GIT | GITHUB

Git : Bir versiyon kontrol sistemidir.
GitHub: Bir bulut sistemidir.

Git: Versiyon Kontrol Sistemi

→ Versiyon kontrol sistemleri bir yazılım projesinin evrimini ve değişikliklerini izlemek, yönetmek ve organize etmek için kullanılan yazılımlardır.

→ Git, özellikle "dağıtık" bir versiyon kontrol sistemidir. Yani bulut sistemini tek bir ana bilgisayar sisteminde değil de dağılmış haldeki sistemler üzerinde uygulamaktadır.

→ Yani git, birçok farklı bölgede bulunan insanların bir arada iş yapmasına olanak sağlıyor. (dağıtık olmanın avantajı)

→ Versiyon kontrol sistemleri, bir yazılım projesinin tüm safhalarının arasında hızlı geçişler yapılabilmesine olanak sağlar.

→ Versiyon kontrol sistemleri, bir yazılım projesinin tüm safhalarının arasında hızlı geçişler yapılabilmesine olanak sağlar. Ve birden fazla insanın bir proje üzerinde çok pratik bir şekilde collabrate olmasını sağlar.

Neden Git: Versiyon Kontrol Sistemi Kullanılır?

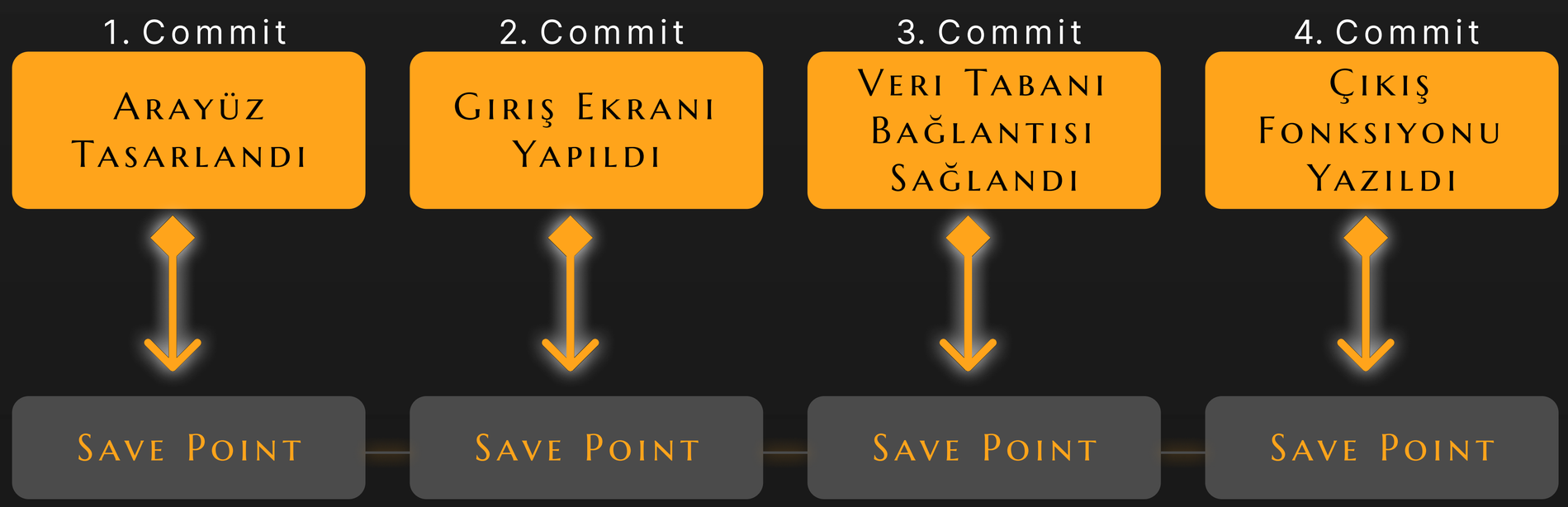
- Eş Zamanlı Çalışma
- Hata İzleme ve Geri Dönme
- İzleme ve Belgeleme
- İşbirliği ve Koordinasyon

Linus Torvalds kurmuştur- Linux PC'nin çekirdeğini yapan Fin'li adam

SVN(Subversion)
Mercurial
Perforce
Darcs
CSV(Concurrent Versions System)
TFS(Team Fundation Server)
Bazaar

} Diğer Versiyon Kontrol Sistemleri

Git: Versiyon Kontrol Sistemi Nasıl Çalışır?



