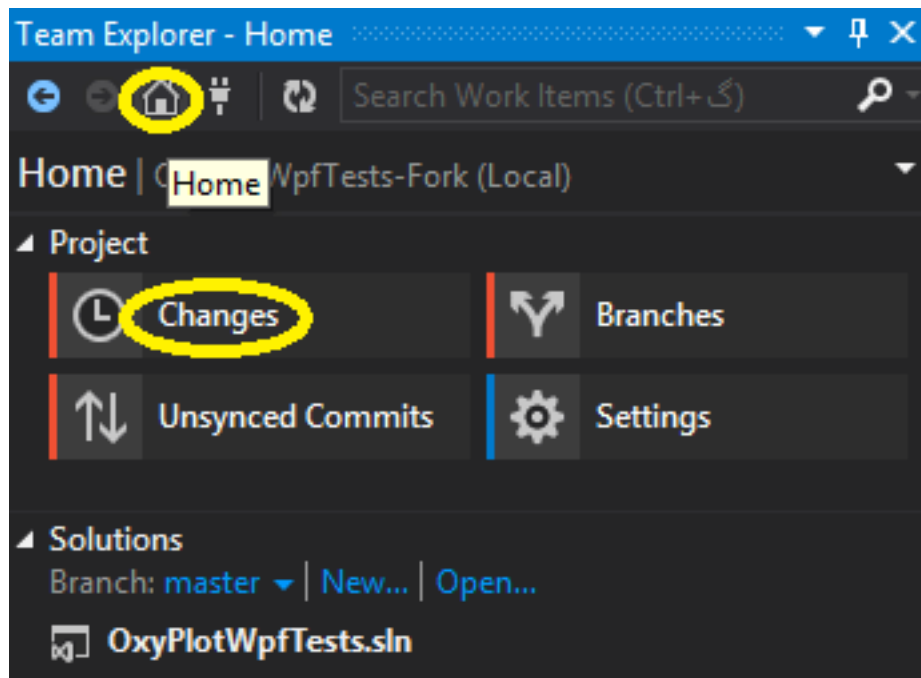
You can clone with HTTPS, SSH,
or Subversion. ?

پس از تکمیل این دو آدرس، بر روی دکمه‌ی Clone کلیک نمایید. پس از پایان کار، اگر به آدرس محلی داده شده بر روی کامپیوتر خود مراجعه کنید، یک کپی از فایل‌های این مخزن، قابل مشاهده هستند.

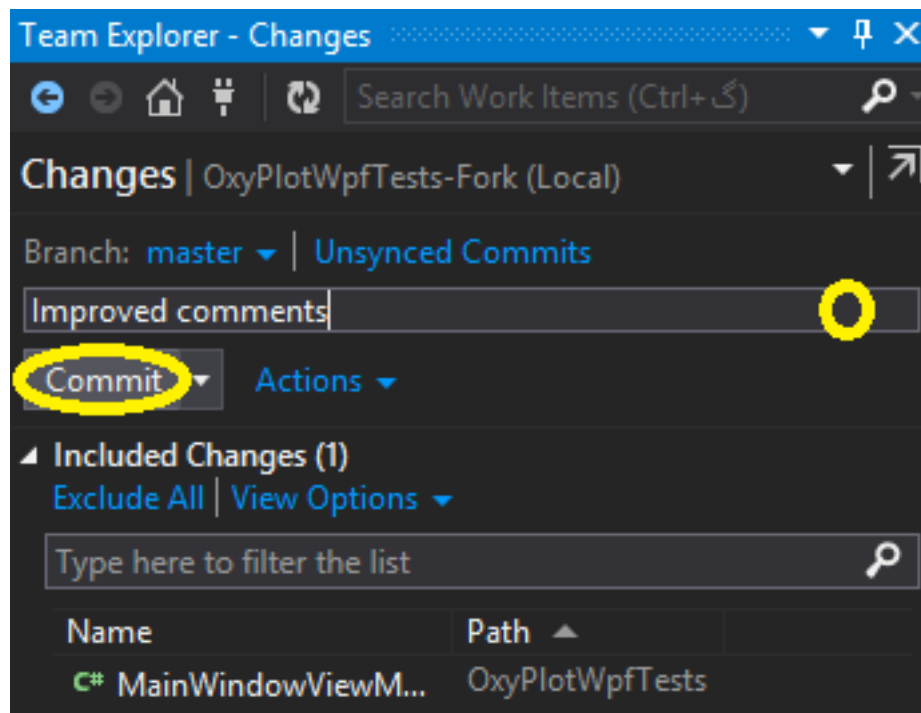
اعمال تغییرات محلی و ارسال آن به سرور GitHub

در ادامه، این پروژه‌ی جدید را در VS.NET باز کرده و تغییرات خود را اعمال کنید. اکنون نوبت به ارسال این تغییرات به سرور GitHub است. برای این منظور به برگه‌ی Team Explorer مراجعه کرده و بر روی دکمه‌ی Home آن کلیک کنید. سپس گزینه‌ی

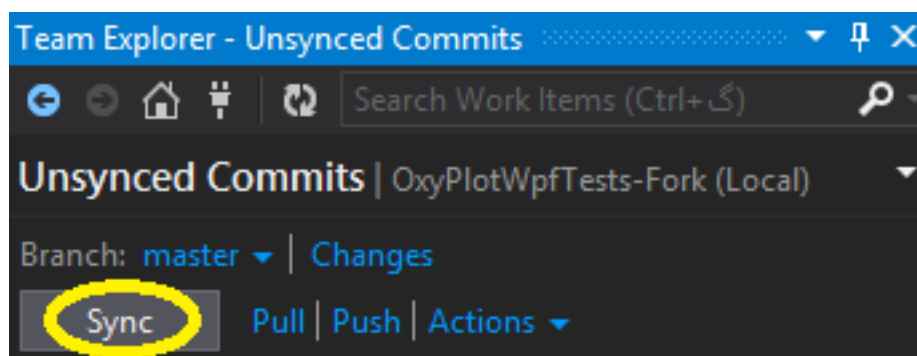
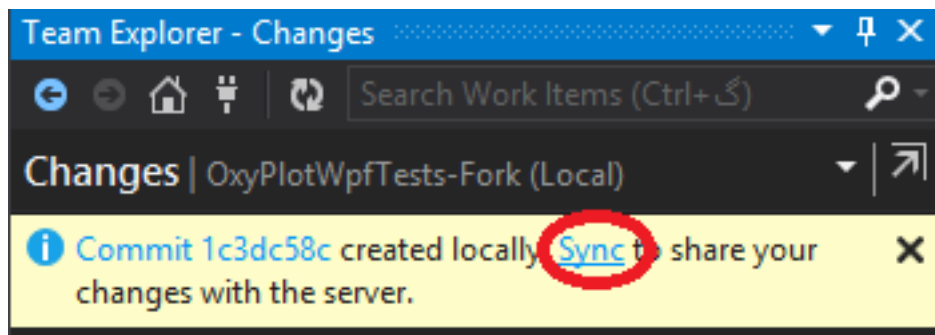
Changes را انتخاب نمایید:



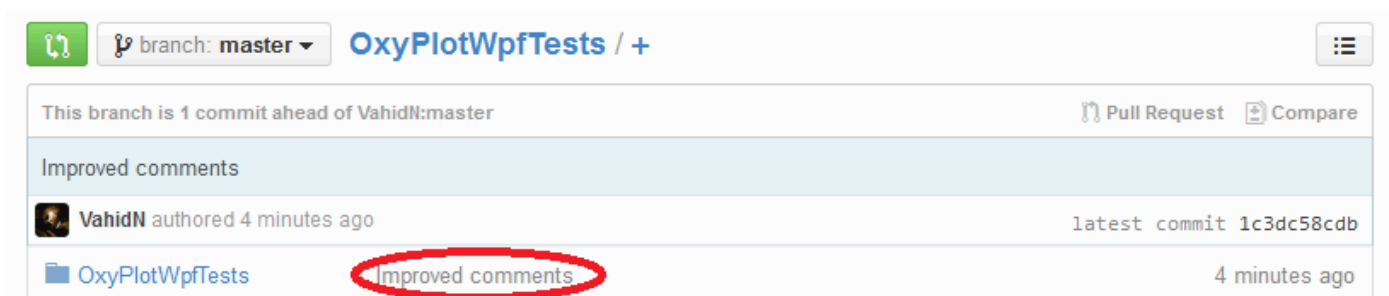
در اینجا توضیحاتی را نوشته و سپس بر روی دکمه‌ی Commit کلیک کنید.



