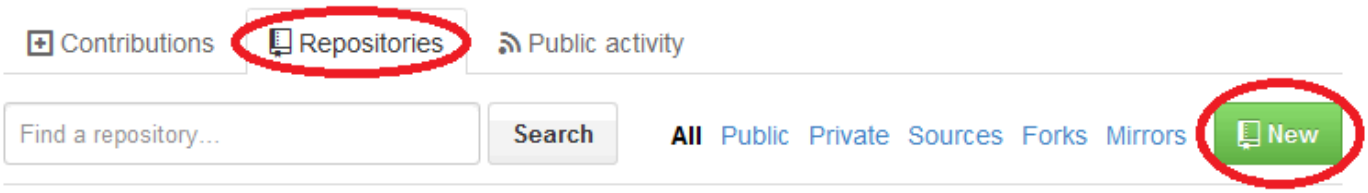
عنوان: مراحل ارسال یک پروژه‌ی Visual Studio به GitHub
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۵ ۲۲:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Git, Source control, GitHub

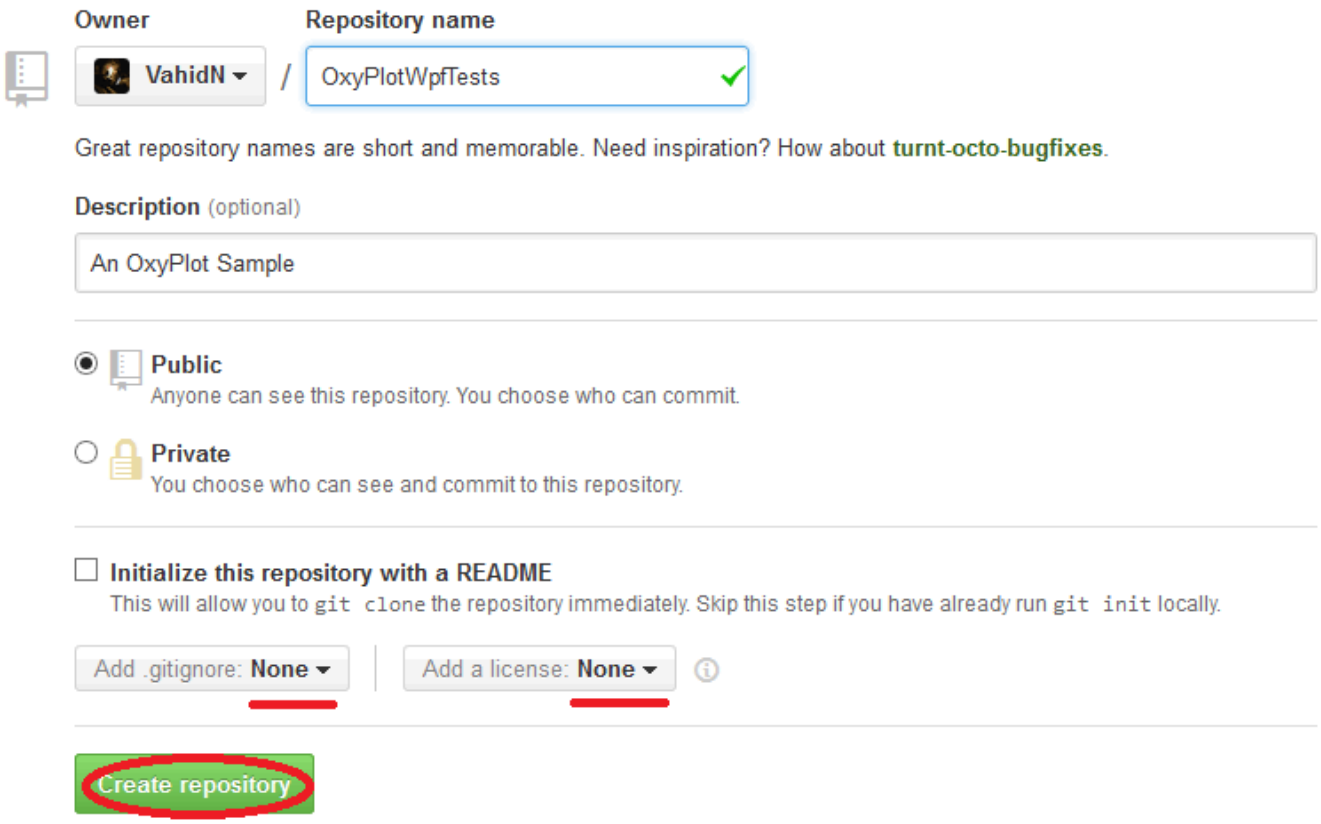
از نگارش 2012 ویژوال استودیو، امکان کار با مخازن Git، به صورت یکپارچه و توکار و بدون نیاز به ابزارهای جانبی، توسط آن فراهم شده‌است. در ادامه قصد داریم به کمک این ویژگی توکار، نحوه‌ی ارسال یک پروژه‌ی از پیش موجود VS.NET را برای اولین بار به GitHub بررسی کنیم.

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:



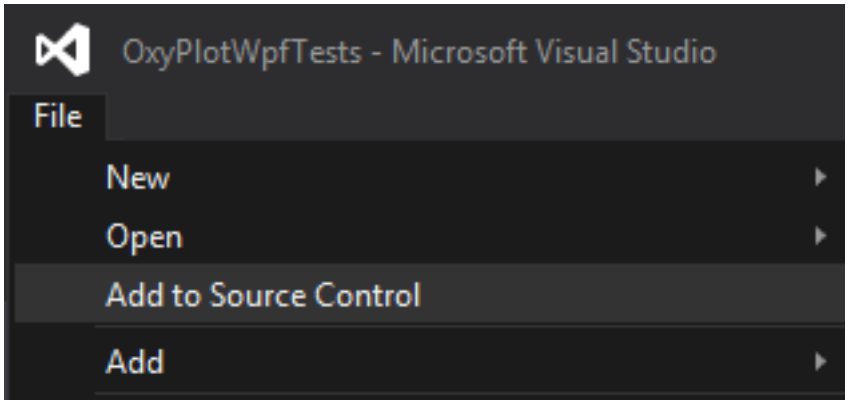
سپس در صفحه‌ی بعدی، نام پروژه را به همراه توضیحاتی وارد نمائید و بر روی دکمه‌ی Create repository کلیک کنید. در اینجا سایر گزینه‌ها را انتخاب نکنید. نیازی به انتخاب گزینه‌ی READ ME و یا انتخاب مجوز و غیره نیست. تمام این کارها را در سمت پروژه‌ی اصلی می‌توان انجام داد و یا VS.NET فایل‌های ignore را به صورت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

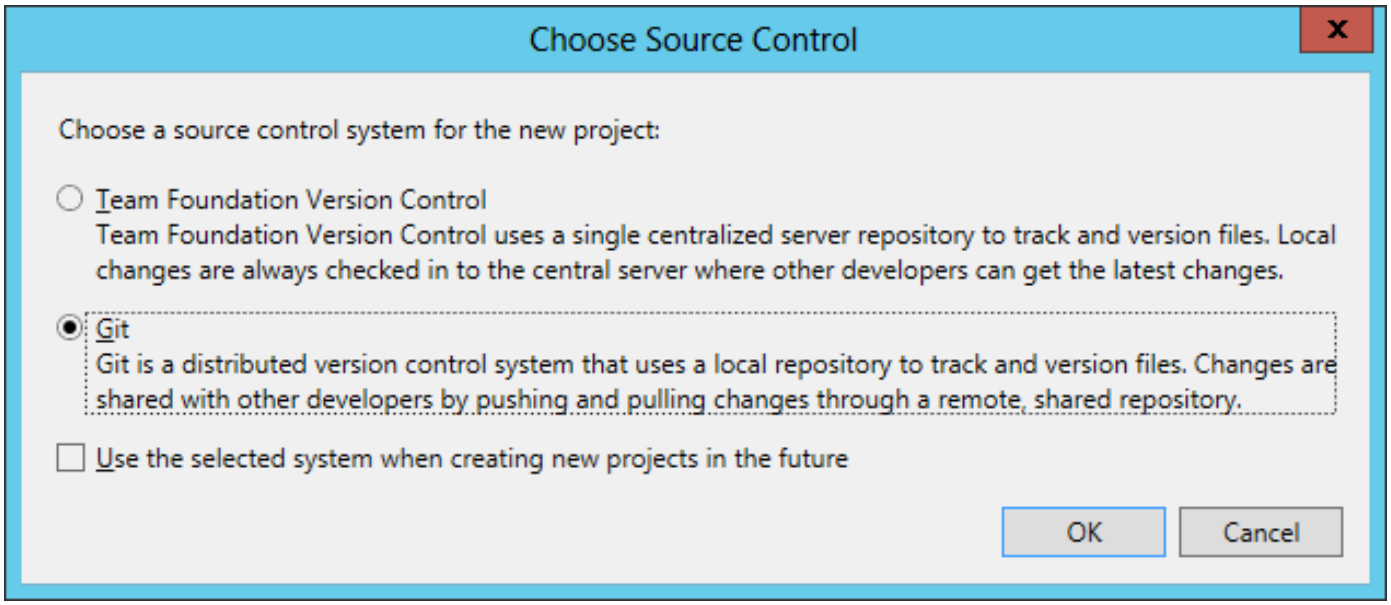


از اطلاعات صفحه‌ی بعدی، تنها به آدرس مخصوص GitHub آن نیاز داریم. از این آدرس در VS.NET برای ارسال اطلاعات به سرور استفاده خواهیم کرد:

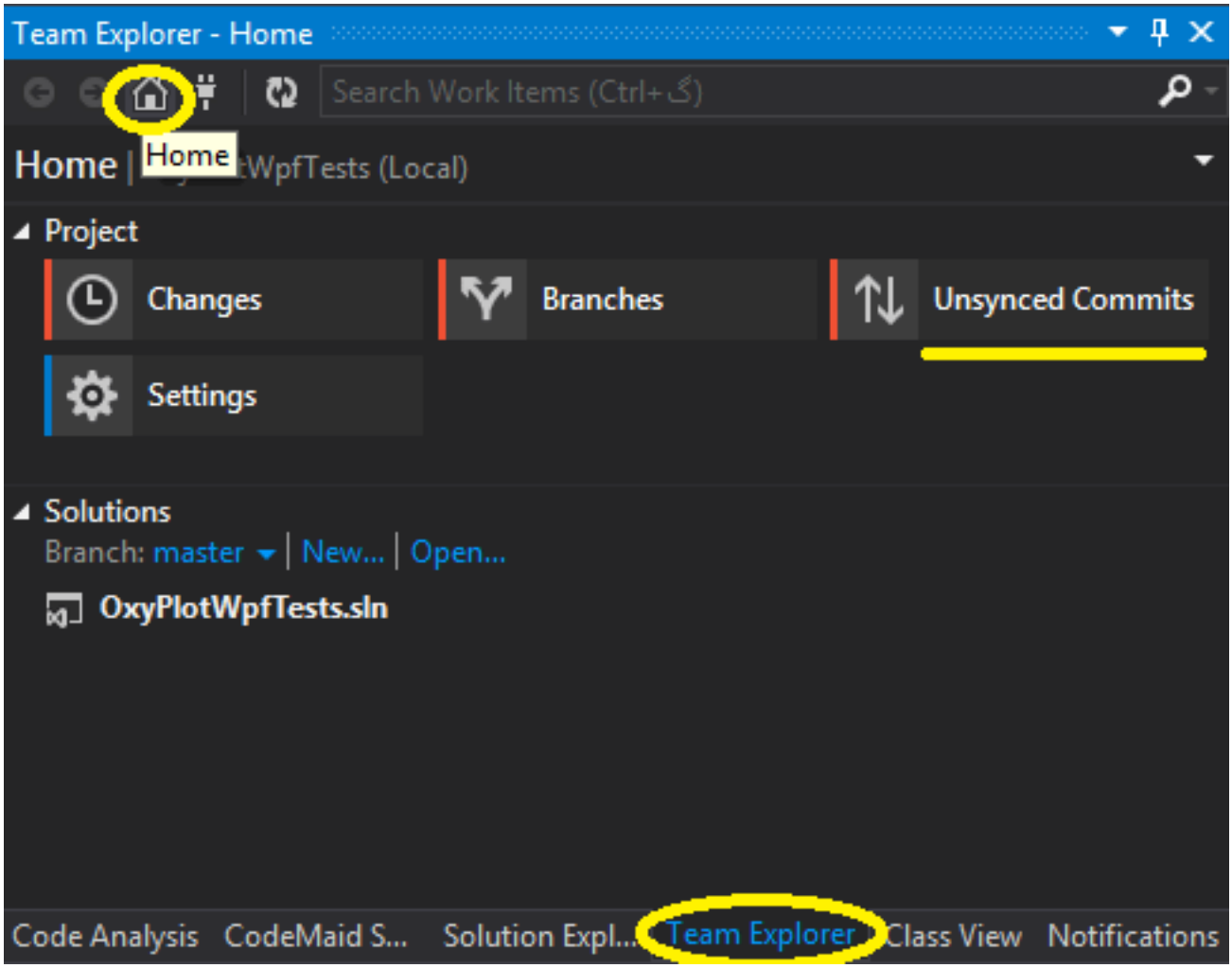
تنظیمات VS.NET برای ارسال پروژه به مخزن GitHub

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 نمائید:

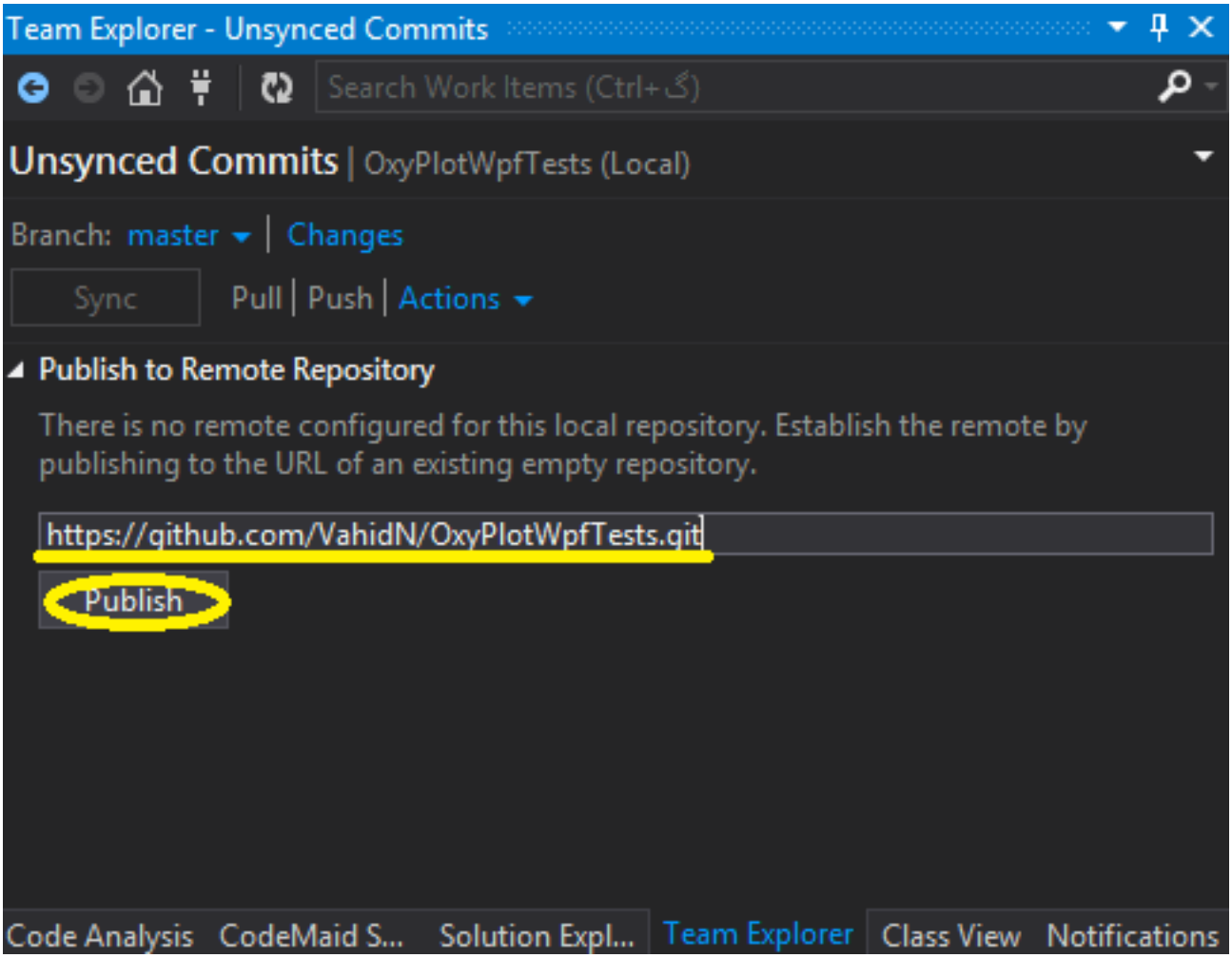




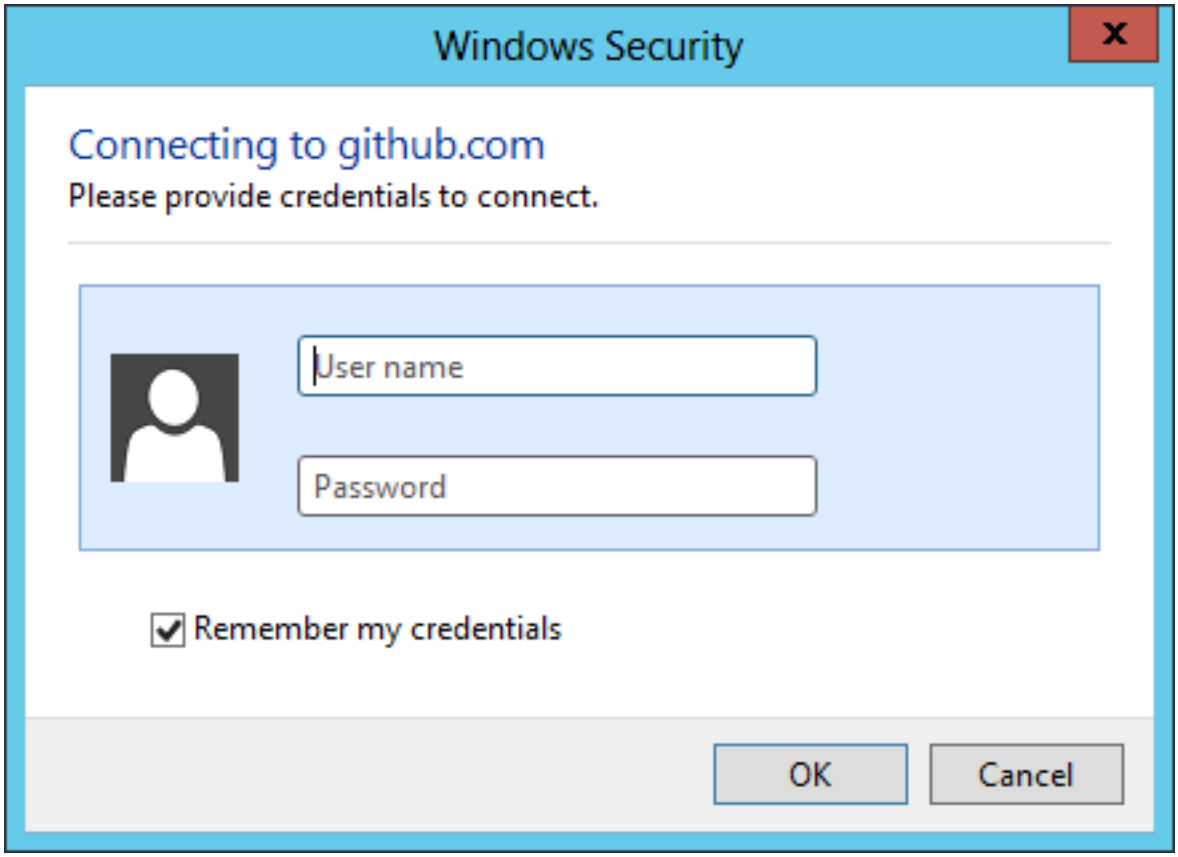
سپس در کنار برگه‌ی Solution Explorer، برگه‌ی Team Explorer را انتخاب کنید. در اینجا بر روی دکمه‌ی Home در نوار ابزار آن کلیک کرده و سپس بر روی دکمه‌ی Unsynced commits کلیک نمائید.



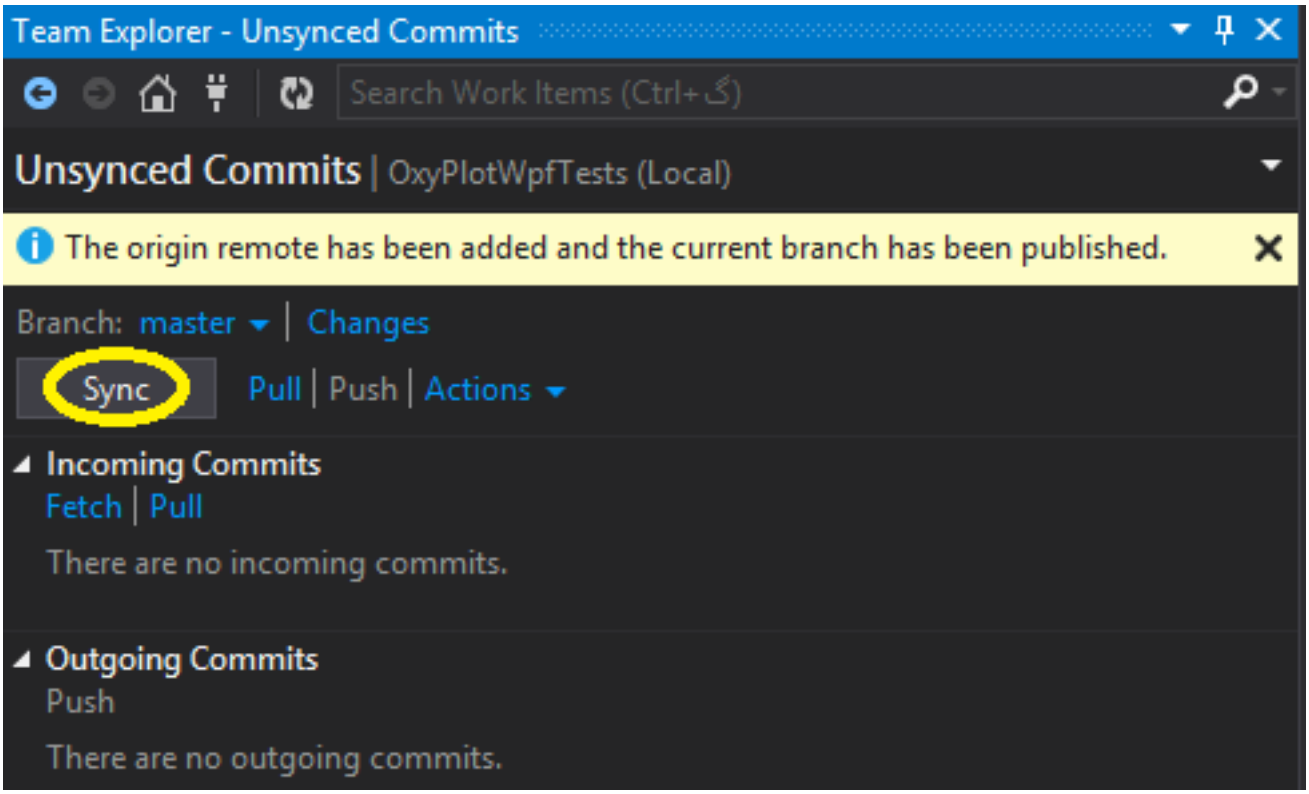
در ادامه در صفحه‌ی باز شده، همان آدرس مخصوص مخزن کد جدید را در GitHub وارد کرده و بر روی دکمه‌ی Publish کلیک کنید:



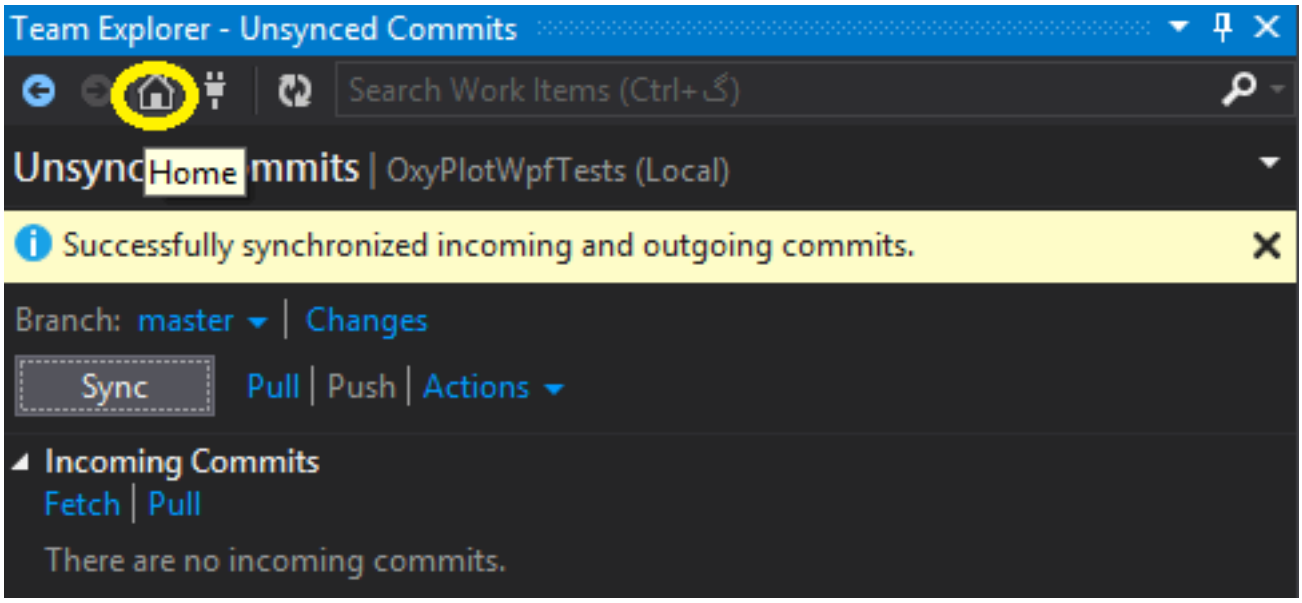
در اینجا بلافاصله صفحه‌ی لاگینی ظاهر می‌شود که باید در آن مشخصات اکانت GitHub خود را وارد نمائید:



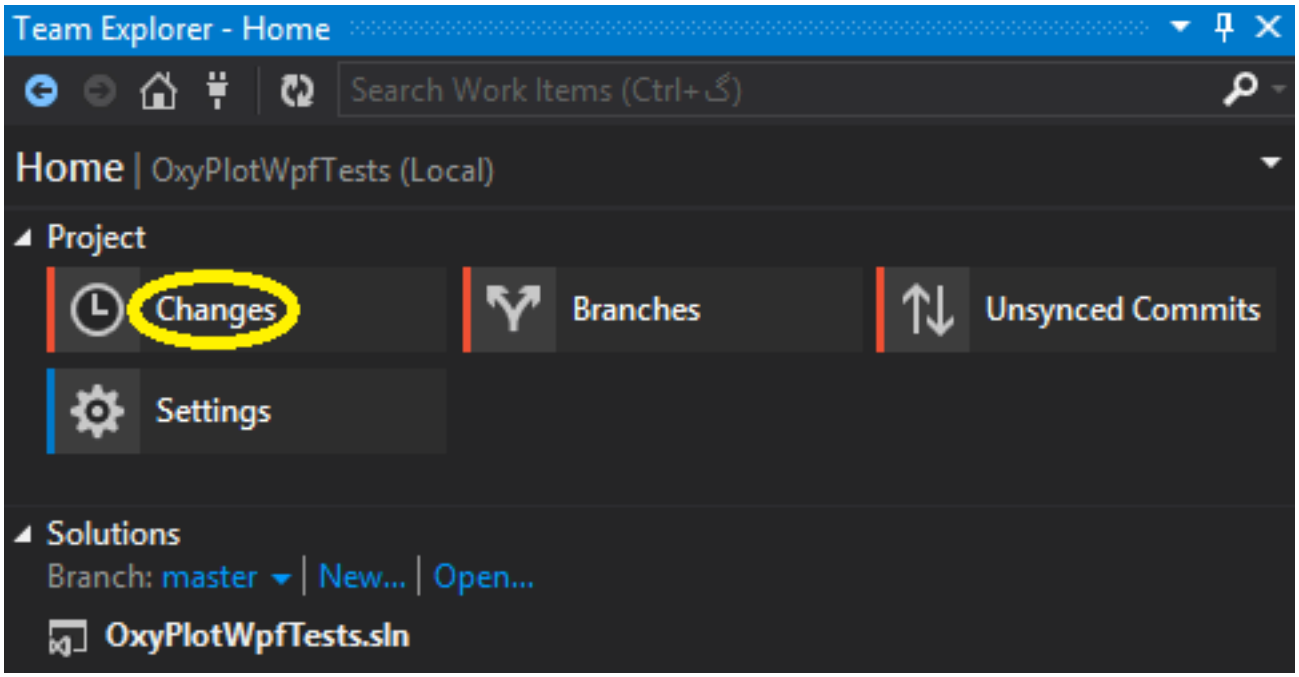
به این ترتیب عملیات Publish اولیه انجام شده و تصویر ذیل نمایان خواهد شد:



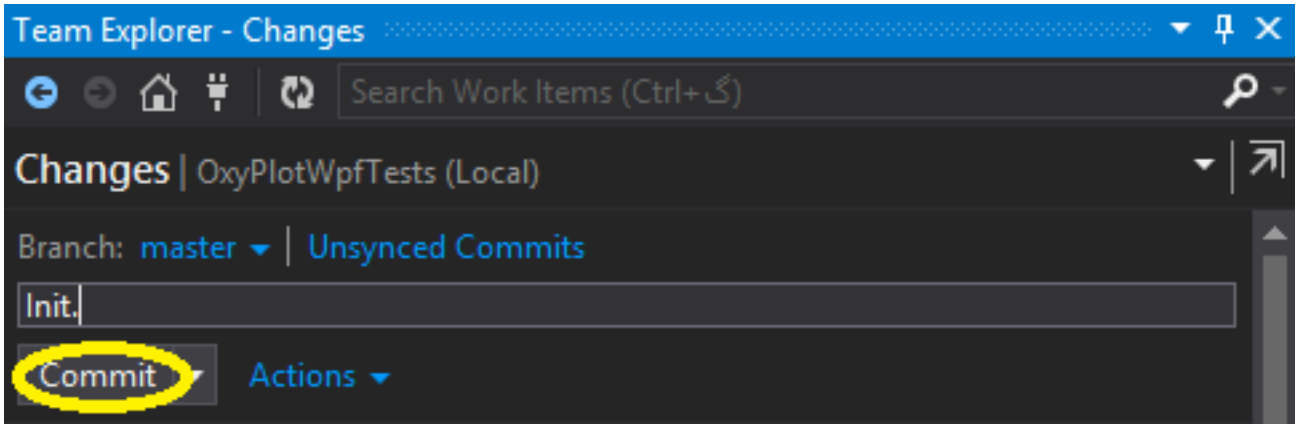
در اینجا بر روی دکمه‌ی Sync کلیک کنید. به این ترتیب مخزن کد GitHub به پروژه‌ی جاری متصل خواهد شد:



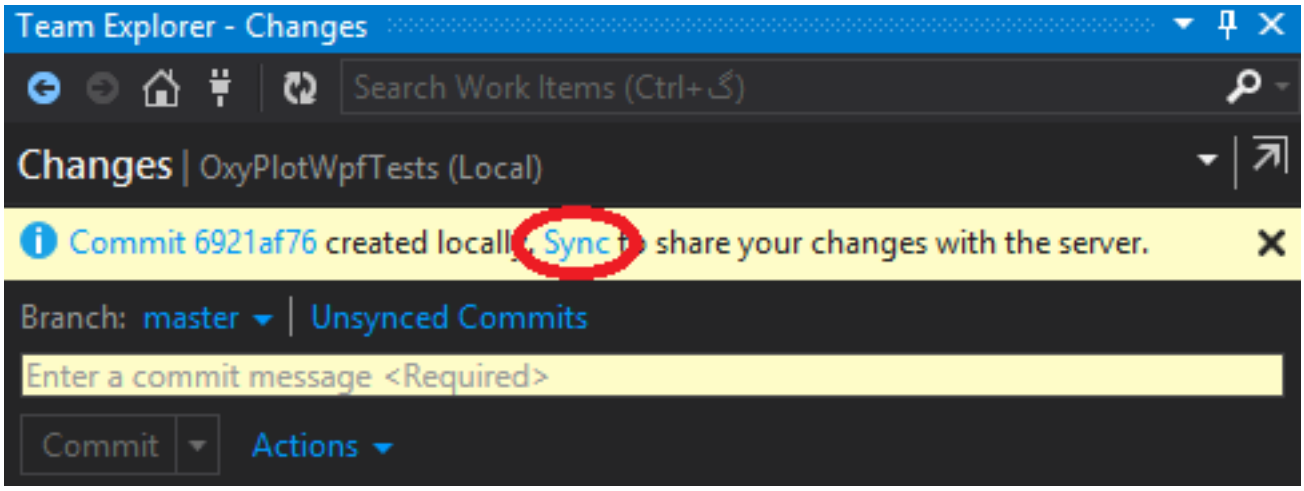
سپس نیاز است فایل‌های موجود را به مخزن کد GitHub ارسال کرد. بنابراین پس از مشاهده‌ی پیام موفقیت آمیز بودن عملیات همگام سازی، بر روی دکمه‌ی Home در نوار ابزار کلیک کرده و اینبار گزینه‌ی Changes را انتخاب کنید:

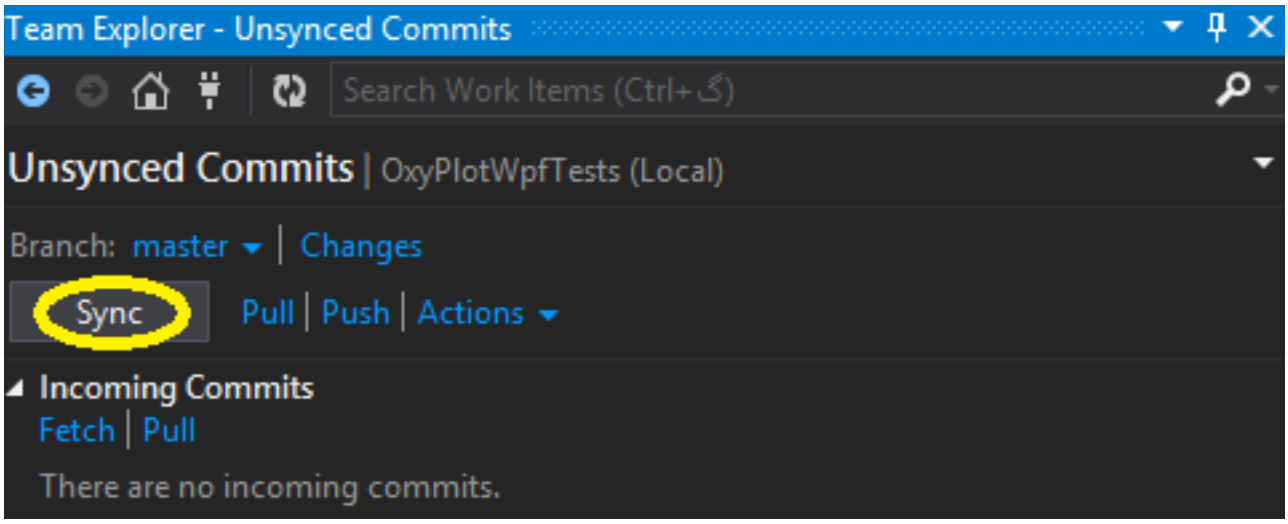


در اینجا پیام اولین ارسال را وارد کرده و سپس بر روی دکمه‌ی Commit کلیک کنید:

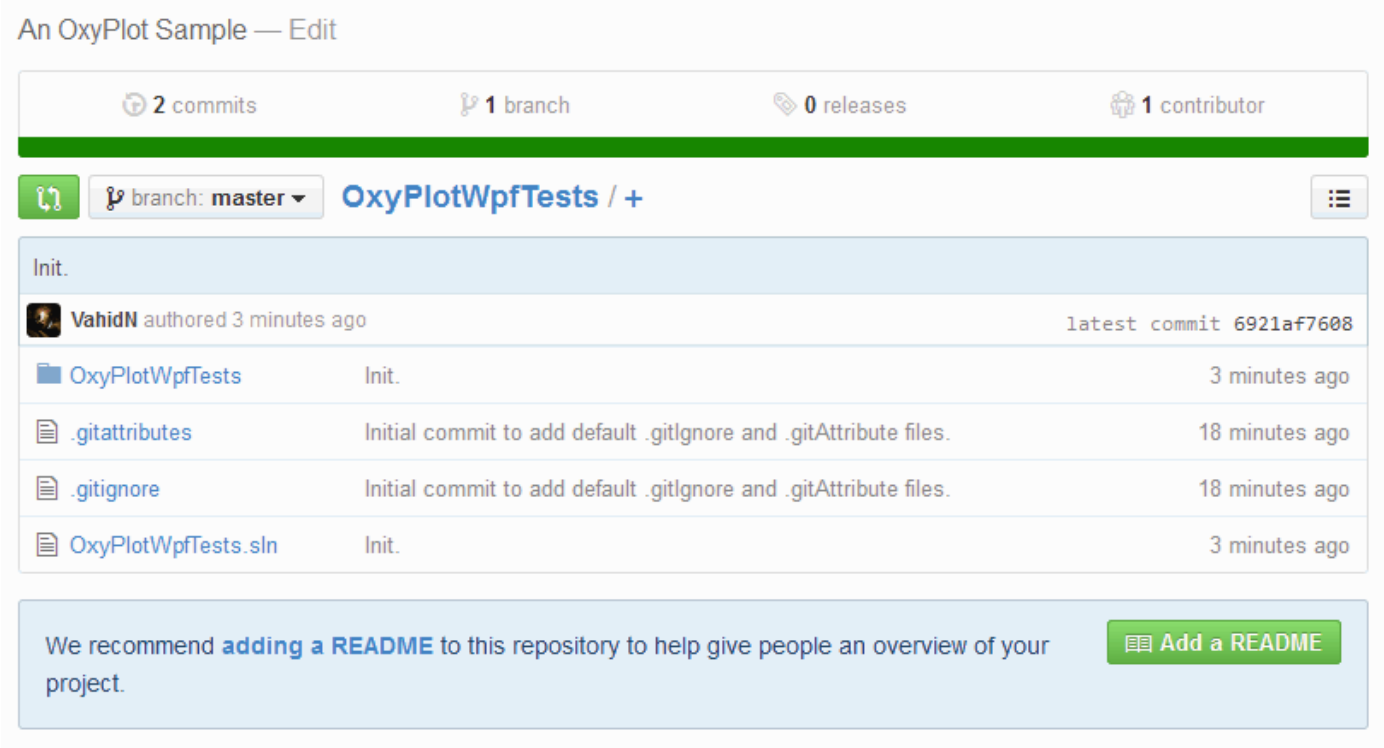


پس از مشاهده‌ی پیام موفقیت آمیز بودن commit محلی، نیاز است تا آن‌را با سرور نیز هماهنگ کرد. به همین جهت در اینجا بر روی لینک Sync کلیک کرده و در صفحه‌ی بعدی بر روی دکمه‌ی Sync کلیک کنید:





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



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 دقیقا 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: ۱۳۹۳/۱۰/۲۵ ۱۸:۴۵

میخواستم بدونم برای پروژهای که نمیخوایم 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: ۱۳۹۳/۱۰/۲۵ ۱۸:۵۷

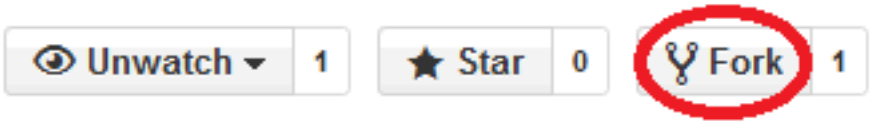
GitHub امکان تهیه مخزن کد خصوصی هم دارد ولی رایگان نیست . سایت BitBucket امکان ایجاد مخزن کد خصوصی رایگان را دارد؛ البته با محدودیت حداکثر 5 کاربر تعریف شده‌ی برای کار با یک مخزن.

عنوان: **نحوه‌ی مشارکت در پروژه‌های GitHub به کمک Visual Studio**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۰/۰۶ ۱۴:۱۰
آدرس: www.dotnettips.info
گروه‌ها: Git, Source control, GitHub

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 این‌کار را به کمک امکانات توکار VS.NET بررسی خواهیم کرد.

ایجاد یک Fork جدید در GitHub

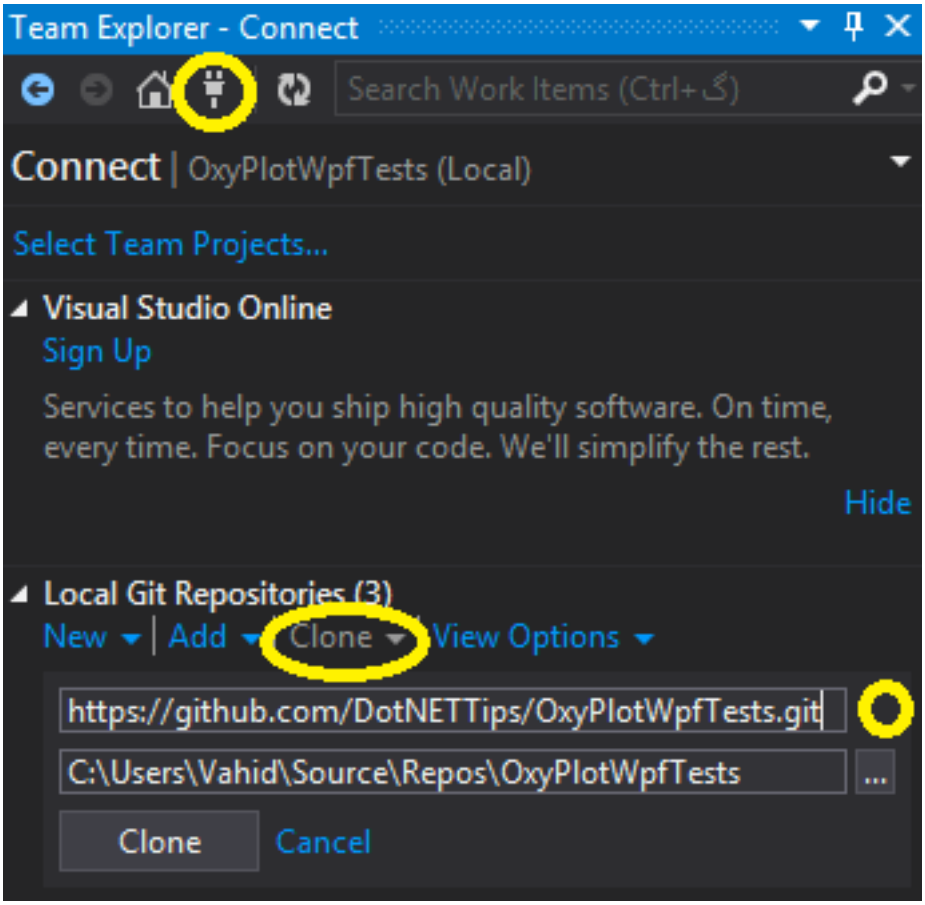
برای ارسال تغییرات انجام شده بر روی یک پروژه، نیاز است به صاحب یا مسئول آن مخزن در GitHub مراجعه و سپس درخواست دسترسی اعمال تغییرات را نمود. در این حالت، احتمال اینکه جواب منفی دریافت کنید، بسیار زیاد است. جهت مدیریت یک چنین مواردی، قابلیتی به نام ایجاد یک Fork پیش بینی شده‌است.



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.

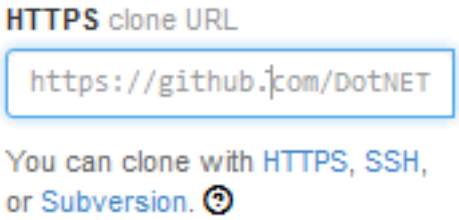
دریافت مخزن کد Fork شده از GitHub به کمک Visual Studio

پس از اینکه Fork جدیدی را از پروژه‌ای موجود ایجاد کردیم، نیاز است یک Clone یا کپی مطابق اصل آن‌را 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 نمائید. 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:

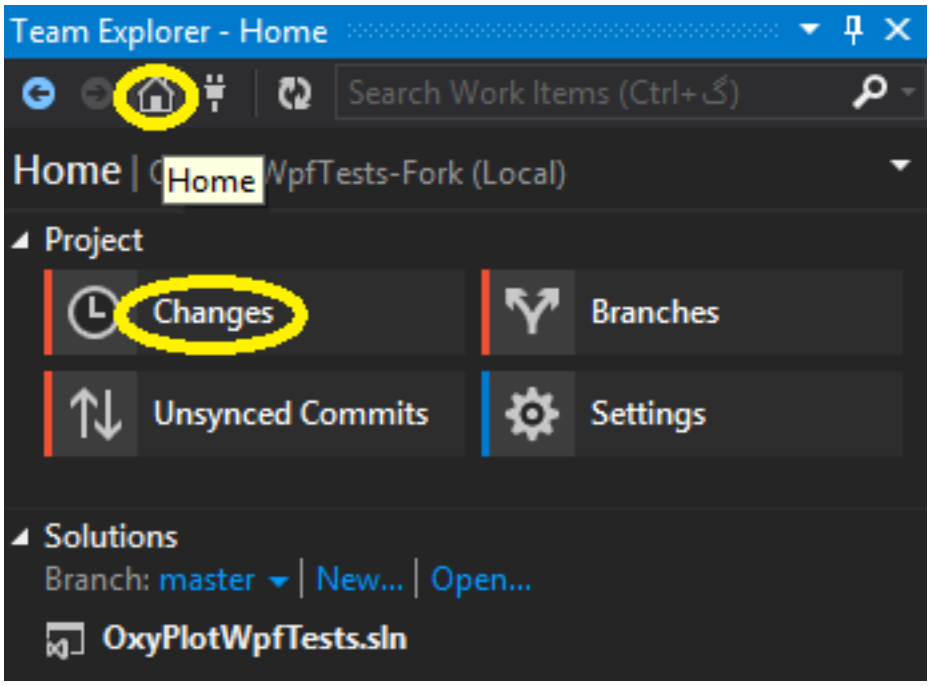


پس از تکمیل این دو آدرس، بر روی دکمه‌ی Clone کلیک نمائید. پس از پایان کار، اگر به آدرس محلی داده شده بر روی کامپیوتر خود مراجعه کنید، یک کپی از فایل‌های این مخزن، قابل مشاهده هستند.

