

Temel Git Eğitimi

Umut ÇAKIR

umutcakirbm@gmail.com

Temel Git eğitiminde neler öğreneceğiz?

- Git versiyon kontrol sistemi kavramı ve önemi
- Git ve Visual Studio Code kurulumu
- Git temel komutları ve örnek kullanımları
- VS Code içinde terminal kullanarak ve kullanmadan Git temel komutlarının kullanılması
- .gitignore dosyası ve versiyon kontrol sistemine birden çok dosyanın eklenebilmesi
- GitHub'a projelerimizin eklenmesi ve diğer repo hosting web platformları hakkında bilgilendirmeler
- Markdown ve temel sözdizimlerinin (syntax) kullanımı
- Git Flow proje geliştirme dal (branch) modelinin kullanımı

Git Nedir?

Git, dağıtık bir sürüm (versiyon) kontrol sistemi ve kaynak kod yönetim aracıdır. Yazılım geliştirme projelerinde kullanılan popüler bir araçtır. Git, projelerin geçmiş versiyonlarını takip etmek, değişiklikleri yönetmek, farklı geliştiriciler arasında işbirliği yapmak ve projeleri güvenli bir şekilde paylaşmak için kullanılır.

- **Dağıtık Yapı:** Her kullanıcının kendi yerel kopyası vardır. İnternet bağlantısı olmadan bile çalışma ve değişiklikleri güvenli bir şekilde kaydedebilme imkanı sağlar.
 - **Hızlı ve Verimli:** Performans odaklı tasarlanmıştır. Sadece değişen dosya bilgilerini ek olarak saklar.
 - **Branching / Merging:** Farklı kod çalışma dalları oluşturulabilir ve birleştirilebilir.
 - **Ekip Halinde Çalışma:** Her geliştirici kendi dalında geliştirmesini yapabilir ve bunu paylaşabilir. Birleştirme özelliği sayesinde bu kodlar birleştirilir.
 - **Güvenlik / İşlemi Geri Alma:** Projenin geçmişi ve her ayrıntı saklanır. Herhangi bir değişikliği geri almak veya geçmişe dönüş yapmak mümkündür.
-

Git ve Visual Studio Code Kurulumu

Bilgisayarlarımıza Git ve VS Code kurulumuna geçebiliriz.

Git Temel Komutları

- **git config:** Git yapılandırma ayarlarını yönetmenizi sağlar. Bu komutla Git ile ilgili kullanıcı adı, e-posta, renk ayarları ve diğer yapılandırmaları yapabilirsiniz. Örneğin, `git config --global user.name "John Doe"` komutuyla kullanıcı adınızı ayarlayabilirsiniz.

```
$ git config --global user.name "Name Surname"
```

```
$ git config --global user.email "test@email.com"
```

Git yapılandırma ayarları, yerel (proje düzeyinde) veya global (sistem düzeyinde) olarak yapılandırılabilir. `--global` seçeneği kullanılarak global yapılandırmalar, tüm projelerinizde geçerli olacak şekilde ayarlanırken, seçeneksiz kullanıldığında yerel yapılandırmalar, yalnızca belirli bir projede geçerli olacak şekilde ayarlanır.

Olası Hata (Windows): warning: LF will be replaced by CRLF in kaynak/dosya/yolu

```
$ git config core.autocrlf true
```

Git Temel Komutları

- **git init:** Bir Git deposu oluşturur. Bu komutu, bir projeyi Git ile izlemeye başlamak için proje dizininde çalıştırabilirsiniz.
- **git clone <repo-url>:** Bir uzak Git deposunu yerel makinenize kopyalar. <repo-url>, kopyalamak istediğiniz reposunun URL'sini temsil eder.
- **git add <dosya-adı>:** Bir dosyayı Git tarafından izlenen dosyalara ekler. Değişiklikleri Git'in takip etmesini istediğiniz dosyaları bu komutla belirtmelisiniz. Örneğin, git add dosya.txt dosyasını izlemeye alır.
- **git commit -m "<açıklama>":** Yapılan değişiklikleri bir commit olarak kaydeder. <açıklama>, commit için açıklayıcı bir mesajdır ve yapılan değişikliği tanımlar.
- **git push:** Yereldeki değişiklikleri uzak bir Git sunucusuna gönderir. Bu komut, yerel deponuzdaki commit'leri uzak sunucuya yükler ve paylaşmanızı sağlar.
- **git pull:** Uzaktaki bir Git deposundan değişiklikleri alır ve yerel projenize birleştirir. Bu komut, başka bir geliştiricinin yaptığı değişiklikleri güncellenenizi sağlar.
- **git branch:** Mevcut dalları listeler. Bu komutu çalıştırarak mevcut dalları görüntüleyebilir ve hangi dalda olduğunuzu öğrenebilirsiniz.

Git Temel Komutları

- **git checkout -b <yeni-dal>**: Yeni bir dal oluşturur ve üzerine geçer. <yeni-dal>, oluşturmak istediğiniz yeni dalın adını temsil eder.
- **git checkout <dal>**: Mevcut dali değiştirir. <dal>, üzerine geçmek istediğiniz dali temsil eder.
- **git merge <dal>**: İlgili dali mevcut dala birleştirir. Bu komut, farklı dallardaki değişiklikleri birleştirerek kodu entegre etmenizi sağlar.
- **git status**: Değişiklikleri ve depo durumunu kontrol eder. Bu komutu çalıştırarak hangi dosyaların değiştiğini, hangi dosyaların staged (hazır) olduğunu ve hangi dosyaların commit edilmesi gerektiğini görebilirsiniz.
- **git log**: Commit geçmişini görüntüler. Bu komut, yapılan commit'leri, commit kimliklerini, tarihleri ve commit mesajlarını görüntüler.
- **git diff**: Değişiklikleri gösterir. Bu komut, çalışma dizinindeki değişiklikleri ve henüz staged (hazır) olmayan değişiklikleri gösterir. Kullanımı git diff veya git diff dosya.txt şeklindedir. İkinci kullanım, belirli bir dosyadaki değişiklikleri görüntüler.
- **git rm**: Bir dosyayı Git deposundan ve dosya sisteminizden kaldırır. Kullanımı git rm dosya.txt şeklindedir. Bu komutla dosyayı silip, bir sonraki commit'te silinmiş olarak işaretleyebilirsiniz. Staged ortamına eklenmiş bir dosyanın takibinin bırakılması yani untracked (izlenmeyen) hale getirilmesi sağlayan komuttur.

