



Git - Github



Git - Github

- › Git – Github nedir?
- › VKS Nedir?
- › Ne amaçla kullanılır

Introduction



Git-Github nedir

Git-Github versiyon kontrol sistemidir

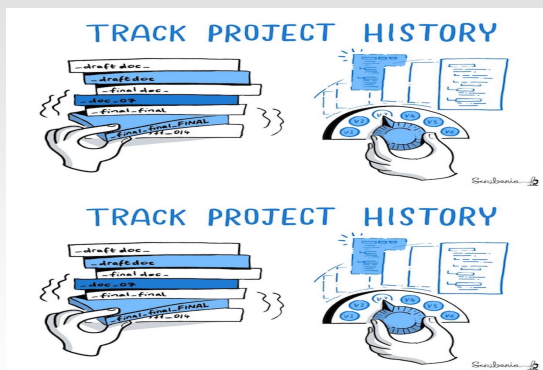


git





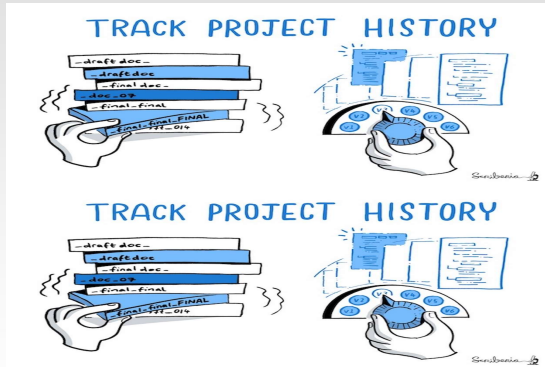
VKS nedir



Versiyon kontrol sistemi, projede dosya ve klasör yapısındaki değişiklikleri kaydeden bir sistemdir.



VKS nedir



- Bazı dosyaların veya projenin tamamının bir önceki versiyona döndürülmesi,
- Zaman içerisinde yapılan değişikliklerin karşılaştırılması,
- Probleme neden olabilecek değişikliklerin en son kimin tarafından yapıldığının tespiti



VKS çeşitleri

3 tip Versiyon Kontrol Sistemi vardır.

YEREL

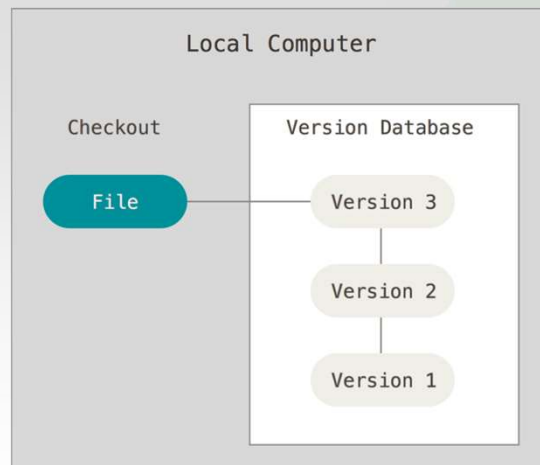
MERKEZİ

DAĞITIK

[Git - Github bu version'u kullanır](#)



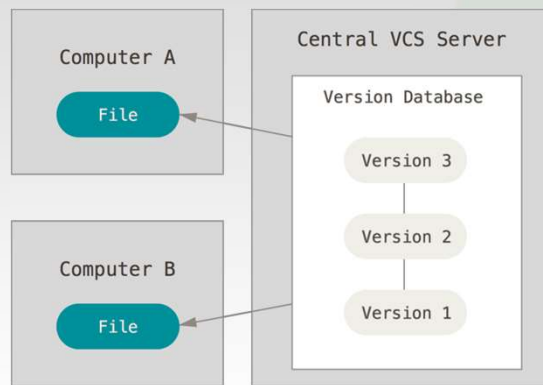
Yerel VKS nedir



<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>



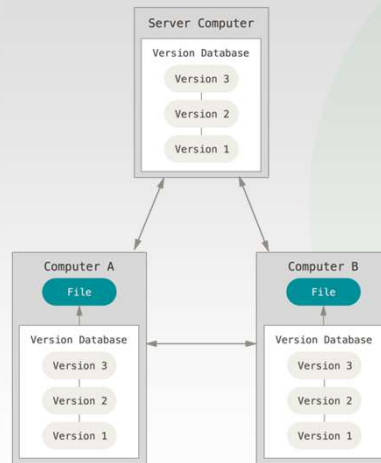
Merkezi VKS nedir



<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>



Dağıtık VKS nedir



<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>



Git-Github Ne Amaçla Kullanılır



git

LOCALE

- Lokalde versiyon yönetimi yapmak
- Offline çalışabilmek
- Hataları geri alabilmek
- Versiyonlar arasında geçiş yapabilmek