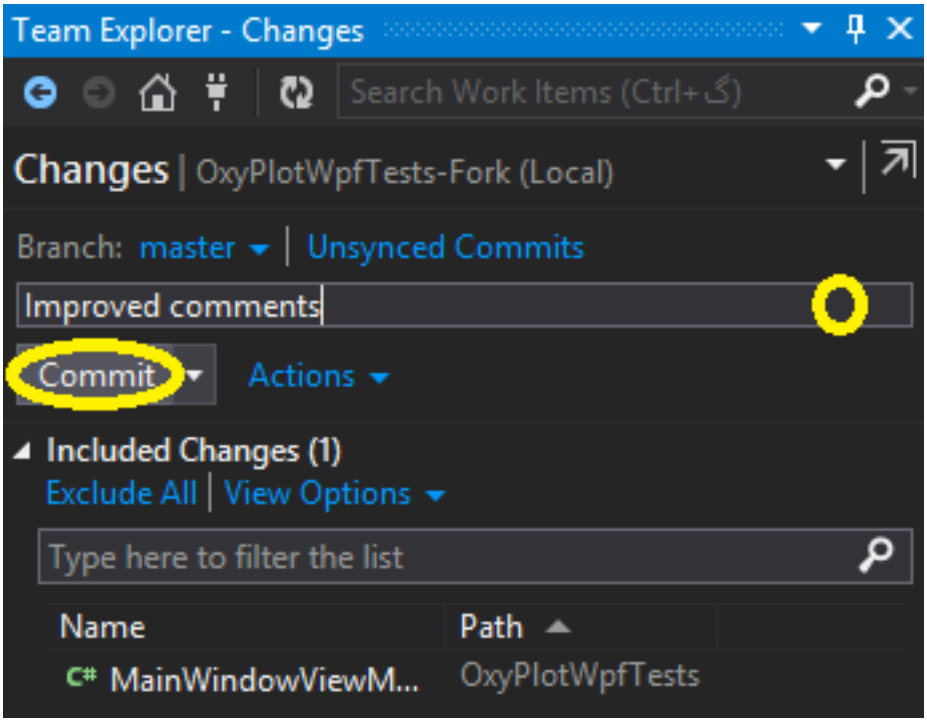
اعمال تغییرات محلی و ارسال آن به سرور GitHub

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

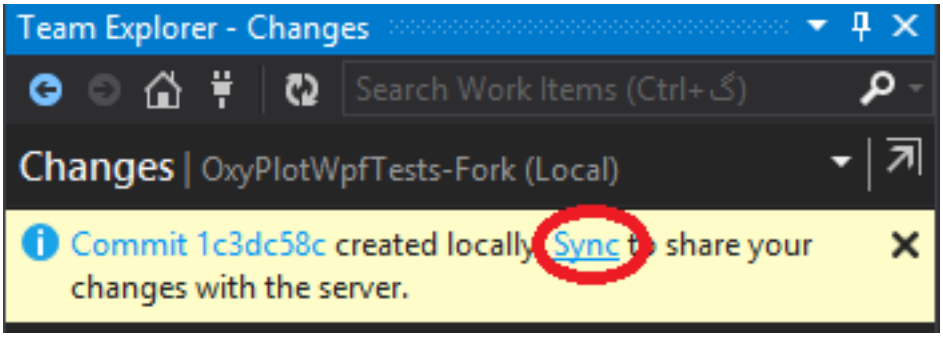
Changes را انتخاب نمائید:

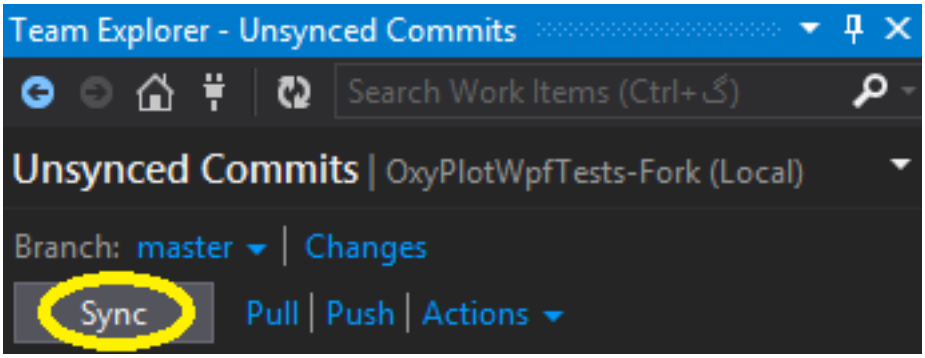


در اینجا توضیحاتی را نوشته و سپس بر روی دکمه‌ی Commit کلیک کنید.

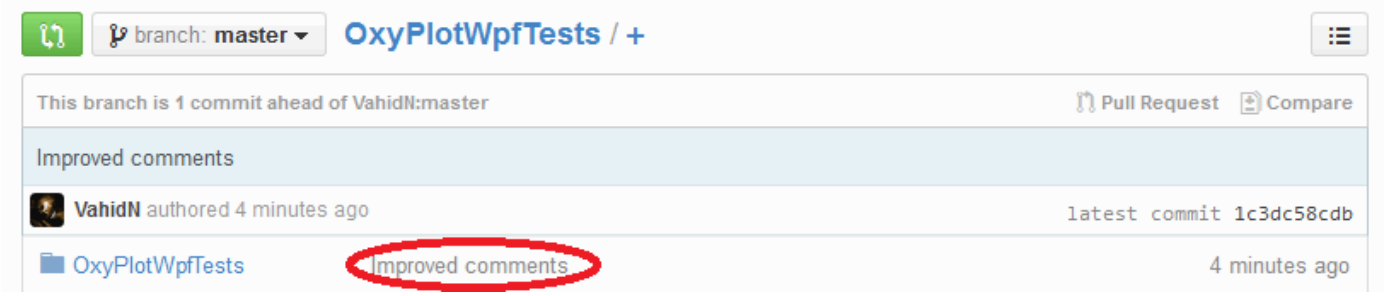


پس از هماهنگ سازی محلی، اکنون نوبت به هماهنگ سازی این تغییرات با مخزن کد GitHub است. بنابراین بر روی لینک Sync در پیام ظاهر شده کلیک کنید و در صفحه‌ی بعدی نیز بر روی دکمه‌ی Sync کلیک نمائید:





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

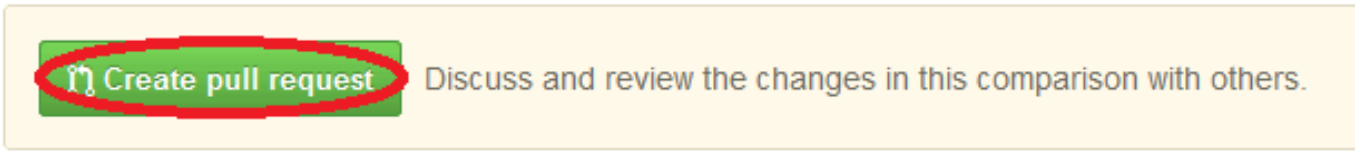


مطلع سازی صاحب اصلی مخزن کد از تغییرات انجام شده

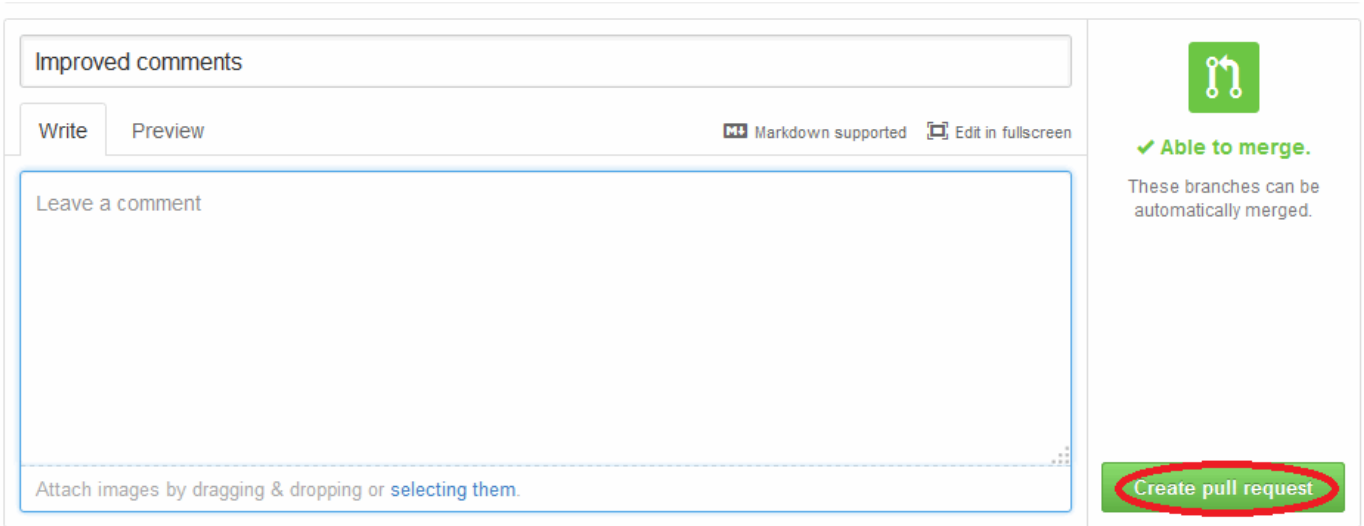
تا اینجا کسی از تغییرات جدید انجام شده‌ی توسط ما باخبر نیست. برای اطلاع رسانی در مورد این تغییرات، به مخزن کد Fork شده که اکنون تغییرات جدید به آن ارسال شده‌اند، مراجعه کنید. سپس در کنار صفحه بر روی لینک Pull request کلیک نمائید:

در اینجا بر روی دکمه‌ی New pull request کلیک کنید:

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

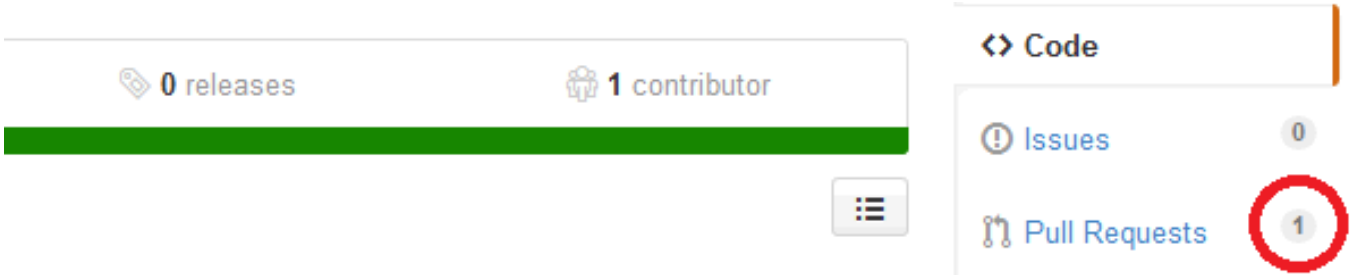


در اینجا عنوان و توضیحاتی را وارد کرده و سپس بر روی دکمه‌ی Create pull request کلیک نمائید:

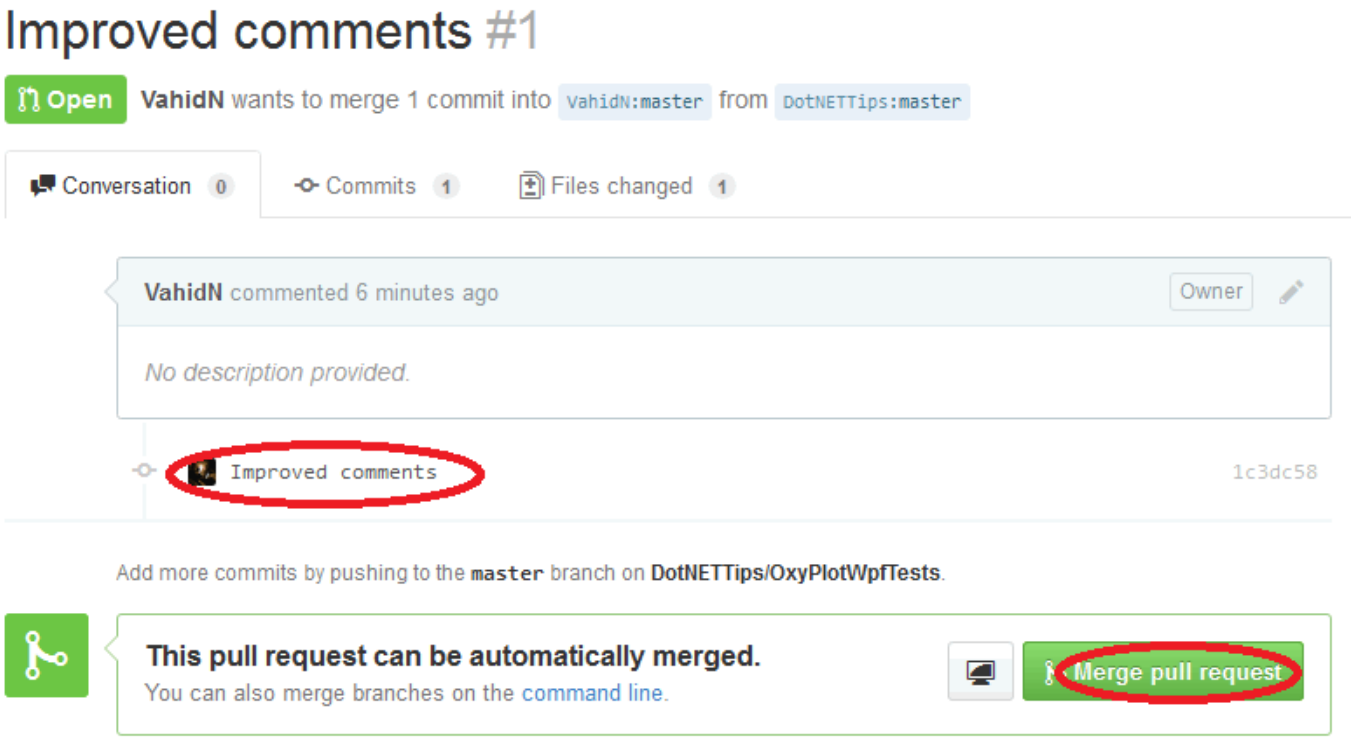


یکی سازی تغییرات با مخزن اصلی

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:

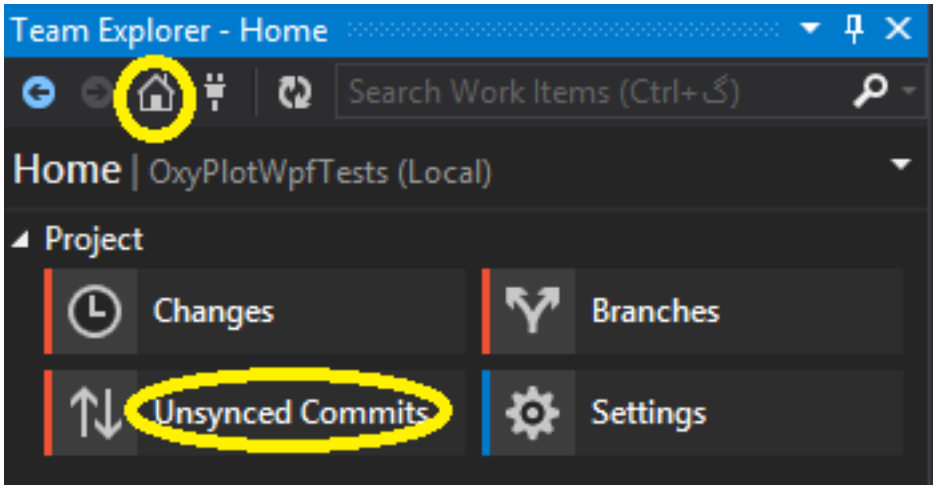


پس از انتخاب یکی از آن‌ها، لینکی برای بررسی تغییرات انجام شده و همچنین دکمه‌ای برای یکی سازی آن‌ها با پروژه‌ی اصلی وجود دارد:

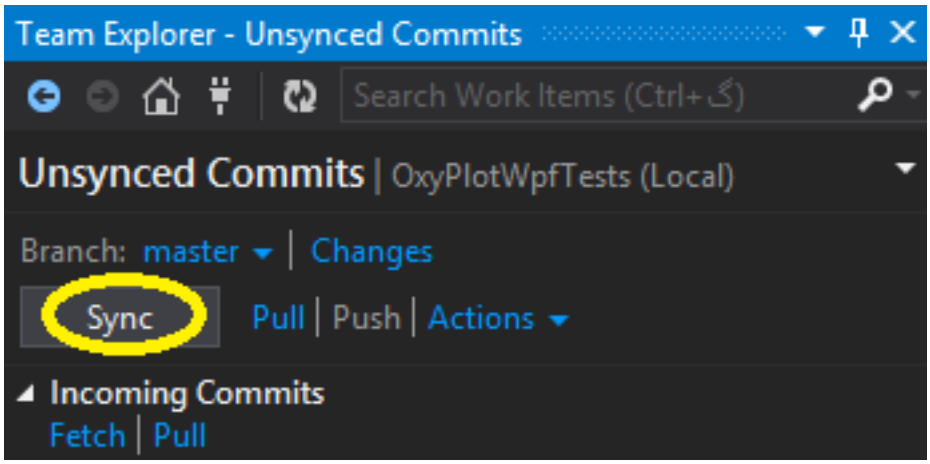


دریافت این تغییرات در مخزن کد محلی توسط صاحب اصلی پروژه

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 نمائید. در صفحه‌ی باز شده بر روی دکمه‌ی Sync کلیک نمائید. 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