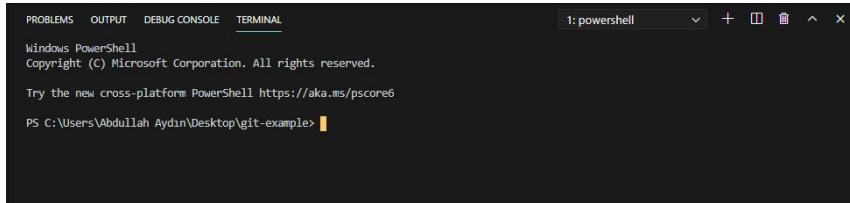
git rm --cached <dosya veya klasor_adi>

Visual Studio Code

Terminal kullanarak temel Git komutlarının kullanımını inceleyeceğiz.

Terminale Erişim

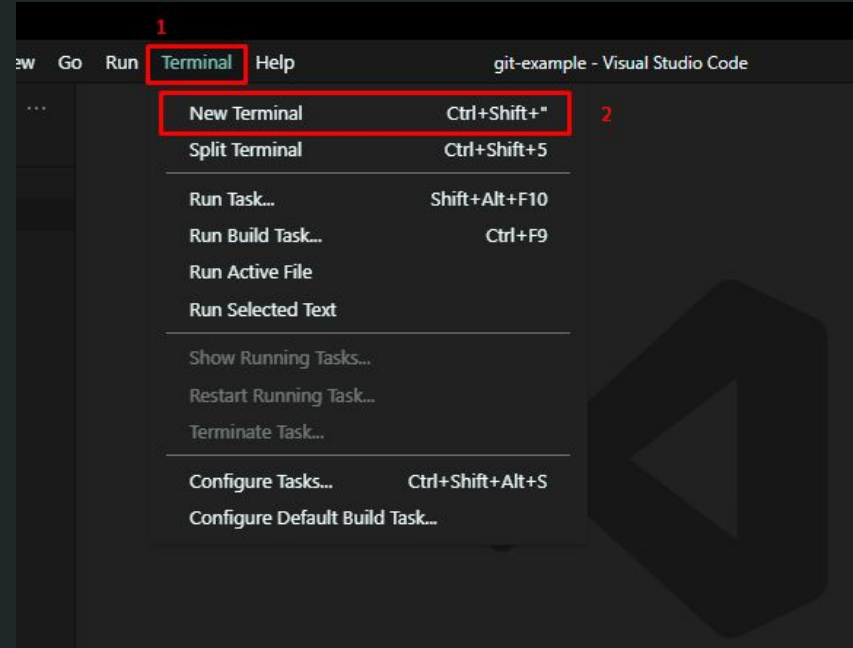
GIT temel komutlarını VS Code içerisindeki terminalde kullanmak için menü barından Terminal'i (1) açarak New Terminal (2) seçeneğini seçmelisiniz.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
Windows PowerShell
Copyright (c) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Abdullah Aydın\Desktop\git-example>
```

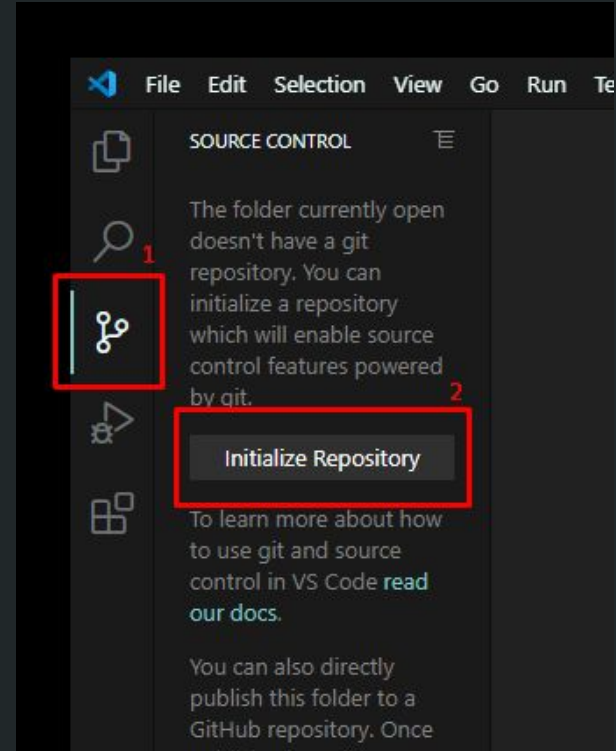


Visual Studio Code

Terminal kullanmadan temel Git
komutlarının kullanımını inceleyeceğiz.

Git'in Aktifleştirilmesi

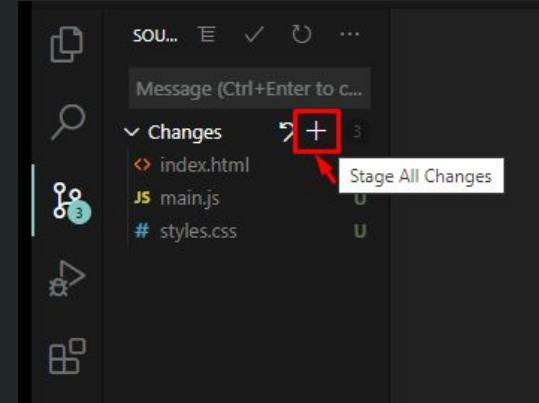
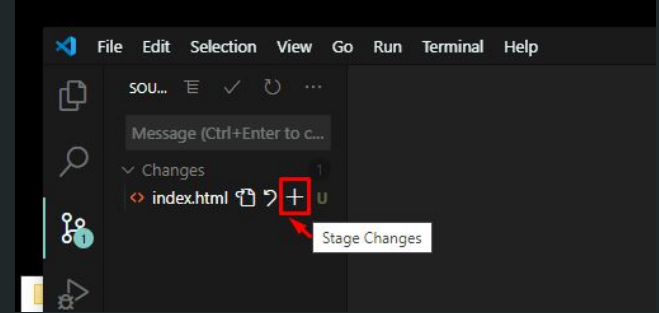
Henüz versiyon kontrolü altında olmayan bir projenin dizininde, boş bir git deposu oluşturmak için Activity Bar bölümünden Source Control (1) ikonuna tıklayıp, Initialize Repository (2) butonuna tıklamalıyız.



Değişikliklerin Git'e Eklenmesi

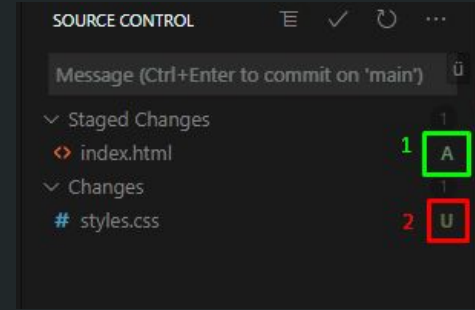
Yeni eklenen veya üzerinde değişiklik yapılan dosyaları staged ortamına göndermek için Stage Changes butonuna tıklamalıyız.

Birden fazla dosyamız olduğu zamanlarda eğer bütün değişiklikleri tek bir seferde staged ortamına göndermek istiyorsak Stage All Changes butonuna tıklamalıyız.



