

Kaynaklar

- [Github Nedir? - Microsoft](#)
- [GitHub'a Giriş](#)

İçincekiler

- [Git Nedir?](#)
- [GitHub Nedir?](#)
- [GitHub Üzerinden Neler Yapılabilir?](#)
- [GitHub'ı Kullanalım](#)

Git Nedir?

- Git bir versiyon kontrol sistemidir. Projenin versiyonlarını sürekli olarak klasörleyerek tutmak yerine, bu ihtiyacı Git üzerinden karşılayabiliriz.
- Örnek olarak bir proje geliştirdiğimizi düşünelim. Proje zaman ilerledikçe büyüyecek, değişecek ve daha karmaşık hale gelecektir. Bu süreç içerisinde bazı nedenlerden dolayı projenin x gün önceki haline dönmemiz veya bir kod parçasının x gün önceki haline erişmemiz gerekebilir. Git teknolojisi aslında bizim bu ihtiyacımız karşılamak için çok uygun bir teknolojidir. İlerleyen kısımlarda bahsedilecek olan **commit**'ler sayesinde, projenin geçmişte **commit** atılmış anlarına kolayca erişebiliyoruz. Eğer Git'i kullanmıyorsak böyle bir ihtiyacı giderebilmek için projeyi geliştirme aşamasında v1, v2, v3, vb. şekilde klasörlememiz gerekecektir.
- Git kullanımının başka bir güzel avantajı projeyi takım olarak geliştirmeyi kolaylaştırır. Örnek olarak 4 modülden oluşan bir proje geliştirdiğimizi düşünelim. Takımdaki 4 kişi birer modül seçer ve geliştirmeye başlar. Git içerisinde bulunan **branch** yapısı ile oldukça basit bir şekilde ayrı çalışmalar yapılabilir ve en son modüller tamamlandıkça **merge** edilerek proje içerisinde birleştirilirler ve proje tamamlanmış olur.
- Son olarak ilk başlarda GitHub ile Git'in aynı şey olduğu sanılsa da aslında Git başlı başına bir teknolojidir, GitHub ise bu teknolojiyi kullanıcılar için hazırlanan arayüzler ile kolay kullanılabilir yapmayı hedefleyen bir servistir.
- Bunu şu şekilde düşünebiliriz, GitHub ve GitLab ortak olarak Git teknolojisini kullanır ama kendileri ayrı servislerdir. İkisinde kendilerine göre avantajları vardır ve kullanıcı ihtiyacına göre Git teknolojisini istediği servis üzerinden kullanabilir.

GitHub üzerinden neler yapılabilir?

- **Sorunlar Alanı:** Projenin kullanıcıları ile projenin geliştiricileri arasındaki iletişim genelde bu alan üzerinden olur. Kullanıcılar projeye eklenmesini istedikleri özellikleri, gördükleri hataları vb. durumları bu alandan projeyi geliştiren kişilere belirtirler.
- **Fork ve Star:** GitHub üzerindeki diğer kullanıcıların yaptığı projeleri, paylaştığı notları veya dökümanları kendi takibimize almak istiyorsak, o projeyi alıp geliştirme yapmak istiyorsak *Fork* işlemi yapabiliriz. Fork'ladığımız Repo'lar bizim Repo'larımızın arasında görülecek ama alt kısmında Repo'nun sahibi belirtilecektir.

Eğer bir Repo'yu beğendiyseniz *Star* kısmını işaretleyebilirsiniz. Profilimizden yıldız verdiğimiz Repo'ları tekrar tekrar inceleyebiliriz.

- **Dal (branch):** Bu kısım aslında geliştiricileri ilgilendiren kısımdır. Projeyi yukarıda örneklendirdiğimiz şekilde, takım olarak proje geliştirirken dallar üzerinde çalışmak aynı anda geliştirme yaparken kodların çakışmasını azaltır. Bu çakışma olayı **conflict** olarak adlandırılır.

Genelde projelerde sabit,kararlı olan bir dal olur, geliştiriciler tamamladıkları kendi dallarını bu kararlı dal ile birleştirirler. Birleştirme işleminden önce kararlı dala gelen yan dal önce kontrollerden geçer, onaylandığı zaman kararlı dal ile birleştirilir.

Dallanmaları kullanılan IDE üzerinden veya GitHub'ın kendi içerisinde takip edip, dallar arasında işlemler yapabiliriz.

- **Commit'ler:** Commit'leri projenin geliştirme sürecinde proje içerisine atılan işaretler olarak düşünebiliriz. Proje üzerinde değişiklikler yaptığımızda commit'ler atarak kendimiz için veya takımımız için projede geriye dönüş noktaları oluşturmuş oluruz. Ayrıca commit sistemi projenin gelişimini daha okunabilir ve daha takip edilebilir hale getirir.
- **Pull Requests:** Projedeki kararlı dala, birleştirilmeye hazır bir yan dal eklenmek isteniyorsa *pull request* oluşturulur. Buradan anlaşılacağı üzere takımdaki herhangi birisinin bitirdiği dalı direkt olarak kararlı dal'a birleştirmesi engellenir (Bunun için **Onay Gerekli** olarak ayarlanmalıdır).

Kararlı dal'a olabildiğince herhangi bir sorun çıkartmayacağını düşündüğümüz ve testlerden geçmiş, kontrol edilmiş kodların eklenmesini istiyoruz. Bu nedenle yan dallar kararlı dal ile birleştirilmeden önce kontrollerden geçer.

GitHub'ı Kullanalım

- Projelerde GitHub hizmetlerini kullanmanın farklı yolları vardır. Git Bash kullanarak, Git Desktop kullanarak, Visual Studio veya VS Code gibi programları kullanarak, eğer Linux tabanlı bir işletim sisteminiz var ise direkt console üzerinden vb. yollar ile GitHub hizmetlerini kullanabiliriz. Şimdilik **Git Bash** ve **Visual Studio** örneklerini Windows cihaz üzerinden göstereceğiz.
- Git teknolojisini kullanmak istiyorsak ilk önce bilgisayarımıza Git'i kurmamız gerekiyor. Hedefimiz kullanımı gösterebilmek olduğundan kurulumu detaylı anlatmayacağız. [Buradan](#) kurulumları yapabilirsiniz.

Git Bash ile GitHub'a Proje Yükleme

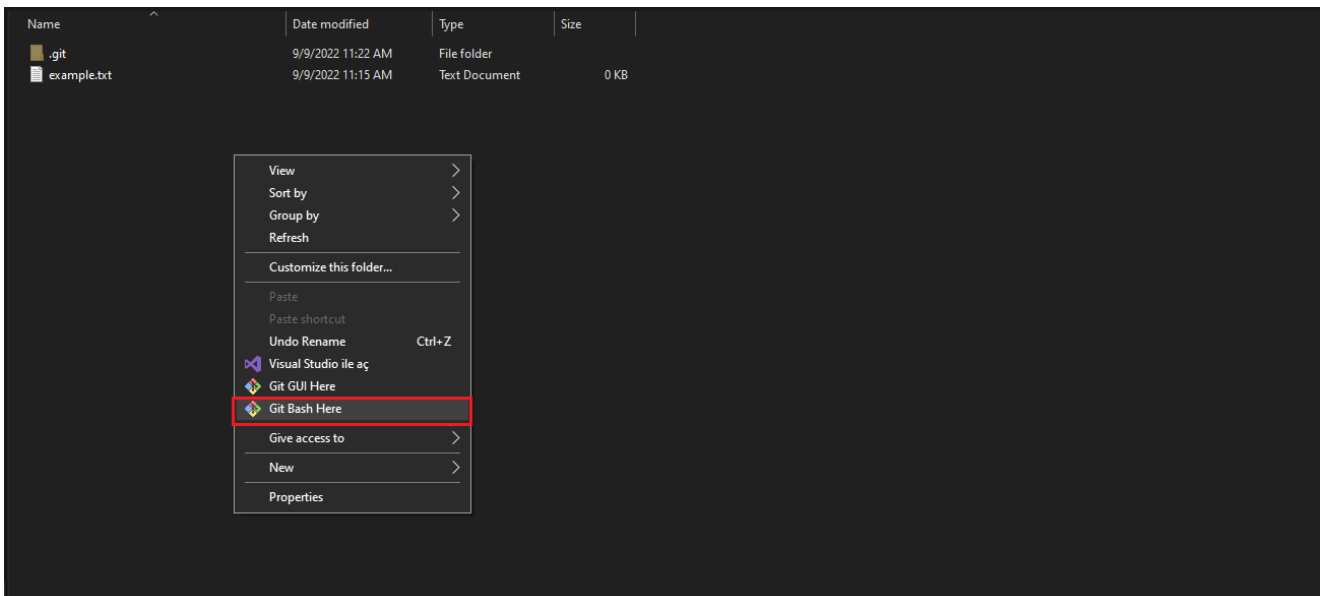
- Öncelikle GitHub hesabımızı bilgisayarımıza indirdiğimiz Git'e tanıtmamız gerekiyor. Bunun için

```
git config --global user.name "userName"
```

şeklinde işlem yapmamız gerekiyor. Daha detaylı anlatımı için [buraya](#) bakabilirsiniz.

- Örnek olarak sıfırdan bir projeye başladığımızı düşünelim. Öncelikle GitHub'a yüklemek istediğimiz projenin bulunduğu klasöre girmemiz gerekiyor. Bu klasör içinde yapılan değişiklikleri Git'in takip edebilmesi ve hizmetlerinden faydalanabilmemiz için bir komut çalıştırmamız gerekiyor.

Git Bash ile projenin olduğu klasörün içerisine girmek için klasör içerisinde mouse ile sağ tıklayıp **GitBash Here** seçeneğine basabiliriz.

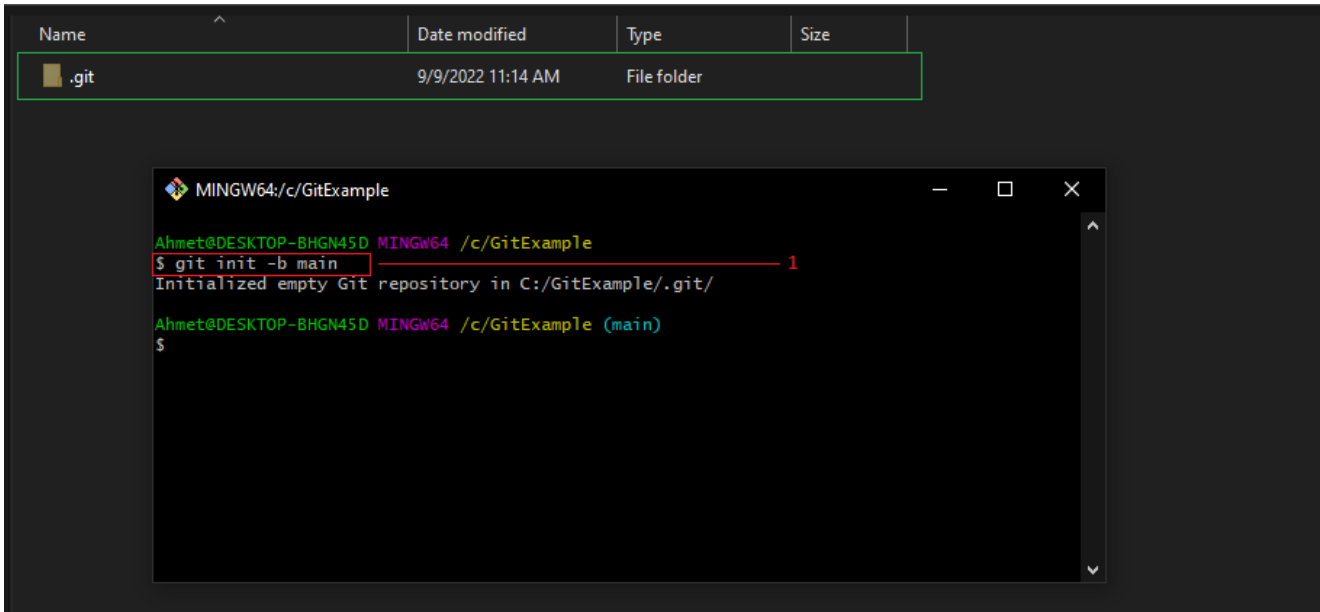


- Eğer sizde bu seçenek gelmiyorsa temel CLI komutları ile GitBash içerisinde ilgili klasöre girmeniz gerekecektir. İsterseniz Git'in kurulum ayarlarını güncelleyerek bu özelliği aktif edebilirsiniz.

