



git

GitHub

İçindekiler

1. GIT Nedir?.....	3
2. "git" Kelimesinin Anlamı.....	3
3. GIT'in Tarihsel Gelişimi	3
4. GIT Versiyon Kontrol Sistemi Nedir?	4
5. GIT Bize Ne Sağlar?.....	4
6. GitHub & GitLab & BitBucket Nedir?	4
7. Bilinmesi Gereken Bazı Terimler	5
8. GIT Komutları	6
9. VS Code İçerisinde Terminal Kullanarak GIT Temel Komutları	10
10. VS Code İçerisinde Terminal Kullanmadan GIT Temel Komutları	11
11. .gitignore Dosyası Ne İşe Yarar? Nasıl Kullanılır?	14
12. Projenin GitHub'a Eklenmesi	16
13. GIT Sitesi ve Kitabı	19
14. Markdown Nedir? Nasıl Kullanılır?	19
Kaynakça.....	22
EK – 1	23
EK – 2	24
GIT CHEAT SHEET	25

1. GIT Nedir?

GIT Kontrol Sistemi'nin ne olduğunu anlatmaya başlamadan önce biraz "git" kelimesinin anlamı ve bu sistemin kısa tarihsel gelişimi üzerine konuşalım. Peki bunu neden yapıyoruz?

Şu sebepten ötürü; Yazılım dehası Bill Gates çok kitap okuyan, okuduklarını iyi şekilde hatırlayan ve konuşmaları sırasında okuduğu kitaplardan alıntılar yapan bir şahsiyet. Bu kadar okuma yapıp da okuduklarını aklında detaylı şekilde nasıl tuttuğu sorulduğunda Bill Gates; Bir konuda okumaya başlamadan önce konu hakkında genel bilgi sahibi olmayı ve konunun tarihini incelemeyi öneriyor.

Bu içeriğimizde biz de *"GIT Versiyon Kontrol Sistemi / Kaynak Kod Yönetim Sistemi (SCM) Nedir?"* sorusuna cevap arayacağız. GIT kurulum videosu için [tıklayınız](#).

2. "git" Kelimesinin Anlamı

Cambridge Sözlüğü göre; aptal, hoş olmayan kişi anlamına geliyor. [1] Bu isimlendirme Torvalds'a sorulduğunda kendisi esprili şekilde şu cevabı veriyor: "Ben bir egoistim ve projelerime kendi ismimi veriyorum. Önceki Linux, şimdiki git.". Torvalds, alternatif olarak kelimenin ruh halinize göre aşağıdaki anlamlara da gelebileceğinden bahsediyor;

- ☞ Yaygın bir UNIX komutu tarafından kullanılmayan, telaffuz edilebilir, rastgele üç harfli bir kombinasyon.
- ☞ Sözlükteki argo anlamı ile; aptal, aşağılık ve basit.
- ☞ "Küresel bilgi izleyici (Global information Tracker)".
- ☞ "Kahrolası aptal kamyon dolusu pislik (Goddamn Idiotic Truckload of sh*t)".

Görüldüğü üzere aslında çok da kesin bir anlamı yok. Özgür yazılım dünyası sizi burada da özgür bırakmış.

3. GIT'in Tarihsel Gelişimi

Linux'un mimarı Linus Torvalds, çok sayıda kişi ile birlikte Linux çekirdeğini geliştirirken projenin yönetimi için o dönem piyasada bulunan BitKeeper adındaki versiyon kontrol sistemini tercih etmiş. Fakat BitKeeper'in telif haklarını elinde bulunduran kişi ile yaşanan yasal sorunlardan ötürü bu kullanımdan vazgeçilmiş. O günlerde piyasada bulunan versiyon kontrol sistemlerinin hiçbiri aslında Torvalds'ın beklentilerini karşılamıyormuş. Bu sebeple ihtiyaçlarına çözüm için kolları sıvayan Torvalds, piyasadaki sistemleri inceleyerek kendi versiyon kontrol sistemini yazmaya başlamış. İlk sürüm 2005 yılında piyasaya sürülmüş. Yayınlanmasından günümüze kadar geçen sürede ise GIT büyük bir pazar hacmine ulaştı.

4. GIT Versiyon Kontrol Sistemi Nedir?

“GIT nedir?” diye sorduğunuzda “Versiyon Kontrol Sistemi” cevabını almışsınızdır. İyi de versiyon kontrol sistemi nedir? Projemi geliştirirken "proje", "proje", "proje_son", "proje_son_5" şeklinde klasörlere ayırsam olmaz mı? GIT öğrenmeden olmaz mı? - OLMAZ!

Son sorudan başlayayım. Eğer profesyonel işler yapacaksanız GIT öğrenmek zorundasınız. Klasör isminin sonuna fazladan harfler, rakamlar ekleyerek proje geliştirmek ilerleyen süreçlerde başa çıkılabilecek bir yöntem değildir.

“Versiyon kontrol sistemi nedir?” sorusuna gelirsek; Bir döküman (yazılım projesi, ofis belgesi vb.) üzerinde yaptığınız değişiklikleri adım adım izleyen, istediğinizde kayıt eden ve isterseniz bunu internet üzerindeki bir bilgisayarda veya yerel bir cihazda (repository / repo / depo) saklamanızı ve yönetmenizi sağlayan bir sistemdir. [2]

“Versiyon Kontrol Sistemi” yerine “Kaynak Kod Yönetim Sistemi (SCM)” ifadesini de duymuş olabilirsiniz. İkisi de aynı şeyi ifade etmektedir.

5. GIT Bize Ne Sağlar?

Birden fazla yerde (dağıtık olarak) dosyalarınızı ve versiyon kontrol bilgilerinizi depolayabilirsiniz. Böylelikle cihaz bağımsız olarak dosyalarınıza erişebilirsiniz.

“commit” yaparak **SnapShot (anlık görüntü)** özelliği ile istediğiniz zaman proje veya dosyaların o anki halini kayıt altına alabilirsiniz. Böylelikle ileride bir hata ile karşılaşırsanız herhangi bir zamandaki herhangi bir versiyona dönüş yapabilirsiniz.

SnapShot alındıktan sonra değişiklik yapılan dosyaları görebilirsiniz.

Takımların aynı projede beraber çalışmasına imkan verir. “Kim neyi düzenledi, ne ekledi, ne çıkarttı, son değişiklik ne zaman yapıldı?” gibi bilgilere erişebilirsiniz. Bu sayede topluluk ile proje geliştirme süreçlerini kolaylaştırabilirsiniz.

Projede versiyonlanmasını istemediğiniz dosyaları belirtebilirsiniz. (node_modules, .mp4, .log, .env gibi)

6. GitHub & GitLab & BitBucket Nedir?

En sade şekilde GIT versiyon kontrol sistemini kullanan depolama servisleri diyebiliriz.