پس از هماهنگ‌سازی محلی، اکنون نوبت به هماهنگ‌سازی این تغییرات با مخزن کد GitHub است. بنابراین بر روی لینک Sync در پیام ظاهر شده کلیک کنید و در صفحه‌ی بعدی نیز بر روی دکمه‌ی Sync کلیک نمایید:

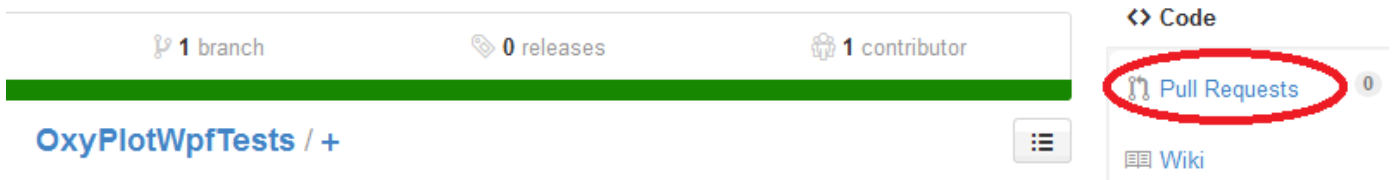


اکنون اگر به پروژه‌ی GitHub خود مراجعه کنید، این تغییر جدید قابل مشاهده‌است:



مطلع سازی صاحب اصلی مخزن کد از تغییرات انجام شده

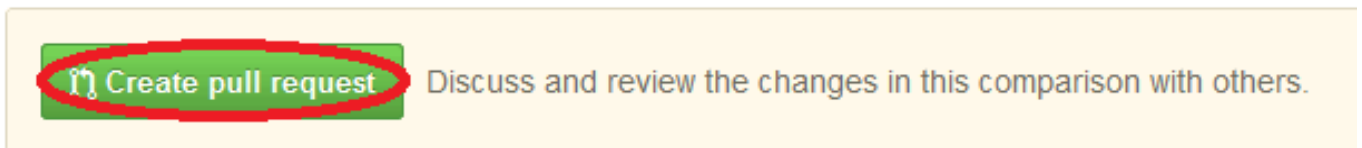
تا اینجا کسی از تغییرات جدید انجام شده‌ی توسط ما باخبر نیست. برای اطلاع رسانی در مورد این تغییرات، به مخزن کد Fork شده که اکنون تغییرات جدید به آن ارسال شده‌اند، مراجعه کنید. سپس در کنار صفحه بر روی لینک Pull request کلیک نمائید:



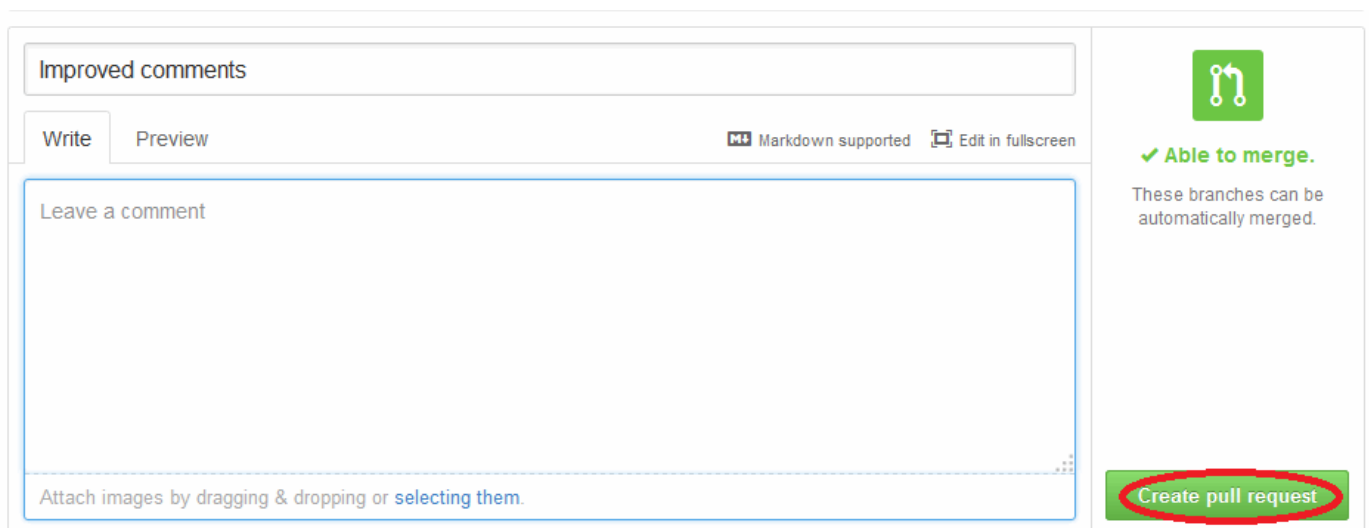
در اینجا بر روی دکمه‌ی New pull request کلیک کنید:



در ادامه تغییرات ارسال شما نمایش داده خواهند شد. آن‌ها را بررسی کرده و مجدداً بر روی دکمه‌ی Create pull request کلیک کنید:

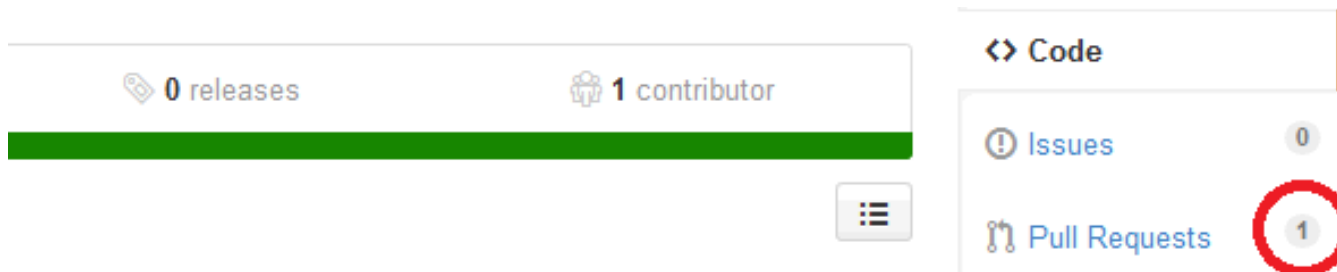


در اینجا عنوان و توضیحاتی را وارد کرده و سپس بر روی دکمه‌ی Create pull request کلیک نمایید:



یکی سازی تغییرات با مخزن اصلی

اکنون صاحب اصلی مخزن کد یک ایمیل را دریافت خواهد کرد؛ همچنین اگر به مخزن کد خود مراجعه نماید، آمار Pull requests دریافتی قابل مشاهده است:

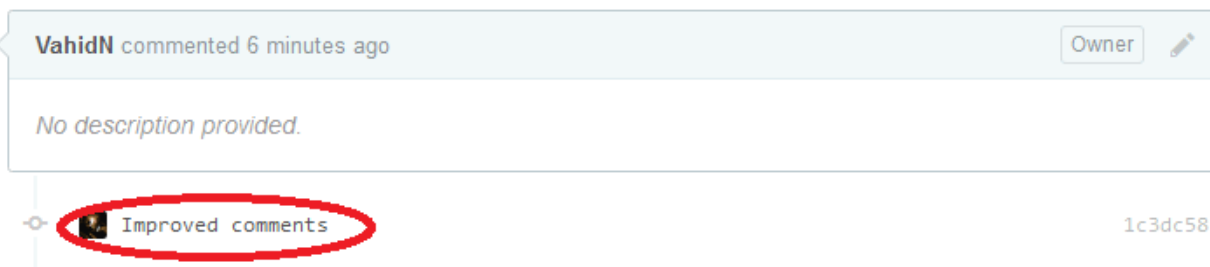


پس از انتخاب یکی از آن‌ها، لینکی برای بررسی تغییرات انجام شده و همچنین دکمه‌ای برای یکی سازی آن‌ها با پروژه‌ی اصلی وجود دارد:

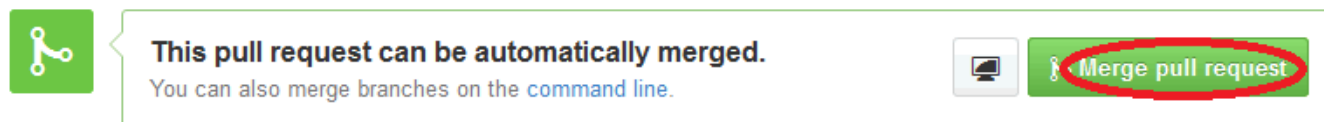
Improved comments #1

Open VahidN wants to merge 1 commit into VahidN:master from DotNETTips:master

Conversation 0 Commits 1 Files changed 1

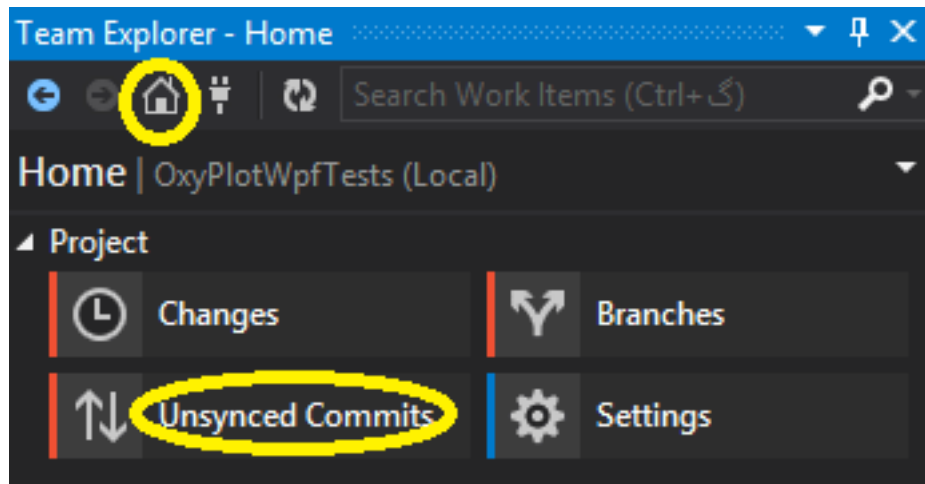


Add more commits by pushing to the master branch on DotNETTips/OxyPlotWpfTests.

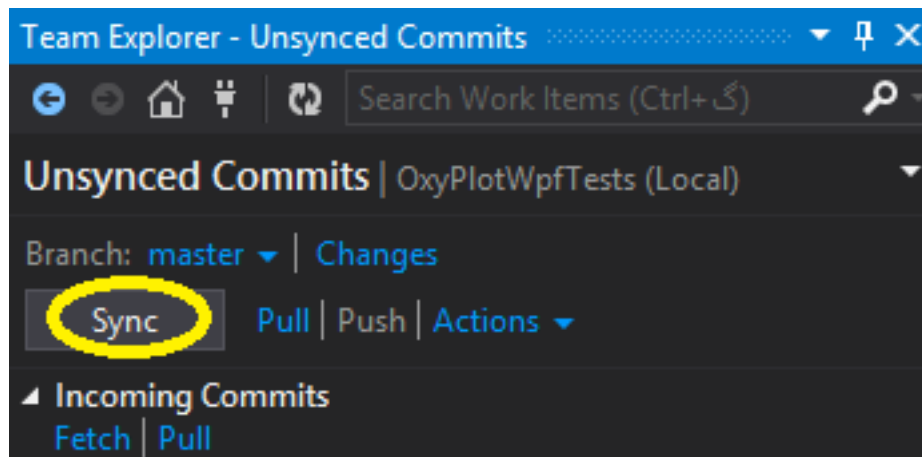


دریافت این تغییرات در مخزن کد محلی توسط صاحب اصلی پروژه

اکنون که این تغییرات با پروژه‌ی اصلی Merge و یکی شده‌اند، صاحب اصلی پروژه جهت تهیه‌ی یک کپی محلی و بهبود یا تغییر آن‌ها می‌تواند به صورت ذیل عمل کند:



ابتدا به برگه‌ی Team explorer مراجعه کرده و بر روی دکمه‌ی Home آن کلیک کنید. سپس گزینه‌ی Unsynced commits را انتخاب نمایید. در صفحه‌ی باز شده بر روی دکمه‌ی Sync کلیک نمایید. به این ترتیب آخرین تغییرات را از مخزن کد GitHub به صورت خودکار دریافت خواهید کرد:



نظرات خوانندگان

نویسنده: بهزاد شیرانی
تاریخ: ۱۳۹۳/۱۱/۲۶ ۱۲:۱۸

چطور می‌تونیم سورس خودمون رو با آخرین تغییرات انجام شده روی سورس اصلی sync کنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۱/۲۶ ۱۲:۴۹

باید عملیات [pull commits](#) انجام شود؛ در همان برگه‌ی Unsynced commits.

در [این صفحه](#) یک برنامه مختص ویندوز قرار داده شده است که شعار آن بدین شکل است: "کار با گیت هاب تا بحال تا این حد آسان نبوده است". موقعی که فایل را دانلود کنید، بعد از اجرا، شروع به دانلود و نصب برنامه اصلی خواهد کرد که در حال حاضر حجم فعلی آن حدود 45 مگابایت است. بعد از اینکه برنامه را نصب کرده و آن را اجرا کنید، از شما درخواست اطلاعات لاگین را می‌کند. اطلاعات ورود به GitHub را وارد کنید تا با اکانت شما در سایت ارتباط برقرار کند و خود را با آن سینک نماید. برای ایجاد یک repository جدید می‌توانید از دکمه‌ی Add، که در بالا سمت چپ قرار دارد استفاده کنید. در اولین کادر متنی، یک نام و در دومین کادر، متن مسیر ذخیره پروژه را اختصاص دهید. در قسمت git ignore می‌توانید مشخص کنید که چه فایل‌هایی توسط سیستم گیت ردیابی نشوند و در زمان سینک کردن یا انتشار محتوا، به سیستم گیت اضافه نشوند. این گزینه را می‌توانید none انتخاب کنید تا شاید بعدا بخواهید دستی آن را تغییر دهید. ولی با این حال این گزینه شامل قالب‌های از پیش آماده‌ای است که ممکن است کار را برای شما راحت کند. مثلا گزینه‌ی پیش فرض Windows، در مورد فایل‌هایی با پسوند doc یا docx و ... می‌باشد. برای اطلاع از روش کار این فایل، مطالب [اینجا](#) را مطالعه فرمایید.

+

master ▾

Create

Clone

Name

dotnettips

Local path

E:\GitHub\dotnettips

Browse

Git ignore

Windows ▾

✓

Create repository

در صورتیکه فایل‌های شما برای انشار نهایی آماده هستند، پروژه خود را در لیست سمت چپ برنامه انتخاب کنید تا در بالا و سمت راست برنامه، گزینه‌ی Publish Repository دیده شود و با انتخاب آن، یک نام را که قبلا وارد کرده اید و یک توضیح مختصر را از شما می‌خواهد. به صورت پیش فرض انتشارها عمومی و رایگان هستند. در صورتی که اگر بخواهید این انتشار را تنها برای خود و به صورت اختصاصی انجام دهید، باید هزینه آن را پرداخت کنید.

Publish Repository

GitHub

Enterprise

Name

dotnettips

Description

a tutorial for 'github for windows'