- Yukarıda yazmış olduğumuz bir projenin 4 farklı kayıt noktası vardır. Bunların her birine biz **versiyon** diyoruz. Ve Git ile amacımız bu versiyonların kontrolünü sağlamak. Bunu da **commit** ile yapıyoruz.
- Bir kayıt noktası oluşturup onu bir yere kaydederiz ve istediğimiz zaman eğer gerekirse o kayıt noktasına geri dönebiliriz.

> git --versiyon → bilgisayarda yüklü olan git'in versiyonu öğrenilir

```
C:\Users\monstr>git --version
git version 2.43.0.windows.1
```

Linux ve MacOS Terminal Komutları

→ Başlat arama çubuğundan aratılarak Git Bash terminali açılır. Ya da , çalışma yapmak istediğimiz klasörün içerisinde sağ tıklayıp "Open Git Bash here" diyerek Git Bash terminaline giriş yaparız.

→ İkinci yöntemi uygulamanın avantajı, açılan Git Bash terminalinin doğrudan çalışma yapılan sayfanın içerisinde açılacak olmasıdır.

→ Buradaki komutları windows'un cmd terminalinde uygulamıyoruz. Bu kodlar Linux veya MacOS işletim sistemleri için geçerli.

> ls → klasör içeriğini listeler (gizliler hariç)

```
monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls
gt_github.txt
```

Bu klasörde sadece "gt_github.txt" dosyası varmış.

> ls -a → klasörde içeriğini listeler (gizliler dahil)

> ls -l → klasör içerikleri ile ilgili detaylı bilgi alırız (gizliler hariç)

> ls -la → klasör içerikleri ile ilgili detaylı bilgi alırız (gizliler dahil)

```
monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls -la
total 16
drwxr-xr-x 1 monstr 197121 0 Feb 25 10:55 ./
drwxr-xr-x 1 monstr 197121 0 Feb 25 10:46 ../
-rw-r--r-- 1 monstr 197121 0 Feb 25 10:55 gt_github.txt
```

Bu klasörde 3 adet dosya var, birisini biz oluşturduk, diğer ikisi gizli dosya. Dosyaların oluşturulma tarihleri var. Ve "r", "w" yani read, write gibi özelliklere sahiplik durumu gösterilir.

> pwd → bulunulan klasörü gösterir

> cd .. → bir üst klasöre geç

```
monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ pwd
/c/Users/monstr/Desktop/Git_GitHub

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ cd ..

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop
$ pwd
/c/Users/monstr/Desktop
```

cd .. ile bir üst klasöre geçtik ve pwd ile hangi klasörde olduğumuzu yazdırdık. Artık Desktop içerisindeyiz yani masaüstündeyiz.

> cd hedef_yer → hedef_yer yazan kısma yazdığımız klasöre gideriz

```
monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop
$ cd Git_GitHub

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ pwd
/c/Users/monstr/Desktop/Git_GitHub
```

Not: hedef_yer olarak belirlediğimiz yer eğer bulunduğumuz dosyanın altında değil ise, o zaman uzantıyı komple şekilde yazmalıyız.

ÖR: ~Desktop/Git_GitHub

> mkdir deneme → yeni klasör(ler) oluşturur

> touch bir.ipynb → yeni dosya(lar) oluşturur

```
monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ mkdir denem

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls
denem/  gt_github.txt

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ touch bir.ipynb

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls
bir.ipynb  denem/  gt_github.txt

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ touch deneme.docx sunu.pptx oyun.py

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls
bir.ipynb  denem/  deneme.docx  gt_github.txt  oyun.py  sunu.pptx
```