Değişikliklerin İzlenmesi

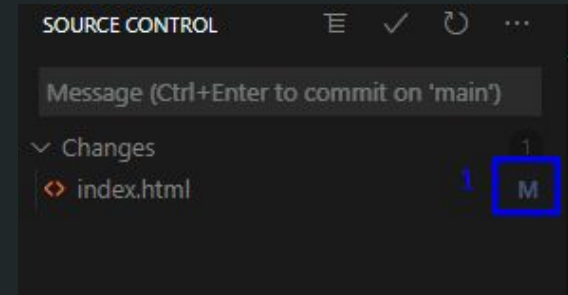
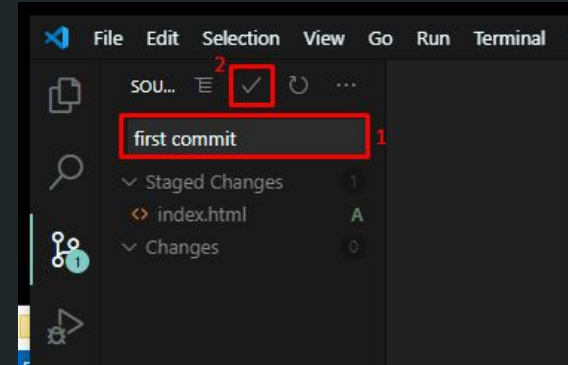
Staged ortamına dosyayı eklediğimizde aşağıdaki resimde olduğu gibi, dosyanın yanında "A" (1) (added) yazacaktır. Staged ortamına eklemediğimiz dosyalar olursa bu dosyaların yanında da "U" (2) (untracked) yazacaktır.



Commit İşlemi

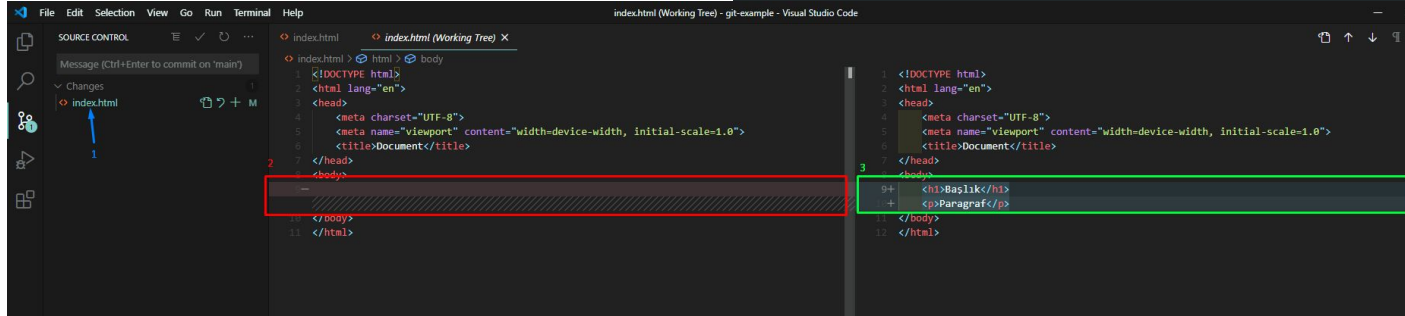
Commit, staged ortamına alınan dosyaların Local Repository'e gönderilmesidir. En iyi uygulama yöntemi her kayıt sırasında yapılan değişiklikleri açıklayıcı bir mesaj eklemektir. Ayrıca her commit benzersiz bir kimliğe (unique ID) sahip olur. Dosyalarımızı commit'lemek için Message bölümüne (1) commit'imizi açıklayıcı bir mesaj yazmalıyız ve ardından Commit butonuna (2) basmalıyız.

Commit'lenen dosya üzerinde değişiklik yaptığımızda, dosyanın yanında "M" (1) (modified) yazacaktır.



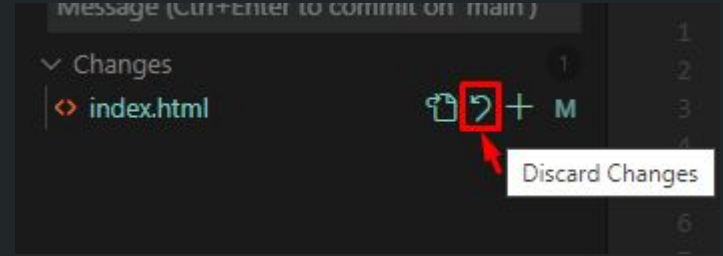
Değişikliklerin Görüntülenmesi

Dosyamızda yapılan değişikliği görüntülemek için, Source Control bölümünde, dosyanın üzerine tıkladığımızda (1), iki farklı bölüm karşımıza geliyor. En sağdaki bölümde (3) dosyamızın üzerinde yaptığımız değişiklikleri görüntüleyebiliriz.



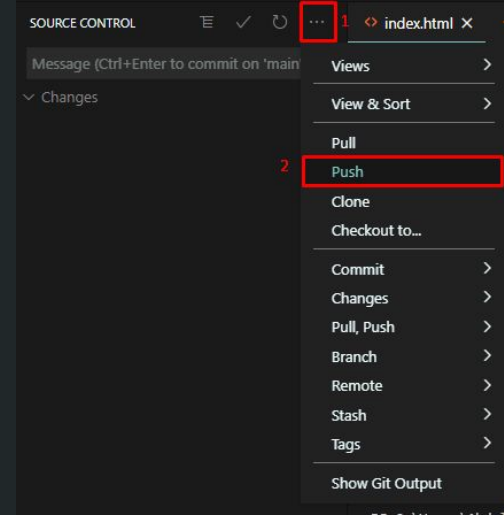
Değişikliklerin Geri Alınması

Yaptığımız değişiklikleri geri almak istersek, tekrar sol bölümdeki (2) gibi olmasını istiyorsak Discard Changes butonuna tıklamalıyız.