cevirimdisi



GitHub

REMOTE

- Yedekleme (backup)
- Proje paylaşımı (share)
- Proje yayınlama (deploy)
- Ortak çalışma (collobration)



Git - Github



- › Kurulum ve ilk ayarlar
- › Repository
- › Lokal repo oluşturma
- › Working directory, staging area, commit changes
- › Değişiklikleri iptal etme
- › Önceki versiyonlara dönme
- › Branchs



Kurulum ve İlk Ayarlar



git

Version Control System

- › Git altyapısını oluşturmak ve git komutlarını kullanabilmek için Git kütüphanesinin kurulması gerekmektedir

[<https://git-scm.com/downloads>]

- › `git --version`



Kurulum ve İlk Ayarlar

Git configuration

geçerli kullanıcı için geçerli

```
git config --global user.name "Ali Gel"  
git config --global user.email "ali@gel.com"
```

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

Yapılan commit leri burada belirtilen isim ve eposta ile ilişkilendirir. Repo da çalışan diğer kişiler bu isim ve epostayı görür.

Terminal de komutların renklendirilmesini sağlar

herkes için geçerli

- **System** parametresi kullanıldığında tüm kullanıcılar ve tüm repolar üzerinde etkili olur
- **Global** parametresi geçerli kullanıcının tüm repolar üzerinde etkili olur
- **Local** parametresi ise sadece geçerli repo üzerinde etkili olur



Genel Kavramlar

depo demek

Repository

Versiyon kontrol ve birlikte çalışma altyapısını ayrı tutmak istediğimiz her bir bağısız yapıya **repository** denir. Genellikle her proje için ayrı bir repository tanımlanır.