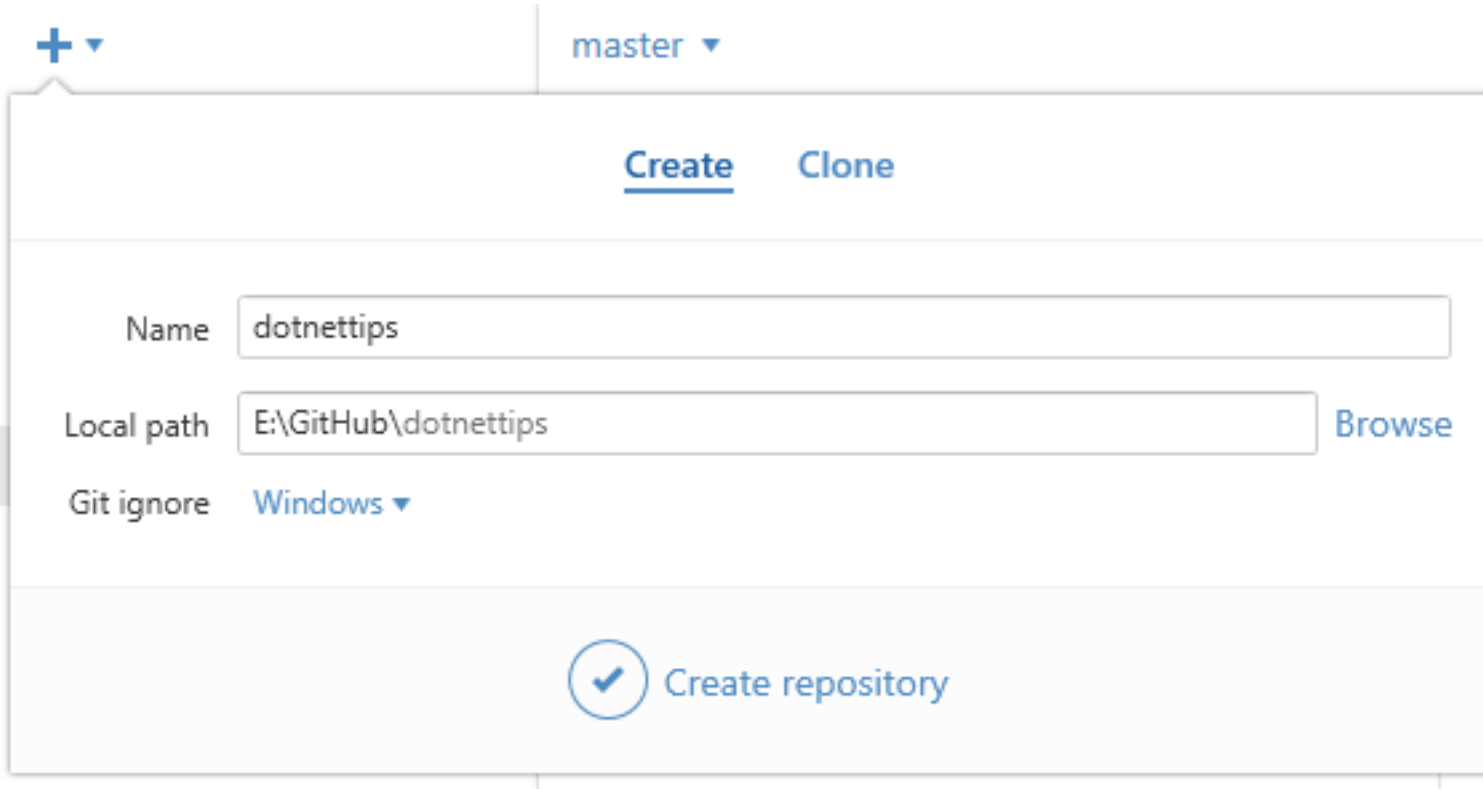
چطور می‌تونیم سورس خودمون رو با آخرین تغییرات انجام شده روی سورس اصلی sync کنیم؟

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

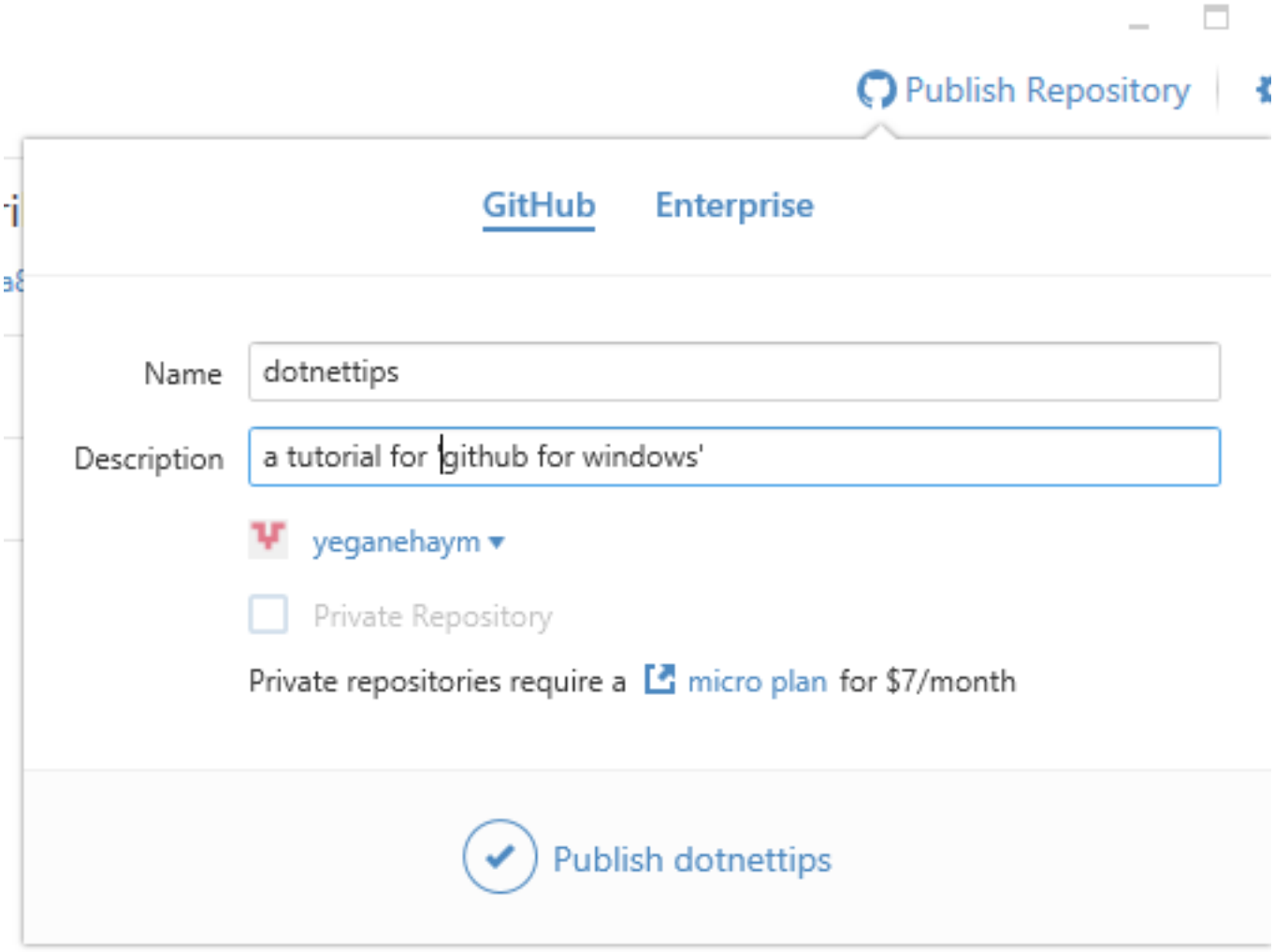
باید عملیات pull commits انجام شود؛ در همان برگه‌ی Unsynced commits .

عنوان: **ساده ترین روش کار با Github در ویندوز**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۰/۲۵ ۱:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Git, Windows, GitHub

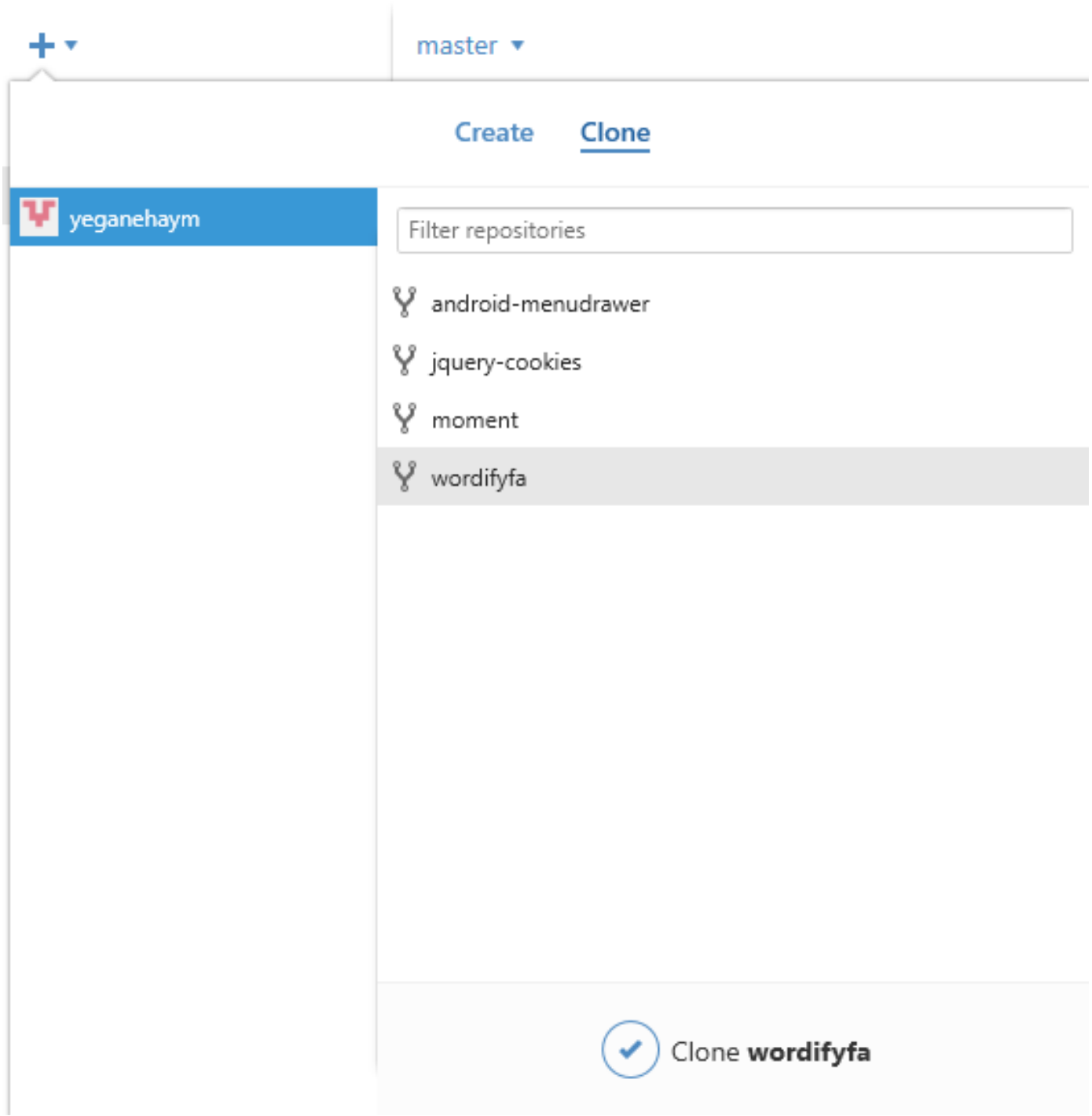
در این صفحه یک برنامه مختص ویندوز قرار داده شده است که شعار آن بدین شکل است :"کار با گیت هاب تا بحال تا این حد آسان نبوده است". موقعی که فایل را دانلود کنید، بعد از اجرا، شروع به دانلود و نصب برنامه اصلی خواهد کرد که در حال حاضر حجم فعلی آن حدود 45 مگابایت است. بعد از اینکه برنامه را نصب کرده و آن را 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.



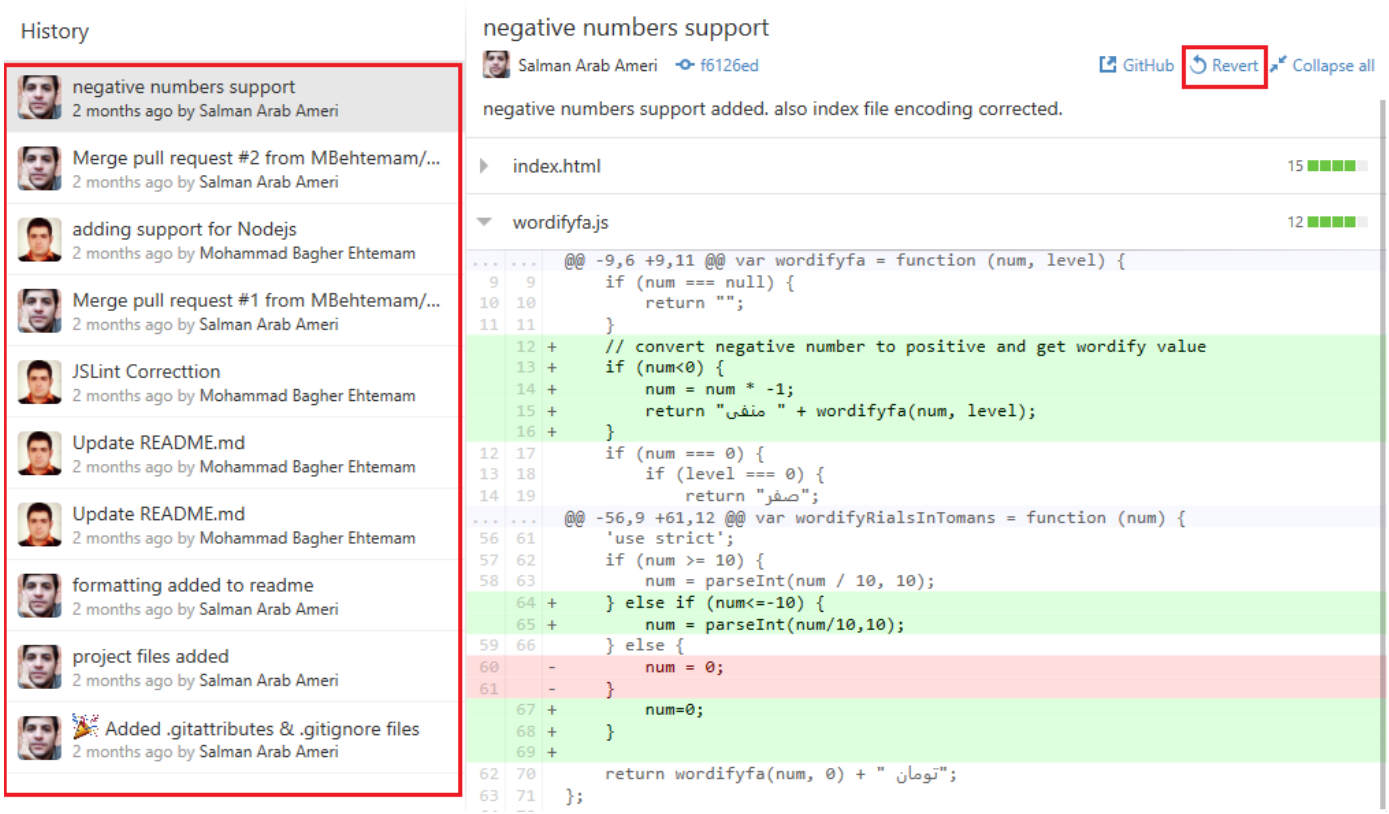
در صورتیکه فایل‌های شما برای انشار نهایی آماده هستند، پروژه خود را در لیست سمت چپ برنامه انتخاب کنید تا در بالا و سمت راست برنامه، گزینه‌ی Publish Repository دیده شود و با انتخاب آن، یک نام را که قبلا وارد کرده اید و یک توضیح مختصر را از شما می‌خواهد. به صورت پیش فرض انتشارها عمومی و رایگان هستند. در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



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 کپی خواهد شد.