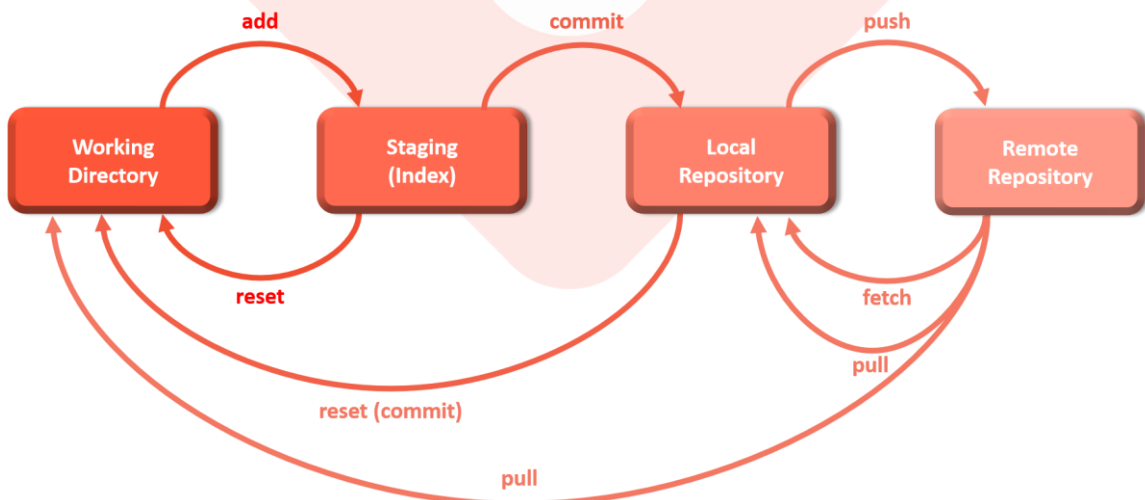
- ☞ **GitHub:** Yazılımcılar için bir kod kütüphanesi ve bir çeşit sosyal medya ortamıdır. Yazılım geliştiriciler projelerini halka açık veya özel olarak saklayabilir. Ücretli ve ücretsiz paket seçenekleri mevcuttur.

- ☞ **GitLab:** GitHub gibi bir GIT servsidir. “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 geçime ve wiki desteğiyle çalışıyor. Başlangıçta GitLab’ın büyük bir çoğunluğu Ruby programlama dili ile yazılmıştır. Farklı olarak firmalara GitLab’ı kendi sunucularına kurma imkanı verildiği için genelde kurumsal tarafta kullanılır. GitLab ile firmalar kendi içerisinde GIT hizmetlerinden faydalanabilir. [3]
- ☞ **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 servsidir. [4] Genelde kişisel kullanıma yöneliktir. GitHub tarafındaki açık kaynak projeler ve sosyal medya ortamı burada gelişmemiştir. [5]

Yukarıda açıkladığımız servisler haricinde GitKraken, SourceTree gibi irili ufaklı farklı servislerde mevcuttur.

7. Bilinmesi Gereken Bazı Terimler

- ☞ **untracked (izlenmeyen):** GIT tarafından henüz takip edilmeyen, yani yeni oluşturulmuş dosyaları ifade eder.
- ☞ **unstaged (hazırlanmamış):** Güncellenmiş ancak commit’lenmek için hazırlanmamış dosyaları ifade eder.
- ☞ **staged (hazırlanmış):** Commit’lenmeye hazır olan dosyaları ifade eder.
- ☞ **deleted (silinmiş):** Projeden silinmiş ama GIT üzerinden kaldırılmamış dosyaları ifade eder.



8. GIT Komutları

- ☞ **git init:** İlk defa projeyi oluştururken kullanılan *“initialization (başlatma)”* komutudur. Henüz versiyon kontrolü altında olmayan bir projenin dizininde, boş bir git deposu oluşturmak için kullanılır.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git init
Reinitialized existing Git repository in C:/Users/Nisa Ceren ÜNNÜ/Desktop/git_example/.git/
```

- ☞ **git config:** GIT'in bir çok konfigürasyon ve ayarı vardır. Bunlardan ikisi, *user.name* ve *user.email*'dir. Bu ayarları yapılandırmak için aşağıdaki komutları kullanırsınız. GIT'i ilk kurduğumuzda genellikle aldığımız ilk hata bu konfigürasyon ayarlarını yapmadığımız için gelir. Burada yazdığınız isim ve email ileride GitHub benzeri bir platforma commit attığınızda görüneceği için bunu bilerek isimlendirme yapmak yararlı olur. Ayrıca görüldüğü gibi bu ayarlar *--global* yani sistem genelinde geçerli ayarlardır. Proje bazlı bu ayarları değiştirebiliriz.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git config --global user.name "Name and Surname"

Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git config --global user.email "example@gmail.com"
```

Konfigürasyon ayarlarının bütünü için aşağıdaki komutu yazınız:

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git config --list
```

Windows işletim sistemi kullanıyorsanız *“warning: LF will be replaced by CRLF in kaynak/dosya/yolu”* hatasını alabilirsiniz. Hatanın çözülebilmesi için aşağıda yer alan komutu kullanabilirsiniz.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git config core.autocrlf true
```

- ☞ **git add:** Herhangi bir dosyayı/klasörü takibin yapılması için depoya (repo) ekler. Yeni eklenen veya üzerinde değişiklik yapılan dosyaları *staged* ortamına göndermek için kullanılır. Terminale *“git add <klasör veya dosya ismi>”* yazılır.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git add example.txt
```

Tek seferde tüm dosyaları eklemek isterseniz aşağıdaki komutlardan birisini kullanmanız yeterli olacaktır.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git add -A

Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git add .

Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git add *
```

(-A ingilizcede all yani hepsi kelimesinden gelmektedir.)

- ☞ **git commit:** SnapShot almak (onay almak) için kullanılan yapıdır. *Commit*, staged ortamına alınan dosyaların Local Repository'ye gönderilmesidir. En iyi uygulama yöntemi her kayıt sırasında yapılan değişiklikleri açıklayıcı bir mesaj eklemektir. Ayrıca her commit benzersiz bir kimliğe (unique ID) sahip olur. Bu sayede eski bir commit'e geri dönebilirsiniz ve herhangi bir kayıp yaşama ihtimaliniz kalmaz.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git commit -m "Gerekli dosyalar eklendi"
[master eff81f6] Gerekli dosyalar eklendi
 2 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 example1
 delete mode 100644 example2
```

Buradaki -m (message) mesaj anlamındadır.

- ☞ **git status:** Proje içerisinde yapılan değişiklikleri görmemizi sağlar. Üzerinde çalışılan projenin o anki durumu hakkında bilgi verir. Yapılan değişiklikler, eklenen ve silinen dosyalar gibi bilgiler listelenir.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    example1
    deleted:    example2

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

On branch master	:	Master branch'inde (dalında) olduğumuzu gösterir.
Changes to be committed	:	Commit'lenmeye hazır değişiklikler olduğunu belirtir.
Modified: <dosya ismi>	:	<dosya ismi> dosyasında değişiklik yaptığımızı gösterir.
Deleted: <dosya ismi>	:	<dosya ismi> dosyasını sildiğimizi gösterir.
Changes not staged for commit	:	Üzerinde değişiklik yapılan ama staged ortamına gönderilmemiş dosyaları ifade eder.
Untracked files	:	Takibi yapılmayan dosyaları ifade eder.

- ☞ **git log:** Projedeki commit geçmişini görüntülememizi sağlar. Bütün commit'ler, id'si, yazarı, tarihi ve mesajı ile beraber listelenir.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git log
commit eff81f6ea67982b2d4a520dbf1b573c50043c34a (HEAD -> master)
Author: Name and Surname <example@gmail.com>
Date: Sun Dec 18 18:25:32 2022 +0300

Gerekli dosyalar eklendi
```

- ☞ **git branch:** Local veya remote repository üzerinde yeni bir branch (dal) eklemek, silmek veya listelemek için kullanılır.