yeganehaym

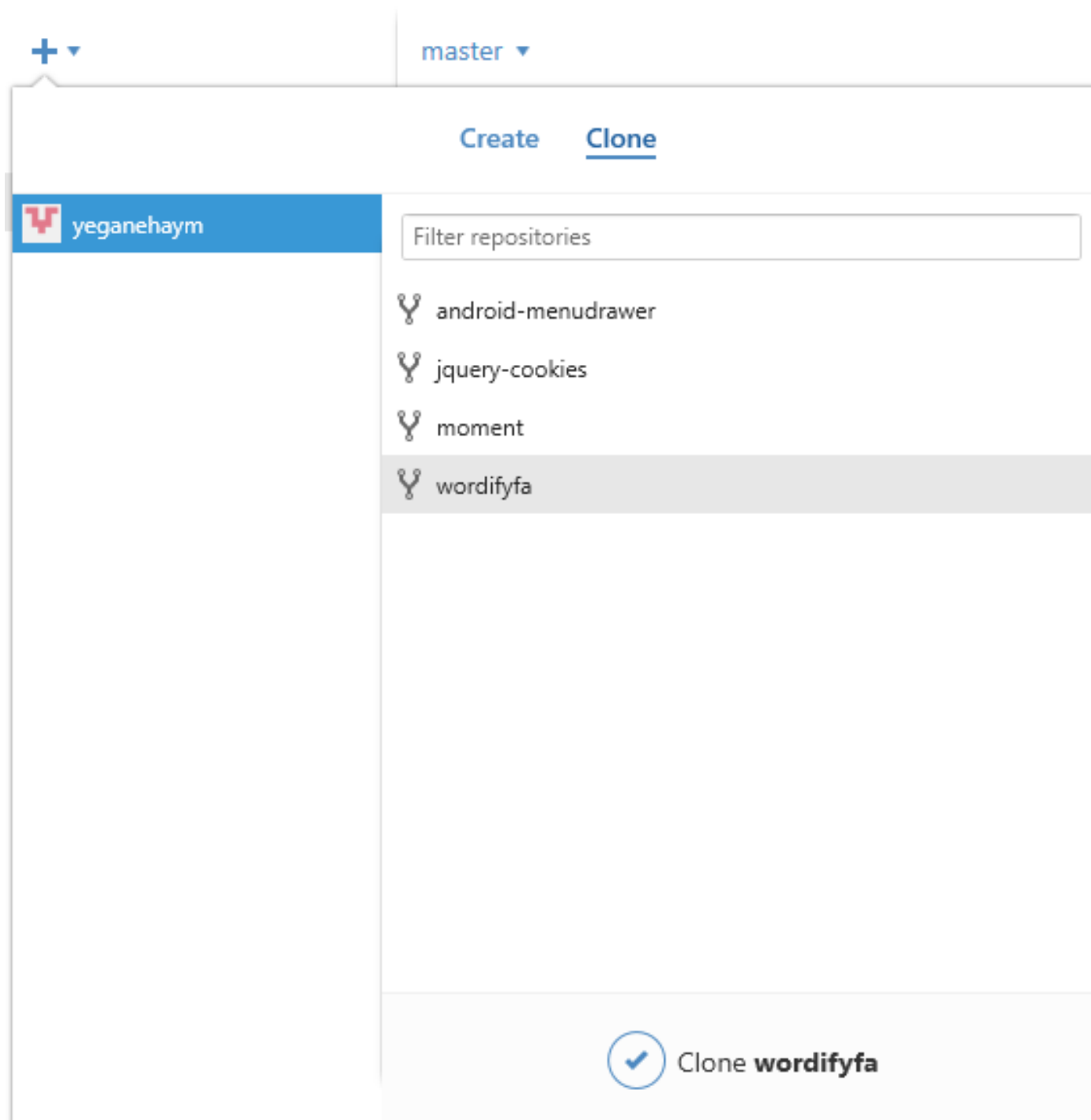
☐ Private Repository

Private repositories require a [micro plan](#) for \$7/month

✓

Publish dotnettips

در صورتیکه دوست دارید در پروژه‌ای مشارکت داشته باشید، ابتدا پروژه مورد نظر را در سایت گیت هاب Fork کنید و سپس از طریق گزینه‌ی Add در برنامه عمل کنید و اینبار در سربرگ‌های بالا، به جای Create گزینه‌ی Clone را انتخاب نمایید. در این حالت لیستی از پروژه‌های Fork شده نمایش داده می‌شوند و با انتخاب هر کدام، پروژه بر روی سیستم شما کیی خواهد شد.



بعد از انتخاب گزینه‌ی Clone، از شما محل ذخیره‌ی پروژه را خواهد پرسید و بعد از تایید آن، مقدار زمان کمی برای کپی کردن پروژه خواهد خواست. پس از آن لیستی از همه‌ی تغییرات و مشارکت‌ها به شما نمایش داده می‌شود و در صورتیکه دوست دارید به تغییری در قبل برگردید تا کارتان را از آن شروع کنید، می‌توانید از گزینه‌ی Revert استفاده کنید. برای یادگیری سایر اصطلاحات فنی گیت و گیت‌هاب می‌توانید از [مسیرهای آموزشی آن](#) استفاده کنید.

History

negative numbers support
2 months ago by Salman Arab Ameri

Merge pull request #2 from MBehtemam/...
2 months ago by Salman Arab Ameri

adding support for Nodejs
2 months ago by Mohammad Bagher Ehtemam

Merge pull request #1 from MBehtemam/...
2 months ago by Salman Arab Ameri

JSLint Correction
2 months ago by Mohammad Bagher Ehtemam

Update README.md
2 months ago by Mohammad Bagher Ehtemam

Update README.md
2 months ago by Mohammad Bagher Ehtemam

formatting added to readme
2 months ago by Salman Arab Ameri

project files added
2 months ago by Salman Arab Ameri

Added .gitattributes & .gitignore files
2 months ago by Salman Arab Ameri

negative numbers support

Salman Arab Ameri
f6126ed

GitHub
Revert
Collapse all

negative numbers support added. also index file encoding corrected.

index.html
15
wordifyfa.js
12

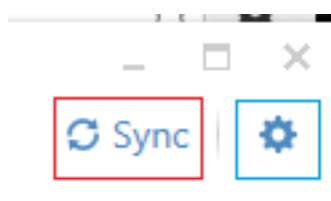
```

... -9,6 +9,11 @@ var wordifyfa = function (num, level) {
9 9     if (num === null) {
10 10         return "";
11 11     }
12 + // convert negative number to positive and get wordify value
13 + if (num<0) {
14 +     num = num * -1;
15 +     return "منفی " + wordifyfa(num, level);
16 + }
12 17     if (num === 0) {
13 18         if (level === 0) {
14 19             return "صفر";
... -56,9 +61,12 @@ var wordifyRialsInTomans = function (num) {
56 61     'use strict';
57 62     if (num >= 10) {
58 63         num = parseInt(num / 10, 10);
64 +     } else if (num<=-10) {
65 +         num = parseInt(num/10,10);
59 66     } else {
60 -         num = 0;
61 -     }
67 +         num=0;
68 +     }
69 + }
62 70     return wordifyfa(num, 0) + " تومان";
63 71 };

```

حال با خیال راحت روی پروژه کار کنید و تغییرات را روی آن اعمال کنید و بعد از اینکه کارتان تمام شد، دوباره به برنامه باز گردید و پروژه را در لیست انتخاب کرده و در سمت راست بالای صفحه، گزینه Sync Now را انتخاب کنید تا مشارکت جدید شما به سیستم گیت هاب اعمال شود و حالا اگر به صفحه‌ی پروژه در سایت گیت هاب بروید، می‌بینید که شما به عنوان یک مشارکت کننده‌ی جدید اضافه شده‌اید. پس با هر بار تغییر نسخه‌ی پروژه می‌توانید آن را با سیستم گیت سینک نمایید.

گزینه‌ی تنظیمات که در کنار عبارت Sync Now قرار دارد و با رنگ آبی در شکل مشخص شده است نیز به شما اجازه‌ی تغییر فایل‌های تنظیماتی از قبیل gitignore یا gitattribute را می‌دهد.



در صورتی که برای پروژه‌ای در گیت هاب شاخه‌ها یا branches تعریف شده باشند، در اینجا هم می‌توانید شاخه‌ی مورد نظر را انتخاب کنید:

ammeep/httpclient-extension ▼

Branches

 Manage

Filter or create new

ammeep/fix-convention-tests

ammeep/httpclient-extension ✓

ammeep/statistics-api

bump-perpage-parameter

dont-pull-down-comments

get-content-spike

hahmed/search-api

master

niik/support-etags-through-wininet

release-docs

shiftkey/rework-build-script

shiftkey/symbol-server-support



3 months ago by Brendan Forster

در مطلب «[نحوه مشارکت در پروژه‌های GitHub به کمک Visual Studio](#)» با مفهوم pull request آشنا شدیم. اما ... یک pull request خوب چه خصوصیتی دارد و فرهنگ ارسال یک PR خوب چیست؟

اخلاق مشارکت در یک پروژه سورس باز

بعضی از توسعه دهنده‌ها در حین مشارکت در یک پروژه سورس باز، برای مثال جهت افزودن قابلیت جدید و یا رفع مشکلی، ابتدا سعی می‌کنند تا کدهای فعلی را برای خودشان «قابل فهم‌تر» کنند. این قابل فهم‌تر کردن پروژه، شامل تغییر نام متغیرها و متدهای فعلی، انتقال کدهای موجود به فایل‌هایی دیگر یا حتی یکی کردن چندین فایل با هم، مرتب سازی متدهای یک کلاس بر اساس حروف الفباء و امثال آن می‌شود. این کارها را نباید در حین مشارکت و توسعه‌ی پروژه‌های سورس باز دیگران انجام دهید! اگر هدفتان رفع مشکلی است یا افزودن قابلیت جدید، باید نحوه‌ی کدنویسی فعلی را حفظ کنید. از این جهت که نگهدارنده‌ی اصلی پروژه، پیش از شما این کار را شروع کرده‌است و زمانیکه شما به پروژه‌ای دیگر رجوع خواهید کرد، باز نیز باید همین کار را ادامه دهید. اگر refactoring گسترده‌ی شما به هر نحوی سبب بهبود پروژه‌ی اصلی می‌شود، ابتدا این مورد را با مسئول اصلی پروژه مطرح کنید. اگر او قبول کرد، سپس اقدام به چنین کاری نمائید.

بحث در مورد تغییرات پیش از ارسال PR

قبل از اینکه PR ایی را ارسال کنید، بهتر است یک issue یا ticket جدید را باز کرده و در مورد آن بحث کنید یا توضیح دهید. در این حالت ممکن است توضیحات بهتری را در مورد سازگار سازی تغییرات خود با کدهای فعلی دریافت کنید.