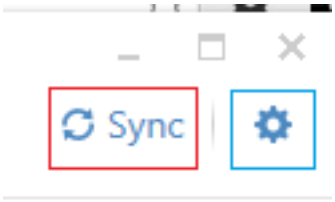


بعد از انتخاب گزینه‌ی Clone، از شما محل ذخیره‌ی پروژه را خواهد پرسید و بعد از تایید آن، مقدار زمان کمی برای کپی کردن پروژه خواهد خواست. پس از آن لیستی از همه‌ی تغییرات و مشارکت‌ها به شما نمایش داده می‌شود و در صورتیکه دوست دارید به تغییری در قبل برگردید تا کارتان را از آن شروع کنید، می‌توانید از گزینه‌ی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

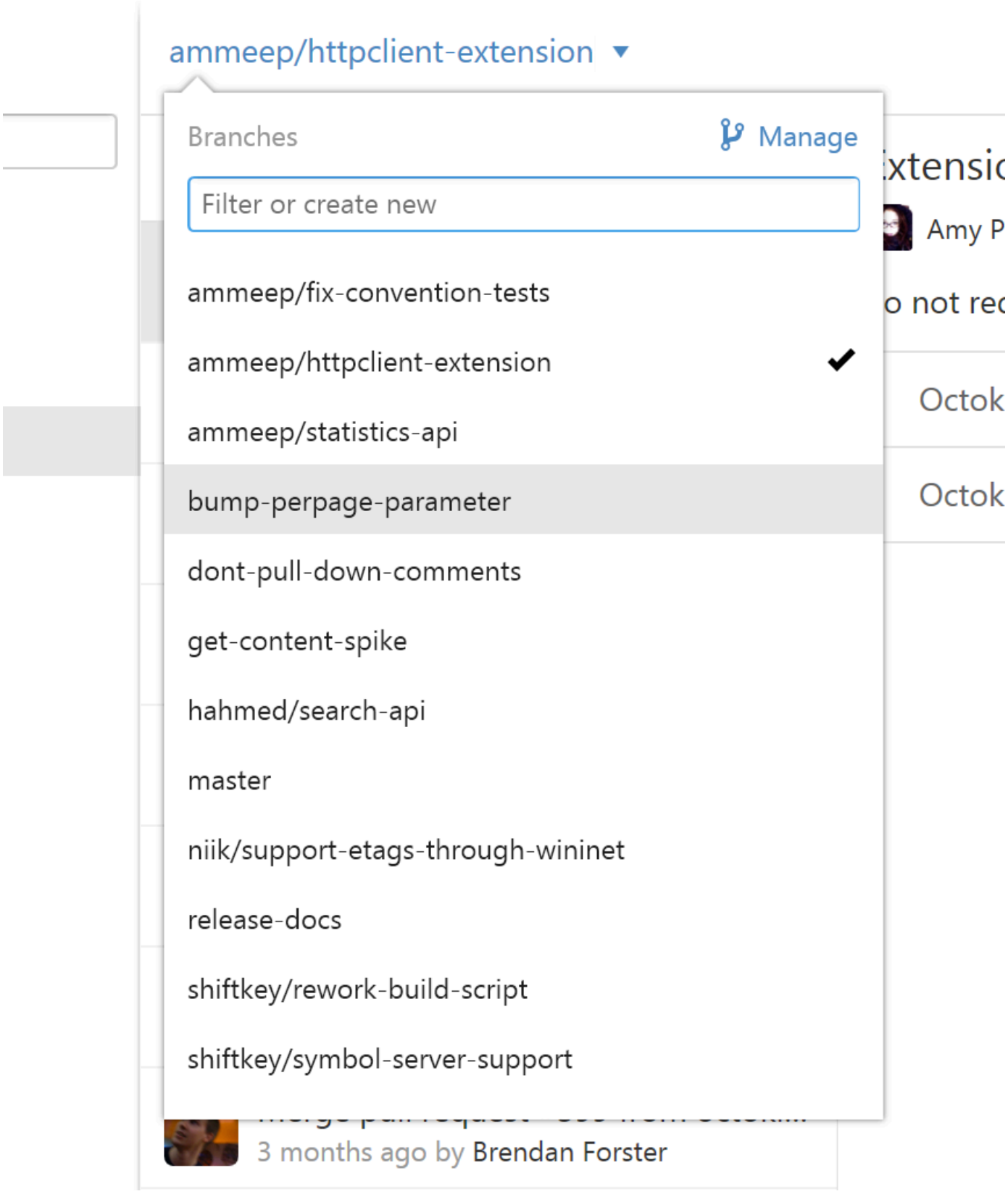


حال با خیال راحت روی پروژه کار کنید و تغییرات را روی آن اعمال کنید و بعد از اینکه کارتان تمام شد، دوباره به برنامه باز گردید و پروژه را در لیست انتخاب کرده و در سمت راست بالای صفحه، گزینه‌ی Sync Now را انتخاب کنید تا مشارکت جدید شما به سیستم گیت هاب اعمال شود و حالا اگر به صفحه‌ی پروژه در سایت گیت هاب 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

عنوان: **آشنایی با ساختار یک Pull Request خوب**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۱۳ ۱۶:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Git, Open Source, GitHub

در مطلب « نحوه‌ی مشارکت در پروژه‌های GitHub به کمک Visual Studio » با مفهوم pull request آشنا شدیم. اما ... یک pull request خوب چه خصوصیاتی دارد و فرهنگ ارسال یک PR خوب چیست؟

اخلاق مشارکت در یک پروژه‌ی سورس باز

بعضی از توسعه دهنده‌ها در حین مشارکت در یک پروژه‌ی سورس باز، برای مثال جهت افزودن قابلیتی 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 قابلیتی جدید، باید نحوه‌ی کدنویسی فعلی را حفظ کنید. از این جهت که نگهدارنده‌ی اصلی پروژه، پیش از شما این‌کار را شروع کرده‌است و زمانیکه شما به پروژه‌ای دیگر رجوع خواهید کرد، باز نیز باید همین کار را ادامه دهد.
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 requestها را کوچک نگه‌دارید

برای اینکه شانس قبول شدن PR خود را بالا ببرید، حجم و تمرکز آن‌را کوچک نگه دارید. بسیاری از توسعه دهنده‌های سورس باز اگر با یک PR حجیم روبرو شوند، آن‌را 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

شبیه 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، حتما آن‌ها را یکبار بررسی کنید تا مطمئن شوید جای دیگری را به هم نریخته‌اید. ضمنا 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: ۱۳۹۳/۱۲/۱۴ ۱۹:۱۱

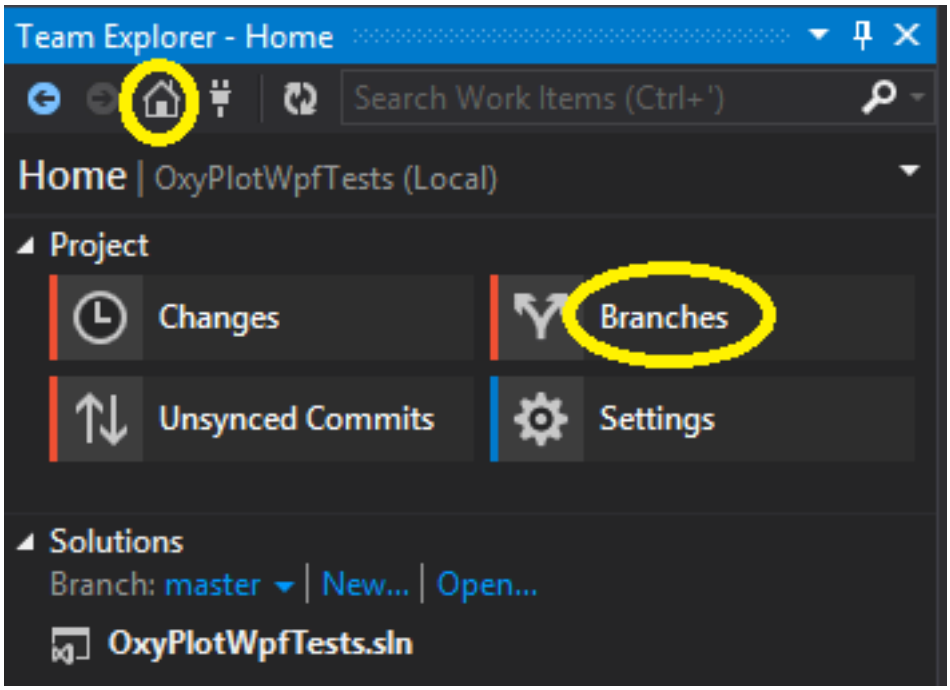
کاش زودتر خونده بودمش :P

عنوان: کار با شاخه‌ها و انشعابات Git در Visual Studio
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۱۵ ۱۴:۲۰
آدرس: www.dotnettips.info
گروه‌ها: Git, GitHub, OpenSource

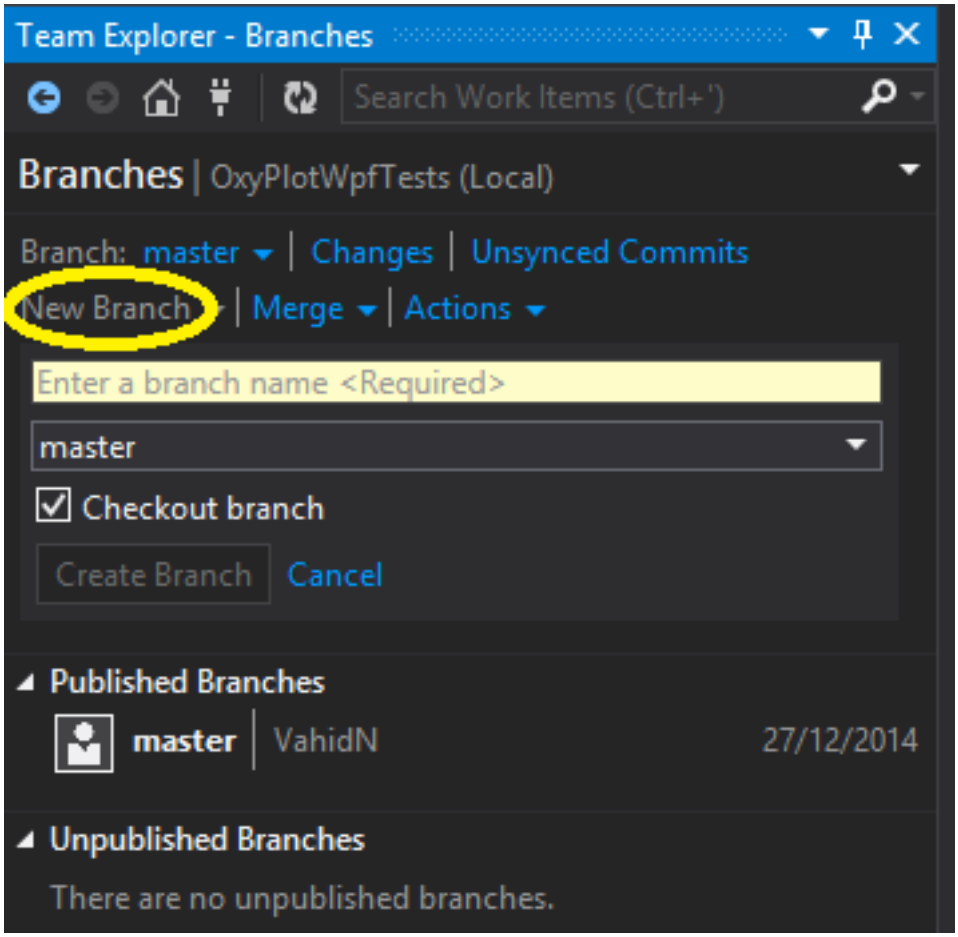
در مطلب « آشنایی با ساختار یک Pull Request خوب » عنوان شد که قابلیت‌های جدید و یا رفع مشکلات را در شاخه‌ی اصلی کار نکنید. اما ... چگونه؟

ایجاد یک شاخه‌ی جدید در Visual Studio و انتشار آن

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



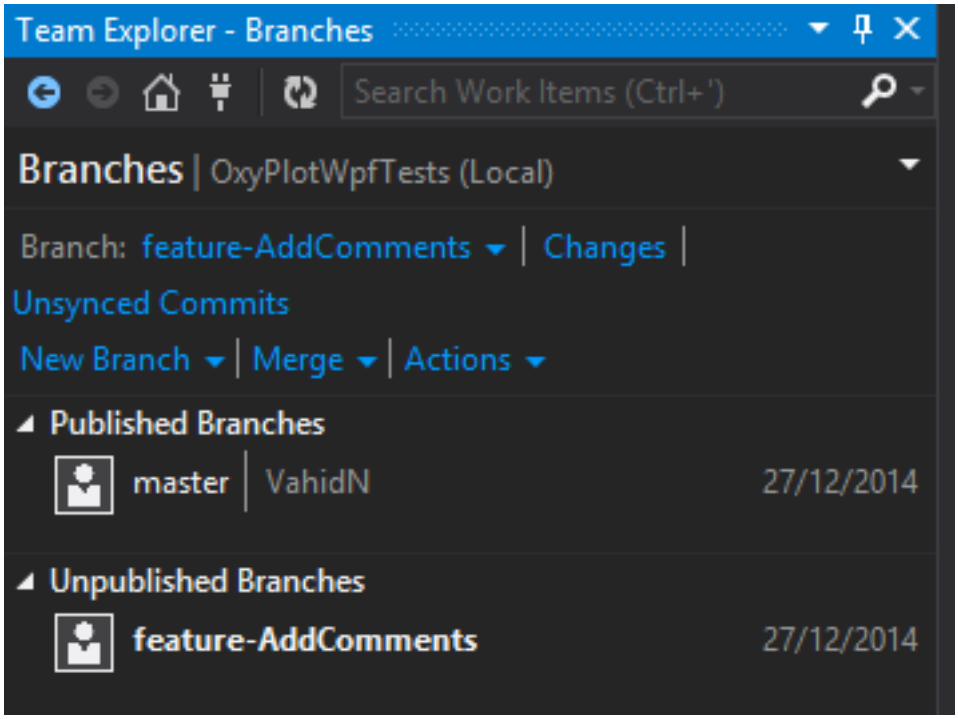
در برگه‌ی باز شده، انشعاب و شاخه‌ی جاری با فونت ضخیم نمایش داده می‌شود. برای مثال در اینجا، انشعاب کاری همان master است:



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 و یا fix-Y استفاده کنید (افزودن قابلیت X و یا رفع مشکل Y) و در آخر بر روی دکمه‌ی Create branch کلیک کنید.