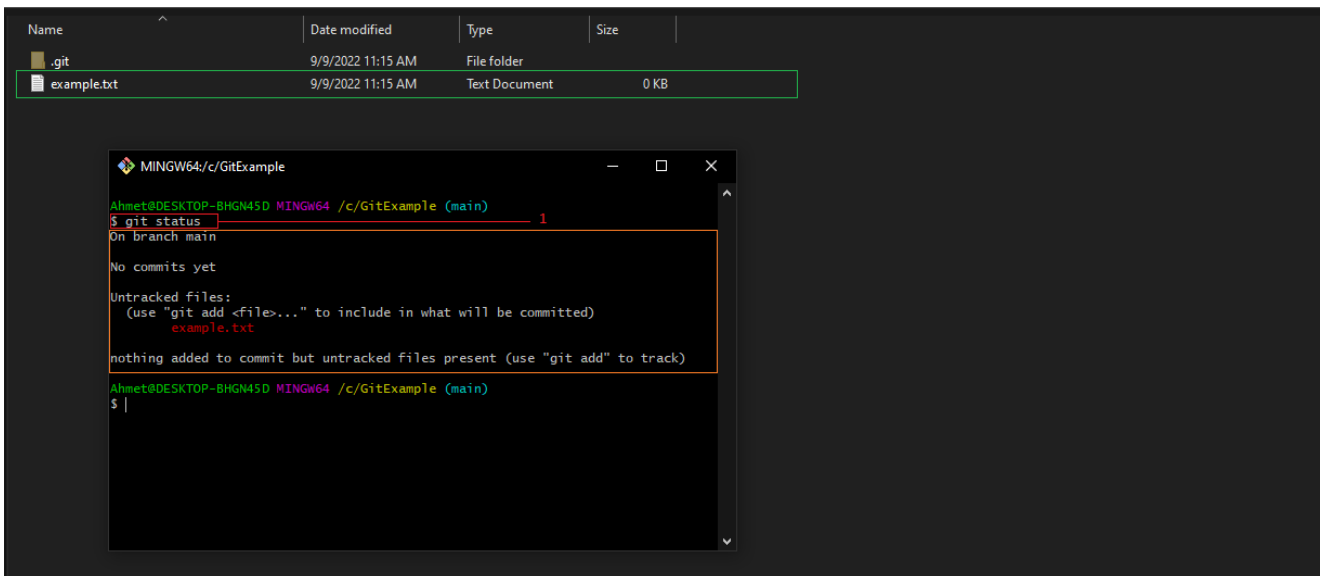
Açılan Git Bash ekranında çalıştırmamız gereken komut:

```
git init -b main
```

Bu komutu çalıştırdığımızda artık klasör içerisinde gizli klasör olarak tutulan bir `.git` klasörü oluşacaktır. Eğer sizde bu klasör görünmüyor ise *File Explorer* ayarlarından bu özelliği açabilirsiniz.



Artık bu klasör içerisindeki değişiklikler Git tarafından takip edilebilir, Git'in hizmetlerinden faydalanılabilir hale geldi. Örnek olarak projemizin içerisine alt kısımda yeşil alan ile gösterilen gibi bir `txt` dosyasının eklendiğini düşünelim.



Proje içerisinde değişiklikler yapıldığında Git üzerinden bu durumları şu komut ile takip edebiliriz:

```
git status
```

Bu komut, Git içerisinde yer alan **Working Tree** kavramından faydalanarak dosyaların durumları hakkında bizlere rapor verir. Örnek olarak yukarıdaki ekran görüntüsünde *turuncu* olarak işaretlenen alanda bir uyarı veriliyor. Projeye eklediğimiz *example.txt* dosyasını **Untracked file** olarak tanımlıyor ve hemen üstünde yönlendirici bir bilgi mesajı veriyor.

Proje içerisine eklediğimiz dosyaların Git'tarafından takibe alınmasını istiyorsak bu dosyaları şu komut ile işaretleriz:

```
git add
```

Bu komutu farklı şekillerde farklı parametreler ile kullanabiliriz. Örnek olarak projeye eklenen dosyalar içerisinde sadece *example.txt* dosyasının takip edilmesini istiyorsak:

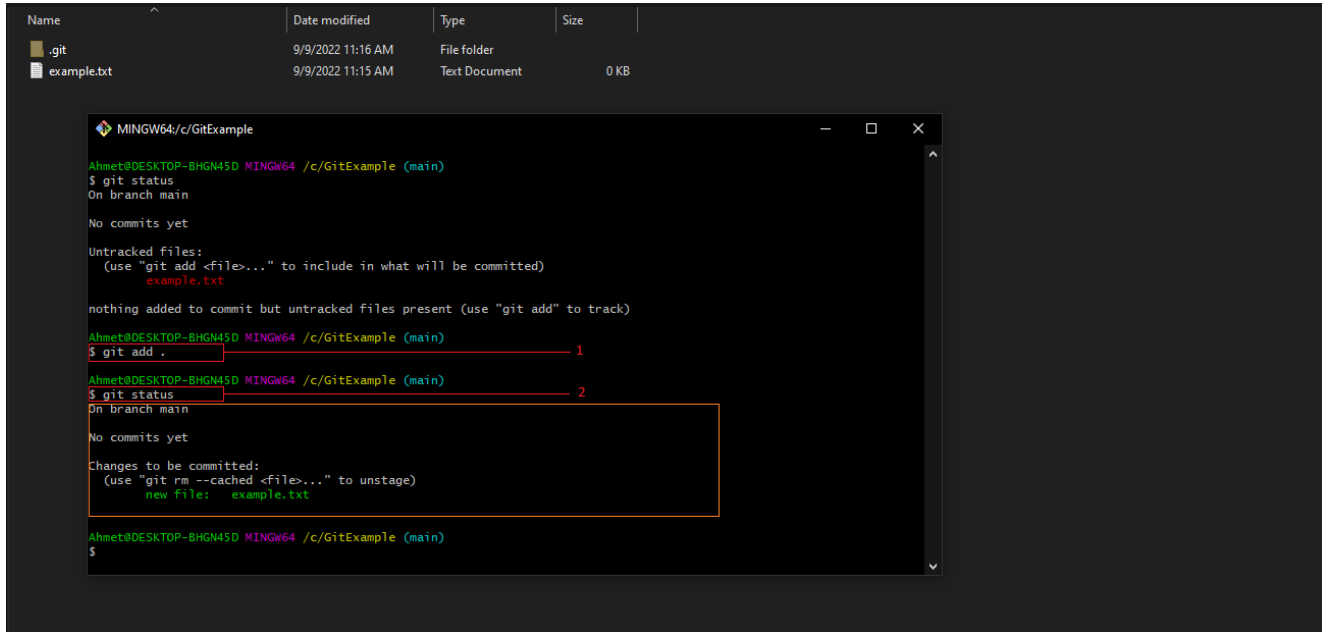
```
git add example.txt
```

şeklinde kullanabiliriz. Eğer proje içerisindeki bütün dosyaları Git'in takip etmesini istiyorsak yukarıdaki ekran görüntüsünde yapıldığı gibi:

```
git add .
```

şeklinde kullanabiliriz

Örnek olarak:



The screenshot shows a Windows File Explorer window with a dark theme. The top bar shows the file name, date modified, type, and size. Below the bar, there are two files: ".git" (File folder) and "example.txt" (Text Document, 0 KB). Both files have a date modified of 9/9/2022 11:16 AM. Below the File Explorer, there is a terminal window titled "MINGW64/c/GitExample". The terminal shows the following commands and output:

```
Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git status
On branch main

No commits yet

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

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

Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git add .
1

Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git status
2
On branch main

No commits yet

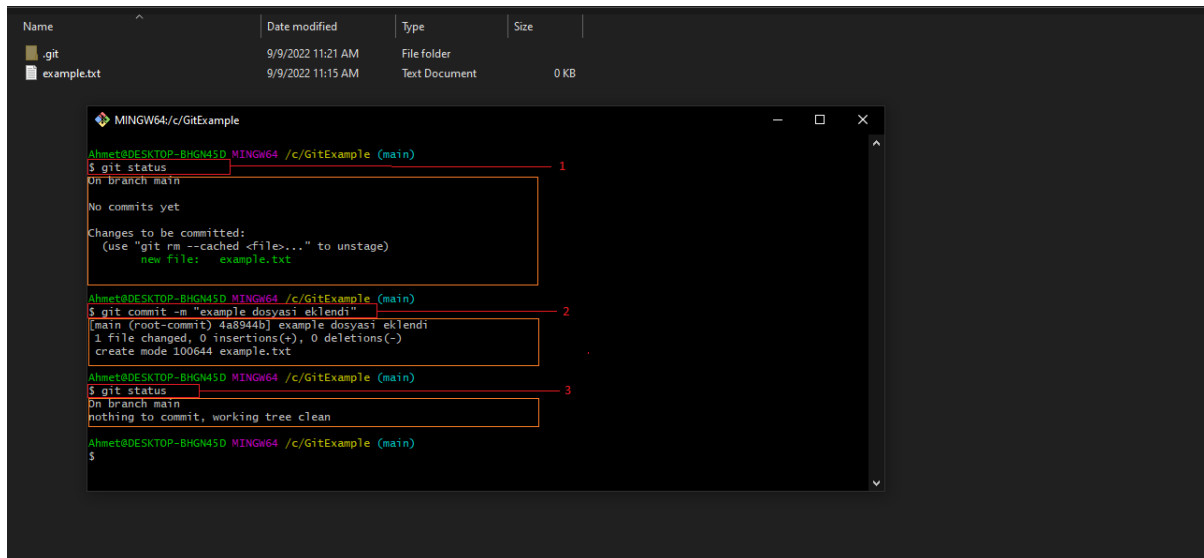
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   example.txt

Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$
```

- Yukarıdaki ekran görüntüsünde olduğu gibi (1) komutundan sonra (2) komutunu çalıştırdığımızda artık farklı bir rapor oluşuyor. Turuncu olarak işaretlenen alanda artık farklı uyarı mesajı var ve görüldüğü gibi dosya ismi kırmızı renkten yeşil renge dönüş durumunda, yani artık *example.txt* dosyası takip ediliyor.
- Sonraki adımda projeye eklenen ve Git tarafından takibe alınan dosyaları GitHub üzerindeki uzak depomuza göndermek istersek şu şekilde bir ara komut çalıştırmamız gerekiyor:

```
git commit -m "commit mesajı"
```

Commit komutu ile artık dosyamızı gönderilebilir olarak işaretledik ve kaydettik. İleriki zamanlardan dosyanın şuan commit'lediğimiz haline erişilebilir bir nokta da oluşturmuş olduk.



The screenshot shows a Windows File Explorer window with a dark theme. The top bar shows the file name, date modified, type, and size. Below the bar, there are two files: ".git" (File folder) and "example.txt" (Text Document, 0 KB). Both files have a date modified of 9/9/2022 11:21 AM. Below the File Explorer, there is a terminal window titled "MINGW64/c/GitExample". The terminal shows the following commands and output:

```
Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git status
1
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   example.txt

Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git commit -m "example dosyası eklendi"
2
[main (root-commit) 4a8944b] example dosyası eklendi
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 example.txt

Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git status
3
On branch main

nothing to commit, working tree clean

Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$
```

- Genel olarak Git komutlarını çalıştırmadan önce *git status* komutu ile dosyaların durumlarını kontrol etmemiz faydalı olacaktır. Yukarıdaki ekran görüntüsünde görüldüğü gibi (1) komutunu çalıştırdıktan sonra alt kısımdaki turuncu alandan dosyaları kontrol ettik. Sonrasında (2) komutu ile dosyalar üzerinde yaptığımız değişiklikleri işaretledik ve (2)'nin alt kısımdaki turuncu alandan gerçekleştirilen işlemlerin detayları gördük.

Commit işleminden sonra tekrar *git status* ile dosyalarının durumlarına bakalım (3). Görüldüğü gibi (3)'ün alt kısımdaki turuncu alan bizlere Git tarafından takip edilen dosyalardan üzerinde değişiklik yapılmış bir dosya olmadığını söylüyor.

- Commit işleminden sonra projemizi uzak depomuza göndermek için GitHub üzerinde bir *Repository* oluşturmamız gerekiyor. Bu işlem de Git Bash üzerinden yapılabilir fakat örnekte GitHub üzerinden yeni bir Repo oluşturacağız.

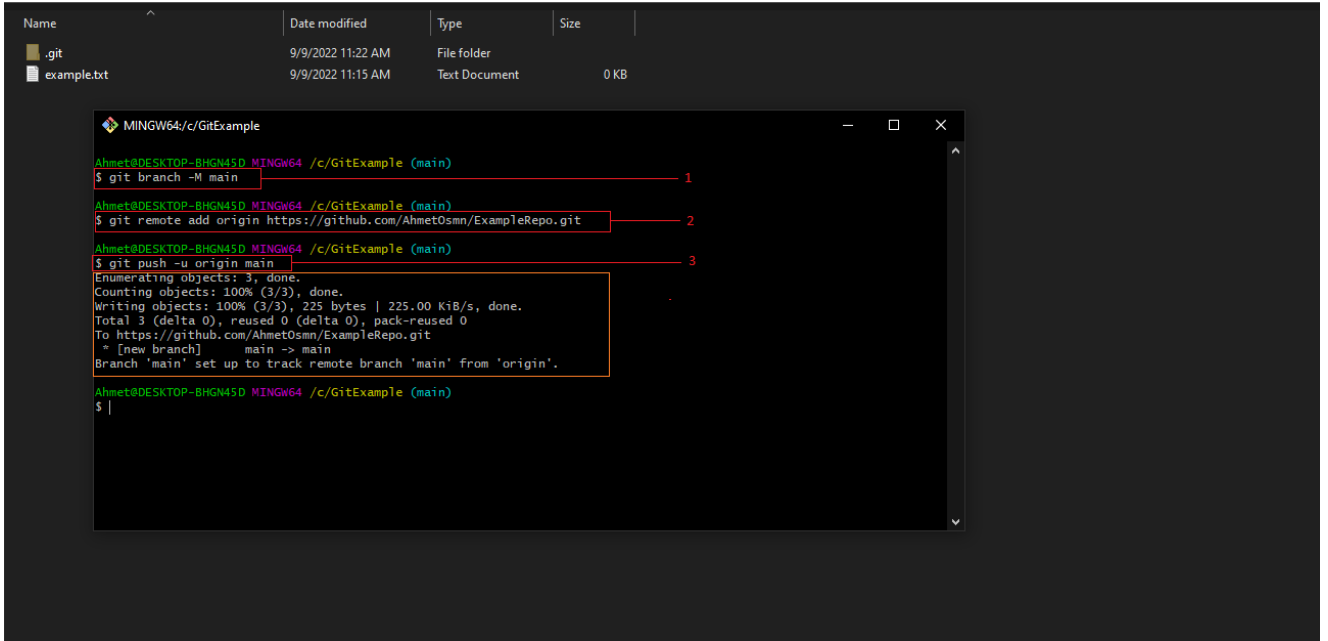
- Bu sayfada Repo için bazı ayarlar yapılabilir ve istenirse Repo'ya açıklama eklenebilir. Şimdilik yukarıdaki ekran görüntüsündeki gibi default şekilde Repo'yu oluşturalım.

