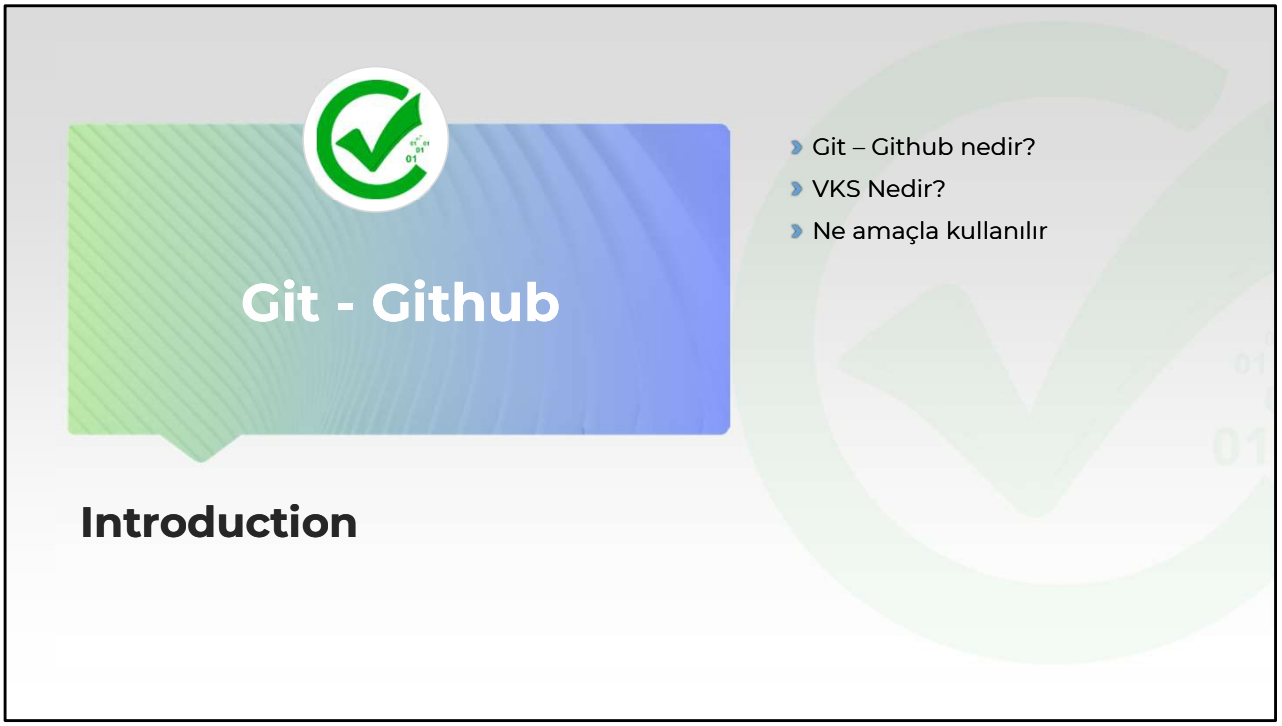
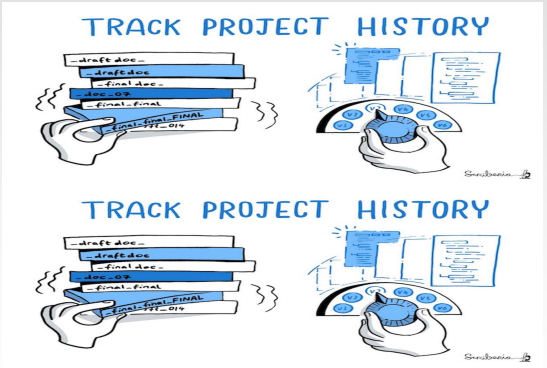