Değişikliklerin Sunucuya Gönderilmesi

Eğer remote repository'e bağlıysak ve commit'lerimizi remote repository'e göndermek istersek Views and More Actions (1) butonuna tıklayıp, Push (2) seçeneğini seçmeliyiz.



.gitignore Dosyası

Git reposunun .gitignore dosyası ile kontrol edilmesini inceleyeceğiz.

.gitignore

Git reposunda izlenmeyen dosyaları ve izinleri belirlemek için kullanılan bir dosyadır. Bu dosya, Git'e hangi dosyaların ve izinlerin izlenmemesini sağlayacağını söyler. İzlenmeyen dosyalar, Git deposuna eklenmez ve değişiklikler takip edilmez.

- **Derleme Çıktıları:** Derlenmiş dosyalar veya derleme sürecinde oluşan geçici dosyalar (*.class, *.o, *.pyc vb.) .gitignore dosyasına eklenerek Git deposunda izlenmez.
 - **Yerel Yapılandırma:** Kullanıcının yerel yapılandırma ayarları veya kimlik bilgileri (config.json, credentials.ini vb.) .gitignore dosyasına eklenerek Git deposuna eklenmez.
 - **Geçici Dosyalar:** Çalışma ortamına veya metin düzenleyiciye özgü geçici dosyalar (*.swp, ~*, .DS_Store vb.) .gitignore dosyasına eklenerek izlenmez.
-

.gitignore Kullanımı

- Proje dizininde .gitignore dosyası oluşturun (eğer yoksa).
- .gitignore dosyasını bir metin düzenleyiciyle açın ve izlenmeyen dosya ve dizin kalıplarını belirtin. Her satırda bir kalıp olmalıdır. Örneğin:

```
*.class  
config.json  
build/
```

- Dosya ve dizinlerin hangi kriterlere göre eşleşeceğini anlamak için Git'in dosya eşleme kurallarını gözden geçirin. Örneğin, *.class kalıbı tüm .class uzantılı dosyaları, build/ kalıbı ise "build" adlı bir dizini belirtir.
- .gitignore dosyasını kaydedin ve Git reposuna ekleyin (git add .gitignore).
- Artık .gitignore dosyasında belirttiğiniz dosyalar ve dizinler izlenmeyecektir. Bu dosyaları veya dizinleri Git deposundan çıkarmak isterseniz, .gitignore dosyasını güncelleyerek değişiklikleri yapabilirsiniz.

.gitignore dosyası, projenin daha temiz ve sade bir şekilde yönetilmesini sağlar. Özellikle geçici veya üretilen dosyalar gibi gereksiz dosyaların Git deposuna eklenmesini engeller. Böylece, sadece önemli kaynak dosyalarını ve proje dosyalarını depoda tutabilir ve gereksiz yığılmayı önleyebilirsiniz.

.gitignore Dosyasında “!” İşareti

- .gitignore dosyasında ! işareti, "hariç tutma" veya "izin verme" anlamında kullanılır. Normalde .gitignore dosyasında belirttiğiniz kalıplar, eşleşen dosyaları izlememek için kullanılır. Ancak ! işaretiyle birlikte kullanıldığında, belirli bir kalıba sahip dosyaları izlemek veya hariç tutulan bir dosyayı yeniden izlemek için kullanılır.

Örneğin, aşağıdaki örnekte .gitignore dosyasında *.txt kalıbı ile tüm .txt uzantılı dosyaların izlenmemesini sağlıyoruz, ancak bu kuralın altında !important.txt kalıbı ile important.txt dosyasını izlemeyi istiyoruz:

****.txt***

!important.txt

Bu durumda, tüm .txt uzantılı dosyaları takip etmeyen bir .gitignore kuralı belirtiyoruz. Ancak important.txt dosyasını hariç tutma kuralı olarak !important.txt eklediğimiz için bu dosya Git tarafından izlenecektir.

GitHub ve Diğer Repo Hosting Platformları

GitHub'a projemizin eklenmesini ve diğer
ünlü repo hosting platform isimlerini
öğreneceğiz.

GitHub'da Yeni Repo Oluřturulması

GitHub'da repo oluřturabilmek iin hesap oluřturma ařamalarından sonra profilimizde bulunan "Repositories" menüsünden saė kısımda bulunan "New" butonu ile repo oluřturma kısmına eriřebiliriz.

Owner: Reponun sahibinin seiyoruz.

Repository name: Oluřturduėumuz reponun adını belirliyoruz.

Description: Repomuz iin bir aıklama girebiliriz.

Public: Repomuzun tm herkesin eriřimini saėlar.

Private: Bu seenek sayesinde repomuzu gizli bir řekilde oluřturabiliriz.

Add a README file: Repomuza ncesinde README dosyası oluřturmuř oluruz, dilersek daha sonrasında kendimiz ekleyebilir ve gncelleyebiliriz.

Add .gitignore: Repomuza ncesinde .gitignore dosyası oluřturabilir ve gndermek istemediėimiz dosyaları seebiliriz, daha sonrasında aynı iřlemi yapabilir ve .gitignore dosyamızı gncelleyebiliriz.

Choose a license: Reponuz iin bir lisans seimi yapabilmenizi saėlar.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

Repository name *



oguuzduran

/

Great repository names are short and memorable. Need inspiration? How about [improved-barnacle?](#)

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☒ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☒ Choose a license

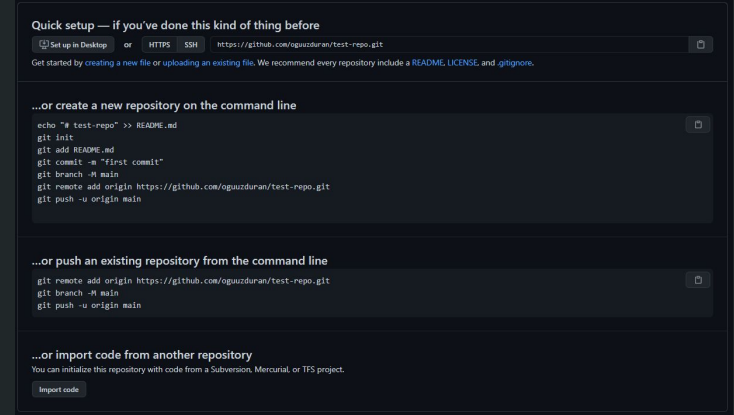
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

Yeni Repo Ekranı

Gerekli işlemleri tamamladıktan sonra "Create repository" diyerek repomuzu oluşturmuş oluruz.

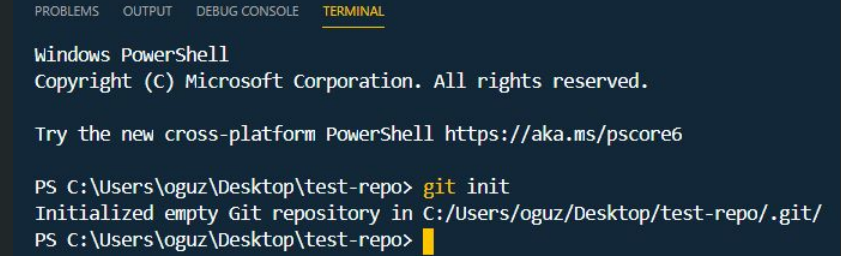
Şimdi, oluşturmuş olduğumuz repomuz bilgisayarımızdan erişebilmek için gerekli işlemleri yapmaya başlayacağız.



Bilgisayarda Repo Kurulumu

Bilgisayarımızda oluşturmuş olduğumuz klasörümüze konsol ekranımızdan veya kullandığımız IDE yardımı ile erişim sağladıktan sonra git init komutumuzu çalıştıralım.

Klasörümüz hazır, öncelikle README.md dosyamızı oluşturalım ve repomuza ilk push işlemini yapabilmek için adımları tamamlayalım.

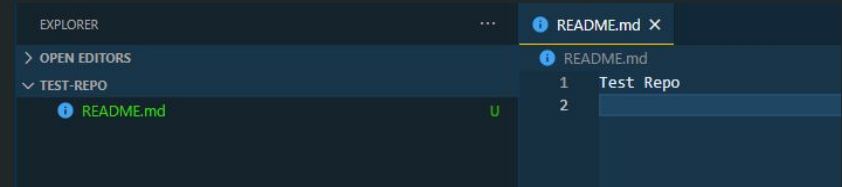


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\oguz\Desktop\test-repo> git init
Initialized empty Git repository in C:/Users/oguz/Desktop/test-repo/.git/
PS C:\Users\oguz\Desktop\test-repo>
```



```
EXPLORER  ...  1 README.md X

> OPEN EDITORS
1 README.md
2 Test Repo

1 README.md
2
```