Yeni bir Repo oluşturduğumuzda, GitHub bizlere örnek bir GitHub kullanım klavuzu veriyor. İsterseniz o kısmı da inceleyebilirsiniz.

- GitHub üzerinde bir Repo oluşturduğumuza göre artık projeyi gönderebiliriz. Gönderme işlemi için:

```
git push
```

Komutunu kullanacağız. *git push* komutuna bazı parametreler vermemiz ve bu komuttan önce bazı işlemler yapmamız gerekecek. Bu işlemler için örnek bir ekran görüntüsü olarak:



```
Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git branch -M main
Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git remote add origin https://github.com/AhmetOsmn/ExampleRepo.git
Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 225 bytes | 225.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AhmetOsmn/ExampleRepo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
Ahmet@DESKTOP-BHGN45D MINGW64 /c/GitExample (main)
$ |
```

- İlk önce yukarıdaki ekran görüntüsünde olduğu gibi (1) komutu ile *main* dalına geçiş yapalım.

```
git branch -M main
```

Ardından projeyi göndereceğimiz GitHub Repo'sunun adresini Git'e belirtmemiz gerekiyor. Bu işlemi de yukarıda olduğu gibi (2) komutu ile yapabiliriz:

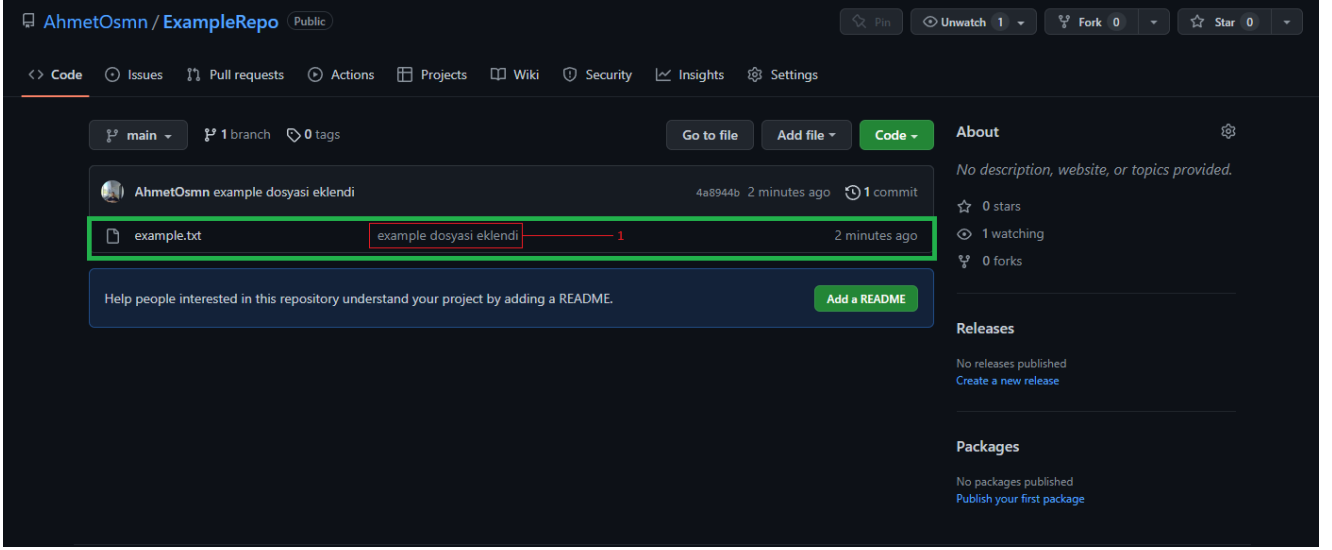
```
git remote add origin https://github.com/AhmetOsmn/ExampleRepo.git
```

Burada verdiğimiz adres GitHub üzerinde projeyi yüklemek istediğimiz Repo'nun adresi olmalıdır. Son olarak yapmamız gereken ekran görüntüsündeki gibi (3) komutunu çalıştırmaktır.

Gönderme işlemi tamamlandıktan sonra, (3)'ün hemen alt kısımdaki turuncu alan gibi bilgi veren bir alan gelecektir. Eğer projeyi gönderirken bir sorun yaşanırsa buradan

görülebilir. Bu alan içerisinde Yapılan işlemler ve proje hakkında bazı detay açıklamalar yer alır.

Gönderme işlemi tamamlandıktan sonra GitHub üzerinden ilgili Repo içerisine girip değişikliklerin gelip gelmediğine bakabiliriz.



- Burada (1) ile gösterilen alan anlaşılacağı üzere, *git push* öncesinde ara işlem olarak söylediğimiz *git commit -m "commit mesajı"* kısmındaki commit mesajıdır.

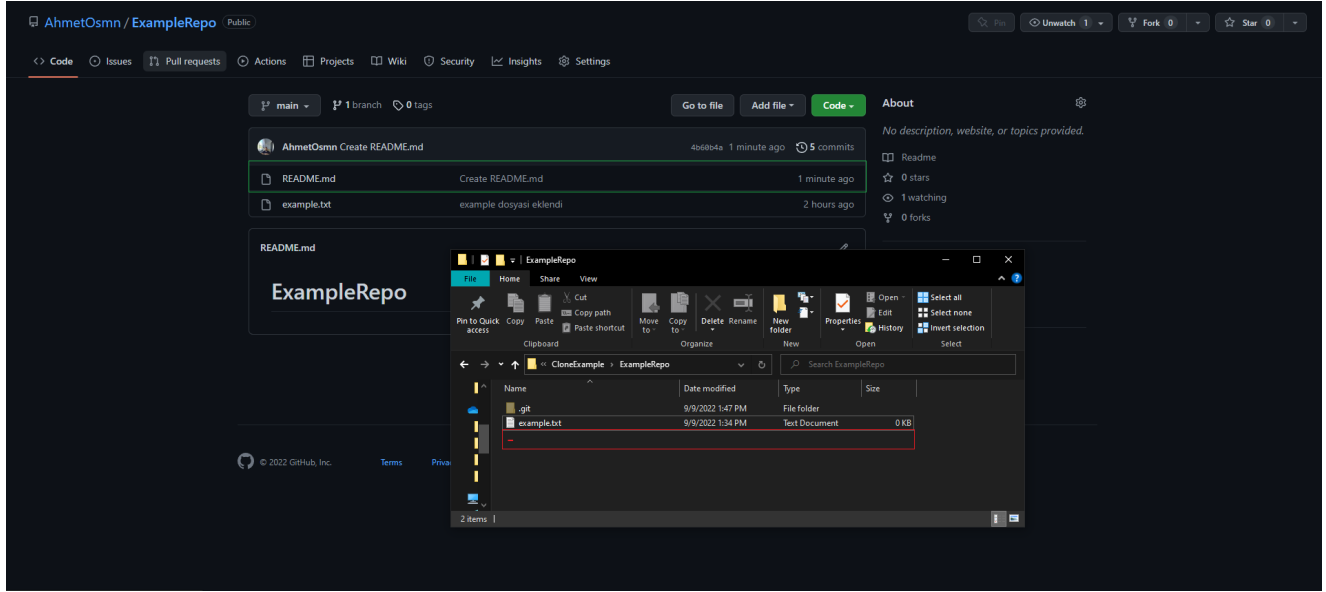
Git Bash ile Projeyi Güncellemek

- Takım halinde çalıştığımız zaman projeye bizim dışımızda diğer arkadaşlarımız da eklemeler, çıkartmalar ve güncellemeler yapıyor. Böyle durumlarda projenin güncel halini elde etmek için sürekli yeni bir klasöre *git clone* komutu ile projeyi çekmeyiz. Bunun yerine

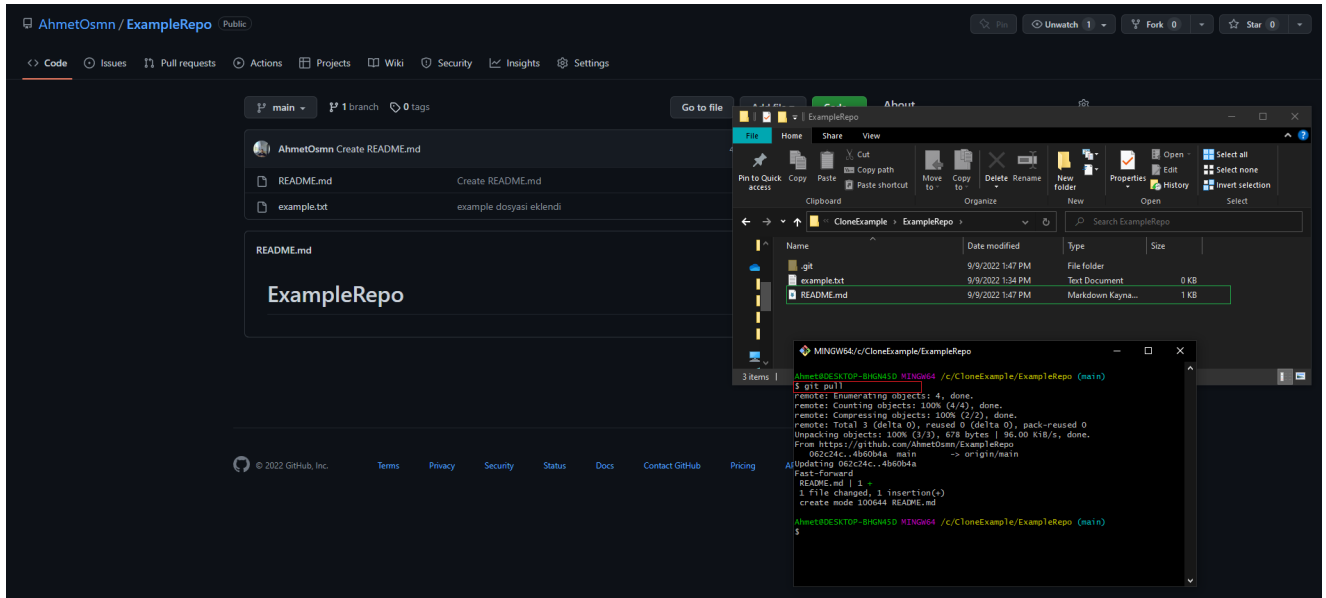
```
git pull
```

komutunu kullanırız. Bu komut farklı parametreler ile daha detaylı da kullanılabilir fakat şimdilik sadece işlevini öğrenmemiz yeterlidir.

Örnek olarak alt kısımdaki ekran görüntüsünde olduğu gibi projeye bizde olmayan bir dosya eklenmiş olsun.



Yapmamız gereken şey projenin güncel halini GitHub üzerinden çekmektir. Burada dikkat edilmesi gereken bir nokta vardır. Projenin son halini çekmeden, kendi bilgisayarınızda proje üzerinde değişiklikler yaptıysanız ve GitHub'taki proje ile aynı alanlarda ise bu değişiklikler *conflict*'ler meydana gelebilir. Bu nedenle geliştirmeye başlamadan önce projenin son halini bilgisayarımıza çekip ondan sonra ayrı bir dal üzerinden çalışmaya devam etmeliyiz.



Projenin güncel halini yukarıdaki ekran görüntüsünde kırmızı olarak işaretlenen alandaki gibi *git pull* komutunu çalıştırarak elde edebiliriz.

Git Bash Üzerinde Dallar (Branches)

- Git Bash ile yeni bir branch oluşturmak istiyorsak 2 şekilde yapabiliriz.
- İlk yöntemde branch oluşturulur fakat geçiş yapılmaz:

```
git branch [branch adı]
```

Yukarıda oluşturduğumuz branch'e geçmek için şunu yapmalıyız:

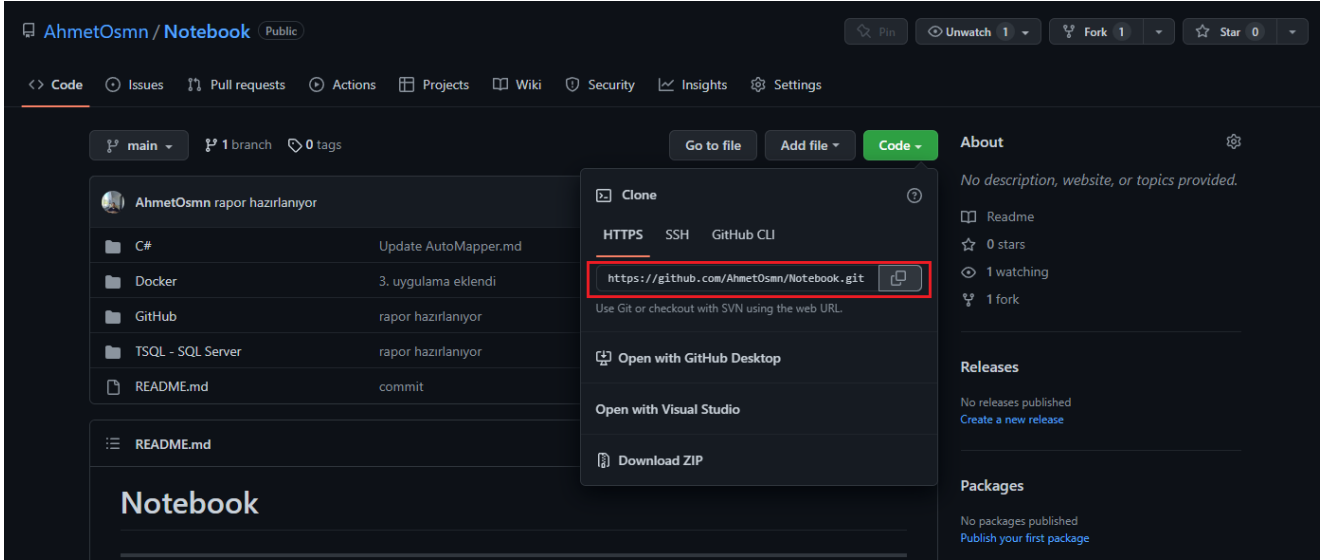
```
git checkout [branch adı]
```

- İkinci yöntemde ilk önce branch oluşturulur sonrasında direkt olarak geçiş yapılır:

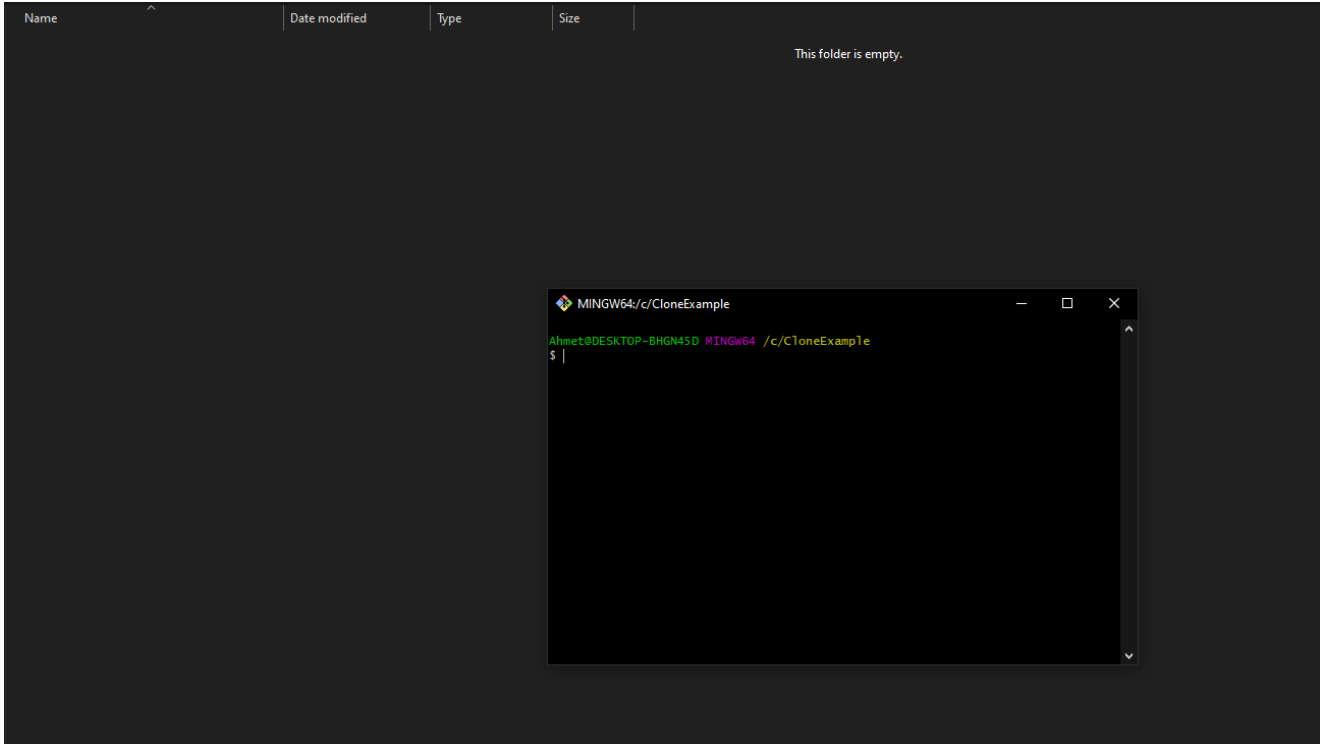
```
git checkout -b [branch adı]
```

GitHub Üzerinden Proje Çekmek

- GitHub üzerinde paylaşılmış bir projeyi kendi bilgisayarımıza indirmek istiyorsak ve bunu Git Bash ile yapacaksak ilk olarak ilgili Reponun adresini almamız gerekiyor.

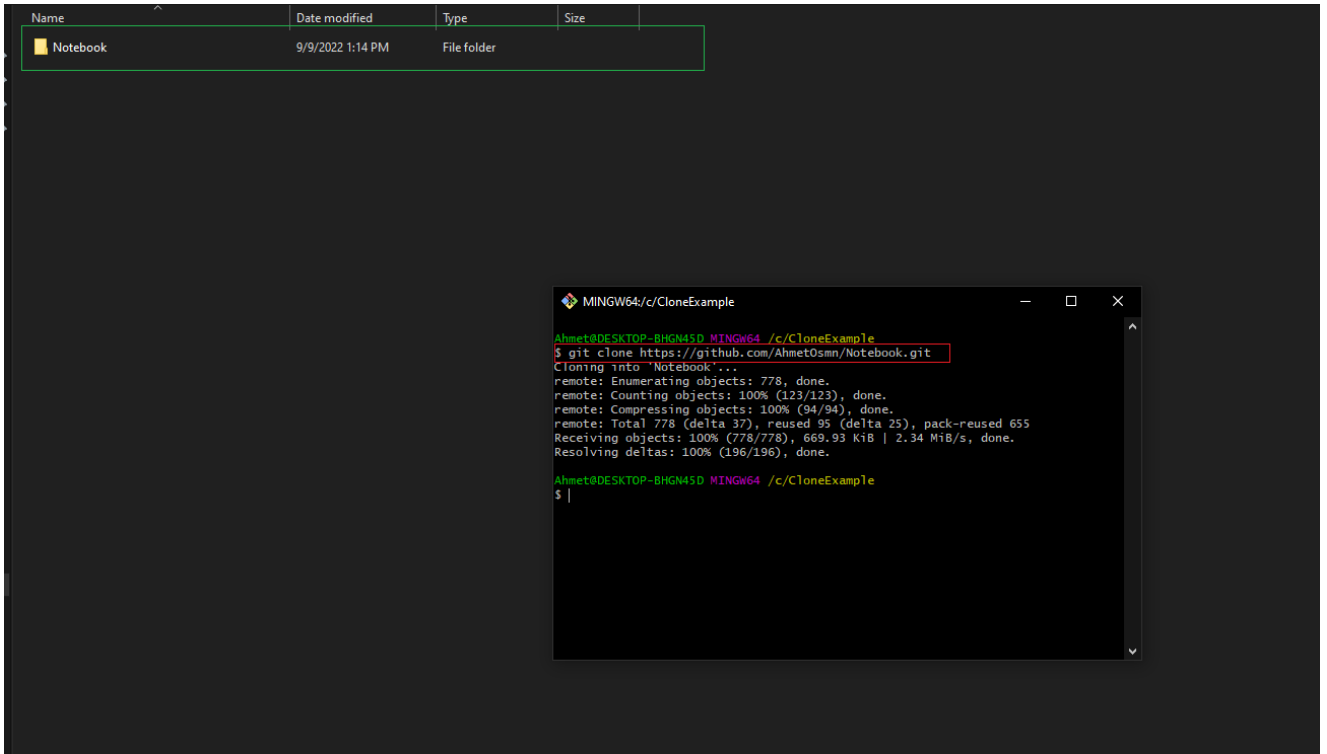


- Daha sonra Git Bash ile projeyi indireceğimiz klasörün içerisine girmemiz gerekiyor.



- Son olarak alt kısımda kırmızı olarak gösterilen alandaki gibi şu komutu çalıştırmalıyız:

```
git clone [projenin github adresi]
```

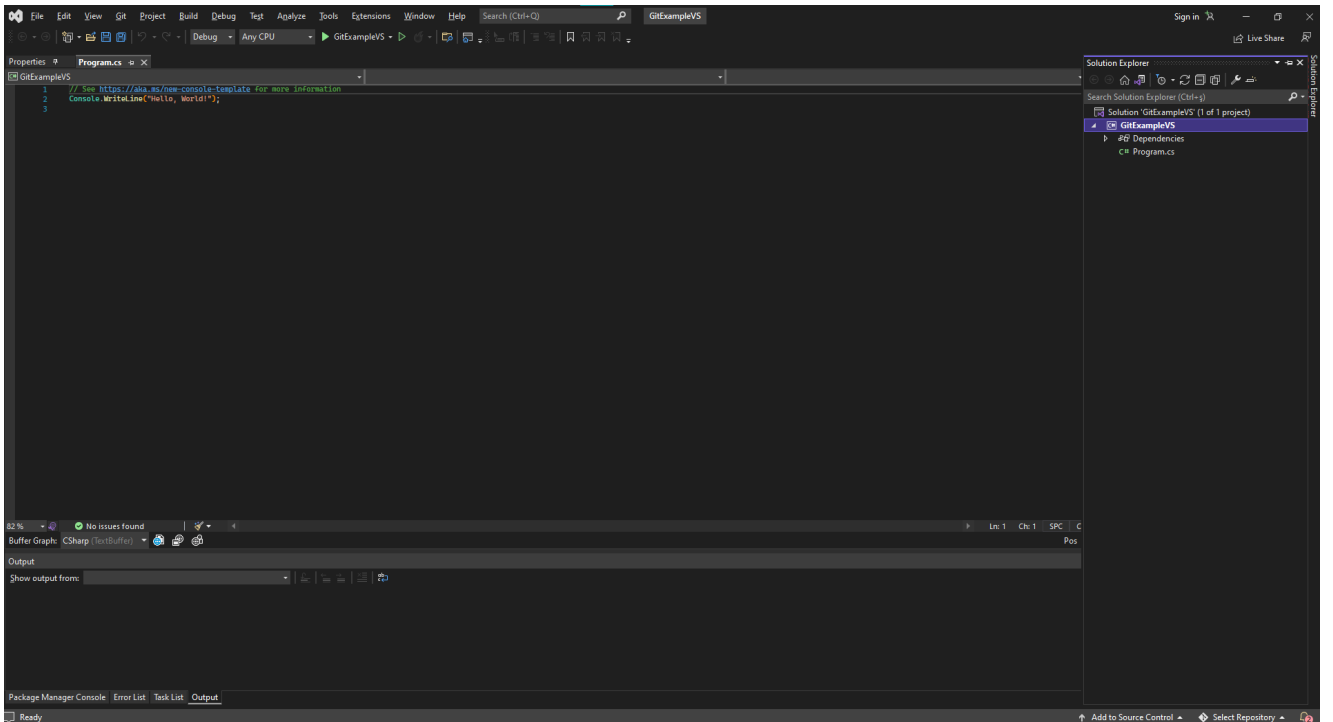
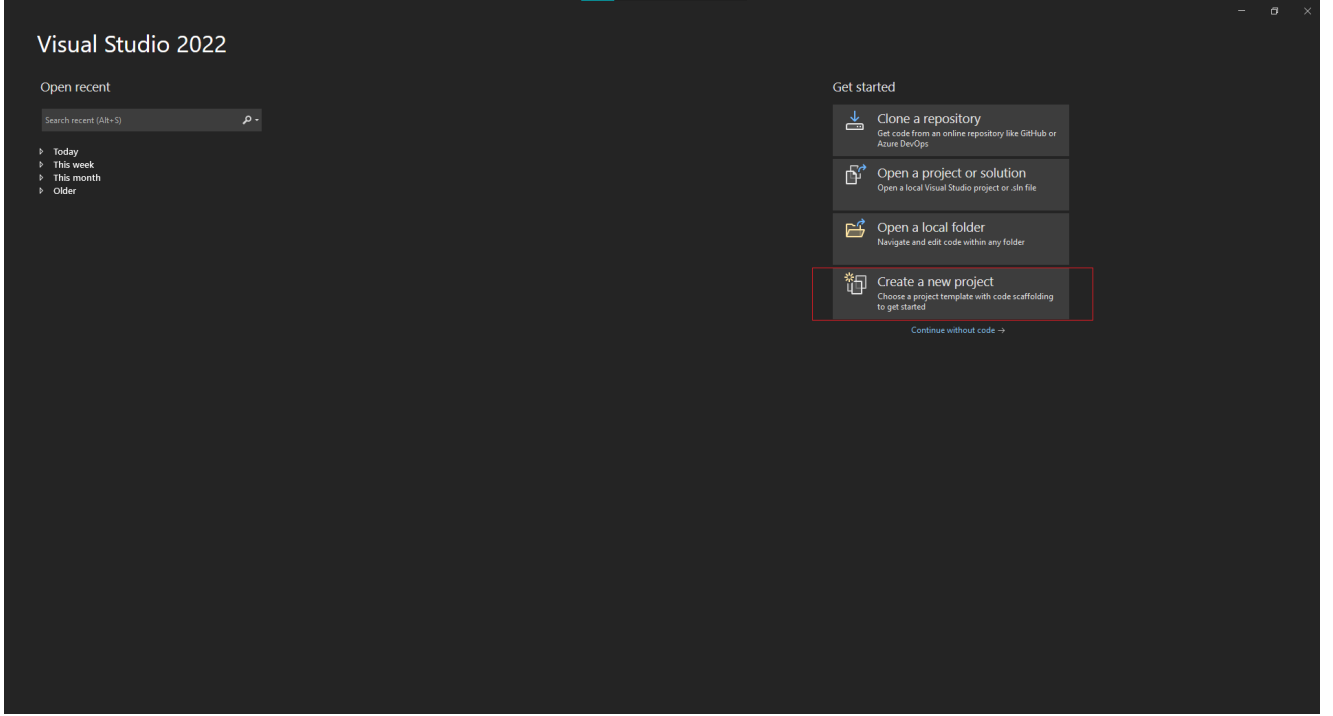


Ekran görüntüsünde yeşil olarak gösterilen alanda görüldüğü üzere indirme işlemi tamamlandıktan sonra proje bilgisayarımıza indirilmiş olacaktır.