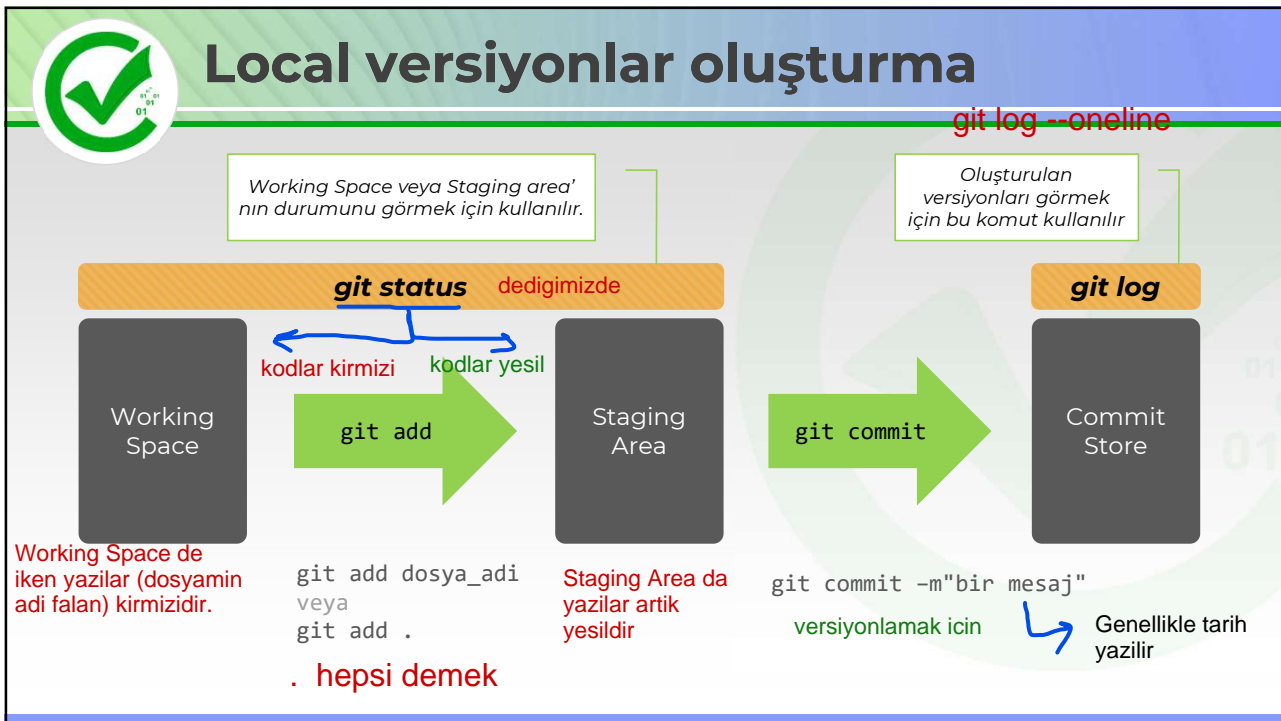


Local repo oluřturma

| **git init** |

Local bilgisayarımızda bir projeyi versiyon sistemine alabilmek için **git init** komutu kullanılır. Bu komut kullanılınca proje klasöründe **.git** klasörü oluřturulur. Bu, local repomuzu saklayacaktır.

COK ÖNEMLİ KAVRAMLAR MUTLAKA BILMEMİZ GEREKEN KAVRAMLAR



Working Space den Staging Area nasıl geçilir? --> git add . ile

git log --oneline --> yaptığımızda ayrıntıları görmek istemediğimizde kullanılır sadece kullanacağımız 7 karakteri veriyor. H kod (36e5032)

daf36ac (HEAD -> master) 1.C

↪ bizim H kodumuz

git log : commit daf36ac6a122a6e6fc566d2aaf6b63631dedbe00 (HEAD -> master)

git log --oneline : daf36ac (HEAD -> master) 1.C

Yanlış yaptığımızda q veya enter basıyoruz

git diff ? Working space deki değişikliği gösterir

git diff --staged ==> Staging deki değişiklikleri bize gösteriyor.



Versiyon detaylarını görme

git show

Bir versiyon içinde, hangi değişikliklerin olduğunu görmek için kullanır.

git show *[hash kodun ilk 7 karakteri]*

git log

```
C:\Users\sariz\Desktop\test>git log
commit c4170fe1afa5deac505808a0a2c8ba05afc8e86d (HEAD -> master)
Author: Your Name <you@example.com>
Date: Sat Aug 7 23:49:17 2021 +0300

    1 satır eklendi

commit 5e063d211454b3bc8846bc0720aef4895b1fdbff
Author: Your Name <you@example.com>
Date: Sat Aug 7 23:40:18 2021 +0300

    first commit
```

git reflog = git log --oneline

git reflog ==> ile versyonlarımızın h kodları ve commitler gösteriyor.



Versiyon oluřturmak için kodlar

Ana komutlar

```
git init  
git add . git add dosya adi da olabilir  
git commit -m "versiyon metni"
```

Repo oluřturur. Her projede en bařta bir kere kullanılır.

Dosyaları staging area ya gönderir

Versiyon oluřturur

Yardımcı komutlar

```
git status  
git log = git reflog = git log --oneline  
git show [hash_kodu] = git reflog
```

Genel durum ile ilgili bilgi verir

Versiyonların listesini verir ve bize H kodlarını veriyor

Versiyondaki deęişiklikleri gösterir



Commit Store & Head

LOCALE REPOSITORY

Commit Store

Commit 32

Commit 31

Commit 30

Commit 29

HEAD

- › Bir repo içinde birden fazla commit olabilir. Bunlardan en son alınan commit'e **HEAD** denir.
- › Bu HEAD değiştirildiğinde önceki versiyonlara dönüş yapılabilir.

```
C:\Users\sariz\Desktop\test>git log
commit c417dfe1afa5deac505808a0a2c8ba05afc8e86d (HEAD -> master)
Author: Your Name <you@example.com>
Date: Sat Aug 7 23:49:17 2021 +0300

    1 satır eklendi

commit 5e063d211454b3bc9846bc0720aef4895b1fdbff
Author: Your Name <you@example.com>
Date: Sat Aug 7 23:40:18 2021 +0300

    first commit
```