Sunucuya Gönderim

Eklemiş olduğumuz README dosyasını repomuza gönderebilmek için git add README.md komutumuzu ile README dosyasının GitHub repomuza göndermek üzere hazırlayalım.

Repomuza son yapılan değişiklikleri göndermeden önce git commit -m "ilk yorum" komutu ile neler yaptığımızı yazalım.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\oguz\desktop\test-repo> git add README.md
PS C:\Users\oguz\desktop\test-repo>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\oguz\desktop\test-repo> git add README.md
PS C:\Users\oguz\desktop\test-repo> git commit -m "readme dosyasi eklendi"
[master (root-commit) cb1f520] readme dosyasi eklendi
1 file changed, 1 insertion(+)
create mode 100644 README.md
PS C:\Users\oguz\desktop\test-repo>
```

Sunucuya Gönderim

Bize belirtilen şekilde git branch -M main komutunu konsol ekranımızda çalıştıralım ve main branch oluşturalım.

Son aşamaya gelmeden önce ise git remote add origin 'repo-link' komutu ile remote bağlantımızı ekleyelim.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\oguz\Desktop\test-repo> git branch -M main
PS C:\Users\oguz\Desktop\test-repo>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

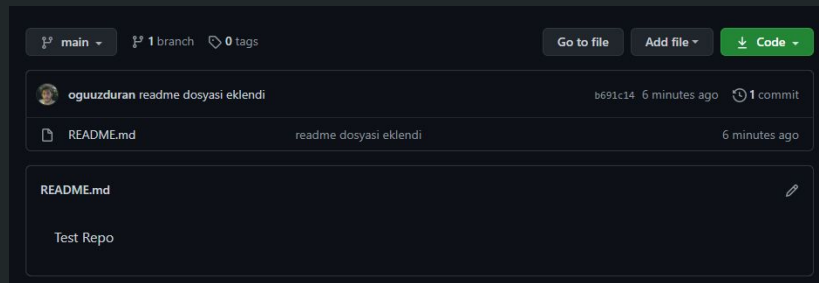
```
PS C:\Users\oguz\Desktop\test-repo> git add README.md
PS C:\Users\oguz\Desktop\test-repo> git commit -m "readme dosyasi eklendi"
[master (root-commit) b691c14] readme dosyasi eklendi
1 file changed, 1 insertion(+)
create mode 100644 README.md
PS C:\Users\oguz\Desktop\test-repo> git branch -M main
PS C:\Users\oguz\Desktop\test-repo> git remote add origin https://github.com/oguzduran/test-repo.git
PS C:\Users\oguz\Desktop\test-repo>
```

Sunucuya Gönderim

Son aşama olarak git push -u origin main komutu ile repomuza dosyalarımızı gönderelim.

Ve ilk push işlemimiz ile birlikte tüm değişikliklerimizi GitHub repomuza göndermiş olduk. GitHub sayfasını yeniden yüklediğimizde böyle bir ekran ile karşılaşmış olacağız.

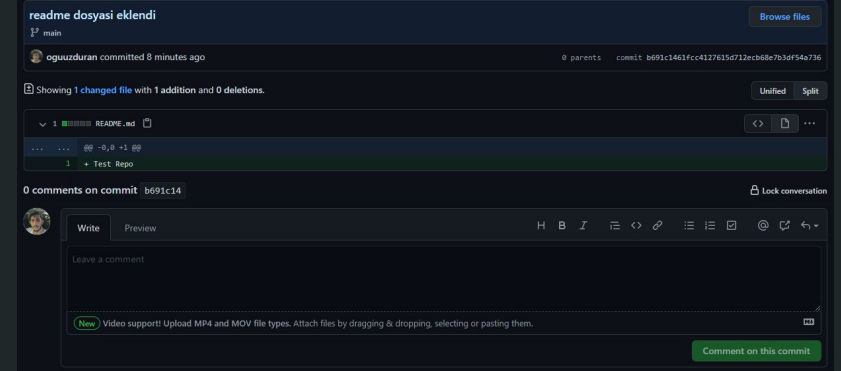
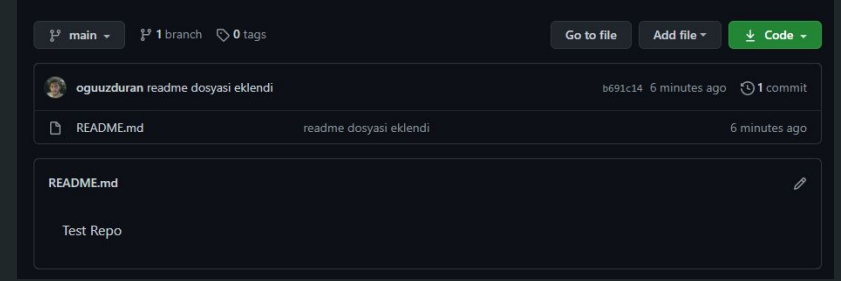
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\oguz\Desktop\test-repo> git add README.md
PS C:\Users\oguz\Desktop\test-repo> git commit -m "readme dosyasi eklendi"
[master (root-commit) b691c14] readme dosyasi eklendi
1 file changed, 1 insertion(+)
 create mode 100644 README.md
PS C:\Users\oguz\Desktop\test-repo> git branch -M main
PS C:\Users\oguz\Desktop\test-repo> git remote add origin https://github.com/oguuzduran/test-repo.git
PS C:\Users\oguz\Desktop\test-repo> git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 229 bytes | 229.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/oguuzduran/test-repo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
PS C:\Users\oguz\Desktop\test-repo>
```