Yeni bir branch eklemek için **"git branch <branch ismi>"** ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git branch origin
```

Tüm branch'ları (uçak ve yerel hepsi) listelemek için **"git branch -a"** ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git branch -a
* master
origin
```

Listelenen branch'lardan yeşil ile yazılan şu an bulunulan branch'ı gösterir.

Branch silmek için **"git branch -d <branch ismi>"** ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git branch -d origin
Deleted branch origin (was eff81f6).
```

- ☞ **git checkout:** Bu komut ile birlikte branch'lerinize (dallarınıza) geçiş yapabilirsiniz. Projein içinde birden fazla kişi veya birden fazla yapı çalışıyorsa bunlar arasında geçiş yapabilirsiniz. Branch'ler arası veya commit'ler arası geçiş yapmak istediğimizde kullanılır.

Mevcutta var olan branch'a geçiş yapmak için **"git checkout <branch_name>"** ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git checkout origin
Switched to branch 'origin'

Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ |
```

Yeni bir branch oluşturup, bu branch'a geçiş yapmak için **"git checkout -b <branch_name>"** ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git checkout -b origin
Switched to a new branch 'origin'

Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ |
```


Commitler arası geçiş yapmak için *"git checkout <commit_ID>"* (eski bir versiyona dönmek istediğimiz zaman) ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git checkout eff81f6ea67982b2d4a520dbf1b573c50043c34a
```

- ☞ **git merge:** Başka bir branch'da olan değişiklikleri, bulunduğumuz branch ile birleştirmek istediğimizde kullanılır *"git merge <branch ismi>"* ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git merge master
Already up to date.
```

- ☞ **git push:** Uzak bilgisayara/sunucuya, projeyi (klasörü/dosyayı) gönderir. Projemizde aldığımız commit'leri, remote repository'e gönderir.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git push origin master
```

Burada bahsi geçen **origin**, remote repository'nin kök dizinini belirtir ve sabit bir isimdir. **master** ise sizin çalıştığınız branch (dal) 'i belirtir. Henüz remote repository'niz yoksa aşağıdaki komut ile local deponuzu uzak sunucudaki depoya bağlayabilirsiniz.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git remote add origin http://uzak_deponun_adresi.git
```

- ☞ **git pull:** Uzak bilgisayardan/sunucudan, projeyi (klasörü/dosyayı) çeker (geri getirir).

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git pull <remote_URL>
```

- ☞ **git clone:** Projenin kopyasını almamızı sağlar. Mevcut bir Remote Repository'de bulunan dosyaların bilgisayarımızda bir kopyasının oluşturulmasını sağlar.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git clone <remote_URL>
```

- ☞ **git rm:** Dosyaları/Klasörleri silmek için kullanılır. Staged ortamına eklenmiş bir dosyanın takibinin bırakılması yani **untracked (izlenmeyen)** hale getirilmesi sağlayan komuttur.

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git rm --cached simple.c
```

"git rm <dosya/klasör ismi>" yazarsanız git deposundan o dosyayı silmiş olursunuz ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ git rm example2
rm 'example2'
```

- ☞ **git reset:** Bu komut, dosyanı aşamasını kaldırır. Ancak dosya içeriğini korur. Komutun kullanımı, *"git reset <dosya_adı>"* ;

```
$ git reset git_example/
```

- ☞ **git tag:** Bu komut, sürüm gibi önemli bir değişiklik grubunu işaret eder. Komutun kullanımı, *"git tag <commit_ID>"* ;

```
$ git tag 858f19ecbe8cac4ae62b30cbd205c658b1437d97
```

- ☞ **git diff:** Repository üzerinde yapılan değişikliklerden sonra dosyalar arasında oluşan farklılıkları gösterir.

Çalışma dizini ile repository (HEAD) arasındaki farklılıkları görmek için *"git diff HEAD"* ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git diff HEAD
```

İki commit arasındaki farklılıkları görmek için *"git diff <commit_id_1>..<commit_id_2>"* ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git diff eff81f6ea67982b2d4a520dbf1b573c50043c34a a5d1d6aa00ee53d3653f5d03f6b99b8949753932
diff --git a/example1 b/example1
new file mode 100644
index 0000000..e69de29
diff --git a/example2 b/example2
new file mode 100644
index 0000000..e69de29
```

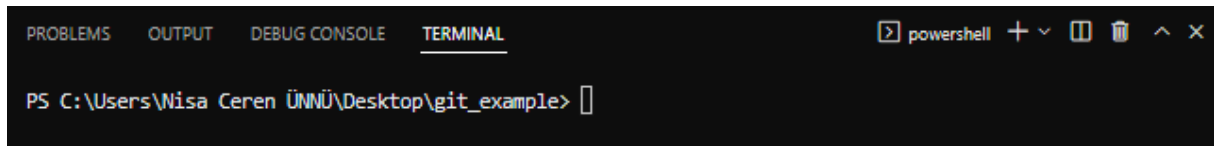
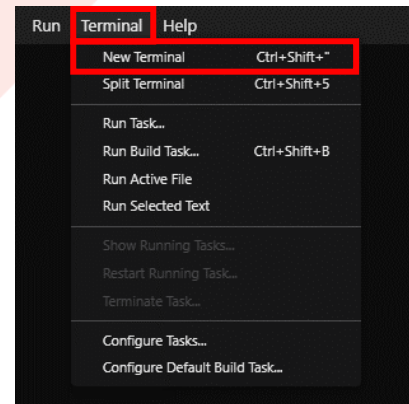
Çalışma dizini ve staged ortamı arasındaki farkları görmek için *"git diff --staged"* ;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (origin)
$ git diff --staged
```

[3] [4] [5]

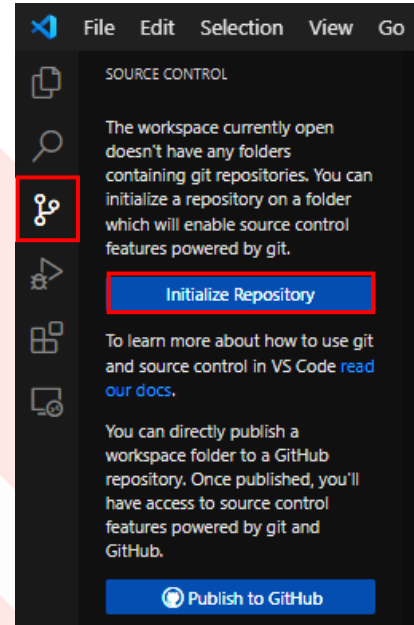
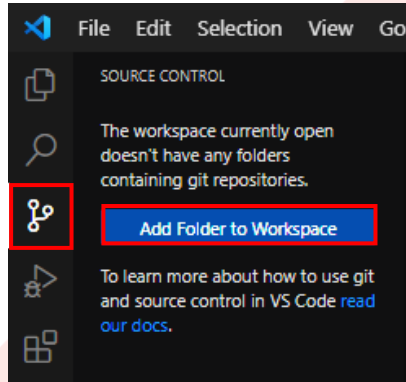
9. VS Code İçerisinde Terminal Kullanarak GIT Temel Komutları

GIT temel komutlarını Visual Studio (VS) Code içerisindeki terminalde kullanmak için menü barından Terminal açarak **New Terminal (Ctrl+Shift+`)** seçeneğini seçmelisiniz. Burada açılan terminal proje klasörünüzün içerisinde açılacak ve editörden ayrılmadan GIT komutlarını kullanabileceksiniz.