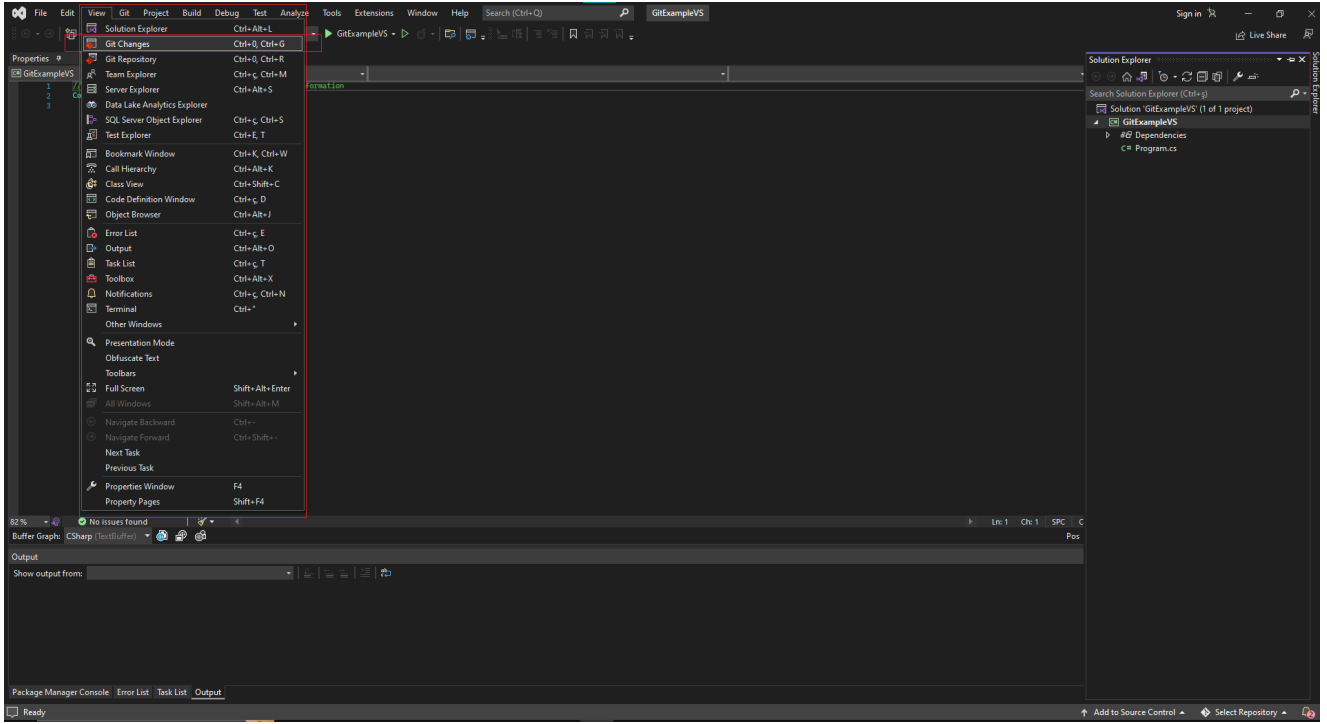
Visual Studio

GitHub'a Proje Yüklemek

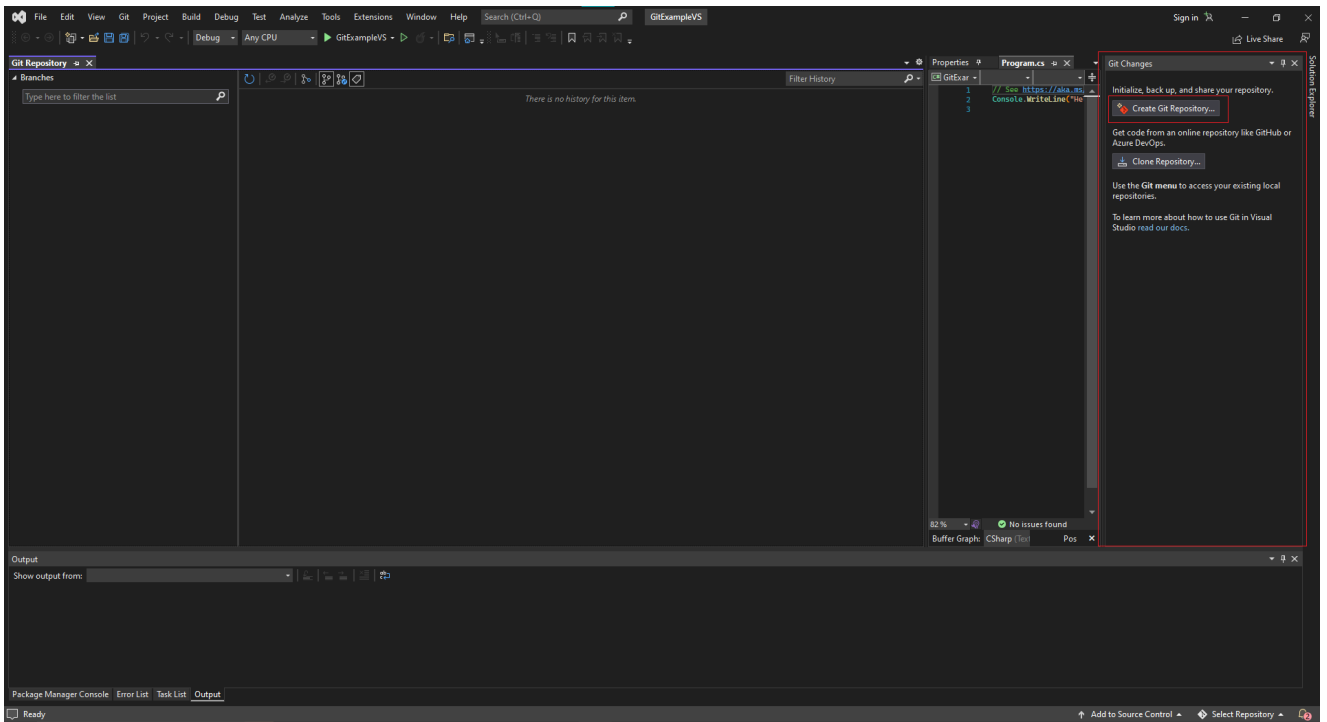
- VS ile bir proje oluşturup GitHub'a yüklemek istiyorsak öncelikle normal bir proje oluşturmalıyız.



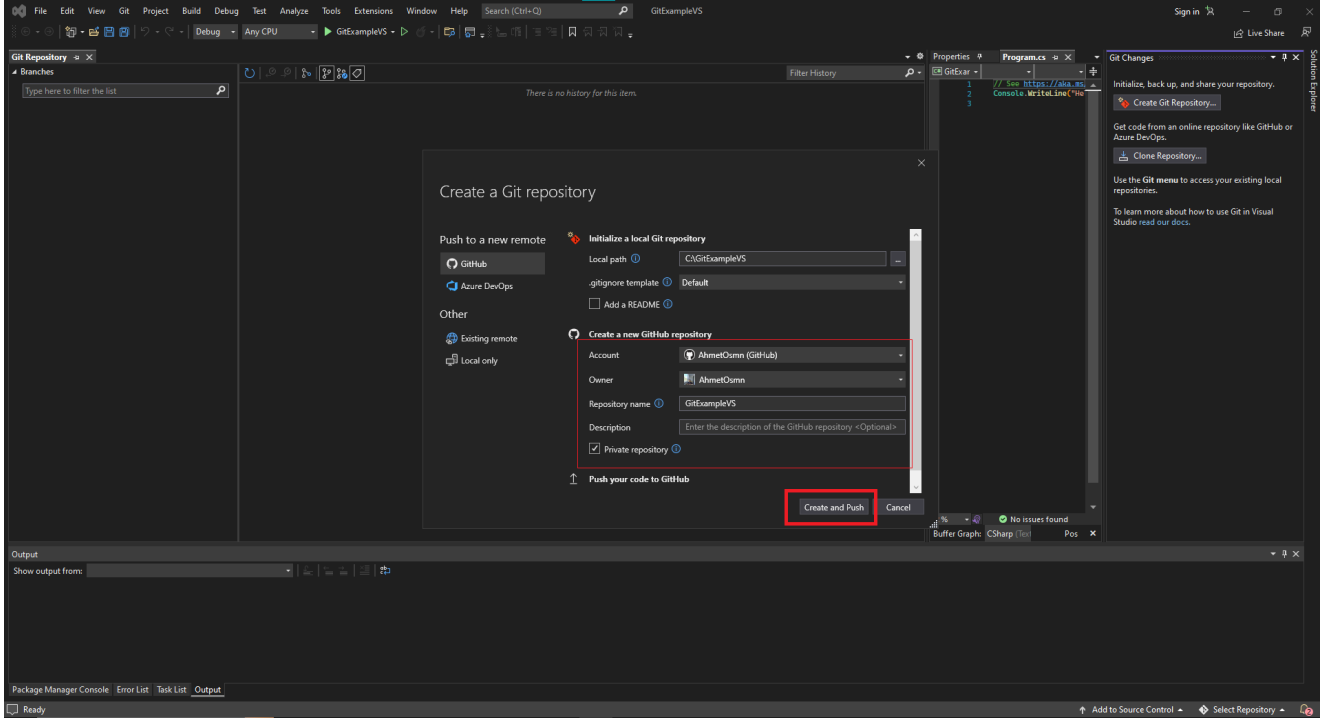
- Sonraki adımda projeyi geliřtirdiđimizi dűřűnelim. İřlemler tamamlanmıř olsun ve artık projeyi GitHub'a yűklemek istiyoruz. VS iēerisindeki **View** sekmesinden **Git Changes**'i aēalım. Burada ve sonrasında yapacađımız iřlemleri VS iēerisinde farklı yollarla da yapabiliriz. Ben űrnek olarak **Git Changes** űzerinden gűstereēeđim.



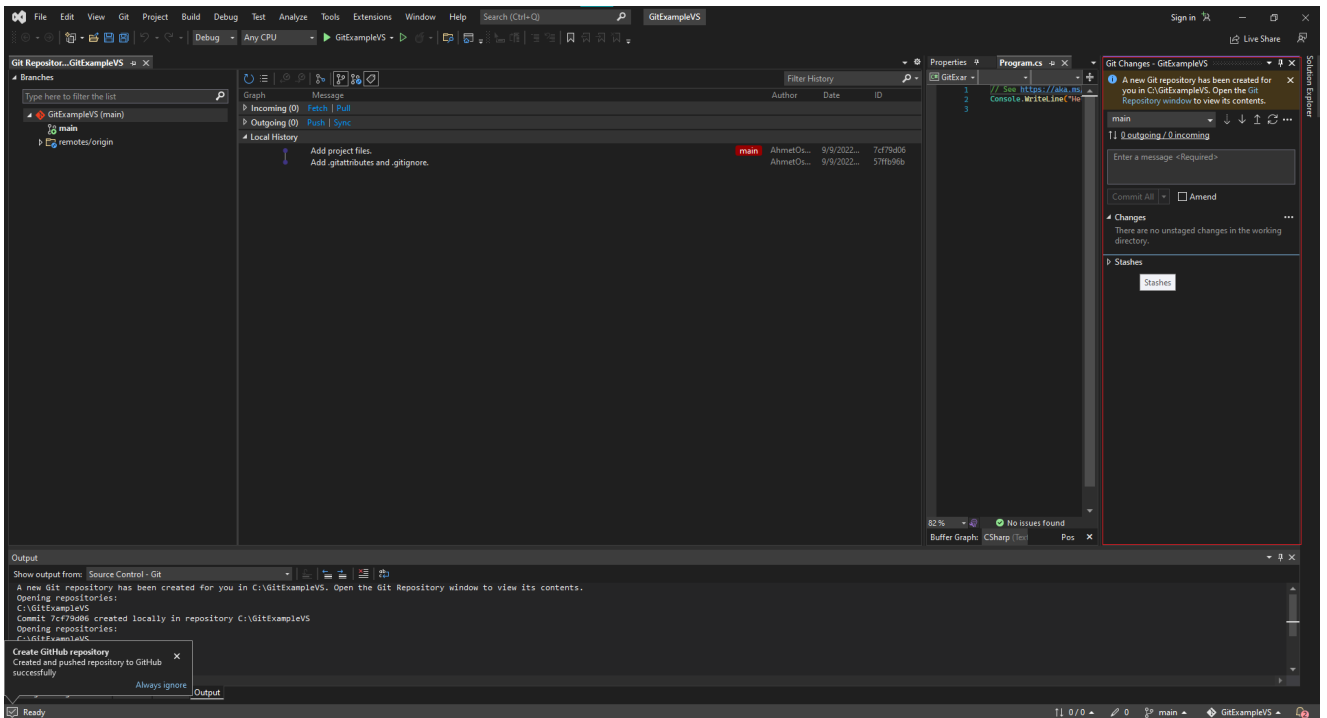
- *Git Changes* seēeneđine tıkladıđımızda *Solution Explorer*'ın yanına *Git Changes* alanı gelecektir.