Sunucuya Gönderim

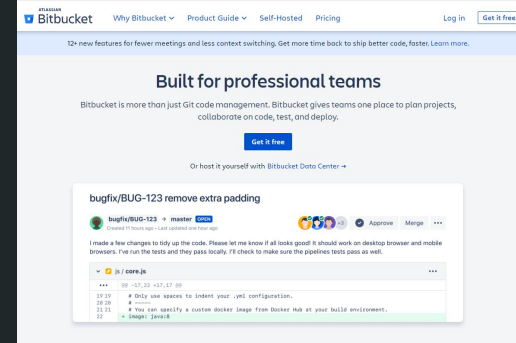
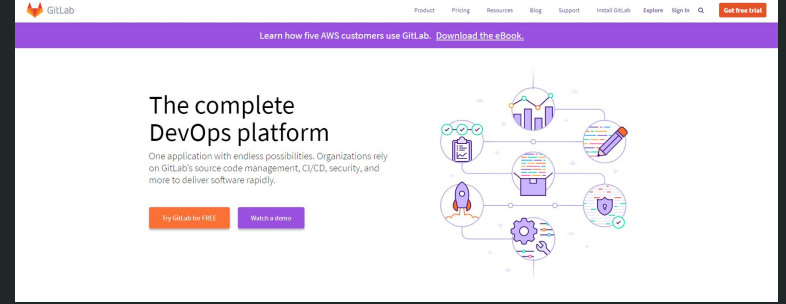
Böylece README dosyamızı repomuza göndermiş olduk. Sağ tarafta bulunan "1 commit" yazısının üstüne tıklayarak ise commit ile ilgili detayları görmüş oluruz.

Daha fazla detay görmek için ise sağ tarafta görünen mavi renk ile belirtilmiş commit işlemimiz için özel olarak random şekilde atanmış benzersiz yazı kısmının üstüne gelerek tıklamanız yeterli olacaktır.



GitHub Benzeri Repo Hosting Platformları

- **Gitlab:** “GitLab nedir?” sorusunun yanıtı GitLab ile GitHub arasındaki farkın daha iyi anlaşılması açısından oldukça önemli. GitLab web tabanlı bir Git depo uygulaması olarak tanımlanabilir. Bu depo servisi sürekli entegrasyon (CI), sürekli teslimat (CD), hata kayıt, kod gözden geçirme ve wiki desteğiyle çalışıyor.
- **Bitbucket:** Bitbucket, 2010 yılında Atlassian firması tarafından satın alınması ile beraber Mercurial ile birlikte Git desteği de vermeye başlayan ve günümüzde de hala sadece Git ve Mercurial versiyon kontrol sistemlerini (VCS) destekleyen, yazılım projeleri kodları için web tabanlı bir depolama servisedir.



Markdown

Projelerimize açıklama metinleri yazmamızı kolaylaştıran Markdown dili ve dosyalarını öğreniyoruz.

Markdown Nedir?

Markdown, basit bir metin biçimlendirme dilidir. Hafif ve kolay anlaşılır bir sözdizimine sahiptir ve özellikle web tabanlı dokümantasyon, notlar, blog yazıları, README dosyaları gibi belgelerin oluşturulmasında yaygın olarak kullanılır.

Markdown, metni daha okunabilir hale getirmek ve biçimlendirme öğeleri eklemek için basit işaretleri kullanır. İşaretler, belgenin içeriğine anlam katan ve belirli biçimlendirme özelliklerini sağlayan kurallara dönüştürülür.

- Başlıklar
 - Paragraflar
 - Vurgulamalar
 - Kalın Yazılar
 - Üstü Çizili Yazılar
 - Listeler
 - Bağlantılar
 - Resimler
 - Kod Blokları
-

Markdown Kullanımı

- **Başlıklar:** Başlıkları oluşturmak için # işaretini kullanabilirsiniz. # işareti ile başlayan satırlar, başlık düzeyini belirtir. Örneğin;

Başlık1

Başlık2

Başlık3

- **Paragraflar:** Paragraf: Paragraflar için normal metin yazabilirsiniz. Paragraflar arasında boş bir satır bırakarak paragrafları ayırabilirsiniz.
- **Vurgulamalar:** Metin içinde vurgulamak istediğiniz kelimeleri veya metin parçalarını * veya _ işaretleri ile sarabilirsiniz. Örneğin;

*Bu bir *vurgulama* örneğidir.*

Bu bir _vurgulama_ örneğidir.

Markdown Kullanımı

- **Kalın Yazılar:** Metni kalın olarak biçimlendirmek için ****** veya **__** işaretleri kullanabilirsiniz. Örneğin;
*Bu bir ****kalın**** yazı örneğidir.*
*Bu bir **__kalın__** yazı örneğidir.*
- **Üstü Çizili Yazılar:** Metni üstü çizili olarak biçimlendirmek için **^^** işaretlerini kullanabilirsiniz. Örneğin;
*Bu bir **^^üstü çizili^^** yazı örneğidir.*
- **Listeler:** Sıralı ve sırasız listeler oluşturabilirsiniz. Sırasız listeler için -, +, veya * işaretlerini kullanabilirsiniz. Sıralı listeler için ise rakamları kullanabilirsiniz. Örneğin;
 - Liste ögesi 1
 - Liste ögesi 2
 - Liste ögesi 3
 1. Sıralı öge 1
 2. Sıralı öge 2
 3. Sıralı öge 3