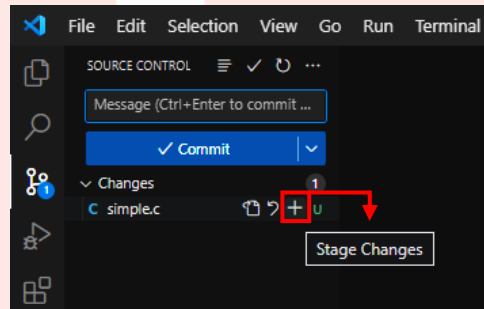


10. VS Code İçerisinde Terminal Kullanmadan GIT Temel Komutları

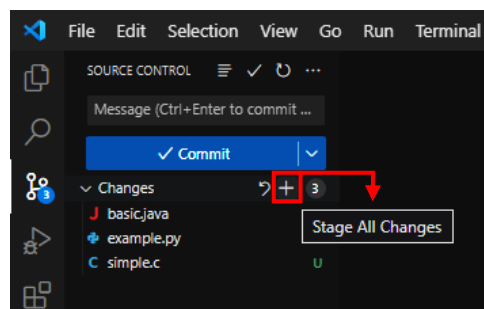
- Henüz versiyon kontrolü altında olmayan bir projenin dizininde, boş bir git deposu oluşturmak için **Activity Bar** bölümünden **Source Control (Ctrl + Shift + G)** ikonuna tıklayıp, **Add Folder to Workspace** yazıyorsa o butona tıklamalıyız. Ardından **Initialize Repository** butonuna tıklayıp git deposunu oluşturabiliriz.



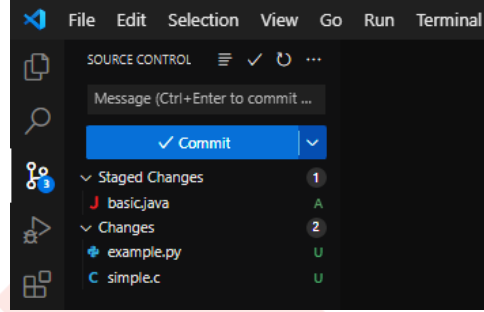
- Yeni eklenen veya üzerinde değişiklik yapılan dosyaları staged ortamına göndermek için **Stage Changes** butonuna tıklamalıyız.



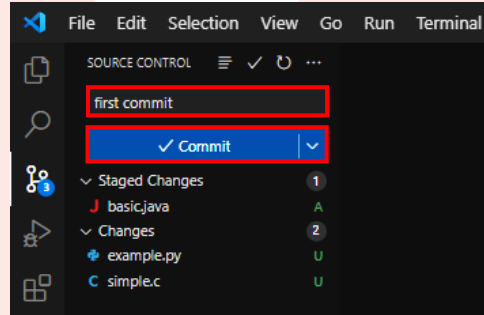
- Birden fazla dosyamız olduğu zamanlarda eğer bütün değişiklikleri tek bir seferde staged ortamına göndermek istiyorsak **Stage All Changes** butonuna tıklamalıyız.



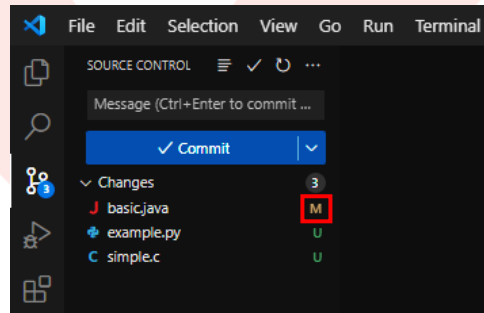
- ☞ Staged ortamına dosyayı eklediğimizde dosyanın yanında “A (added)” yazacaktır. Staged ortamına eklemediğimiz dosyalar olursa bu dosyaların yanında da “U (untracked)” yazacaktır.



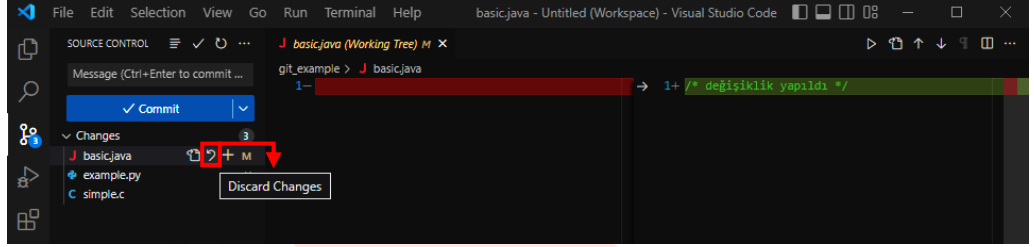
- ☞ Commit, **staged** ortamına alınan dosyaların *Local Repository*'e gönderilmesidir. En iyi uygulama yöntemi her kayıt sırasında yapılan değişiklikleri açıklayıcı bir mesaj eklemektir. Ayrıca her commit benzersiz bir kimliğe (**unique ID**) sahip olur. Dosyalarımızı commit'lemek için **message** bölümüne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



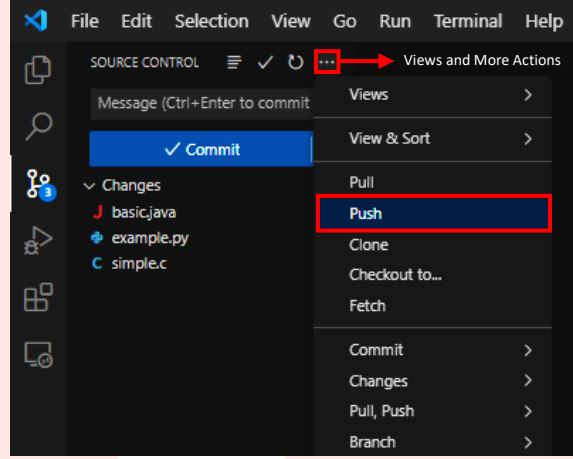
- ☞ Commit'lenen dosya üzerinde değişiklik yaptığımızda, dosyanın yanında “M (modified)” yazacaktır.