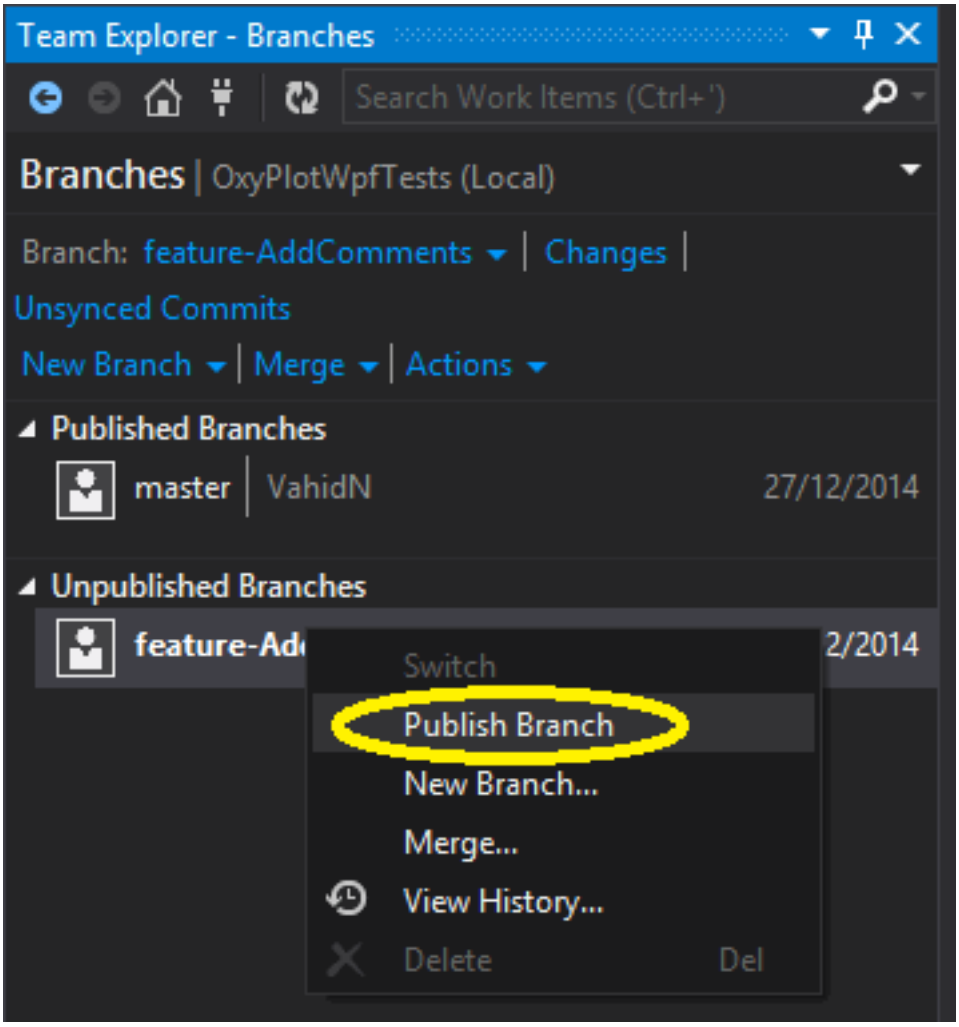
در اینجا می‌توان مشخص کرد که انشعاب ایجاد شده باید بر اساس کدام انشعاب فعلی نیز تهیه شود (دراپ داون ذیل قسمتی که می‌توان نام انشعاب را وارد کرد). برای مثال پروژه‌های مایکروسافت در GitHub، دارای سه شاخه‌ی master، dev و release هستند. شاخه‌ی dev (یا توسعه) جایی است که انشعابات pull requests را قبول خواهند 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:

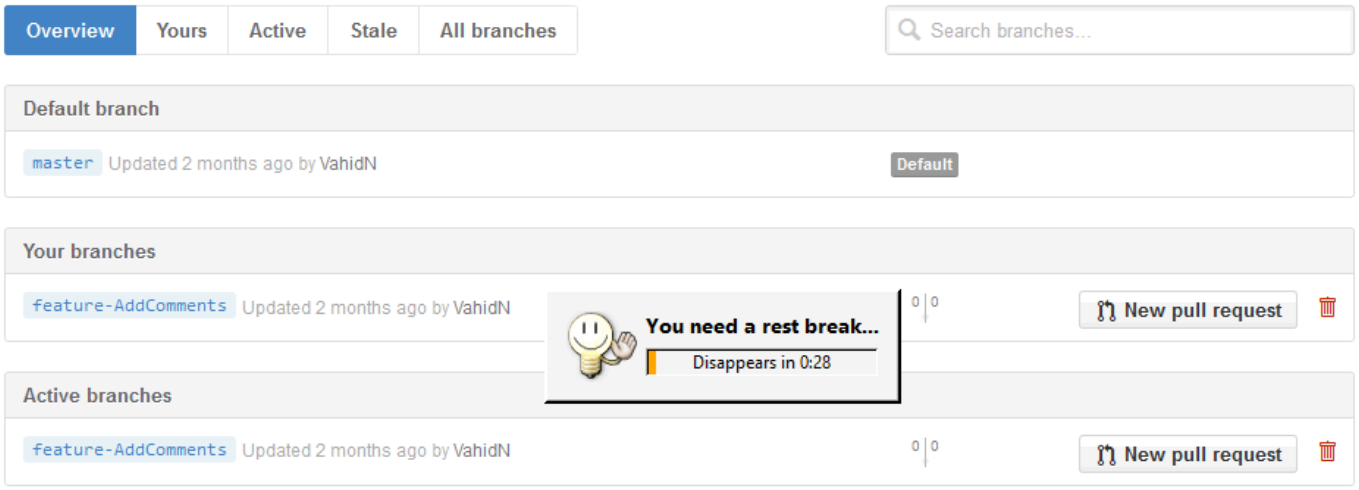


همانطور که ملاحظه می‌کنید، اینبار شاخه‌ی جدید ایجاد شده به صورت bold و ضخیم نمایش داده شده‌است. این bold بودن به معنای شاخه‌ی کاری جاری بودن است. همچنین این شاخه در قسمت unpublished branches قرار دارد. بنابراین کلیه‌ی تغییرات واقع شده‌ی در آن، محلی بوده و هنوز با سرور هماهنگ نشده‌اند.

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



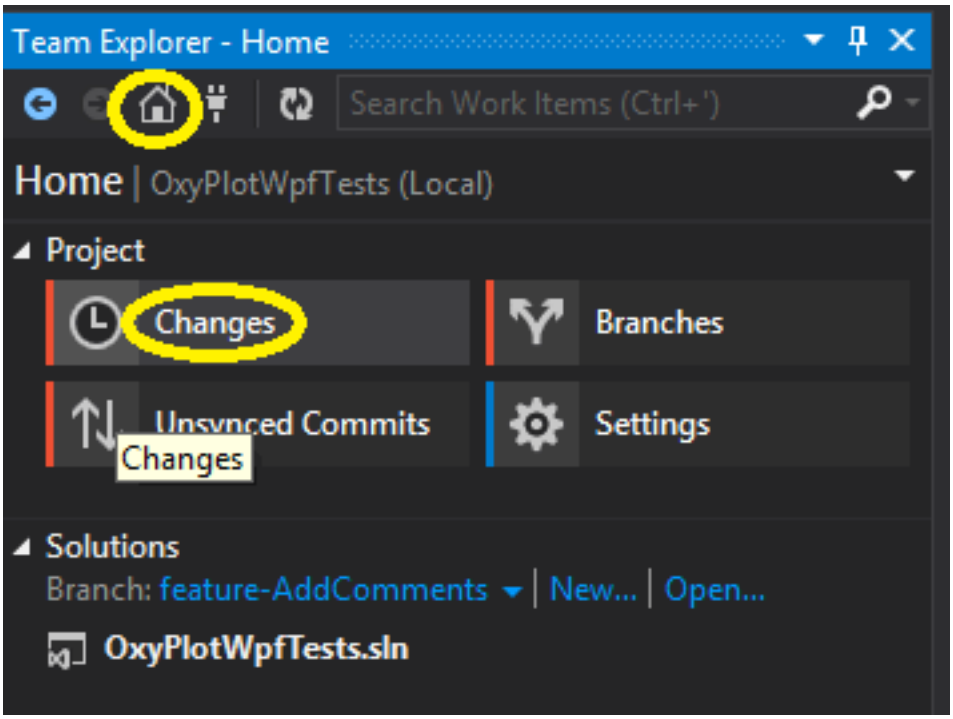
این انتشار سبب نمایش لیستی از تغییرات جدید در برگه‌ی branches پروژه، در GitHub خواهد شد:



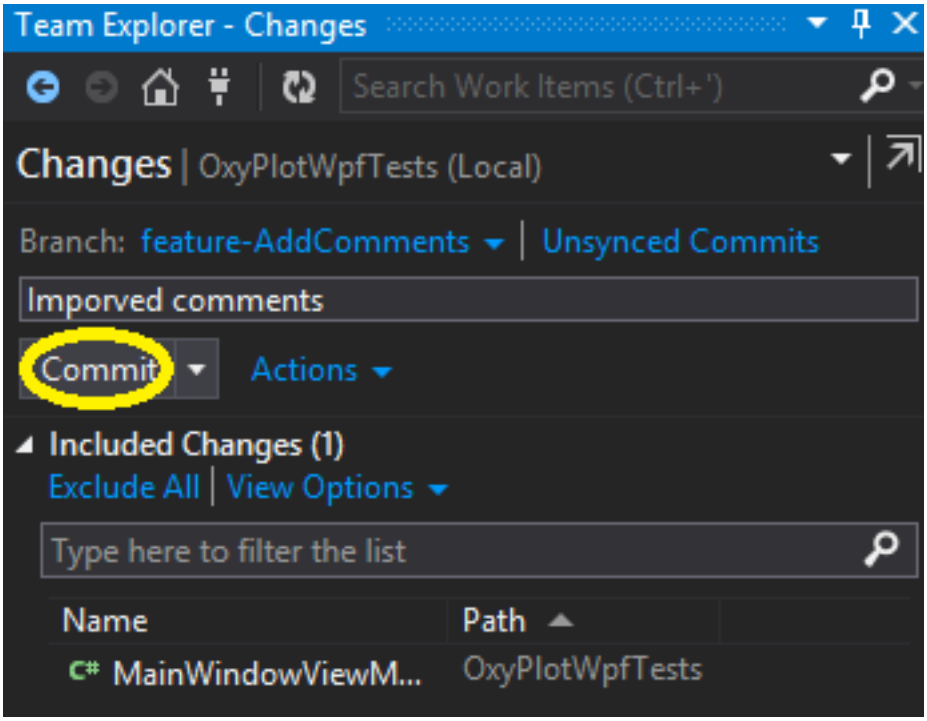
یک نکته: برای تغییر branch فعال جاری، فقط کافی است در برگه‌ی branches در ویژوال استودیو، دوبار بر روی لینک نام آن شاخه کلیک کنید تا به صورت bold ظاهر شود.

ارسال تغییرات انجام شده‌ی در Branch به سرور

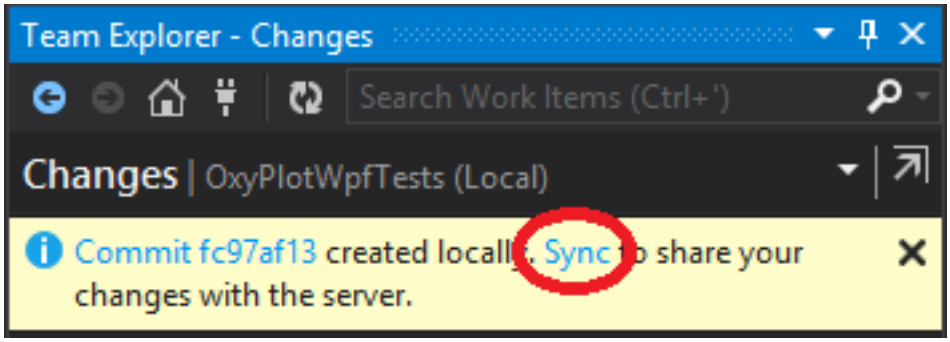
پس از کار بر روی شاخه‌ی جدید ایجاد شده، اکنون نوبت به ارسال و هماهنگ سازی این تغییرات با سرور است. این مورد نیز همانند قبل بوده و ابتدا باید به برگه‌ی Home و گزینه‌ی changes آن مراجعه کرد:

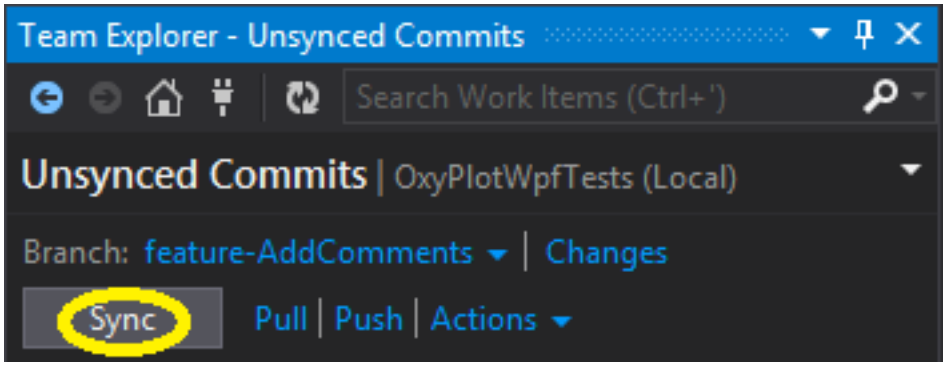


و سپس تغییرات را به همراه توضیحی commit کرد:



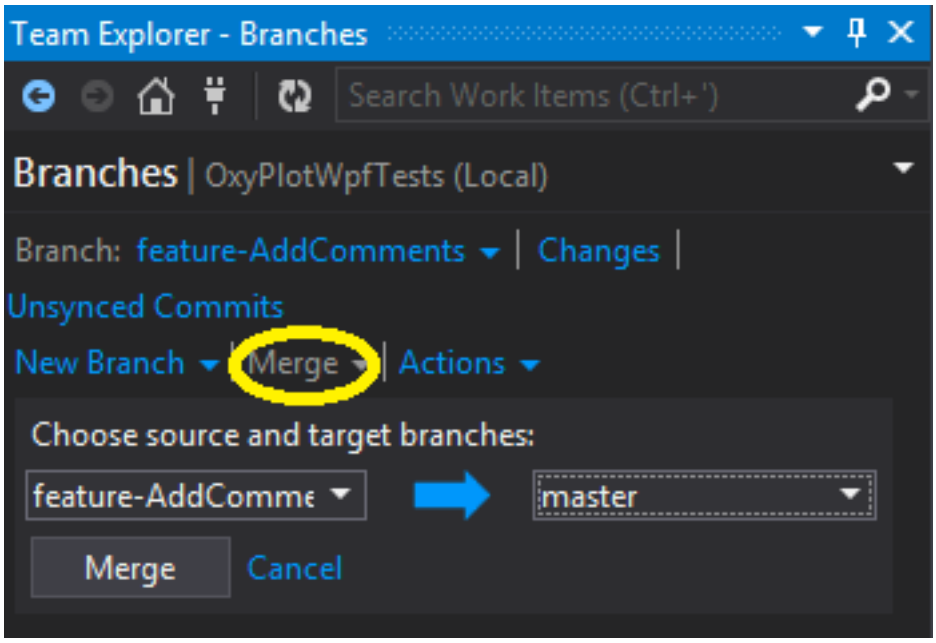
اینکار سبب sync محلی می‌شود. سپس بر روی لینک sync کلیک نمائید و تغییرات را با سرور هماهنگ کنید.



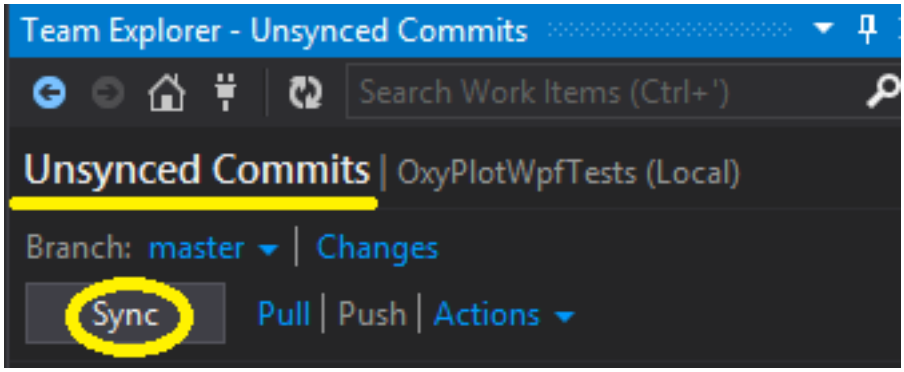


یکی کردن تغییرات شاخه‌ی جدید با شاخه‌ی اصلی

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.



عنوان: **تهیه‌ی گزارش از آخرین ارائه‌های یک پروژه در GitHub**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۱/۰۳ ۱۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: JSON, Software deployment, json.net, GitHub

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 هربار 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 آن‌را 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 نتی 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 ، 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.

عنوان: **مفاهیم پایه سیستم های کنترل نسخه؛ قسمت اول : گیت**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۱۳ ۱۴:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Git, SVN, Source control, GitHub

در این مقاله با دو سیستم کنترل نسخه git و SVN آشنا شده و تفاوت‌های آن‌ها را برای تازه‌کاران بررسی می‌کنیم. ایده اولیه نوشتن این مقاله زمانی بود که برای یک پروژه‌ای، اعضای تیم ما دور هم جمع شده و در مورد ابزارهای مورد استفاده بحث کردند و یک عده از گیت و عده‌ای از SVN صحبت می‌کردند. بر این شدم که مقاله‌ای نوشته و ابتدا به معرفی آن‌ها و سپس به مزایا و معایب هر کدام بپردازیم.