Markdown Kullanımı

- **Bağlantılar:** Metne bağlantılar ekleyebilirsiniz. Bağlantı metni ve hedef URL'sini [bağlantı metni](hedef-url) biçiminde kullanabilirsiniz. Örneğin;
[Google](https://www.google.com)
- **Resimler:** Belgeye resim ekleyebilirsiniz. Resim etiketi ve kaynak URL'sini ![resim açıklaması](resim-url) biçiminde kullanabilirsiniz. Örneğin;
![Logo](https://example.com/logo.png)
- **Kod Blokları:** Kod örneklerini veya kod bloklarını göstermek için ` işaretlerini kullanabilirsiniz. Tek satırlık kod örnekleri için arka arkaya üç tane ` işareti kullanabilirsiniz. Birden çok satırlık kod blokları için ise üç tane ` işaretinden sonra dilin adını belirtebilirsiniz. Örneğin;
Tek satırlık kod: `print("Merhaba dünya!")`
Blok:

```
'''  
  
def hello_world():  
    print("Merhaba dünya!")  
'''
```

Markdown Kullanımı

- Markdown dilinin sunduğu özellikler bunlarla sınırlı değildir. Tablolar, alıntılar, dipnotlar ve daha birçok özelliği Markdown ile kullanabilirsiniz. Markdown sözdizimine uygun olarak oluşturduğunuz belgeleri, çeşitli Markdown düzenleyicileri veya platformlar aracılığıyla HTML, PDF veya diğer formatlara dönüştürebilirsiniz.
- Markdown, basit ve hızlı bir şekilde belgeler oluşturmak için kullanışlı bir araçtır. Herhangi bir metin düzenleyiciyle veya Markdown destekli bir yazılımla Markdown belgelerini oluşturabilirsiniz.

Git Flow

Projelerimizi Git ile geliştirirken uygulanması önerilen dal (branch) modelini öğreniyoruz.

Git Flow

Git Flow, bir yazılım geliştirme süreci ve Git tabanlı bir dal modelidir. Vincent Driessen tarafından önerilen bir yaklaşımdır ve popüler bir sürüm yönetimi stratejisidir. Git Flow, projenin düzenli, paralel ve sürdürülebilir bir şekilde geliştirilmesini sağlar.

Git Flow, yazılım projelerinde ekip çalışmasını kolaylaştırır, paralel geliştirme ve sürdürülebilir bir sürüm yönetimi sağlar. Git Flow modelini kullanarak, geliştiricilerin işbirliği yapması, yeni özelliklerin doğru bir şekilde entegrasyonu ve yayınların güvenli bir şekilde gerçekleştirilmesi kolaylaşır.

- **Ana Dal (Main Branches):**
 - **master:** Ana dal olarak da bilinen master dalı, yayınlanan ve üretimdeki stabil sürümleri temsil eder. Bu dal üzerindeki kod, yayınlanmaya hazır, hatasız ve test edilmiş olmalıdır.
 - **develop:** Geliştirme sürecindeki kodu ve bir sonraki yayın için hazırlanan özellikleri içeren bir dal olarak develop dalı kullanılır. Geliştiriciler, yeni özelliklerin üzerinde çalışırken develop dalından türetilen kendi çalışma dallarını oluşturur.
 - **Destekleyici Dallar (Supporting Branches):**
 - **feature:** Yeni bir özelliği geliştirmek için feature dalları kullanılır. Yeni bir özellik üzerinde çalışmaya başlamadan önce, develop dalından feature dalı türetilir ve geliştirme tamamlandıktan sonra develop dala birleştirilir.
 - **release:** Yeni bir sürüm yayınlamak için release dalları kullanılır. develop daki kodun yayınlanmaya hazır olduğu düşünüldüğünde, release dalı oluşturulur. Bu dal üzerinde son düzeltmeler, belgelendirme ve sürüm numarası güncellemeleri yapılır. Sonuçta, release dalı hem develop hem de master dallarına birleştirilir.
 - **hotfix:** Canlıda ortaya çıkan acil hataları düzeltmek için hotfix dalları kullanılır. master daki hataları düzeltmek için hotfix dalı oluşturulur ve hızlı bir şekilde yayınlanır. Sonuçta, hotfix dalı hem master hem de develop dallarına birleştirilir.
-

Temel Git eğitimimizi tamamladınız!

Git Ödevi

Aşağıdaki kriterlere dikkat ederek Github üzerinde bir repo oluşturmanızı bekliyoruz. Repo içeriğini size bıraktık.

- İsmi “**patikaodev01**” olsun ve “**public**” erişim olsun.
- Repomuzda minimum 5 commit olmasını bekliyoruz.
- Commit mesajlarımızın anlaşılır ve uyumlu olmalarına dikkat edilmesi gerekiyor. Aynı zamanda commit atan kişi bilgisi de önemli.
- **README.md** dosyamız zorunlu ve dosyamızda öğrendiğimiz sözdizimi örneklerini görmek istiyoruz. Daha fazlasını kullanmakta özgürsünüz.
- **Git Flow** stratejisini uygulayarak farklı branchler kullandığınızı görmek bizi mutlu eder.
- Oluşturduğunuz repo URL bilgisini bizimle paylaşabilirsiniz.



```
git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
cat > README << 'EOF'
Git is a distributed revision control system.
EOF
git add README
git commit
[master (root-commit) e4dcc69] You can edit locally and push to
any remote.
1 file changed, 1 insertion(+)
commit mode 100644 README
git remote add origin git@github.com:cdown/thats.git
git push -u origin master
```

Bir sonraki adımımız ne olacak?

Temel Git kullanımını öğrendik.
Bundan sonraki dersimizde temel SQL eğitimi ile devam edeceğiz.
Gelecek konuları ve ödevlerimizi de Github üzerinden paylaşıyor olacağız.



Teşekkürler! :)