Pull request ها را کوچک نگه‌دارید

برای اینکه شانس قبول شدن PR خود را بالا ببرید، حجم و تمرکز آن را کوچک نگه دارید. بسیاری از توسعه دهنده‌های سورس باز اگر با یک PR حجم روبرو شوند، آن را رد می‌کنند چون مشکل اصلی، مدت زمان بالایی است که باید جهت بررسی این PR اختصاص داد. هرچقدر حجم آن بیشتر باشد، زمان بیشتری را خواهد برد.

فقط یک کار را انجام دهید

شبهه به اصل تک مسئولیتی کلاس‌ها، یک PR نیز باید تنها یک کار را انجام دهد و بر روی یک موضوع خاص تمرکز داشته باشد. فرض کنید PR ایی را ارسال کرده‌اید که سه مشکل A، B و C را برطرف می‌کند. از دیدگاه مسئول اصلی پروژه، موارد A و C قابل قبول هستند؛ اما نه مورد C مطرح شده. در این حالت کل PR شما برگشت خواهد خورد. به همین جهت بهتر است بجای یک PR، سه PR مختلف و مجزا را جهت رفع مشکلات A، B و C ارسال کنید.

سازگاری تغییرات ارسالی را بررسی کنید

حداقل کاری را که پیش از ارسال PR باید انجام دهید این است که بررسی کنید آیا این تغییرات قابل Build هستند یا خیر. همچنین اگر پروژه دارای یک سری Unit tests است، حتماً آن‌ها را یکبار بررسی کنید تا مطمئن شوید جای دیگری را به هم نریخته‌اید. ضمناً وجود این تست‌ها به صورت ضمنی به این معنا است که تغییرات جدید شما نیز باید به همراه تست‌های مرتبطی باشند تا پذیرفته شوند.

PR ایی را بر روی شاخه‌ی master ارسال نکنید

پس از اینکه یک fork از پروژه‌ای سورس باز را ایجاد کردید و سپس آنرا clone نمودید تا به صورت Local بتوانید با آن کار کنید، فراموش نکنید که در همینجا باید یک branch و انشعاب جدید را جهت کار بر روی ویژگی مدنظر خود ایجاد کنید (برای مثال feature-X, fix-Y). بسیاری از پروژه‌های سورس باز به هیچ عنوان PRهای کار شده‌ی بر روی انشعاب master را قبول نمی‌کنند.

برای مطالعه بیشتر

[Open Source Contribution Etiquette](#)

[ten tips for better Pull Requests](#)

[Getting a Pull Request Accepted](#)

[Optimize Your Pull-request](#)

نظرات خوانندگان

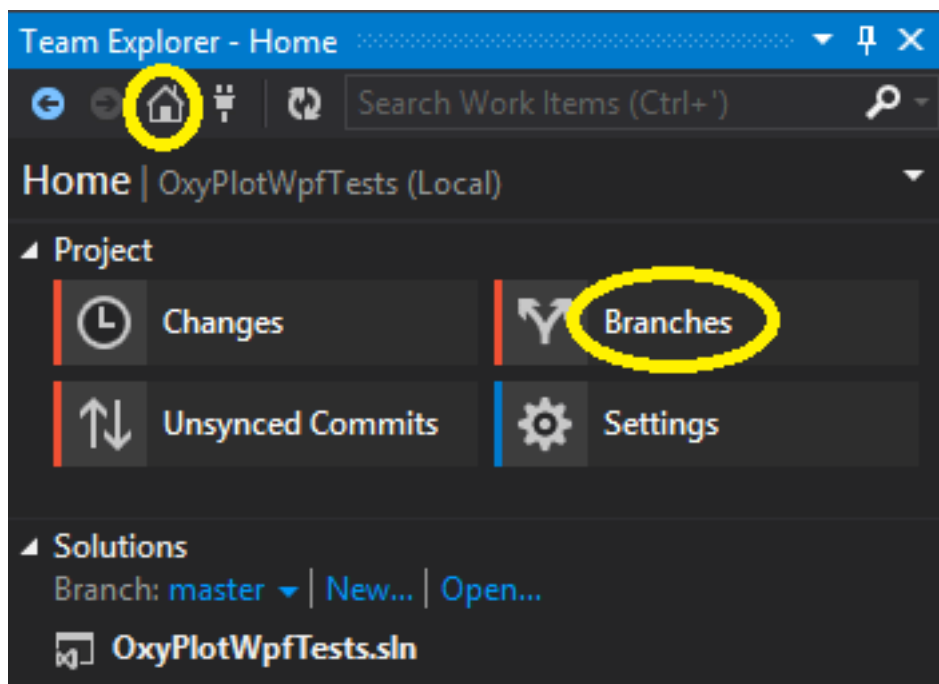
نویسنده: جلال
تاریخ: ۱۹:۱۱ ۱۳۹۳/۱۲/۱۴

کاش زودتر خونده بودمش P:

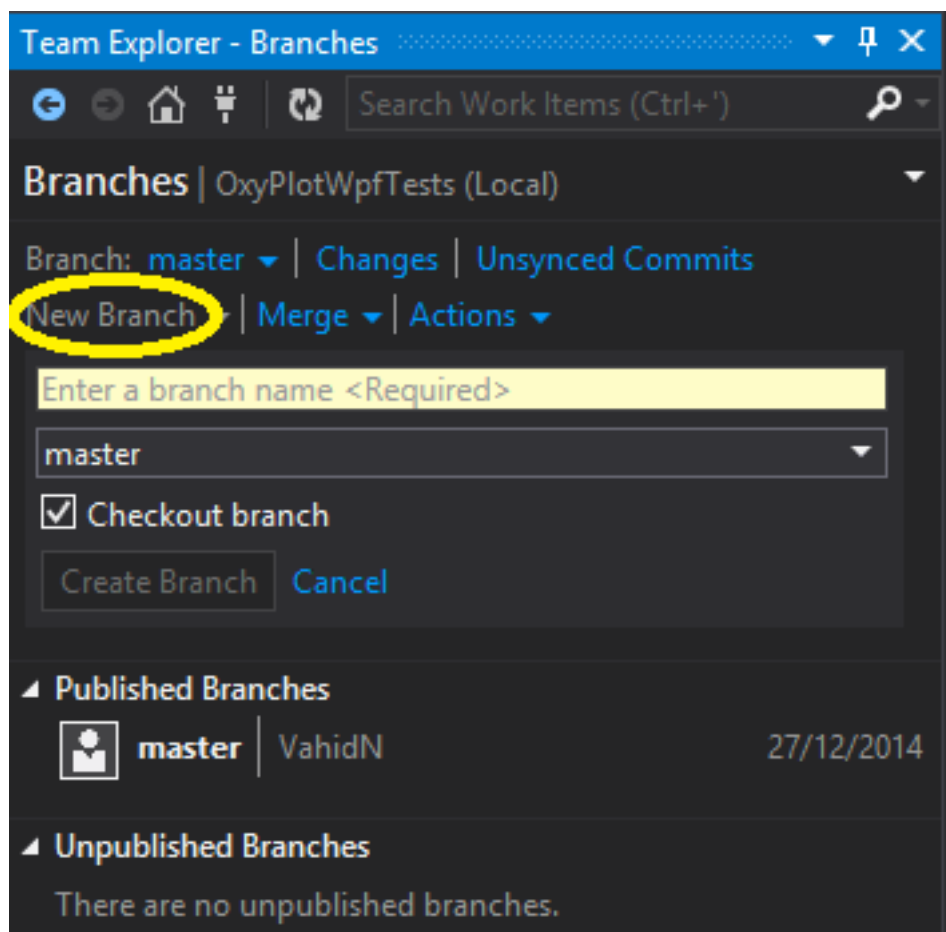
در مطلب «[آشنایی با ساختار یک Pull Request خوب](#)» عنوان شد که قابلیت‌های جدید و یا رفع مشکلات را در شاخه‌ی اصلی کار نکنید. اما ... چگونه؟

ایجاد یک شاخه‌ی جدید در Visual Studio و انتشار آن

به برگه‌ی Team explorer مراجعه کرده و سپس گزینه‌ی Branches آن را انتخاب کنید:



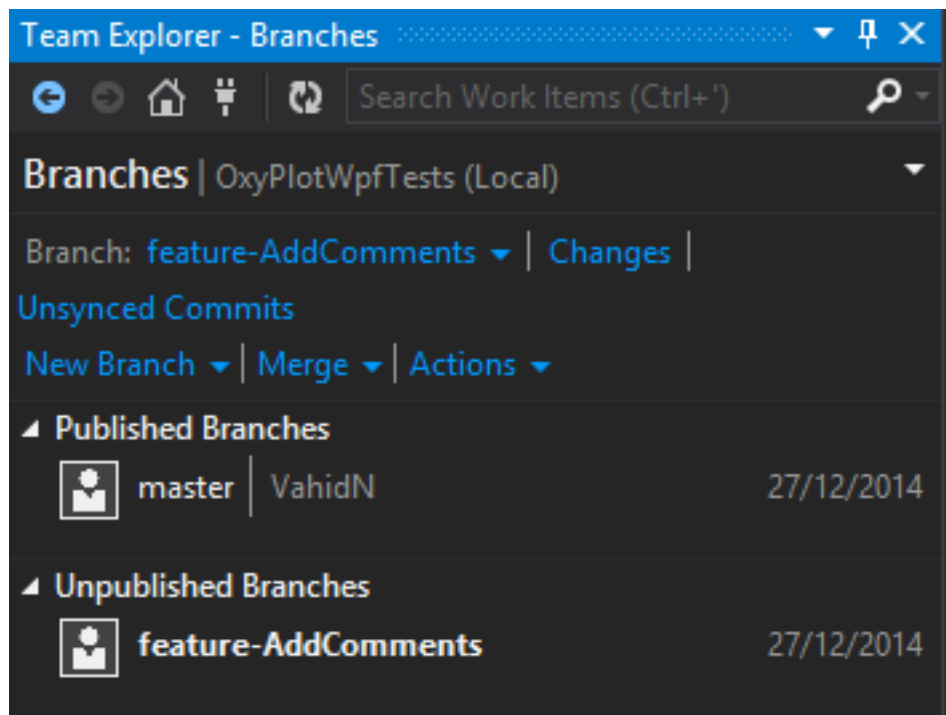
در برگه‌ی باز شده، انشعاب و شاخه‌ی جاری با فونت ضخیم نمایش داده می‌شود. برای مثال در اینجا، انشعاب کاری همان master است:



برای ایجاد یک شاخه‌ی جدید، بر روی لینک new branch کلیک کنید تا بتوان نامی را برای این منظور وارد کرد. بهتر است از نام‌های با مفهومی مانند feature-X یا fix-Y استفاده کنید (افزودن قابلیت X و یا رفع مشکل Y) و در آخر بر روی دکمه‌ی Create branch کلیک کنید.

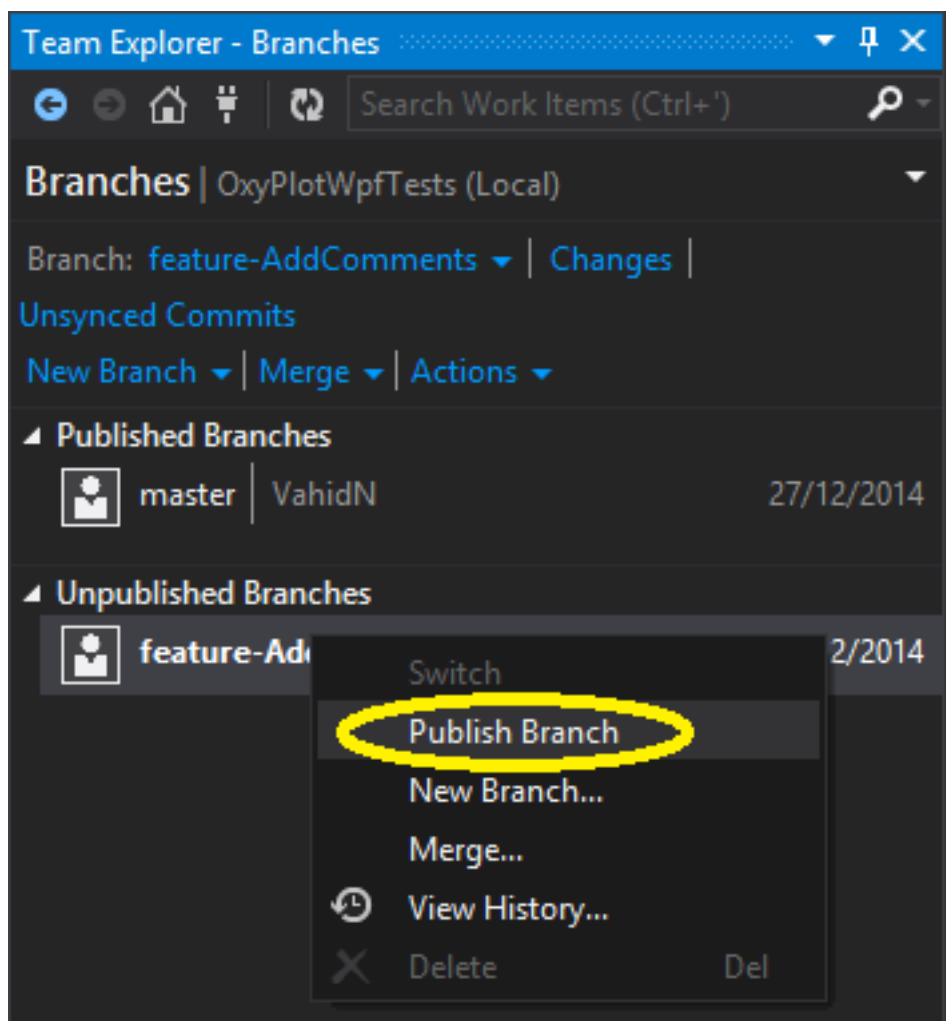
در اینجا می‌توان مشخص کرد که انشعاب ایجاد شده باید بر اساس کدام انشعاب فعلی نیز تهیه شود (دراپ داون ذیل قسمتی که می‌توان نام انشعاب را وارد کرد). برای مثال پروژه‌های میکروسافت در GitHub، دارای سه شاخه‌ی master، dev و release هستند. شاخه‌ی dev (یا توسعه) جایی است که انشعابات pull requests را قبول خواهند کرد. بنابراین بر اساس ساختار و طراحی پروژه‌ی جاری به این موضوع نیز باید دقت داشت.

پس از ایجاد شاخه‌ی جدید، تصویر ذیل نمایان خواهد شد:



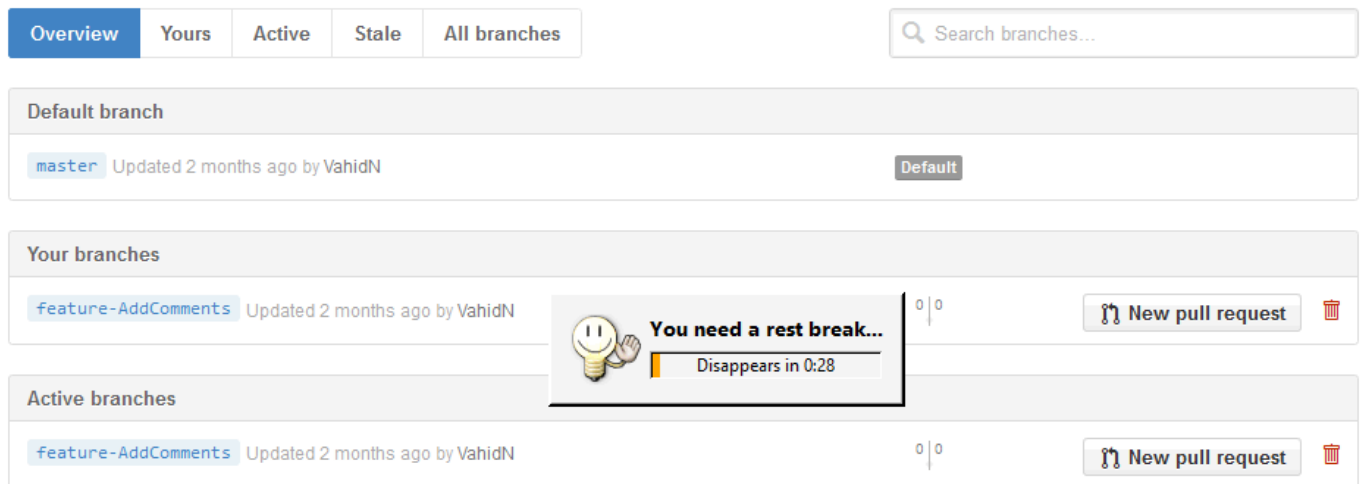
همانطور که ملاحظه می‌کنید، اینبار شاخه‌ی جدید ایجاد شده به صورت **bold** و ضخیم نمایش داده شده‌است. این **bold** بودن به معنای شاخه‌ی کاری جاری بودن است. همچنین این شاخه در قسمت **unpublished branches** قرار دارد. بنابراین کلیه‌ی تغییرات واقع شده‌ی در آن، محلی بوده و هنوز با سرور هماهنگ نشده‌اند.