> rm → dosya siler

> rm -rf → klasör siler (içindekilerle birlikte)

```
monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ rm deneme.docx

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls
bir.ipynb  denem/  gt_github.txt  oyun.py  sunu.pptx

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ rm -rf denem

monstr@DESKTOP-OL3QQTV MINGW64 ~/Desktop/Git_GitHub
$ ls
bir.ipynb  gt_github.txt  oyun.py  sunu.pptx
```

Git Komutları

1. **git komutlarını listeler:**

   > git

2. **git versiyonunu gösterir:**

   > git --version

3. **Kullanıcı tanımlar (veya değiştirir):**

   > git config --global user.name “KullaniciAdi”

4. **Tanımlanmış kullanıcı adını gösterir:**

   > git config user.name

5. **Kullanıcı Email’i tanımlar (veya değiştirir):**

   > git config --global user.email kullanici@gmail.com

6. **Tanımlanmış kullanıcı Email’i gösterir:**

   > git config user.email

NOTLAR:
- config → git’in ayarlarıyla ilgili bir değişiklik olduğu zaman kullanılır
- global → yapılan değişikliğin her yerde geçerli olması için kullanılır

- Working Directory: Bizim çalışma yaptığımız klasör'dür.
- Bizim bu WD'de yaptığımız her şeyden git haberdar olmaz. Eğer yaptığımız değişiklikleri git'in denetimi altına almak için git'i başlatmalıyız.
- Bir dosyayı git'in kontrolü altına almak istiyorsak, öncelikle klasörün url kısmına cmd yazıp, aşağıdaki işlemlerle devam edebiliriz.

7. git'in klasörü kontrol edip etmediğini kontrol eder:

> git status

```
C:\Users\monstr\Desktop\Git_GitHub>git status
fatal: not a git repository (or any of the parent directories): .git
```

- Yukarıdaki gibi bir fatal alırsak, bulunduğumuz klasör henüz git'in kontrolü altında değil demek oluyor.

8. git kontrolü başlatır:

> git init

```
C:\Users\monstr\Desktop\Git_GitHub>git init
Initialized empty Git repository in C:/Users/monstr/Desktop/Git_GitHub/.git/
```