HEAD



inteliji kapatmadan yapmamız lazım yapılan çalışmayı iptal eder. eski versiyona eski haline döner

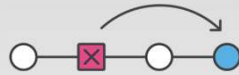
git status : ---> dan sonra ---> nothing to commit, working tree clean (Çalışma alanınız temiz)

herşey karıştıgında ne yapmamız lazım? Bilgisayarım ==> C:\ gidiyoruz (projeyi nerede oluşturduysak)
==> proje klasörü ==> .git klasörünü siliyoruz
!! Dikkat : Önce Working space'de ki çalışmalarımızı muhafaza etmemiz lazım



Önceki versiyonlara dönmek

1.Yöntem: CHECKOUT



Commits

Commit 32



Commit 31

Commit 30

Commit 29

Commit 33

Önceki versiyonu incelemek için

git checkout *[hash]*.

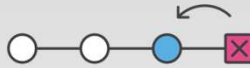
Bu işlemi kalıcı hale getirmek için

git commit -m"..."



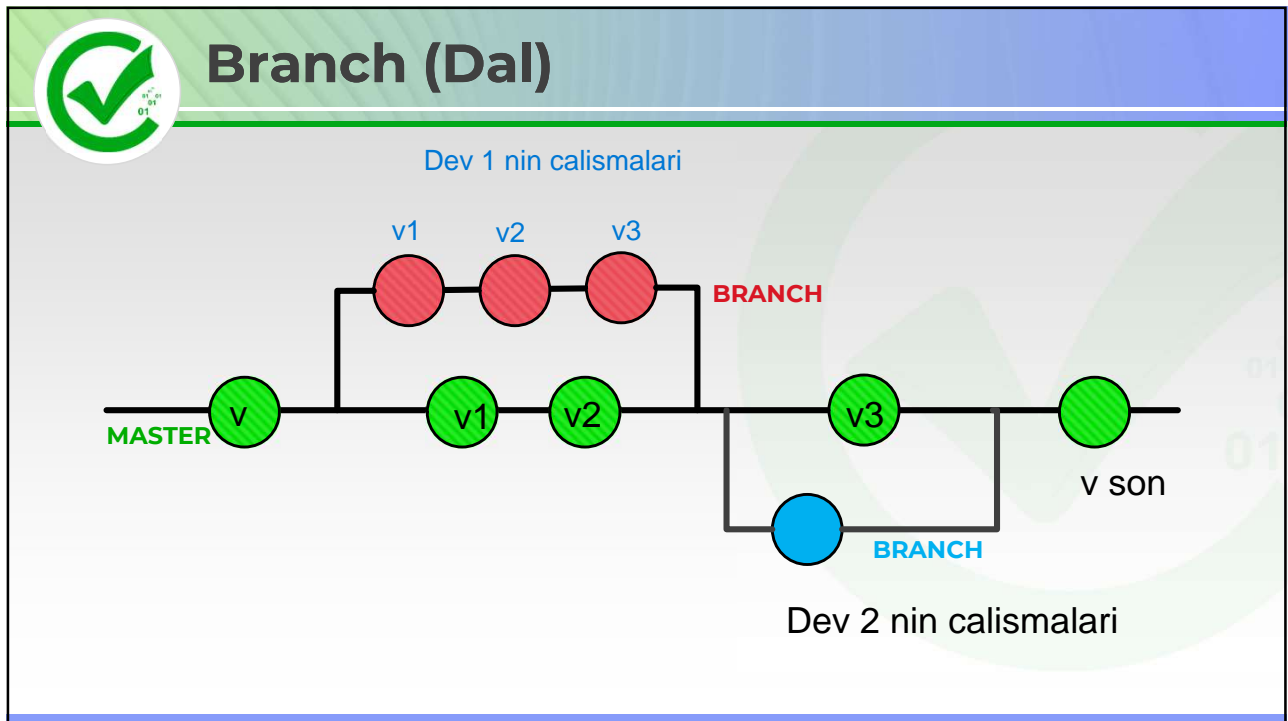
Önceki versiyonlara dönmek

2.Yöntem: RESET



Geri alınamayacak şekilde önceki versiyona dönmek

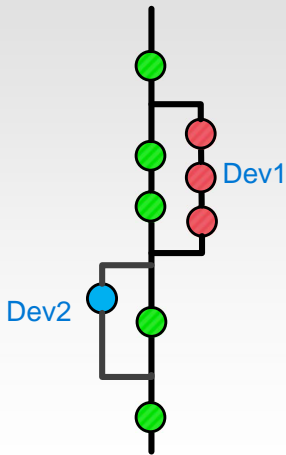
git reset --hard *[hash]*



Branch--> projenin ana yapiya dokunmadan yan dallar olusturarak projeye yeni eklemeler yapilir