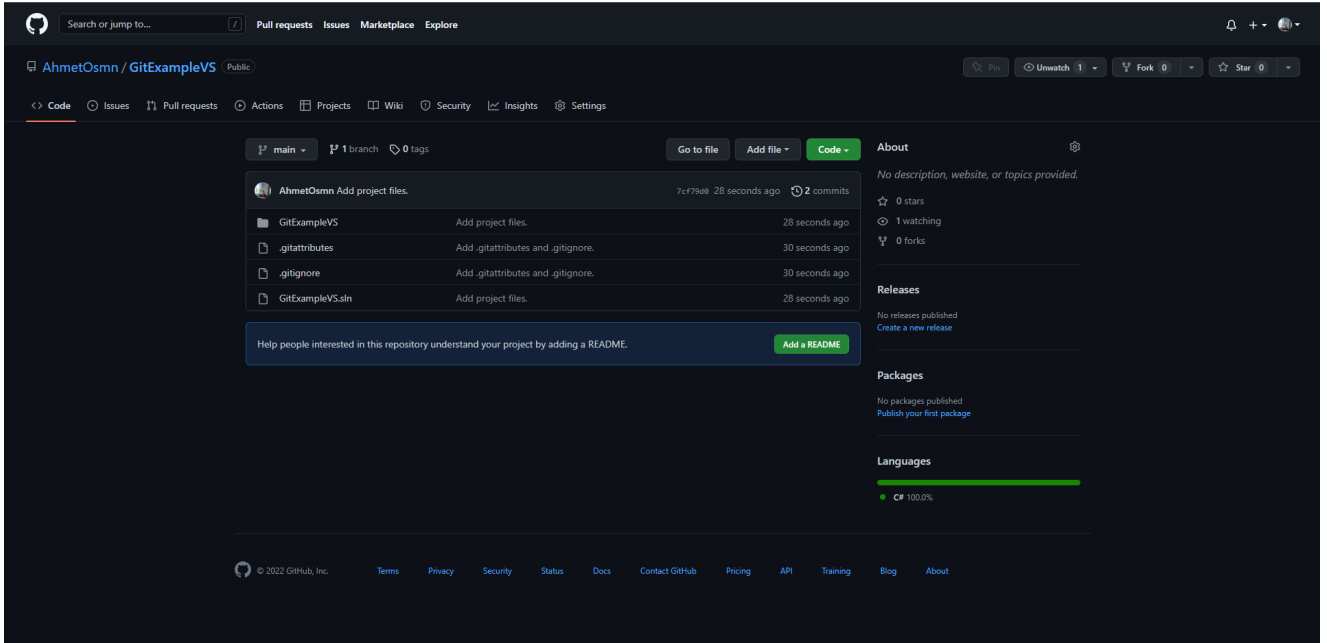


- Yukarıda kırmızı olarak işaretlenen *Create Git Repository* seçeneğine tıkladığımızda projeyi GitHub'ta paylaşırken yapacağımız bazı ayarları seçeceğimiz alt kısımdaki ekran açılacaktır.



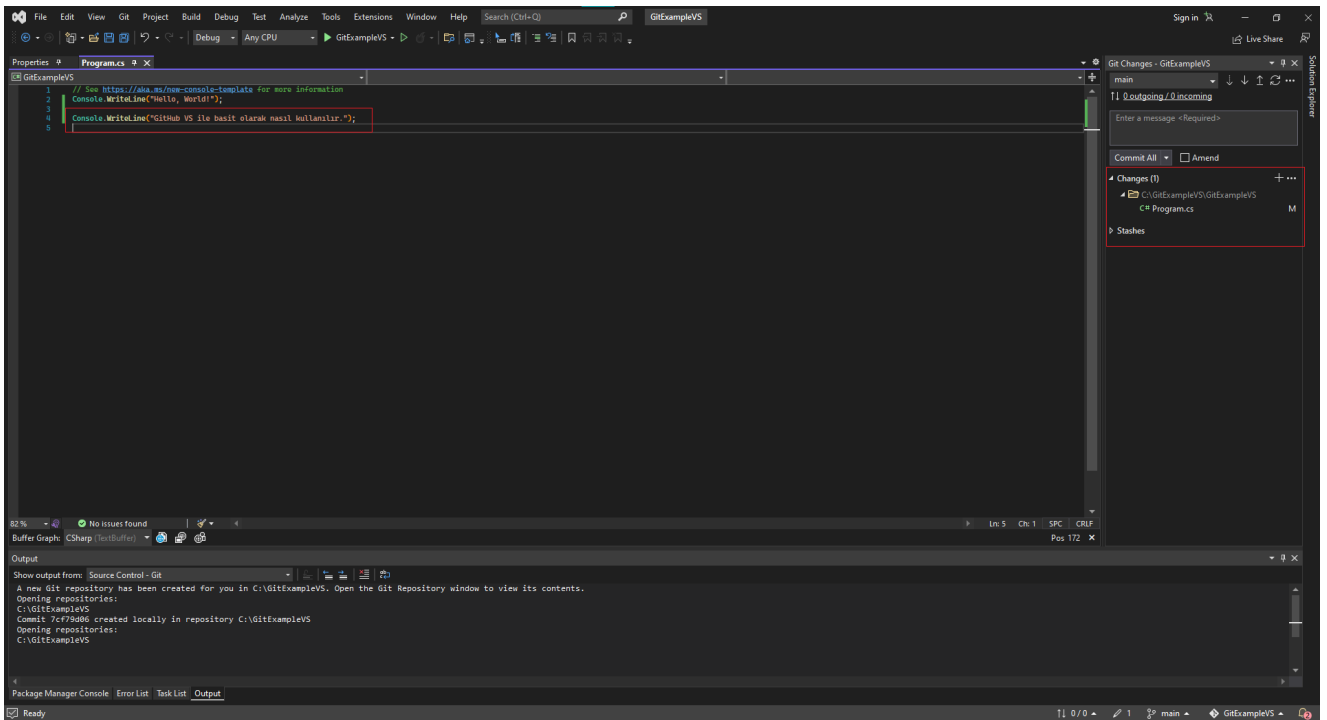
Projeyi paylaşırken yapmak istediğimiz ayarları da düzenledikten sonra kırmızı olarak seçilen *Create and Push* seçeneğine tıkladığımızda artık proje GitHub'ta paylaşılmış olacaktır. İşlemin bilgilendirme mesajını ve daha fazla detayını ekranda açılan alanlardan inceleyebiliriz.



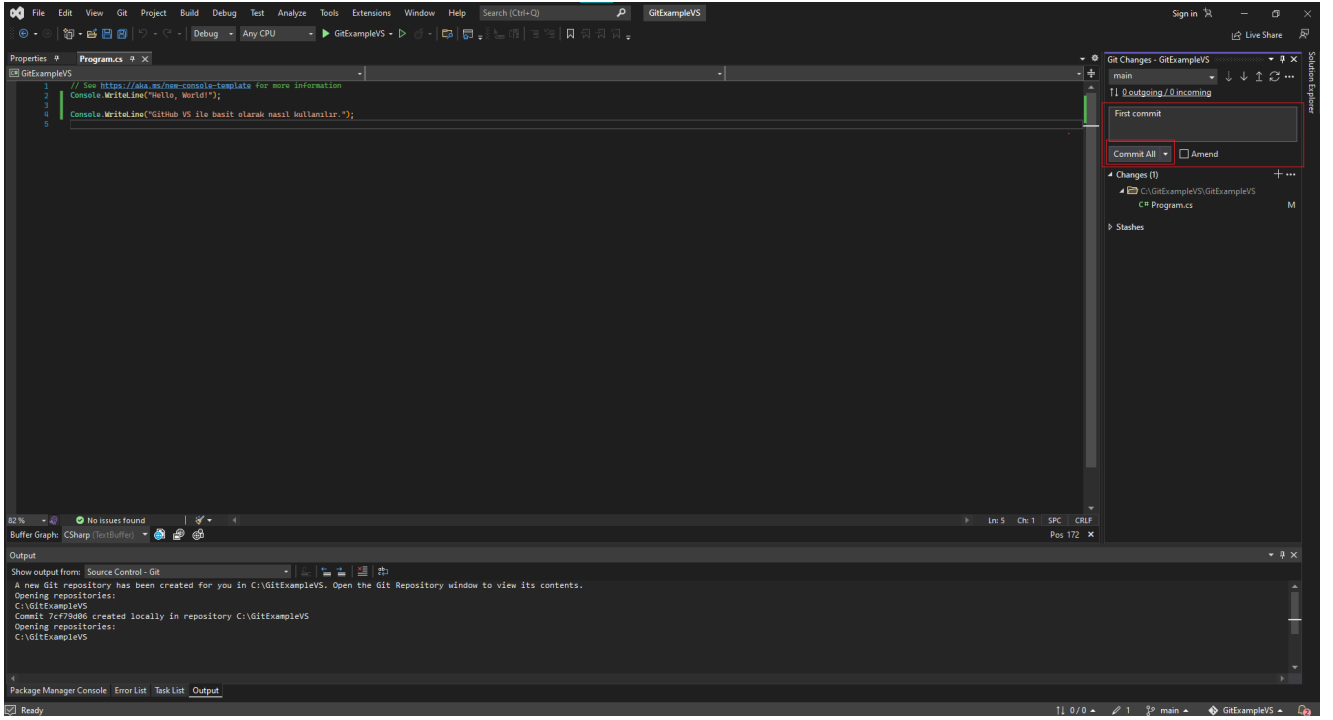


- Projeyi GitHub'ta paylaştıktan sonra geliştirme yaptığımızda, değişiklikleri GitHub'a göndermek için de *Git Changes* menüsünü kullanabiliriz.

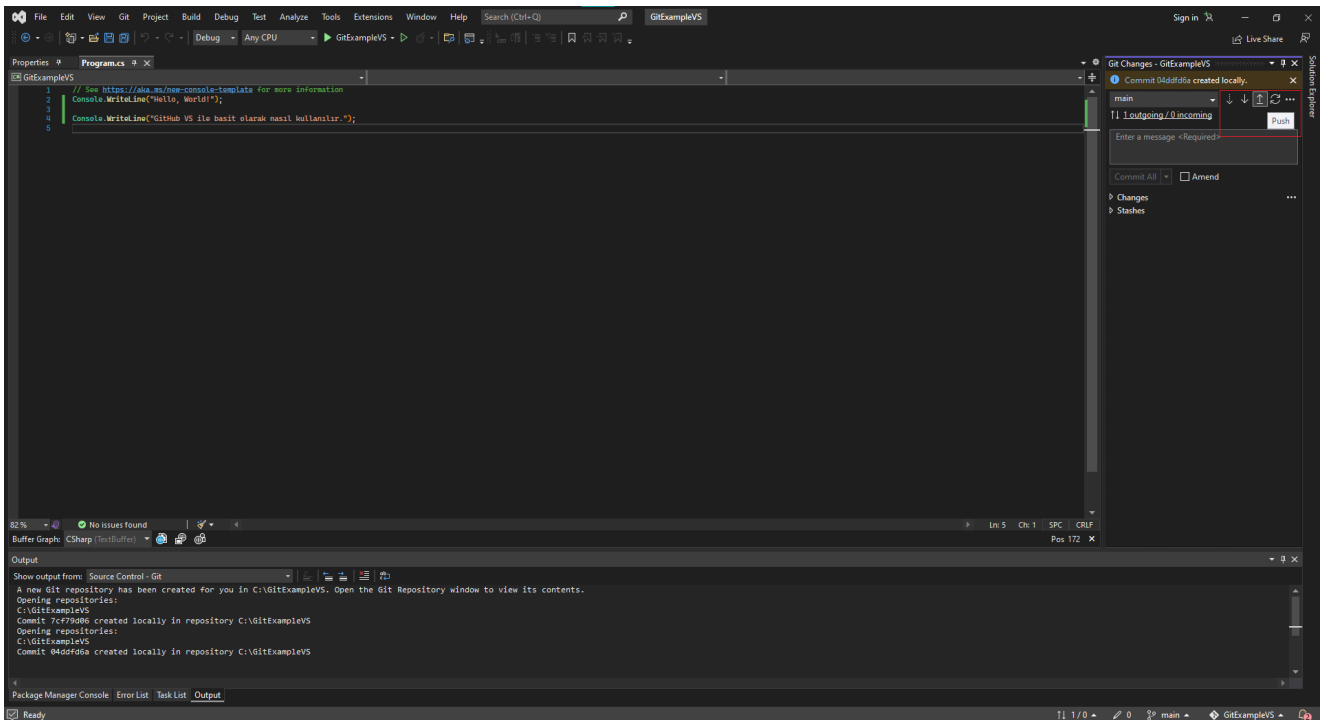
Projenin GitHub'ta olan halinin sonrasında yaptığınız değişiklikler alt kısımdaki gibi *Git Changes* içerisindeki *Changes* alanından görülecektir. Burada değişiklik yapılan dosyaları, bu dosyalarda nelerin değiştirildiğini eski hali ile birlikte kıyaslama yaparak görebiliriz. Bunun dışında yaptığımız değişiklikleri komle geri alabiliriz vb. bir çok detaylı işlemi buradan gerçekleştirebiliriz.