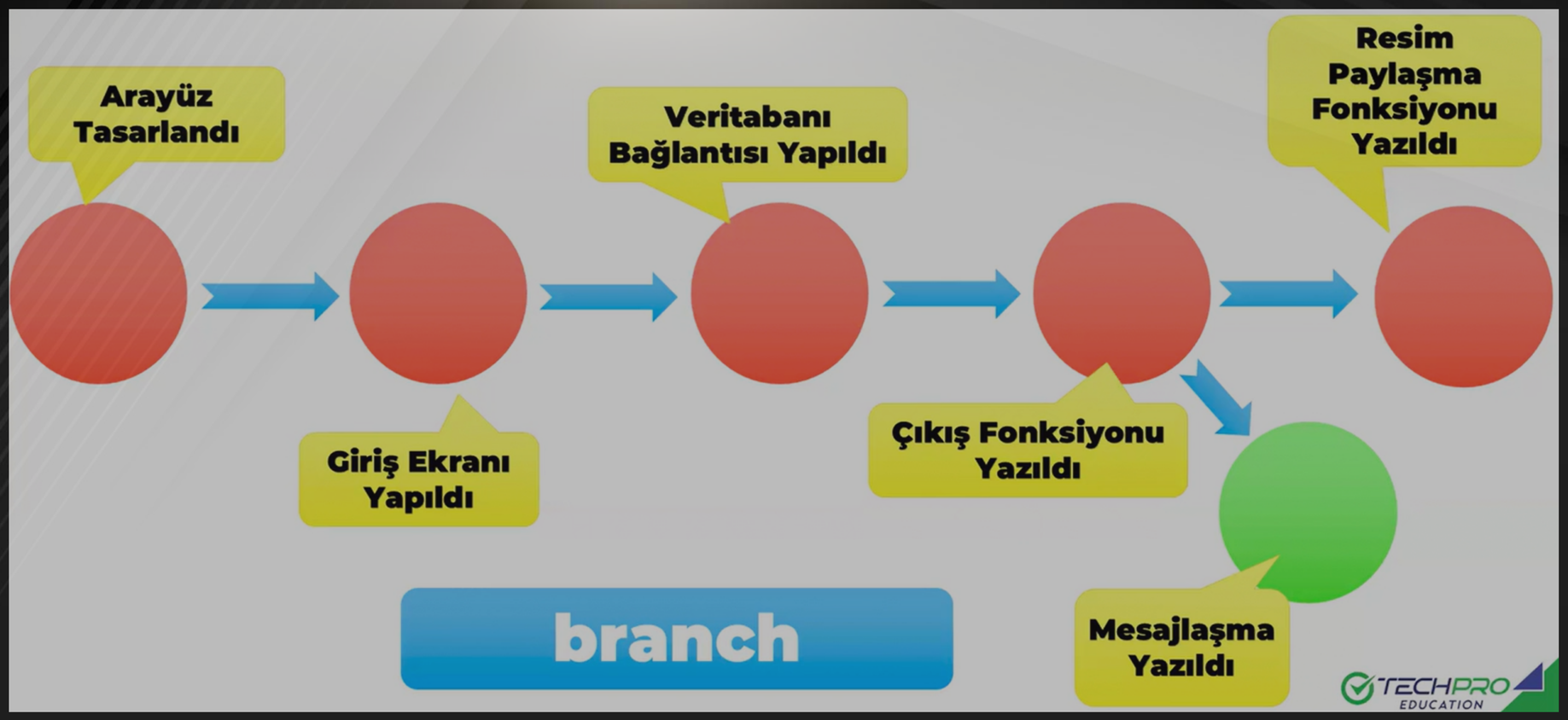
- Yukarıdaki gibi bir cevap alırsak, bulunduğumuz klasör içerisinde git kontrolü başlamıştır demek oluyor.

```
C:\Users\monstr\Desktop\Git_GitHub>git init
Initialized empty Git repository in C:/Users/monstr/Desktop/Git_GitHub/.git/

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        bir.ipynb
        deneme/
        gt_github.txt

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

On branch main: bizim için "main" isimli bir branch oluşturdu

No commits yet: henüz hiçbir şey commit edilmemiş (git sisteminin kontrolünde olan versiyonlanmış bir sürüm yok demek)

Untracted files: takip edilmeyen dosyaları belirtir



- Eğer biz ana fikir'den uzaklaşıp farkı bir fikir denemek istiyorsak, yeni bir branch oluşturup öyle devam ederiz.
- Ya da bir işin belli bir kısmına gelindikten sonra iş ayrılıklarına gidilip sonradan ayrı ayrı yapılan işler birleştirilip devam edilebilir.
- Yani farklı branch'lar üzerinden bir çok iş yapılıp o işler bir araya getirilebilir.

9. klasördeki bir değişikliği git'in Staging denilen bölümüne aktarır:

```
> git add <dosya_adı>
> git add <klasör_adı>
> git add <klasör_adı/dosya_adı>
> git add .    (tamamını Staging bölgesine alır)
```

Çalışma Alanı → Stage → Local Repository

```
C:\Users\monstr\Desktop\Git_GitHub>git add bir.ipynb

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

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

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

Changes to be committed: commitlen'meye yani versiyonlanmaya hazır bir dosya var. Çünkü bir önceki komutta biz git add bir.ipynb yaparak bir dosyayı Stage bölgesine göndermiştik.

```
C:\Users\monstr\Desktop\Git_GitHub>git add deneme/

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   bir.ipynb
        new file:   deneme/oyun.py
        new file:   deneme/sunu.pptx

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

git add deneme/

yukarıdaki komut yazılarak, deneme klasörünün içerisindeki her şeyi Stage bölgesine göndermiş olduk.

```
C:\Users\monstr\Desktop\Git_GitHub>git rm --cached bir.ipynb
rm 'bir.ipynb'

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   deneme/oyun.py
        new file:   deneme/sunu.pptx

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

git rm --cached bir.ipynb

yukarıdaki komut yazılarak, Stage bölgesinden bir tane dosyayı çıkartmış olduk

```
C:\Users\monstr\Desktop\Git_GitHub>git add bir.ipynb

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

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

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

Changes to be committed: commitlen'meye yani versiyonlanmaya hazır bir dosya var. Çünkü bir önceki komutta biz git add bir.ipynb yaparak bir dosyayı Stage bölgesine göndermiştik.