Branch lerin faydaları



- ◉ Original kodların güvenliği sağlanır
- ◉ Her developer kendi bölümünden sorumlu olur
- ◉ Daha hızlı geliştirme yapılır
- ◉ Daha az hata oluşur
- ◉ Sorunlar daha hızlı düzeltilir.
- ◉ Organize kod yapısı sağlanır
- ◉ Kaos olmaz



Branch Komutları

`git branch [isim]`

Yeni branch oluşturur

`git checkout [isim]`

Branch aktif hale gelir

`git branch -m [isim]`

Branch ismini değiştirir.

`git branch`

Mevcut branch leri listeler

`git branch -d [isim]`

Branch i siler

`git merge [isim]`

İki branch i birleştirir

Hangi branch te olduğumuzu öğrenmek için `git branch` ---> sonra hangi branch'in basında (*) varsa ve rengide yesil ise o branch'teyiz

`git merge [isimle]` ---> ile iki brabch'i birleştirmis oluyoruz. Bunun icinde hangi branch asagadaysa ve master branch'imiz yukarida HEAD olarak geciyorsa o branch'e gecim master branch ile birlestiriyoruz



Stashing

Working directory ve stage area daki –henüz *commit haline gelmemiş*-değişikliklerin **geçici olarak geri alınması** için stashing işlemi yapılır.

git stash

Working space ve staging area daki değişiklikleri geçici olarak hafızaya alır ve bu bölgeleri temizler

git stash list

Hafızaya alınan değişiklikleri görmek için kullanılır

git stash pop


Hafızaya alınan değişiklikleri geri uygulamak için kullanılır.




Git - Github







- › Hesap oluřturma
- › Repo oluřturma
- › Genel kavramlar
- › Github alıřma prensibi
- › Clone
- › Push & Pull
- › Gitignore
- › Merge & Conflicts




Github hesabı oluşturma




Github.com



[Why GitHub?](#)  [Team](#) [Enterprise](#) [Explore](#)  [Marketplace](#) [Pricing](#) 



[Sign in](#) [Sign up](#)



Github hesabı oluşturma



```
Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ techproed11@gmail.com

Create a password
✓ .....

Enter a username
✓ techproed11

Would you like to receive product updates and announcements via
email?
Type "y" for yes or "n" for no
✓ n
```



Eposta adresinizi giriniz

Şifre belirleyiniz

Bir kullanıcı adı belirliyoruz

Ürün güncelleştirmeleri ve tanıtımlardan email yoluyla haberdar olmak istemiyorsak n yazıyoruz

Github hesabı oluşturma

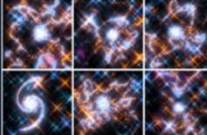


Verify your account

Doğrulama
Gerçek bir kişi olduğunuzu anlamamız için lütfen bu bulmacayı çözün

Doğrula

Sarmal galaksiyi seçin





Create account

Doğrula butonuna basıyoruz

Doğrulama adımlarını geçiyoruz

Create Account butonuna basıyoruz

Github hesabı oluřturma



You're almost done!
We sent a launch code to `techproed11@gmail.com`
→ Enter code

Eposta adresine gönderilen kodu girerek işlemi tamamlıyoruz

Github repo oluřturma



The screenshot shows the GitHub 'Create a new repository' page. The page has a header with the GitHub logo and a navigation bar. Below the navigation bar, there are tabs for 'Overview', 'Repositories' (selected), 'Projects', and 'Packages'. A search bar and filters for 'Type', 'Language', and 'Sort' are present. A green 'New' button is highlighted with a red circle and the number 1. The main content area is titled 'Create a new repository' and includes a description of what a repository is. Below this, there are fields for 'Owner' (set to 'techproeducation1'), 'Repository name' (with a red circle and the number 2), and 'Description (optional)'. There are also radio buttons for 'Public' (selected) and 'Private' (with a red circle and the number 3). A green 'Create repository' button is at the bottom, highlighted with a red circle and the number 4. The background of the slide features a large, faint GitHub logo.