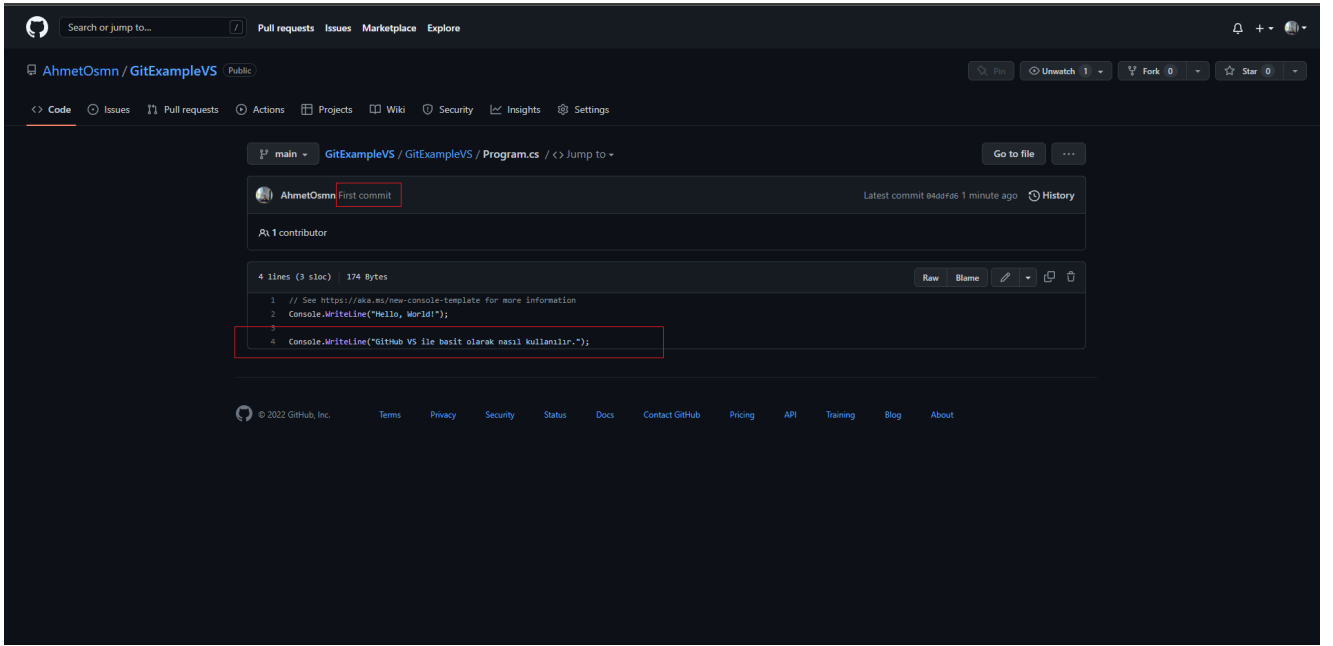
- Projenin üzerinde yapacağımız değişiklikler tamamlandığında, GitHub'a atmak için hazır hale geldiğinde alt kısımdaki gibi *Commit* alanına değişiklikler ile ilgili açıklayıcı bir commit yazarız. Daha sonrasında yaptığımız bütün değişiklikleri commitlemek için, commit alanının hemen altındaki *Commit All* butonunu kullanırız.



Commit işlemi tamamlandığında biz bir bilgilendirme mesajı veriliyor. Daha sonrasında Committediğimiz dosyaları GitHub'a göndermek için yapmamız gereken sadece alt kısımda kırmızı olarak seçilen alandaki *Push* butonuna (yukarı ok, ↑) basmamız yeterlidir.



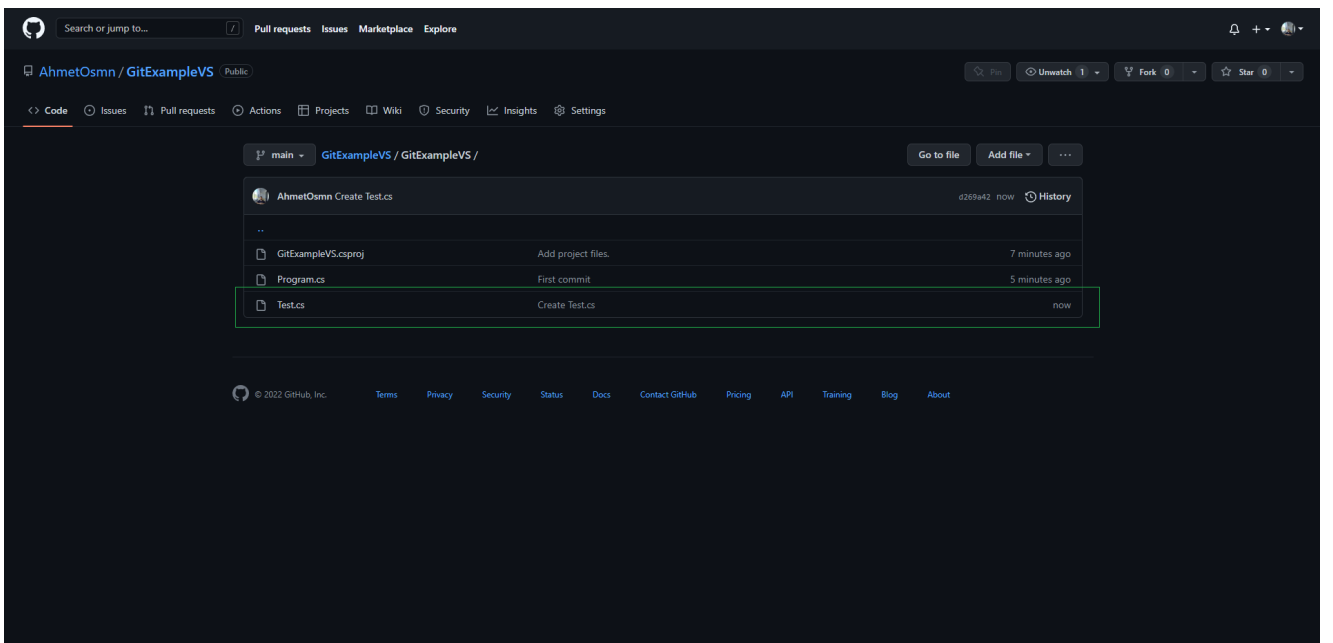
- GitHub'a gelip projenin içerisine girdiğimizde yaptığımız değişiklikleri ve commit mesajlarımızı görebiliriz.



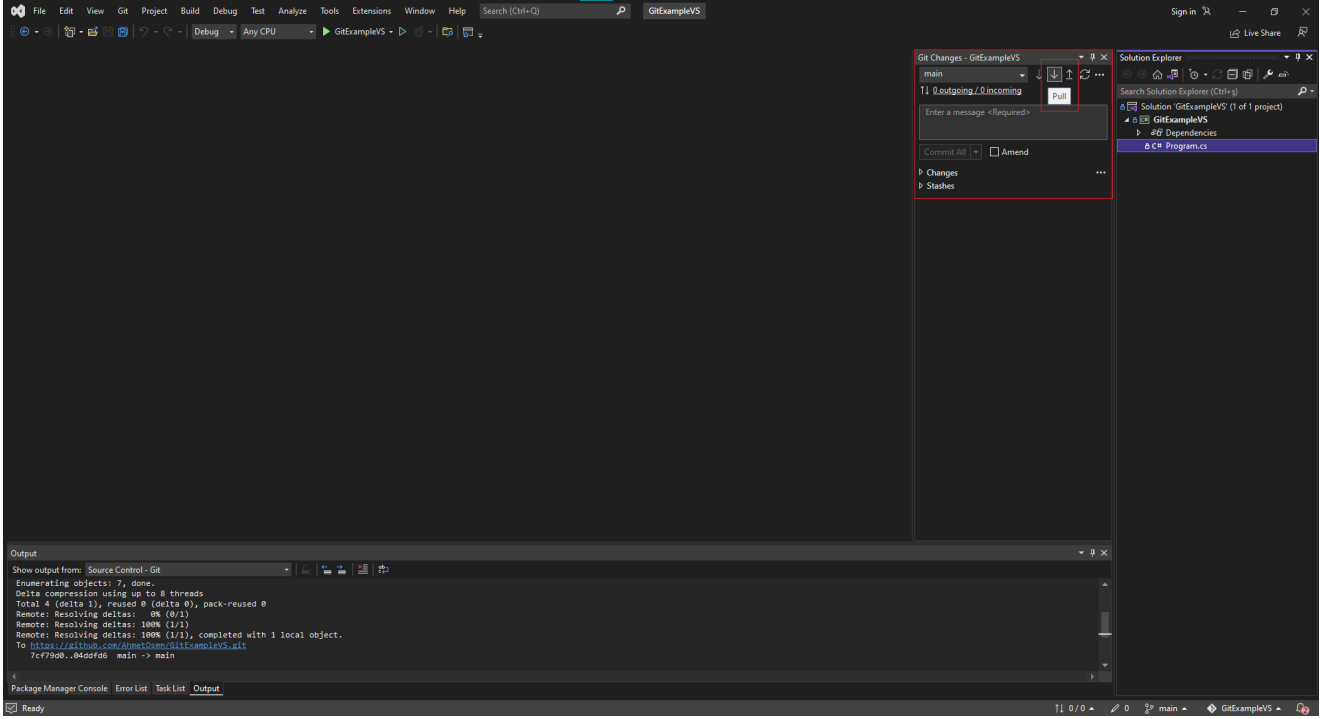
VS ile Projeyi Güncellemek

- Geliştirdiğimiz proje üzerinde takım arkadaşlarımız değişiklik yaptığında kendi bilgisayarımızdaki projeye güncellemeleri alıp sonrasında geliştirmeye devam etmeliyiz. GitHub üzerinden projenin güncel halini bilgisayarımıza çekmek istiyorsal alt kısımdaki örneğe bakabiliriz.

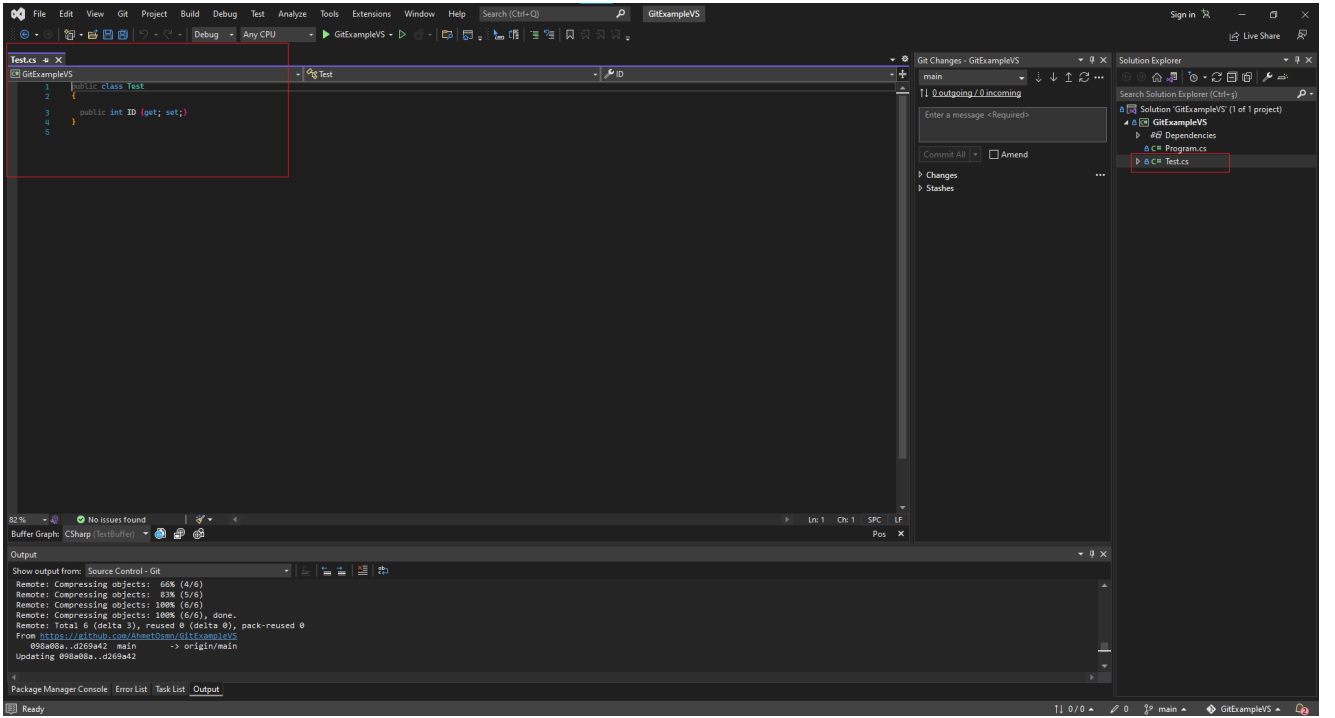
Örnek olarak bir takı arkadaşımız **Test.cs** adında bir dosya oluşturmuş olsun.



Bu dosya bizim kendi bilgisayarımızdaki projede yer almıyor. Bilgisayarımızdaki projeyi güncellemek için *Git Changes* alanındaki *Pull* butonuna (aşağı ok, ↓) basmamız yeterlidir.



İşlem tamamlandığında artık projemizin içerisinde yeni gelen dosyaları görebiliriz. Örnekteki **Test.cs** dosyası artık bizim projemizin içine gelmiş oldu.



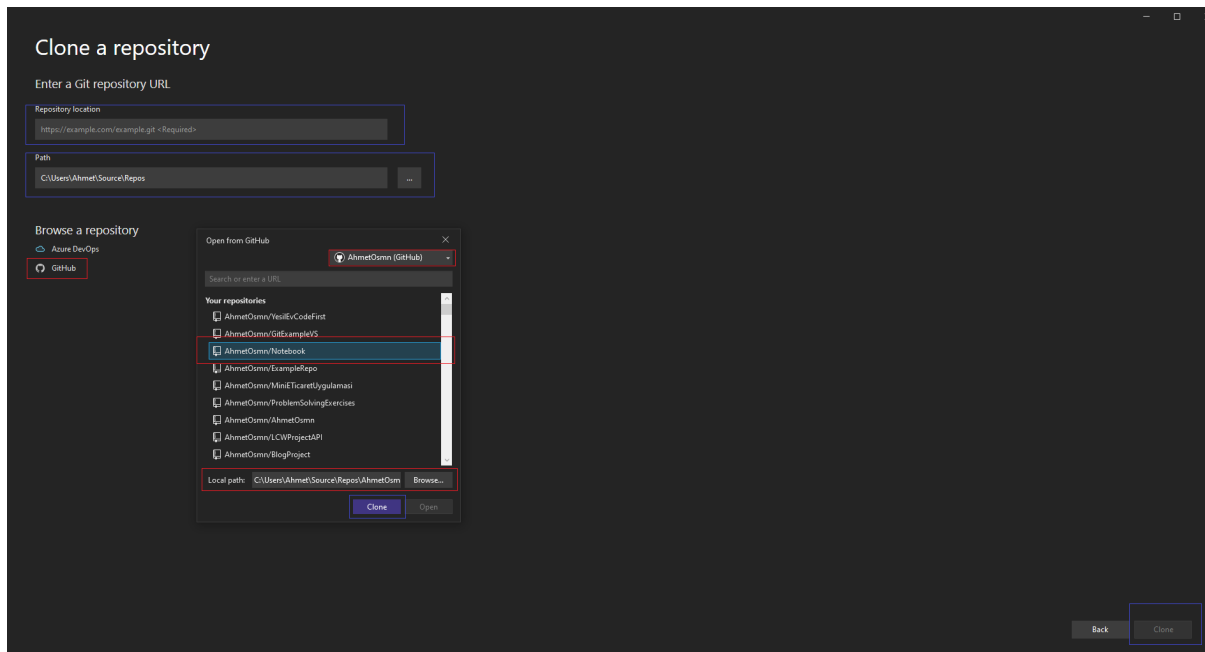
GitHub Üzerinden Proje Çekmek

- GitHub üzerindeki bir proje VS içerisinde nasıl kendi bilgisayarımıza çekmenin birden fazla yöntemi vardır. Biz burada biraz daha basit olarak görebileceğimiz adımları göstereceğiz. Eğer VS açıldığında alt kısımdaki pencere sizde gelmiyor ise VS içerisindeki yukarıda gösterdiğimiz *View* sekmesinin yanındaki *Git* sekmesinden de bu işlemleri yapabilirsiniz.