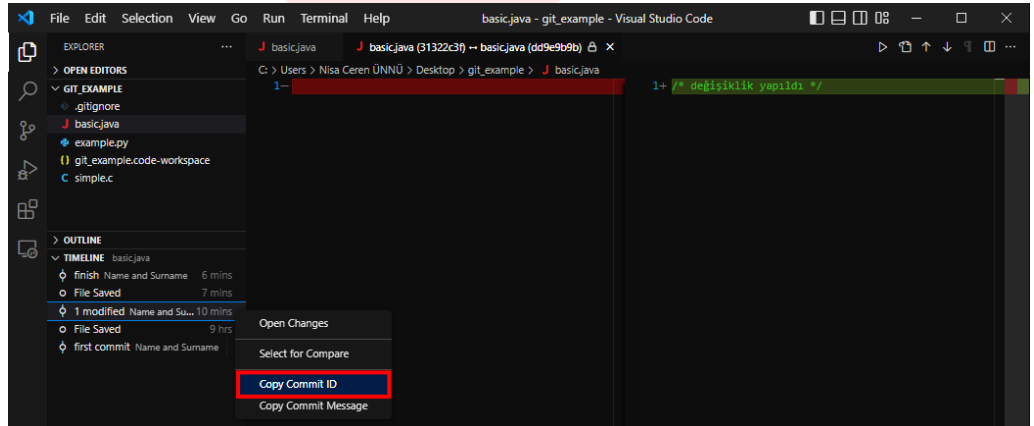
- ☞ Dosyamızda yapılan değişikliği görüntülemek için, **Source Control** bölümünde, dosyanın üzerine tıkladığımızda, iki farklı bölüm karşımıza geliyor. En sağdaki bölümde dosyamızın üzerinde yaptığımız değişiklikleri görüntüleyebiliriz. Bu değişiklikleri eğer geri almak istersek, tekrar sol bölümdeki gibi olmasını istiyorsak **Discard Changes** butonuna tıklamalıyız.



- ☞ Eğer remote repository'e bağlıysak ve commit'lerimizi remote repository'e göndermek istersek **Views and More Actions** butonuna tıklayıp, **Push** seçeneğini seçmeliyiz.



- ☞ Timeline'a gelerek yapılan tüm commit işlemlerini görebilirsiniz ve dosyada istediğiniz eski bir versiyona geri dönebilirsiniz. Mesela dosyada eski bir versiyona dönmek istiyorsak aşağıdaki fotoğrafta "1 modified" mesajıyla commit'lenmiş dosyanın üzerine tıklayıp açtığımızı ve sağa tıklıyoruz. "**Copy Commit ID**" deyip terminale "**git checkout <commit_ID>**" yazarak eski versiyona dönmüş oluyoruz.



11. .gitignore Dosyası Ne İşe Yarar? Nasıl Kullanılır?

.gitignore dosyası projemizin kök dizinine oluşturulan düz bir metin dosyasıdır. Adından anlaşıldığı gibi diyor ki, “beni göz ardı et”. Daha doğrusu “Göz ardı etmek istediğin, local çalışma alanındaki takip edilmesini istemediğin, takım arkadaşların için gerekmeyen dosyaların varsa veya bu dosyaların boyutu reponuza atmanıza gerek olmayacak kadar büyük ölçekli ise buyur beni kullan” diyor.

Gel bu dosyaları .gitignore dosyasına koy ki GIT’de senin bu dosyalarını artık takip etmesin. Üstelik bu işlemler yapılırken senin halihazırdaki dosyalarını da hiçbir şekilde etkilemesin.

☞ .gitignore’a eklenen dosyalar nedir?

- ★ Paket yöneticisinden indirilen bağımlılıklar,
- ★ image ve video dosyalarınız (dosya boyutları çok fazla olabilir),
- ★ IDE eklentileri (örneğin .vscode),
- ★ Sadece kendi çalışma alanınızda olması gereken başkaları tarafından görülmemesi gereken dosyalarınız (veritabanınıza ilişkin konfigürasyonlar),
- ★ API anahtarları, kimlik bilgileri veya hassas bilgiler içeren dosyalar (.env),
- ★ Çalışma dizinizdeki geçici dosyalar,
- ★ Log dosyaları,
- ★ Yararsız sistem dosyaları (örneğin MacOS işletim sisteminin .DS_Store dosyası),
- ★ dist gibi oluşturulan dosyalar,
- ★ herhangi istediğiniz bir dosyanız da olabilir.

☞ .gitignore dosyası nasıl oluşturulur?

- ★ Reponuzu oluştururken verilen seçeneklerde *add gitignore file* dosyasına tıklayarak reponuzla beraber oluşturabilirsiniz. Aynı şekilde editörünüzde .gitignore şeklinde de oluşturabilirsiniz.

- ★ MacOS / Unix terminalinde .gitignore dosyası oluşturmak için;

```
$ touch .gitignore
```

- ★ Windows terminalinde .gitignore dosyası oluşturmak için;

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ echo > .gitignore

Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ echo some-text or nothing > .gitignore
```

Terminale *echo > .gitignore* yazdığınızda sadece .gitignore dosyasını oluşturmuş olursunuz. *echo some-text or nothing > .gitignore* yazdığınızda ise .gitignore dosyasının içerisine *some-text or nothing* yazmış olursunuz.

- ★ Terminalde .gitignore dosyasını oluşturduktan sonra içerisine bir şey yazmak/düzeltilmek için (MacOS ve Windows için geçerlidir);

```
Nisa ÜNNÜ@DESKTOP-3M21MVB MINGW64 ~/desktop/git_example (master)
$ vim .gitignore
```

- ★ .gitignore dosyasının içerisine yazdığımız dosya uzantıları ve dosya isimleri ile yazılımın hangi dosyaları takip etmesi gerektiğini gösteririz.
- ★ Programlama dillerinin kendilerine göre farklı .gitignore dosyaları vardır. Kullanacağınız programlama diline göre internetten bakmanız gerekir.
- ★ Kullandığınız program dilindeki gereksiz/önemsiz/durması anlamsız dosyaları bilmiyorsanız google'a "*gitignore (programlama dilinin ismi)*" şeklinde arama yapıp öğrenebilirsiniz (Örnek: gitignore Python).

☾ .gitignore nasıl çalışır, nasıl kullanılmalı?

- ★ Direkt dosyanın ismini uzantısı ile birlikte yazabilirsiniz.
- ★ Tüm klasörlerin yalnızca dosya yolunu .gitignore'a yazarak programın görmezden gelmesini sağlayabilirsiniz. (node_modules/ veya logs/ vs.)
- ★ "/" sembolünü kullanarak yazılan dosyaya erişilebildiği gibi "**.(dosya uzantısı)*" kullanılarak devamında yazılan tüm dosya uzantısındaki olan dosyaları .gitignore'a eklersiniz.
- ★ "!" ön ekini kullanarak .gitignore'a dahil etme diyebilirsiniz. Örnek: .gitignore'a "**.log*" yazarak tüm .log uzantılı dosyaları görmezden gel demiş olursunuz ama bir tane dosyayı görmezden gelmesini istemiyorsunuz. Burada yapmanız gereken "*!(dosya ismi).log*" demek olacaktır.
- ★ Dikkat edilmesi gereken nokta: eğer ki .gitignore içerisine "*log/*" (log dosyasının içerisindekileri görmezden gel) yazmışsanız "*!logs/example.log*" deseniz bile example.log yok sayılacaktır. O yüzden dosyalarınız hangi klasörde olduğuna ve .gitignore dosyasının içerisine yazdıklarınıza dikkat etmelisiniz.
- ★ .gitignore dosyasında yorum satırı oluşturmak için "*#*" sembolü kullanılır.
- ★ Daha detaylı bilgi isterseniz [1](#) ve [2](#) numaraların üzerine tıklayınız. [6]

☞ Dikkat edilmesi gereken yerler

- ★ Eğer projenizi “*git add .*” veya “*git commit*” etmişseniz sonrasında *.gitignore* dosyasına eklemek istediğiniz dosyayı ekleseniz de bu işlem gerçekleşmeyecektir ve o dosyanız reponuzda hala GIT ile takip edilecektir. Tabi her şeyin bir çözümü olduğu gibi bu sorunu da çözmenin bir yolu var.

- ★ MacOS için çözüm;

```
$ git rm --cached FILENAME
```

- ★ Windows için çözüm; *C:\Users\{myusername}* adresine giderek *.gitignore_global* dosyası oluşturup içerisine global olmasını istediğiniz dosyaları ekledikten sonra *git bash* terminalinizi açarak aşağıdaki komut ile konfigürasyon sağlayabilirsiniz.