```
C:\Users\monstr\Desktop\Git_GitHub>git add deneme/

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   bir.ipynb
        new file:   deneme/oyun.py
        new file:   deneme/sunu.pptx

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

git add deneme/

yukarıdaki komut yazılarak, deneme klasörünün içerisindeki her şeyi Stage bölgesine göndermiş olduk.

```
C:\Users\monstr\Desktop\Git_GitHub>git rm --cached bir.ipynb
rm 'bir.ipynb'

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   deneme/oyun.py
        new file:   deneme/sunu.pptx

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

git rm --cached bir.ipynb

yukarıdaki komut yazılarak, Stage bölgesinden bir tane dosyayı çıkartmış olduk

add komutunu kullanarak Staging bölgesine aldıklarımızı artık versiyon olarak kaydedebiliriz. Yani Commit edebiliriz.

```
C:\Users\monstr\Desktop\Git_GitHub>git add .

C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   bir.ipynb
        new file:   deneme/oyun.py
        new file:   deneme/sunu.pptx
        new file:   gt_github.txt
```

git add .

yukarıdaki komut yazılarak, çalışma alanımızda yer alan ve henüz git'in kontrolü altına alınmamış olan tüm dosyaları Stage bölgesine göndermiş olduk. Artık hepsi commit etmeye hazır halde.

add komutunu kullanarak Staging bölgesine aldıklarımızı artık versiyon olarak kaydedebiliriz. Yani Commit edebiliriz.

10. **Staging bölgesindeki dosyaları Local Repository'e aktarır:**

    > git commit -m "commit mesajı"

11. **Commitlenen değişikliklerin versiyonlarını gösterir:**

    > git log

```
C:\Users\monstr\Desktop\Git_GitHub>git commit -m "versiyon1"
[main (root-commit) 306d245] versiyon1
 4 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 bir.ipynb
 create mode 100644 deneme/oyun.py
 create mode 100644 deneme/sunu.pptx
 create mode 100644 gt_github.txt
```

git commit -m "versiyon1"

yukarıdaki komut yazılarak, "versiyon1" mesajıyla birlikte Stage bölümündeki dosyalar Local Repository olan main'e gönderildi.

```
C:\Users\monstr\Desktop\Git_GitHub>git log
commit 306d245ca50740e0f455035b5ff3d4df0c4c7475 (HEAD -> main)
Author: Emre Çakır <cakiiremre66@gmail.com>
Date:   Sun Feb 25 12:55:23 2024 +0300

    versiyon1
```

git log

yukarıdaki komut yazılarak, geçmiş versiyonları görmüş oluruz

```
C:\Users\monstr\Desktop\Git_GitHub>git status
On branch main
nothing to commit, working tree clean
```

git status

yukarıdaki komut yazılarak, ağacımızın temiz olduğunu görürüz

bütün değişikliklerimiz Stage bölgesinden Local Repository'e gönderildiğine göre artık son işlemi de yapabiliriz. Yani GitHub'a gönderebiliriz.

GitHub Nedir?

Git kullanarak kaydettiğimiz dosyalarımızı gönderebileceğimiz bir depodur.

Git, sadece bir sürüm kontrol sistemidir ve yerel bir bilgisayarda çalışırken GitHub, Git üzerine inşa edilmiş bir bulut barındırma hizmetidir.

Git, bir projenin sürüm kontrolünü yönetirken, GitHub, projenin paylaşımı, işbirliği, sorun takibi ve kod incelemesi gibi ek özellikler sunar.

Git için kullanılan Repository → Local Repository

GitHub için kullanılan Repository → Remote Repository

Git ile Local'de yaptığımız işlemleri GitHub'a kayıt etmemiz için işlemler:

1. git remote add origin <link> {github'da var zaten}
2. git branch -M main
3. git push -u origin main

Yukarıdaki işlemleri yapıp GitHub'ta açık olan sayfayı yenilersek, daha önceden commit ettiğimiz her şeyin GitHub'ımıza eklendiğini görürüz.