برای انتشار و **publish** این شاخه، تنها کافی است تا بر روی آن کلیک راست کرده و گزینه‌ی **publish branch** را انتخاب کنیم:



این انتشار سبب نمایش لیستی از تغییرات جدید در برگه‌ی branches پروژه، در GitHub خواهد شد:

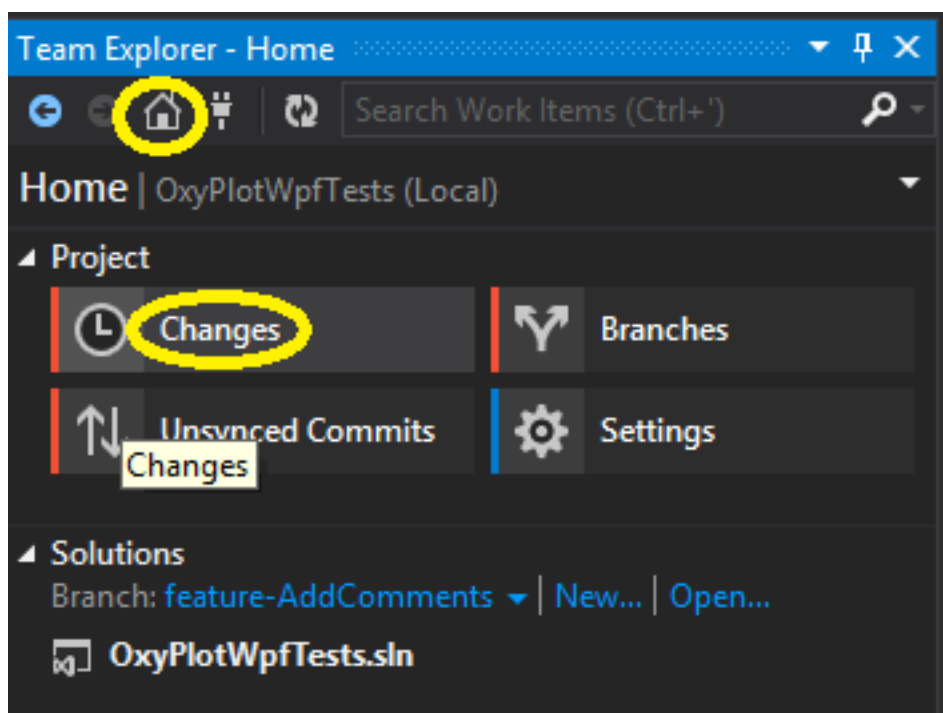




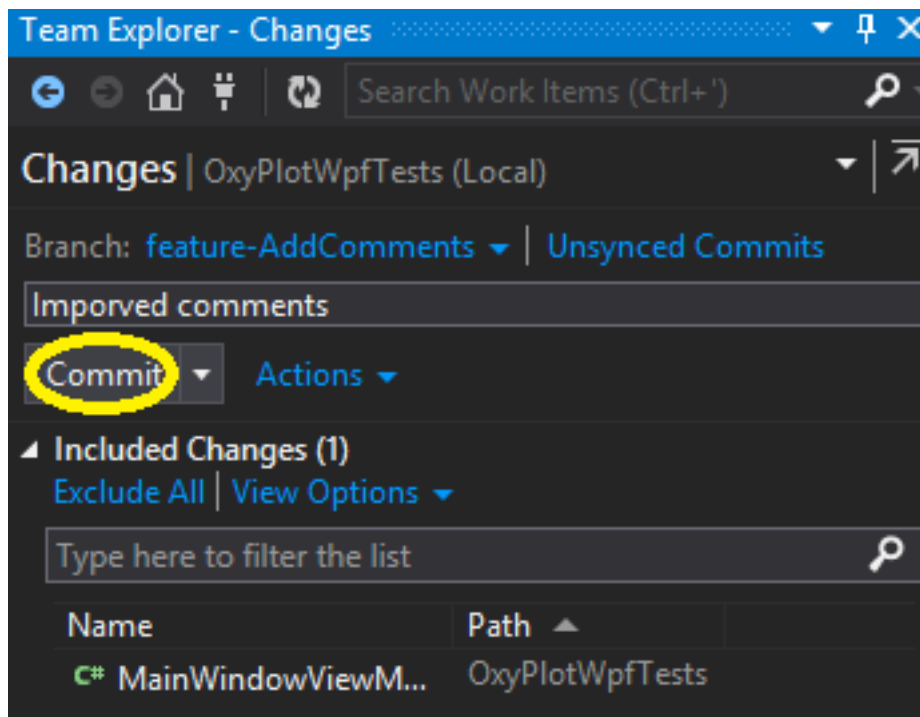
یک نکته: برای تغییر branch فعال جاری، فقط کافی است در برگه‌ی branches در ویژوال استودیو، دوبار بر روی لینک نام آن شاخه کلیک کنید تا به صورت bold ظاهر شود.

ارسال تغییرات انجام شده‌ی در Branch به سرور

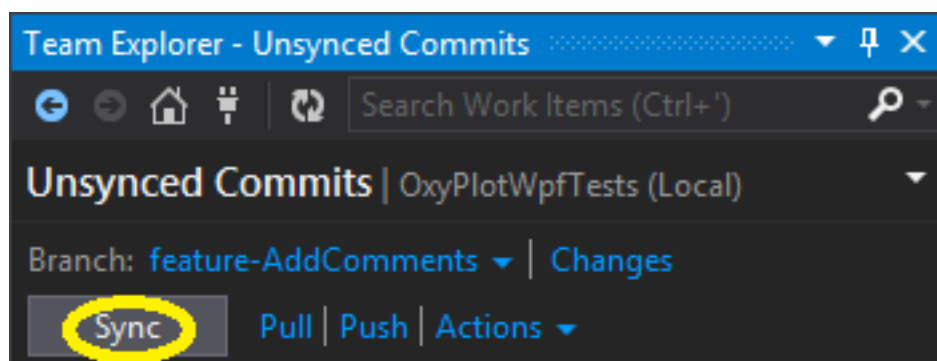
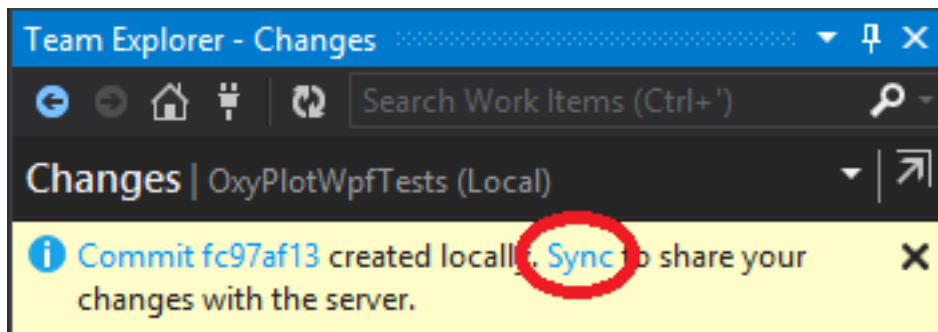
پس از کار بر روی شاخه‌ی جدید ایجاد شده، اکنون نوبت به ارسال و هماهنگ سازی این تغییرات با سرور است. این مورد نیز همانند قبل بوده و ابتدا باید به برگه‌ی Home و گزینه‌ی changes آن مراجعه کرد:



و سپس تغییرات را به همراه توضیحی commit کرد:

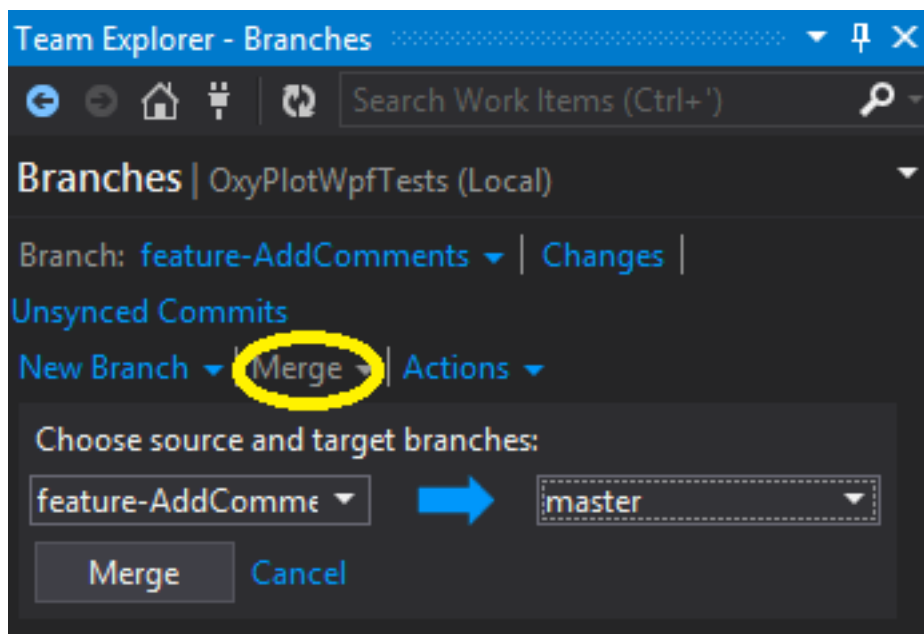


اینکار سبب sync محلی می‌شود. سپس بر روی لینک sync کلیک نمایید و تغییرات را با سرور هماهنگ کنید.