```
$ git config --global core.excludesfile "%USERPROFILE%\gitignore"
```

Dosyanızın doğru çalıştığını kontrol etmek için ise aşağıdaki komutu çalıştırarak aşağıdaki çıktıyı aldığınızda sorunsuz çalıştırabilmiş olmanız gereklidir (Aşağıdaki kodu kopyala yapıştır yapmadan önce kullanıcı adını değiştirin).

```
$ git config --global core.excludesfile  
> C:/Users/user-name/.gitignore_global
```

- ★ Hangi *.gitignore* dosyalarını eklemeliyim dersiniz [buradan](#) hangi dil, framework vs. kullanıyorsanız ona ait *.gitignore* dosyalarını bulabilirsiniz. Global olarak düzenlemek istediğiniz *.gitignore* dosyalarına da [buradan](#) tıklayarak erişebilirsiniz.

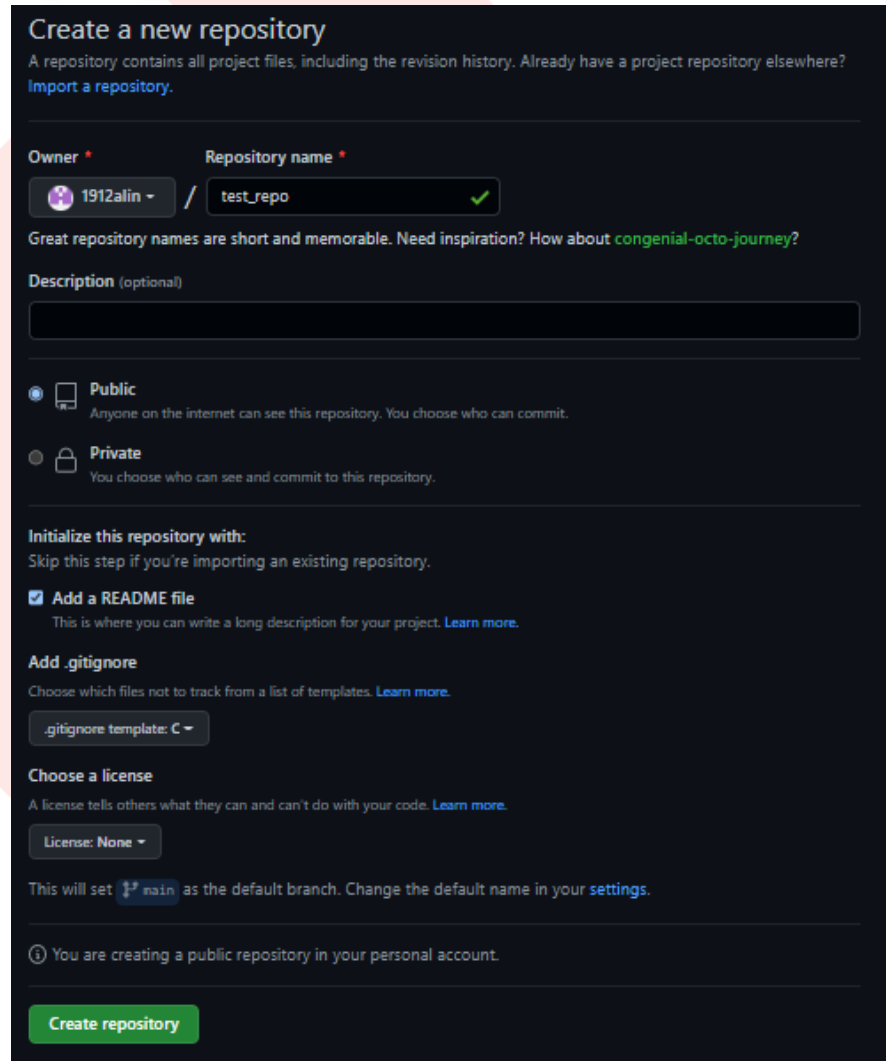
12. Projenin GitHub’a Eklenmesi

- ☞ GitHub’da repo oluşturabilmek için hesap oluşturma aşamalarından sonra sağ tarafta profil resmimize tıklıyoruz. “*Your repositories*” dosyasına tıkladığımızda bulunan repo’ları bize gösterecektir. Hiç repo oluşturmadıysanız boş görünecektir. Sağ tarafta yer alan “*New*” butonuna tıklıyoruz.

- ★ **Owner:** Reponun sahibinin seçiyoruz.
- ★ **Repository name:** Oluşturduğumuz reponun adını belirliyoruz.
- ★ **Description:** Repomuz için bir açıklama girebiliriz.
- ★ **Public:** Repomuzun tüm herkesin erişimini sağlar.
- ★ **Private:** Bu seçenek sayesinde repomuzu gizli bir şekilde oluşturabiliriz.

- ★ **Add a README file:** Repomuzda öncesinde README dosyası oluşturmuş oluruz, dilersek daha sonrasında kendimiz ekleyebilir ve güncelleyebiliriz.
- ★ **Add .gitignore:** Repomuzda öncesinde .gitignore dosyası oluşturabilir ve göndermek istemediğimiz dosyaları seçebiliriz, daha sonrasında aynı işlemi yapabilir ve .gitignore dosyamızı güncelleyebiliriz.
- ★ **Choose a license:** Reponuz için bir lisans seçimi yapabilmenizi sağlar. Lisans için ilgili sayfaya [buradan](#) ulaşabilirsiniz.

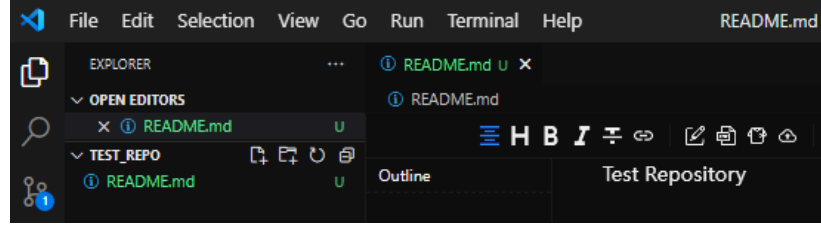
Gerekli işlemleri tamamladıktan sonra **"Create repository"** diyerek repomuzu oluşturmuş oluruz.



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

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> git init
Initialized empty Git repository in C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo/.git/
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> |
```

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



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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> git add .\README.md
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo>
```

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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> git commit -m "readme added"
[master (root-commit) 52af941] readme added
1 file changed, 1 insertion(+)
create mode 100644 README.md
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo>
```