- 1 New butonuna basılır
- 2 Repository için bir isim veriyoruz
- 3 Herkes tarafından ulaşılabilir mi olsun, yoksa sadece bizim belirlediğimiz kullanıcılar mı ulaşabilsin
- 4 Create repository butonuna basılır



Kavramlar

Clone

Github daki bir repoyu lokale indirme işlemidir

Push

Lokalde oluşturulan commit lerin github a gönderilmesi işlemidir.

Fetch

Github daki en son versiyonun, lokal ile karşılaştırılarak –varsa- değişikliklerin indirilmesi işlemidir.

Merge

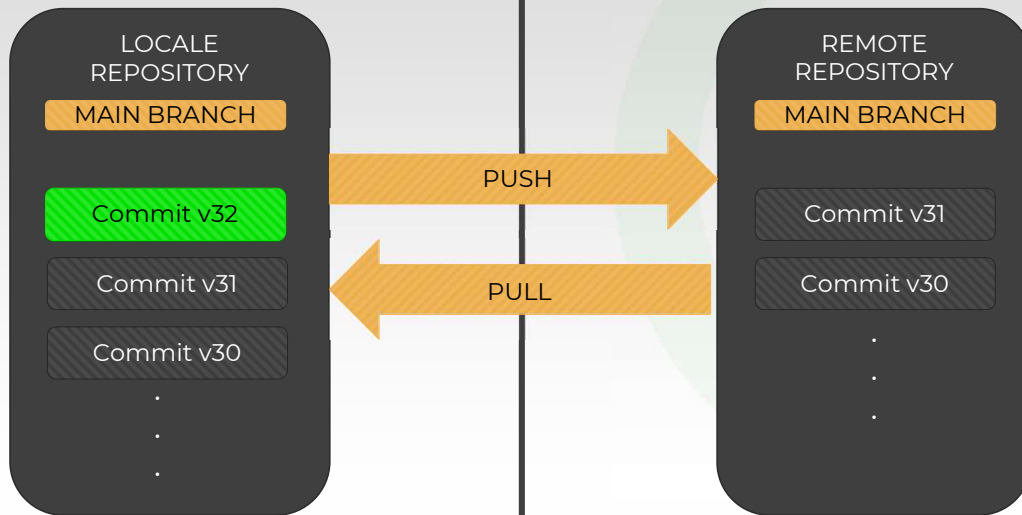
İndirilen değişikliklerin lokale uygulanması işlemidir

Pull

Fetch ve Merge işlemini tek başına yapar



Github Çalışma Prensipleri





Cloning

| **git clone** |

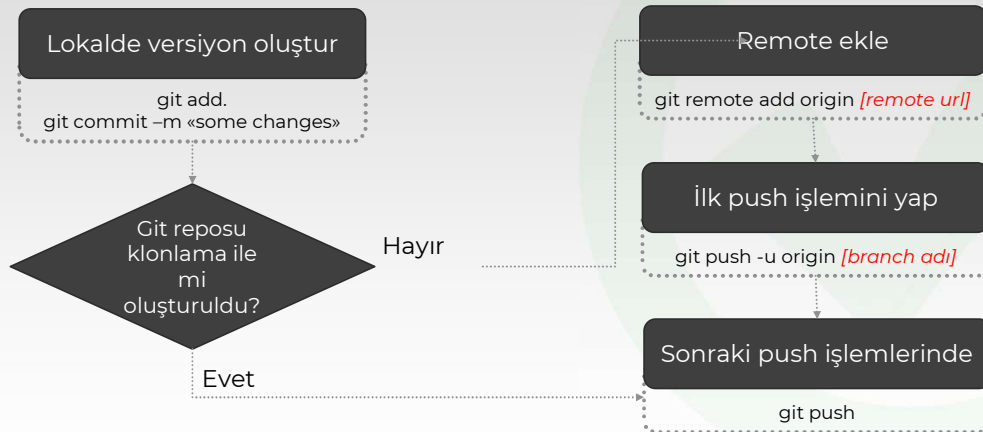
Github daki bir reponun lokale indirilmesi işlemine klonlama denir. Public olan veya gerekli izinlere sahip olunan private repolar klonlanabilir.