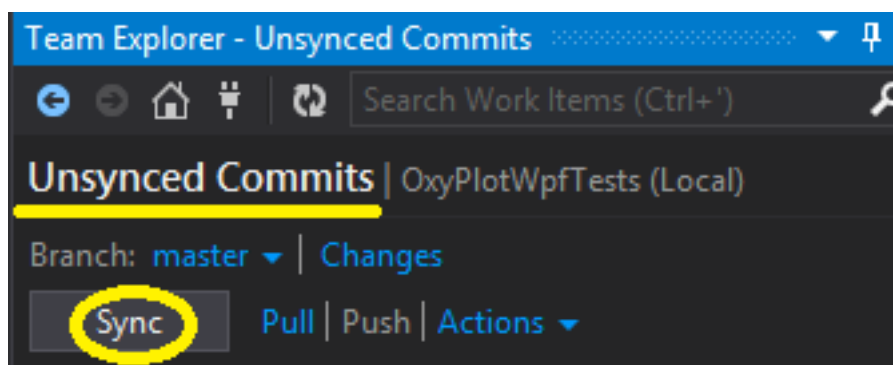
یکی کردن تغییرات شاخه‌ی جدید با شاخه‌ی اصلی

هرچند این تغییرات به سرور ارسال شده‌اند، اما چون در یک انشعاب کاری دیگر قرار دارند، با انشعاب اصلی یکی نخواهند شد. برای انجام عملیات merge، ابتدا به برگه‌ی Home و سپس گزینه‌ی branches مراجعه کنید. در ادامه بر روی لینک merge کلیک نمائید (تصاویر اول و دوم بحث).



در اینجا می‌خواهیم اطلاعات موجود در شاخه‌ی افزودن توضیحات را با شاخه‌ی اصلی یکی کنیم (انتخاب منبع و مقصد). سپس بر روی دکمه‌ی merge کلیک نمائید.

اکنون برای ارسال این تغییرات به سرور، به برگه‌ی Home و سپس گزینه‌ی unsynced commits مراجعه کرده و بر روی دکمه‌ی sync کلیک نمائید تا تغییرات یکی شده به سرور ارسال شوند.



چندی قبل مطلب «[اطلاع از بروز رسانی نرم افزار ساخته شده](#)» را در سایت جاری مطالعه کردید. در این روش بسیار متداول، شماره نگارش‌های جدید برنامه در یک فایل XML و مانند آن قرار می‌گیرند و برنامه هر بار این فایل را جهت یافتن شماره‌های مندرج در آن اسکن می‌کند. اگر پروژه‌ی شما سورس باز است و در GitHub هاست شده، روش دیگری نیز برای یافتن این اطلاعات وجود دارد. در GitHub می‌توان از طریق آدرسی به شکل https://api.github.com/repos/user_name/project_name/releases به اطلاعات آخرین ارائه‌های یک پروژه (قرار گرفته در برگه‌ی releases آن) با فرمت JSON دسترسی یافت (یک مثال). در ادامه قصد داریم روش استفاده‌ی از آن را بررسی کنیم.

ساختار JSON ارائه‌های یک پروژه در GitHub

ساختار کلی اطلاعات ارائه‌های یک پروژه در GitHub چنین شکلی را دارد:

```
[
  {
    release_info,
    "assets": [
      {
      }
    ]
  }
]
```

در اینجا آرایه‌ای از اطلاعات ارائه‌ی یک پروژه ارسال می‌شود. هر ارائه نیز دارای دو قسمت است: لینکی به صفحه‌ی اصلی release در GitHub و سپس آرایه‌ای به نام assets که در آن اطلاعات فایل‌های پیوستی مانند نام فایل، آدرس، اندازه و امثال آن قرار گرفته‌اند.

تهیه‌ی کلاس‌های معادل فرمت JSON ارائه‌های برنامه در GitHub

اگر بخواهیم قسمت‌های مهم خروجی JSON فوق را تبدیل به کلاس‌های معادل دات نت کنیم، به دو کلاس ذیل خواهیم رسید:

```
using Newtonsoft.Json;
using System;

namespace ApplicationAnnouncements
{
    public class GitHubProjectRelease
    {
        [JsonProperty(PropertyName = "url")]
        public string Url { get; set; }

        [JsonProperty(PropertyName = "assets_url")]
        public string AssetsUrl { get; set; }

        [JsonProperty(PropertyName = "upload_url")]
        public string UploadUrl { get; set; }

        [JsonProperty(PropertyName = "html_url")]
        public string HtmlUrl { get; set; }

        [JsonProperty(PropertyName = "id")]
        public int Id { get; set; }

        [JsonProperty(PropertyName = "tag_name")]
        public string TagName { get; set; }

        [JsonProperty(PropertyName = "target_commitish")]
        public string TargetCommitish { get; set; }

        [JsonProperty(PropertyName = "name")]
    }
```

```

    public string Name { get; set; }

    [JsonProperty(PropertyName = "body")]
    public string Body { get; set; }

    [JsonProperty(PropertyName = "draft")]
    public bool Draft { get; set; }

    [JsonProperty(PropertyName = "prerelease")]
    public bool PreRelease { get; set; }

    [JsonProperty(PropertyName = "created_at")]
    public DateTime CreatedAt { get; set; }

    [JsonProperty(PropertyName = "published_at")]
    public DateTime PublishedAt { get; set; }

    [JsonProperty(PropertyName = "assets")]
    public Asset[] Assets { get; set; }
}

public class Asset
{
    [JsonProperty(PropertyName = "url")]
    public string Url { get; set; }

    [JsonProperty(PropertyName = "id")]
    public int Id { get; set; }

    [JsonProperty(PropertyName = "name")]
    public string Name { get; set; }

    [JsonProperty(PropertyName = "label")]
    public string Label { get; set; }

    [JsonProperty(PropertyName = "content_type")]
    public string ContentType { get; set; }

    [JsonProperty(PropertyName = "state")]
    public string State { get; set; }

    [JsonProperty(PropertyName = "size")]
    public int Size { get; set; }

    [JsonProperty(PropertyName = "download_count")]
    public int DownloadCount { get; set; }

    [JsonProperty(PropertyName = "created_at")]
    public DateTime CreatedAt { get; set; }

    [JsonProperty(PropertyName = "updated_at")]
    public DateTime UpdatedAt { get; set; }
}

```

در اینجا از ویژگی [JsonProperty](#) جهت معرفی نام‌های واقعی خواص ارائه شده‌ی توسط GitHub استفاده کرده‌ایم. پس از تشکیل این کلاس‌ها، مرحله‌ی بعد، دریافت اطلاعات JSON از آدرس API ارائه‌های پروژه در GitHub و سپس نگاشت آن‌ها می‌باشد:

```

using (var webClient = new WebClient())
{
    webClient.Headers.Add("user-agent", "DNTProfiler");
    var jsonData = webClient.DownloadString(url);
    var githubProjectReleases = JsonConvert.DeserializeObject<GithubProjectRelease[]>(jsonData);

    foreach (var release in githubProjectReleases)
    {
        foreach (var asset in release.Assets)
        {
            // ...
        }
    }
}

```

حین کار با WebClient یا هر روش دیگری که برای دریافت اطلاعات از وب استفاده می‌کنید، حتما نیاز است user-agent را ذکر

کرد. در غیر اینصورت GitHub درخواست شما را برگشت خواهد زد. پس از دریافت اطلاعات JSON، با استفاده از متد `JsonConvert.DeserializeObject` کتابخانه‌ی [JSON.NET](https://www.json.net/)، می‌توان آن‌ها را تبدیل به آرایه‌ای از `GitHubProjectRelease` کرد.

یک نکته: اگر به صفحه‌ی اصلی ارائه‌های یک پروژه در GitHub دقت کنید، شمارهی تعداد بار دریافت یک ارائه مشخص نشده‌است. در این API، عدد `DownloadCount`، بیانگر تعداد بار دریافت پروژه‌ی شما است.