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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> git branch -M main
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo>
```

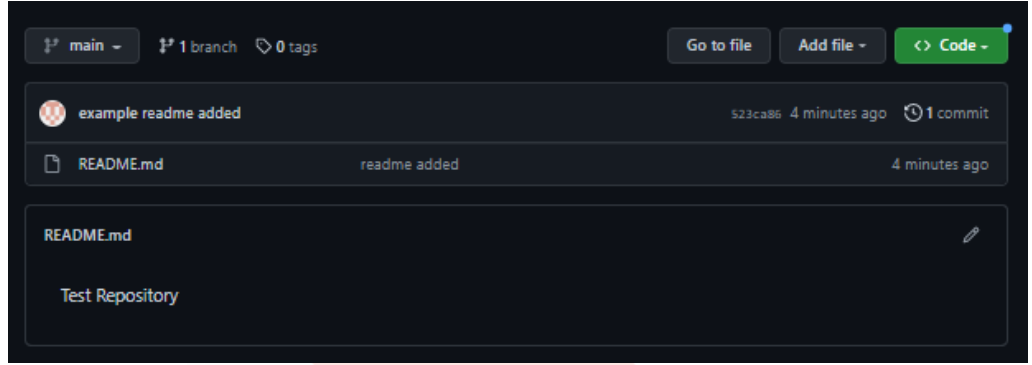
- ☞ Son aşamaya gelmeden önce ise **"git remote add origin 'repo-link'"** komutu ile remote bağlantımızı ekleyelim (origin isimli remote'umuzu ekliyoruz). Önceden uzak repo oluşturduysanız ve başka bir projeye repo oluşturmanız gerekiyorsa **"git remote add (dosya adı) 'repo_link'"** yazın.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> git remote add origin https://github.com/1912alin/test_repo.git
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo>
```

- ☞ Son aşama olarak **"git push -u origin main"** komutu ile repomuza dosyalarımızı gönderelim (main'de yaptıklarımızı origin'e gönder diyoruz). Repo önceden varsa origin yerine dosya adını yazın.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Nisa Ceren ÜNNÜ\Desktop\test_repo> git push -u origin main
```

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



- README.md içerisine alt başlık oluşturmak isterseniz “#” sembolünü kullanabilirsiniz. “#” işaretini 1.başlık, “##” işaretini 2.başlık olarak düşünebilirsiniz.
- README.md içerisine `[link_ismi](link)` yazarsanız GitHub ekranına geldiğinizde artık link_ismi’ne tıkladığınızda yazdığınız link sayesinde sizi siteye yönlendirecektir.

13. GIT Sitesi ve Kitabı

- Git sitesi: <https://git-scm.com/>
- Git kitap linki: <https://git-scm.com/book/tr/v2>

14. Markdown Nedir? Nasıl Kullanılır?

Markdown, John Gruber ve Aaron Swartz tarafından geliştirilen ve 2004 yılından bu yana kullanılan metinden HTML'ye (text-to-HTML) dönüşüm için kullanılan hafif bir işaretleme dilidir.

GitHub gibi platformları kullananların aşına olduğu Markdown formatı, yaygın kanının aksine sadece README dosyaları oluşturmak kullanılmaz. Temel amaç okunabilirliği ve kullanılabilirliği arttırmaktır. Basitliği ve sadeliği sayesinde forumlarda ileti yazmaktan, kitap yazmaya kadar pek çok yerde kullanılabilir. Asıl güçlü olduğu kısım klavyeden elinizi kaldırmadan tablolardan, matematiksel ifadelerle kadar ihtiyaç duyduğunuz her şeyi oluşturabilmeniz ve sonrasında biçimlendirebilmenizdir. [10]

☞ Başlıklar

HTML'de `<h1>`, `<h2>`, `<h3>` etiketleri ile aç-kapat yaparak oluşturduğumuz başlıkları, Markdown ile sadece `#` karakteri kullanarak oluşturabiliyoruz. Burada önemli olan nokta `#` karakterinden sonra boşluk bırakmaktır.

NOT: `h1` ve `h2` başlıklarda GitHub'ın yaptığı özelleştirme sebebiyle otomatik olarak gri bir çizgi geliyor.

☞ Paragraf

Paragraf oluşturmak için haricen bir işlem yapmak gerekmiyor. Markdown formatında yazıyorsanız yeni bir satır oluşturmak paragraf için yeterli. Bir paragraf tek satırdan oluşabileceği gibi, arada boşluk bırakmadan alt satırdan devam etmek de mümkün.

☞ Kalın, Eğik ve Üstü Çizili İfadeler

Yaygın kullanımda **kalın** yazmak için `**`, *eğik* yazmak için `*`, **hem kalın hem eğik** yazmak için `***` kullanılmaktadır.

☞ Tek ve Çok Satırlı Kod Blokları

Tek satır kod bloğu için kodun başına ve sonuna ``` (backtick) karakteri eklenir.

Çok satır kod bloğu için kodun başına ve sonuna **3 adet** ````` backtick karakteri eklenir (`''' (kod bloğu) '''`).

Yazılım diline göre kod bloğundaki ifadelerin stillendirilmesini isterseniz, kod bloğunun başındaki **3 adet backtick** ifadesinden sonra `javascript`, `python`, `css` gibi etiket ekleyebilirsiniz.

☞ Yatay Çizgi

İçerikte bölümlleme yapmak için `---` kullanabilirsiniz. HTML'deki karşılığı `<hr>` olan bu ifade;

☞ Listeler

HTML'de `` `` ve `` `` etiketleri ile oluşturulan listeler Markdown formatında `-` ve `*` ile oluşturulur. Sıralı liste elde etmek için tek yapmanız gereken liste elemanlarının başına **sıra numarası** ve **.** (nokta) eklemek.

Buradaki önemli nokta şu: Siz farklı sıra numaraları vermek isterseniz de Markdown sıra numaralarını otomatik olarak biçimlendirmektedir.

Markdown ile iç içe listeler yapmak da oldukça kolay. Alt listelere tab ile girinti verdiğinizde otomatik olarak nested list yapısına dönüşmekte.

☞ Tablolar

Tablo oluşturmak için aşağıdaki yapı kullanılır. Satır çizgisi için kullanılan - karakterine, : işareti eklenerek tabloda sola, sağa veya ortaya hizalama yapılabilir.

☞ Bağlantı Ekleme

Köşeli parantez bağlantı açıklamasını, küme parantezi ise linki barındırır. HTML'den aşına olduğumuz <a> etiketi yerine Markdown'da []() karakterleri ile yapılır.

☞ Resim Ekleme

Bağlantı resimleri de aynı şekilde eklenir. Sadece köşeli parantezden önce bir tane ! ünlem işareti eklenmelidir. Köşeli parantezin için doldurmak zorunlu değildir. Boş da kalabilir ;

☞ Alıntı

Yazınız içinde alıntı kullanmak isterseniz yapmanız gereken, metnin başına > karakteri koymaktır.

Normal hayatınızda da markdown'u kullanacaksanız yazdığınız markdown kodunu text/word gibi yazıya çeviren (markdown editörü) siteye [buradan](#) ulaşabilirsiniz.

VS Code için markdown extension linkine [buradan](#) ulaşabilirsiniz. Dosyayı .md uzantılı olarak kaydettikten sonra **Ctrl + Shift + V** 'ye bastığınızda yazdığınız kod artık doküman olarak karşınızda olacak.

Örnek bir Markdown yazılımı EK - 1'de yer almaktadır.

Örnek bir Markdown çıktısı EK - 2'de yer almaktadır.

En sonda yer alan **"GIT Cheat Sheet"** sayfalarına bakarak git komutlarına genel olarak bakabilirsiniz.

Kaynakça

- [1] Cambridge Advanced Learner's Dictionary & Thesaurus, «Git,» Cambridge University, [Çevrimiçi]. Available: <https://dictionary.cambridge.org/dictionary/english/git>.
- [2] F. Alaybeg, «Versiyon Kontrol Sistemi Nedir?,» 17 06 2019. [Çevrimiçi]. Available: <https://furkanalaybeg.medium.com/versiyon-kontrol-sistemi-nedir-2f47bb830064>.
- [3] «GitLab,» GitLab, [Çevrimiçi]. Available: <https://about.gitlab.com/>.
- [4] S. Gürgen, «BitBucket Nedir?,» Medium, 9 Mart 2019. [Çevrimiçi]. Available: <https://medium.com/pluginie/bitbucket-nedir-2cc5820b51a6>.
- [5] «BitBucket,» [Çevrimiçi]. Available: <https://bitbucket.org/>.
- [6] N. Uludaş, «GİT Bash ile Komut Versiyonlama,» Medium, 27 Mayıs 2017. [Çevrimiçi]. Available: <https://medium.com/fedeveloper/git-bash-ile-komut-komut-versiyonlama-a354efd3063f>.
- [7] S. Maple, «Git Sheet,» JRebel, 25 Şubat 2016. [Çevrimiçi]. Available: <https://www.jrebel.com/blog/git-cheat-sheet>.
- [8] B. Kerr, «Common Git Commands,» Beanstalk Guides, [Çevrimiçi]. Available: <http://guides.beanstalkapp.com/version-control/common-git-commands.html>.
- [9] A. Garrett-Harris, «How to Use a .gitignore File,» Plural Sight, 23 Ağustos 2019. [Çevrimiçi]. Available: <https://www.pluralsight.com/guides/how-to-use-gitignore-file>.
- [10] «CommanMark,» CommanMark, [Çevrimiçi]. Available: <https://commonmark.org/>.

EK – 1