İlk olarak VS' açtığımızda karşımıza çıkan ekranda *Clone a Repository* seçeneğine tıklayalım. Alt kısımdaki ekran görüntüsünde yeşi olarak işaretlenen kısım.



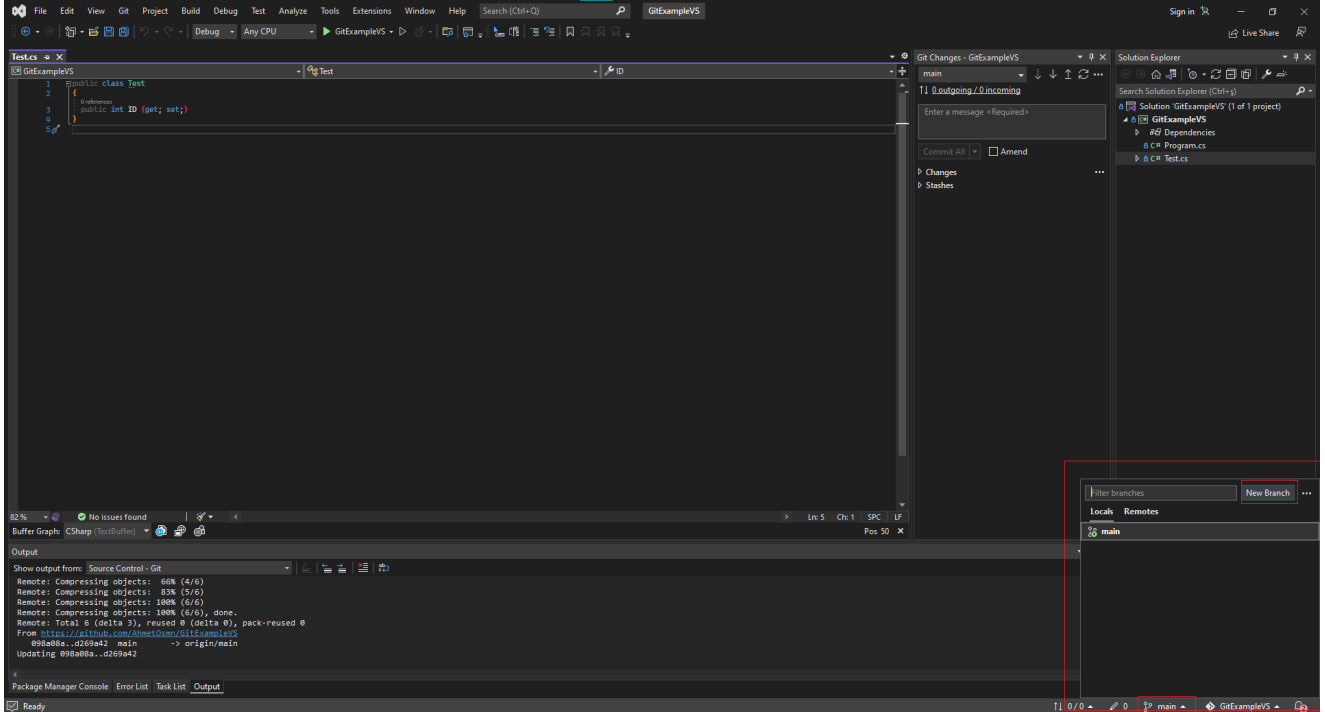
Daha sonrasında bizi çekeceğimiz Repo'ile ilgili yolların (path'lerin) düzenlendiği alt kısımdaki gibi bir sayfa karşılıyor.



- Yukarıdaki örnekte 2 farklı durum için gösterim mevcut:
 - Direkt Repo Adresi İle Repo Çekmek:
 - Bir projeyi direkt olarak adresini vererek çekmek istiyosak mavi alan ile işaretlenen kısımda üst alana çekilecek olan Repo'nun adresini yapıştırıyoruz.
 - Alt kısımdaki alanda ise projeyi kendi bilgisayarımızda nere kurmak istediğimizi seçiyoruz.
 - Daha sonra sağ alt kısımdaki *Clone* butonuna basıp projeyi çekebiliriz.
 - VS ile bağladığımız GitHub Hesabımızdaki Bir Repoyu Çekmek:
 - Profilimizdeki bir projeyi bilgisayarımıza çekmek için yukarıdaki ekran görüntüsünde kırmızı olarak işaretlenen alanları kullanabiliriz.
 - Öncelikle *Browse a repository* alanından *GitHub* seçeneğine tıklıyoruz.
 - Daha sonra yanda açılan pencerede, sağ üst kısımdan istediğimiz GitHub hesabının seçili olduğundan emin oluyoruz.
 - Alt kısımda seçili olan GitHub hesabının içerisinde var olan Repo'lar gelecektir. Buradan bilgisayarımıza çekmek istediğimiz Repo'yu seçiyoruz.
 - *Local path:* alanında Repo'yu bilgisayarımızda nereye kurmak istediğimizi belirtiyoruz.
 - Son olarak *Clone* butonuna basarak projeyi çekebiliriz.

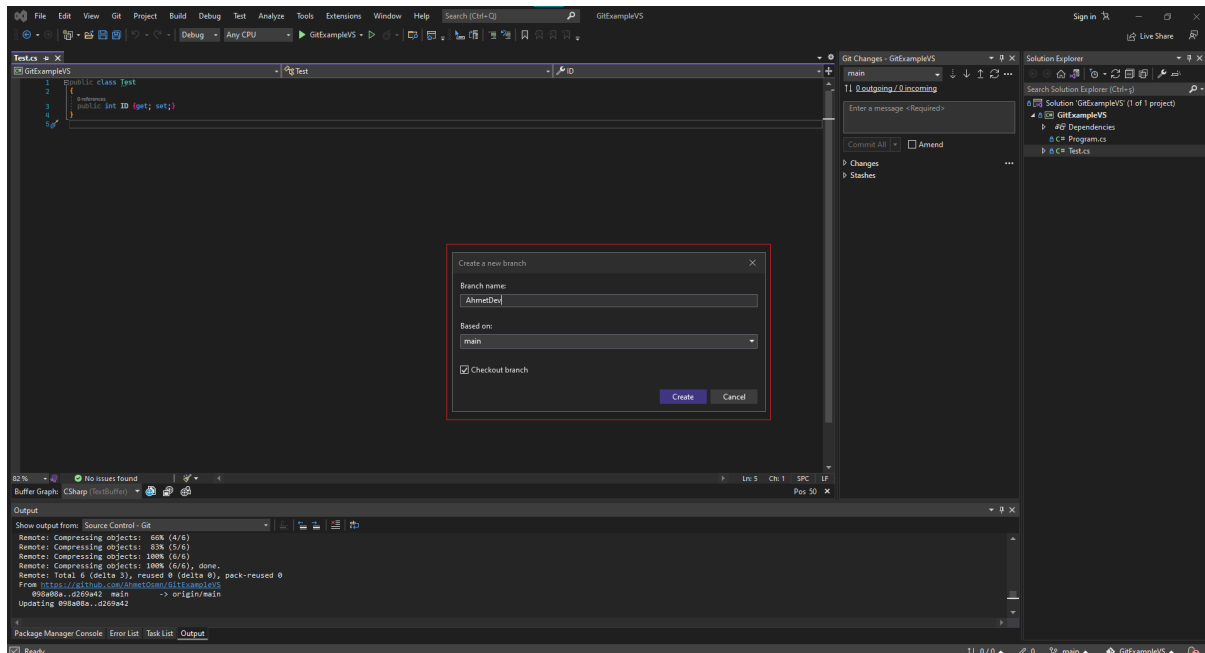
Dallar (Branches)

- VS üzerinde dallar arasında geçişler için de farklı yöntemler vardır. Örnek bir yol olarak ekranın sağ alt kısmında bulunan *main* butonuna tıkladığımızda alt kısımdaki bir pencere açılacaktır.



Burada daha önce açılmış ve silinmemiş branch'leri görebiliriz ve bu branch'ler arasında geçişler yapabiliriz. Örnek olarak yeni bir branch açmak istersek kırmızı olarak işaretlenmiş alandeki *New Branch* butonuna basmalıyız.

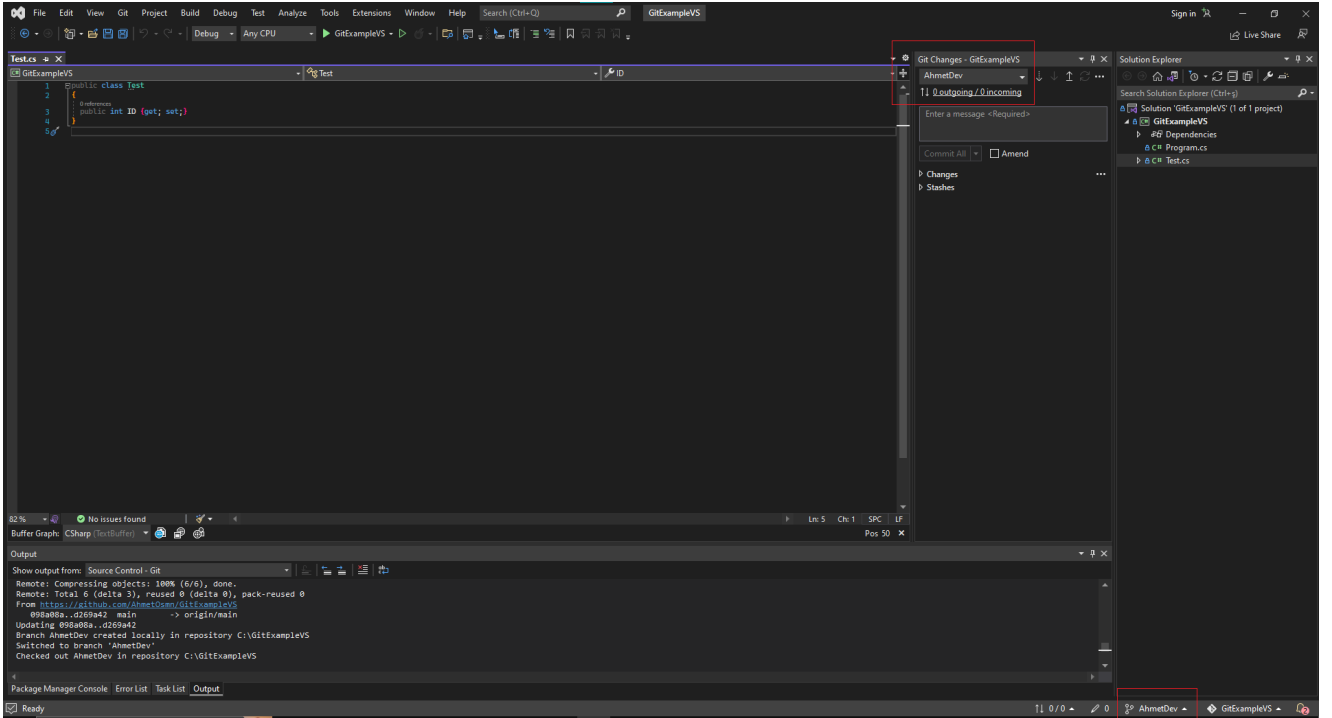
Butona bastığımızda alt kısımdaki pencere açılacaktır.



Açılan ekranda *Branch Name* yazan kısma oluşturacağımız branch'ın ismini giriyoruz. Alt kısımdaki *Based ON* yazan kısımda ise hangi branch üzerinde yeni branch açacağımızı seçiyoruz.

Bu seçeneklerin alt kısmında bulunan *Checkout branch* seçeneğinin seçili olması, yeni branch oluşturulduktan sonra açılan branch üzerinden geliştirmeye devam etmemizi sağlar. Eğer bu seçeneği seçmezsek branch oluşturulacaktır fakat *main* branch'inde kalmaya devam edeceğiz ve daha sonrasında tekrar yeni açtığımız branch'e tıklayıp manuel olarak *checkout* yapmamız gerekecektir.

Gerekli işlemleri yaptıktan sonra *Create* butonuna basarak branch'i oluşturmuş oluruz ve *checkout branch* seçeneği seçili olduğundan direkt olarak yeni branch üzerinden geliştirmeye devam ederiz.



Yukarıdaki ekran görüntüsünden de görüldüğü gibi artık *main* yazan alanlarda yeni açtığımız branch'in ismi var.

Son olarak branch'ler arasında geişler yapacaksak tekrar branch ismine tıkladığımızda açılan pencerelerden bunu gerçekleştirebiliriz.