Bunun için **git clone** komutu kullanılır.

git clone <https://github.com/techproeducation-batches/B-71-FED-TR.git>



Github'a yükleme (pushing)





Github dan commit çekme (pulling)

Github üzerinden local repo güncellenmek istenirse aşağıdaki komutlar kullanılır

git fetch

Değişiklikleri remote'dan local'e indirir

git merge

İndirilen değişiklikleri local repoya uygular

VEYA

git pull

fetch & merge



.gitignore

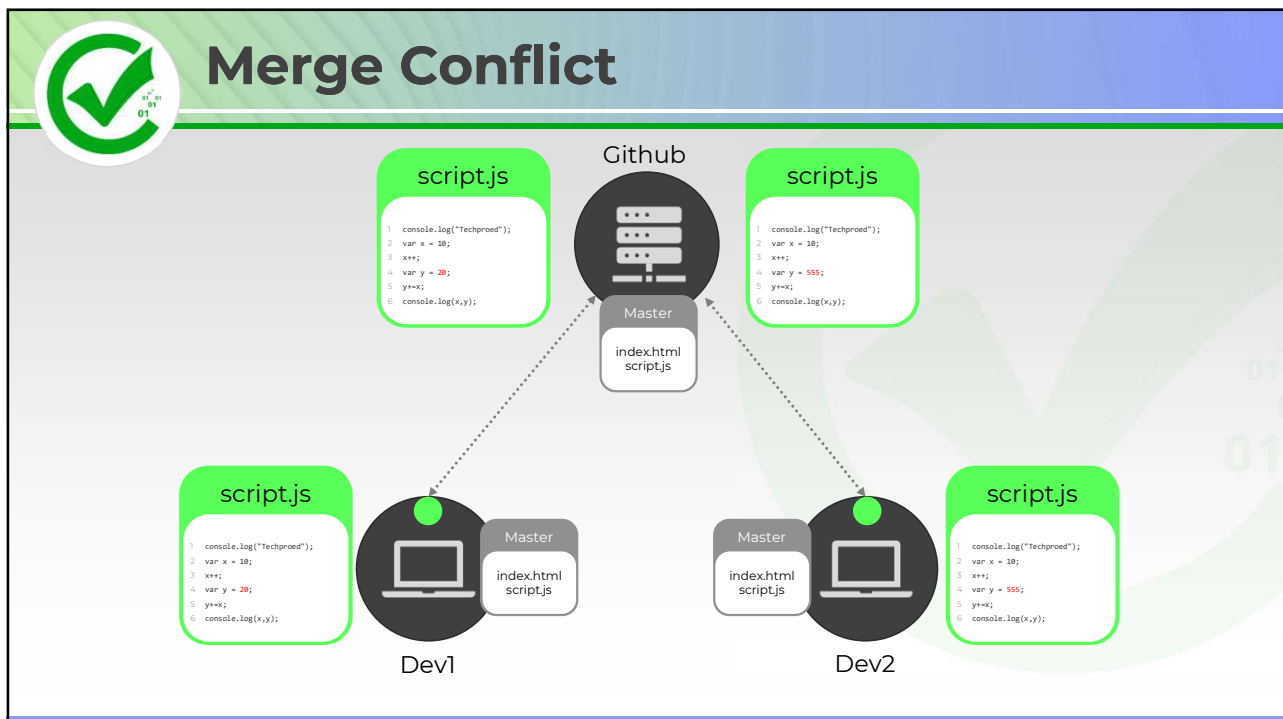
Staging area'ya gitmesini istemediğimiz, yani versiyon kontrol sistemine dahil etmek istemediğimiz dosya ve klasörlerimizi tanımladığımız özel bir dosyadır.

```
out/  
.idea/  
.idea_modules/  
*.iml  
*.ipr  
*.iws
```

.gitignore



Merge Conflict





Git – Github Çalışma döngüsü

BEST
PRACTISE