```
1
2 # MarkDown'a Genel Bakış
3
4 ## Yazı Tipleri
5
6 **GİT Kılavuzu**
7
8 *GİT Kılavuzu*
9
10 ***GİT Kılavuzu***
11
12 ## Blok Code Yazımı
13
14 ` print("Hello World!") `
15
16 ```python
17 print("GitHub")
18 print("MarkDown Olusturuyoruz!")
19 ```
20
21 ## Listeleme
22
23 * Liste 1
24 * Liste 2
25 * Liste 3
26
27 ---
28
29 - Liste 4
30 - Liste 5
31 - Liste 6
32
33 ---
34
35 1. Liste Elemanı 1
36     1. Alt liste elemanı 1
37     2. Alt liste elemanı 2
38 12. Liste Elemanı 2
39 21. Liste Elemanı 3
40
41 ## Tablo
42
43 | Ad | Soyad | Yaş |
44 | :--- | :---: | ---: |
45 | Nisa | Ünnü | 21 |
46
47 ## Bağlantılı Link
48
49 [google](https://www.google.com.tr/)
50
51 ## Resim
52
53 ![Google Logo](https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSK5q0FP74V9wbfwP378_7kj7iDomHuKr-xkXsxDdUT28V9d1VMNJe-EMzaLwaFhneeuzI&usqp=CAU)
54
55 ## Alıntı
56
57 > "Yıldızları hedef alın, ama olur da kaçırırsanız, onlar yerine ayı hedef alın." - Neil Armstrong
58
```

README.md

Preview README.md

MarkDown'a Genel Bakış

Yazı Tipleri

GIT Kılavuzu

GIT Kılavuzu

GIT Kılavuzu

Blok Code Yazımı

```
print("Hello World!")
```

```
print("GitHub")
print("MarkDown Olusturuyoruz!")
```

Listeleme

- Liste 1
- Liste 2
- Liste 3

- Liste 4
- Liste 5
- Liste 6

- Liste Elemanı 1
 - Alt liste elemanı 1
 - Alt liste elemanı 2
- Liste Elemanı 2
- Liste Elemanı 3


Tablo

Ad	Soyad	Yaş
Nisa	Ünnü	21

Bağlantılı Link

[google](#)

Resim



Alıntı

"Yıldızları hedef alın, ama olur da kaçırsanız, onlar yerine ayı hedef alın." - Neil Armstrong

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIs

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows

<https://windows.github.com>

GitHub for Mac

<https://mac.github.com>

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms

<http://git-scm.com>

SETUP

Configuring user information used across all local repositories

```
git config --global user.name "[firstname lastname]"
```

set a name that is identifiable for credit when review version history

```
git config --global user.email "[valid-email]"
```

set an email address that will be associated with each history marker

```
git config --global color.ui auto
```

set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

```
git init
```

initialize an existing directory as a Git repository

```
git clone [url]
```

retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

```
git status
```

show modified files in working directory, staged for your next commit

```
git add [file]
```

add a file as it looks now to your next commit (stage)

```
git reset [file]
```

unstage a file while retaining the changes in working directory

```
git diff
```

diff of what is changed but not staged

```
git diff --staged
```

diff of what is staged but not yet committed

```
git commit -m "[descriptive message]"
```

commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

```
git branch
```

list your branches. a * will appear next to the currently active branch

```
git branch [branch-name]
```

create a new branch at the current commit

```
git checkout
```

switch to another branch and check it out into your working directory

```
git merge [branch]
```

merge the specified branch's history into the current one

```
git log
```

show all commits in the current branch's history



INSPECT & COMPARE

Examining logs, diffs and object information

git log

show the commit history for the currently active branch

git log branchB..branchA

show the commits on branchA that are not on branchB

git log --follow [file]

show the commits that changed file, even across renames

git diff branchB...branchA

show the diff of what is in branchA that is not in branchB

git show [SHA]

show any object in Git in human-readable format

TRACKING PATH CHANGES

Versioning file removes and path changes

git rm [file]

delete the file from project and stage the removal for commit

git mv [existing-path] [new-path]

change an existing file path and stage the move

git log --stat -M

show all commit logs with indication of any paths that moved

IGNORING PATTERNS

Preventing unintentional staging or committing of files

```
logs/  
*.notes  
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

git config --global core.excludesfile [file]

system wide ignore pattern for all local repositories

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

git remote add [alias] [url]

add a git URL as an alias

git fetch [alias]

fetch down all the branches from that Git remote

git merge [alias]/[branch]

merge a remote branch into your current branch to bring it up to date

git push [alias] [branch]

Transmit local branch commits to the remote repository branch

git pull

fetch and merge any commits from the tracking remote branch

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

git rebase [branch]

apply any commits of current branch ahead of specified one

git reset --hard [commit]

clear staging area, rewrite working tree from specified commit

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

git stash

Save modified and staged changes

git stash list

list stack-order of stashed file changes

git stash pop

write working from top of stash stack

git stash drop

discard the changes from top of stash stack

GitHub Education

Teach and learn better, together. GitHub is free for students and teachers. Discounts available for other educational uses.

✉ education@github.com

🌐 education.github.com