امروزه، استفاده از سیستم‌های کنترل نسخه (*Version Control System*) رواج زیادی پیدا کرده است. این سیستم‌ها به شما اجازه می‌دهند تا تغییراتی را که در پروژه ایجاد می‌شوند، ضبط و ثبت کرده تا از تغییراتی که در سطح پروژه اتفاق می‌افتد آگاه شوید. با ذکر یک نمونه این تعریف را باز میکنم:

شما به صورت تیمی در حال انجام یک پروژه هستید و باید نسبت به تغییراتی که اعضای تیم در یک پروژه می‌دهند، آگاه شوید. هر برنامه نویس بعد از انجام تغییرات باید این تغییرات را در سیستم کنترل نسخه به روز کند تا بتوان به سوالات زیر پاسخ داد: آیا اگر در بین راه به مشکل برخوردید می‌توانید پروژه خود را به یک یا چند گام عقب‌تر برگردانید؟ آیا می‌توانید به هر یک از اعضاء تیم دسترسی‌هایی را به قسمت هایی از پروژه تعیین کنید؟ می‌توانید تفاوت فایل‌های تغییر یافته را بیابید؟ آیا میتوان خطاهای یک برنامه را گزارش داد و به بحث در مورد آن پرداخت؟ چه کسی کدها را تغییر داده است؟ روند کار و تغییرات به چه صورت است؟ (این مورد برای به روز کردن نمودارهای burndown در توسعه چابک می‌تواند بسیار مفید باشد.)

پی نوشت: نه تنها در یک تیم بلکه بهتر هست در یک کار انفرادی هم از این سیستم‌ها استفاده کرد تا حداقل بازبینی روی پروژه‌های شخصی خود هم داشته باشیم.

سیستم کنترل گیت: این سیستم در سال 2005 توسط لینوس تورووالدز خالق لینوکس معرفی شد و از آن زمان تاکنون یکی از پر استفاده‌ترین سیستم‌های کنترل نسخه شناخته شده است. ویکی پدیا گیت را به این شکل تعریف می‌کند: « *یک سیستم بازبینی توزیع شده با تاکید بر جامعیت داده‌ها، سرعت و پشتیبانی جهت توزیع کار.* »

از معروف‌ترین سیستم‌های هاستینگ که از گیت استفاده می‌کنند، می‌توان به گیت هاب اشاره کرد.

اکثر سیستم‌های هاستینگ گیت، دو حالت را ارائه می‌دهند: عمومی : در این حالت کدهای شما به عموم بازدیدکنندگان نمایش داده می‌شود و دیگران هم می‌توانند در تکمیل و ویرایش کدهای شما مشارکت کنند و این امکان به صورت رایگان فراهم است. سیستم گیت هاب به دلیل محبوبیت زیادی که دارد، در اکثر اوقات انتخاب اول همه کاربران است. خصوصی: در این حالت کد متعلق به شما، یا شرکت یا تیم نرم افزاری شماست و غیر از افراد تعیین شده، شخص دیگری به کدهای شما دسترسی ندارد. اکثر سیستم‌های مدیریتی این مورد را به صورت premium پشتیبانی می‌کنند. به این معنا که باید اجاره آن را به طور ماهانه پرداخت کنید. سیستم گیت هاب ماهی پنج دلار بابت آن دریافت می‌کند. سیستم دیگری که در این زمینه محبوبیت دارد سیستم BitBucket هست که که اگر تیم شما کوچک است و در نهایت پنج نفر هستید، می‌توانید از حالت خصوصی به طور رایگان استفاده کنید ولی اگر اعضای تیم شما بیشتر شد، باید هزینه‌ب اجاره آن را که از 10 دلار آغاز می‌گردد، به طور ماهیانه پرداخت کنید.

پی نوشت: میتوانید از سیستم‌های متن باز رایگان هم که قابل نصب بر روی هاست ها هم هستند استفاده کنید که در این حالت تنها هزینه هاست یا سرور برای شما می‌ماند.

در سیستم گیت اصطلاحات زیادی وجود دارد: **Repository یا مخزن: برای** هر پروژه‌ای که ایجاد می‌شود، ابتدا یک مخزن ایجاد شده و کدها داخل آن قرار می‌گیرند. کاربرانی که قصد تغییر پروژه را دارند باید یک مخزن جداگانه ایجاد کنند تا بعدا تمامی تغییرات آن‌ها را روی پروژه‌ی اصلی اعمال کنند.

Fork: هر کاربری که قصد تغییر را بر روی سورس کدی، داشته باشد، ابتدا باید پروژه‌ی نویسنده اصلی پروژه را به یک مخزنی که متعلق به خودش هست انتقال دهد. به این عمل Fork کردن می‌گویند. حال کاربر تغییرات خودش را اعمال کرده و لازم هست که این تغییرات با پروژه‌ی اصلی که به آن Master می‌گوییم ادغام شوند. بدین جهت کاربر فرمان pull request را می‌دهد تا به نویسنده‌ی اصلی پروژه این موضوع اطلاع داده شود و نویسنده‌ی اصلی در صورت صلاحدید خود آن را تایید کند.

Branching یا شاخه بندی: نویسنده‌ی مخزن اصلی می‌تواند با مفهومی با نام شاخه بندی کار کند. او با استفاده از این مفهوم، پروژه را به قسمت یا شاخه‌های مختلف تقسیم کرده و همچنین با ایجاد دسترسی‌های مختلف به کاربران اجازه تغییرات را بدهد. به عنوان مثال بخش‌های مختلف پروژه از قبیل بخش منطق برنامه، داده ها، رابط کاربری و ... می‌تواند باشد. بعد از انجام تغییرات روی یک شاخه می‌توانید درخواست merge ادغام شدن یا کل پروژه را داشته باشید. در عمل شاخه بندی، هیچ کدام از شاخه‌های بر

روی یک دیگر تاثیر یا دخالتی ندارند و حتی می‌توانید چند شاخه را جدا از بخش master با یکدیگر ادغام کنید.

به غیر از ارتباط خط فرمانی که میتوان با گیت هاب برقرار کرد، میتوان از یک سری ابزار گرافیکی خارجی هم جهت ایجاد این ارتباط، استفاده کرد: GitHub For Windows : نسخه‌ی رسمی است که از طرف خود گیت هاب تهیه گردیده است و استفاده از آن بسیار راحت است. البته یک مشکل کوچک در دانلود آن وجود دارد که دانلود آن از طریق یک برنامه‌ی جداگانه صورت گرفته و اصلا سرعت خوبی جهت دانلود ندارد. Visual Studio .Net : (+) خود ویژوال استودیو شامل سیستمی به اسم Microsoft Git Provider است که در بخش تنظیمات می‌توانید آن را فعال کنید (به طور پیش فرض فعال است) و به هر نوع سیستم گیتی می‌توانید متصل شوید. تنها لازم است که آدرس Url گیت را وارد کنید. SourceTree : از آن دست برنامه‌های محبوبی است که استفاده آسانی دارد و خودم به شخصه از آن استفاده می‌کنم. شامل دو نسخه‌ی ویندوز و مک است و میتوانید با چندین سیستم گیت مثل «گیت هاب» و «بیت باکت» که در بالا به آن‌ها اشاره شد، به طور همزمان کار کند.

نظرات خوانندگان

نویسنده: سید محمد حسین موسوی
تاریخ: ۱۳۹۴/۰۵/۱۴ ۰:۵۱

سلام؛ خیلی ممنون. چندتا سوال :
«پی نوشت: نه تنها در یک تیم بلکه بهتر هست در یک کار انفرادی هم از این سیستم‌ها استفاده کرد تا حداقل بازبینی روی پروژه‌های شخصی خود هم داشته باشیم.»
-1این یعنی اینکه اگر من بخوام برای خودم هم به تنهایی استفاده کنم و خصوصی هم باشه باید پول بدم؟ حالا اگر عمومی باشه می‌تونم به هیچ کس اجازه دستری ندم؟ فرق عمومی که اجازه دسترسی ندی با خصوصی تو چیه؟ دیدن و ندیدن کدها ؟
2-team foundation ماکروسافت هم برای اینکارهاست؟
-3می شه کمی بیشتر در این مورد توضیح بدید؟
«پی نوشت: میتوانید از سیستم‌های متن باز رایگان هم که قابل نصب بر روی هاست‌ها هم هستند استفاده کنید که در این حالت تنها هزینه هاست یا سرور برای شما می‌ماند.»

نویسنده: محسن خان
تاریخ: ۱۳۹۴/۰۵/۱۴ ۱:۹

بحث git با هاست‌های عمومی git مثل github متفاوت هست. شما خودت هم می‌تونی یک هاست git راه اندازی کنی: راه اندازی سرور Git با استفاده از Bonobo Git Server و انتقال از ساب ورژن به گیت

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۱۴ ۱:۳۰

مبحث TFS کاملا با مباحث سیستم‌های کنترل نسخه متفاوت است و یک سیستم ALM به حساب میاد نه VCS

فرقی نمی‌کند، پروژه عمومی همیشه نمایش داده می‌شود، این دسترسی‌ها مربوط به شاخه بندی پروژه است که چه کسانی بتوانند تا چه حدی روی هر شاخه تغییرات را اعمال کنند ولی بحث خصوصی سازی نیاز به پرداخت هزینه دارد. هنگامی که در گیت هاب پروژه خودتون رو به صورت عمومی انتخاب کنید هیچ گزینه اضافی ندارد ولی وقتی روی خصوصی تنظیم کنید با مجموعه‌ای از آیکن‌های کارت‌های اعتباری روبرو می‌شوید.

همینطور که دوست عزیزمان "محسن خان" گفتند شما میتوانید از طریق یک سیستم متن باز و رایگان به ایجاد یک سیستم گیت جداگانه (شخصی)اقدام کنید و تنها لازم است هزینه هاستی که خریدید را به سرویس دهنده هاست پرداخت کنید.

عنوان: **مفاهیم پایه سیستم های کنترل نسخه؛ قسمت سوم : جمع بندی**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۴/۰۵/۱۸ ۲۱:۲۰
آدرس: www.dotnettips.info
گروه‌ها: Git, SVN, Source control, GitHub

در اولین قسمت این سری، گیت و در قسمت دوم ، SVN را بررسی کردیم؛ در این مقاله قصد داریم یک جمع بندی از این دو مقاله داشته باشیم.

احتمالا در مورد این دو سیستم حرف‌های زیادی شنیده‌اید و احتمالا بیشتر آن‌ها در مورد گیت نظر مساعدتری داشته‌اند؛ ولی تفاوت‌هایی بین این دو سیستم هست که باید به نسبت هدف و نیازی که دارید آن را مشخص کنید. یکی از اصلی‌ترین این تفاوت‌ها این است که svn یک سیستم مرکزی است؛ ولی گیت اینگونه نیست که در ادامه تفاوت این دو مورد را تشریح می‌کنیم.

یک. SVN یک مخزن مرکزی دارد که همه‌ی تغییراتی که روی کپی‌ها انجام می‌شود، باید به سمت مخزن مرکزی Commit یا ارسال شوند. ولی در سیستم گیت یک سیستم مرکزی وجود ندارد و هر مخزنی که fork یا Clone می‌شود، یک مخزن جداگانه به حساب می‌آید و Commit شدن تنها به مخزن کپی شده صورت میگیرد و در صورت pull request ادغام با مخزن اولیه خودش صورت میگیرد. دو. گیت به نسبت svn از پیچیدگی بیشتری برخوردار است؛ ولی برای پروژه‌های بزرگتر که کاربران زیادی با آن کار می‌کنند و احتمال شاخه بندی‌های زیادتر، در آن وجود دارد بهتر عمل می‌کند. موقعی که یک پروژه یا تیم کوچکی روی آن کار می‌کنند به دلیل commit شدن مستقیمی که svn دارد، کار راحت‌تر و آسان‌تر صورت می‌گیرد ولی با زیاد شدن کاربران و حجم کار، گیت کارآیی بالاتری دارد. سه. از آن جا که گیت نیاز به fork شدن دارد و یک مخزن کاملا مجزا از پروژه اصلی تولید می‌کند؛ سرعت بهتری نسبت به svn که یک کپی از زیر مجموعه ساختار اصلی ایجاد می‌کند دارد. *چهار*. شاخه بندی یک مفهوم اصلی و مهم در گیت به شمار می‌آید که اکثر کاربران همه روزه از آن استفاده می‌کنند و این اجازه را می‌دهد که که تغییرات و تاریخچه فعالیت هر کاربر را بر روی هر شاخه، جداگانه ببینیم. در svn پیاده سازی شاخه‌ها یا تگ‌ها سخت و مشکل است. همچنین شاخه بندی کار در svn به شکل سابق با کپی کردن صورت گرفته که گاهی اوقات به دلایلی که در قسمت قبل گفتیم، باعث ناسازگاری می‌گردد. پنج. حجم مخازن گیت به نسبت svn خیلی کمتر است برای نمونه پروژه موزیلا 30 درصد حجم کمتری در مخزن گیت دارد. یکی از دلایلی که svn حجم بیشتری میگیرد این است که به ازای هر فایل دو فایل موجود است یکی که همان فایل اصلی است که کاربر با آن کار می‌کند و دیگری یک فایل دیگر در شاخه svn. است که برای کمک به عملیاتی چون وضعیت، تفاوت ها، ثبت تغییرات به کار می‌رود. در صورتی که در آن سمت، گیت، تنها به یک فایل شاخص 100 بایتی برای هر دایرکتوری کاری نیاز دارد *شش*. گیت عملیات کاربری را به جز fetch و push، خیلی سریع انجام میدهد. این عملیات شامل یافتن تفاوت‌ها، نمایش تاریخچه، ثبت تغییرات، ادغام شاخه‌ها و جابجایی بین شاخه‌ها می‌گردد. *هفت*. در سیستم SVN به دلیل ساختار درختی که دارد، می‌توانید زیر مجموعه‌ی یک مخزن را بررسی کنید ولی در سیستم گیت اینکار امکان پذیر نیست. البته باید به این نکته توجه داشت که برای یک پروژه‌ی بزرگ شما مجبور هستید همیشه کل مخزن را دانلود کنید. حتی اگر تنها نسخه‌ی خاصی از این زیرمجموعه را در نظر داشته باشید. به همین علت در شهرهایی که اینترنت گرانقیمت و یا سرعت پایین عرضه می‌شود، گیت به صرفه‌تر است و زمان کمتری برای دانلود آن می برد. **موارد تعریف شده زیر طبق گفته ویکی سایت Kernel.Org ذکر می‌شود:**

گیت از سیستم SVN سریعتر عمل می‌کند.

در سیستم گیت هر شاخه بندی کل تاریخچه خود را به دنبال دارد.

فایل git که تنظیمات مخزن داخلش قرار دارد، ساختار ساده‌ای دارد و به راحتی می‌توان در صورت ایجاد مشکل، آن را حل کرد و به ندرت هم پیش می‌آید که مشکلی برایش پیش بیاید.

پشتیبانی گیری از یک سیستم مرکزی مثل SVN راحت‌تر از پشتیبانی گیری از پوشه‌های توزیع شده در مخزن گیت است.

ابزارهای کاربری svn تا به الان پیشرفت‌های چشمگیری داشته است. پلاگین‌ها و برنامه‌های بیشتری نسبت به سیستم گیت دارد. یکی از معروفترین این پلاگین‌ها، ابزار tortoisesvn است (البته ابزارهای گیت امروز رشد چشمگیرتری داشته اند که در قسمت اول نمونه‌های آن ذکر شد).

سیستم svn برای نسخه بندی و تشخیص تفاوت‌ها از یک سیستم ساده اعداد ترتیبی استفاده می‌کند که اولین ثبت با شماره یک آغاز شده و به ترتیب ادامه می‌یابد و برای کاربران هم خواندنش راحت است و هم قابل پیش بینی است. به همین جهت برای بررسی تاریخچه‌ها و دیگر گزارش‌ها تا حدی راحت عمل می‌کند. در سیستم شاخه بندی این سیستم شماره گذاری چندان مطلوب نیست و متوجه نمی‌شوید که این شاخه از کجا نشات گرفته است. در حال حاضر برای پروژه‌ی موزیلا این عدد به 6 رقم رسیده است ولی در آن سمت، سیستم گیت از هش SH-1 استفاده می‌کند که یک رشته 40 کاراکتری است و 8 رقم اول آن به منشاء اشاره می‌کند که باعث می‌شود متوجه بشویم که این شاخه از کجا آمده است ولی از آنجا که این عدد یکتا ترتیبی نیست، برای خواندن و

گزارشگیری‌هایی که در SVN راحت صورت می‌گیرد، در گیت ممکن نیست یا مشکل است.
گیت رویدادهای ادغام و شاخه بندی را بهتر انجام می‌دهد.