Git - Github


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

Git-Github nedir

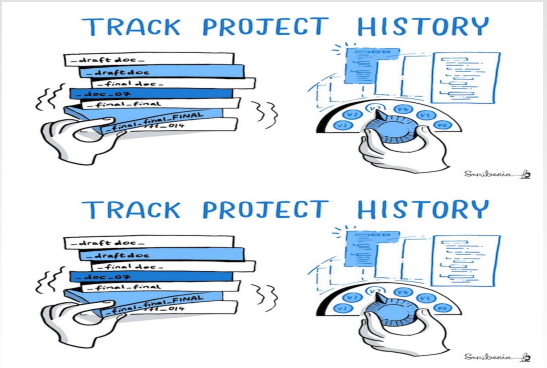
Git-Github versiyon kontrol sistemidir

VKS nedir



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

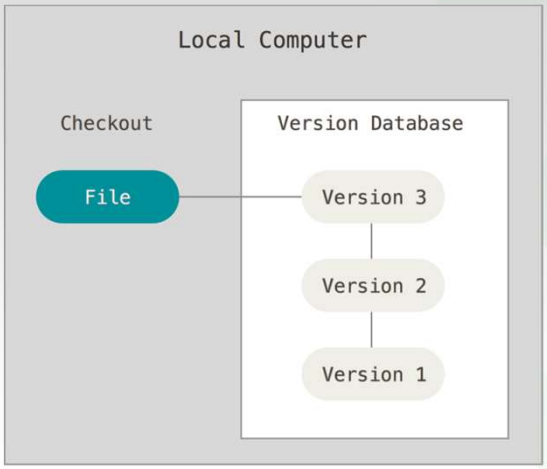
VKS nedir



- 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

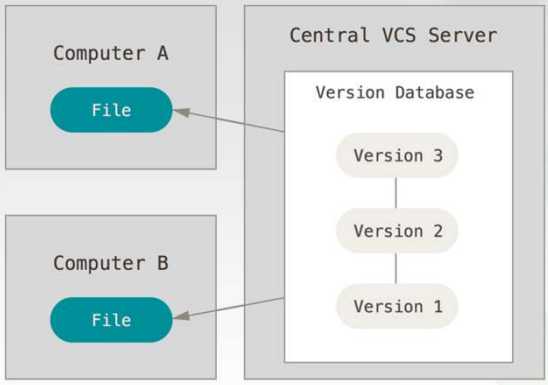
Yerel VKS nedir



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

YVKS, versiyon kontrol sisteminin lokal bilgisayarda tutulduğu sistemlerdir.
Bu sistemde geliştirici kendi lokal bilgisayarında uygulama ile ilgili versiyon sistemi kullanabilir ancak farklı developerlar ile çalışmak isterse YVKS sistemi bunun için bir çözüm üretemez.

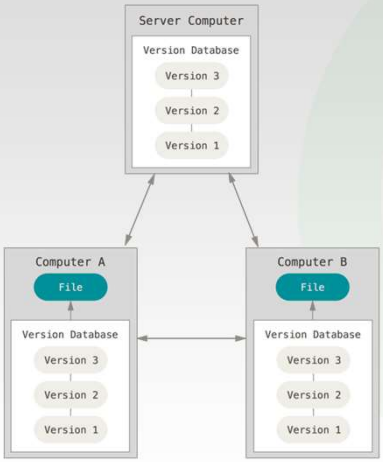
Merkezi VKS nedir



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

Bu sistemde verisyonların depolanması ve kontrolü uzaktaki bir sunucu üzerinden yapılmaktadır. ***Lokal cihazlarda herhangi bir depolama ve kontrol yapılmaz***.
Bu sistemin en büyük sorunu eğer o sunucuda bir sorun oluştuğu andan itibaren hiç kimse iş yapamaz veya üzerinde çalışmakta oldukları herhangi bir şeye sürüm değişikliklerini kaydedemezler.

Dağıtık VKS nedir



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

İşte tam da burada devreye Dağıtık Versiyon Kontrol Sistemleri (DVKS) giriyor. Bir DVKS'de hem merkezi bir sunucu bulunmaktadır, hem de client larda da aynı yapının bir kopyası bulunmaktadır.

Dolayısıyla eğer bir sunucu devre dışı kalırsa, client larda da aynı yapı bulunduğundan sunucu devreye girene kadar her bir geliştirici lokalde çalışabilirken, sunucu devreye alındığında client lar tarafından sunucu rahatlıkla güncelleyebilir. Her client, en nihayetinde tüm verilerin tam bir yedeğidir aslında.

Git-Github Ne Amaçla Kullanılır

git LOCALE

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

GitHub REMOTE

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


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

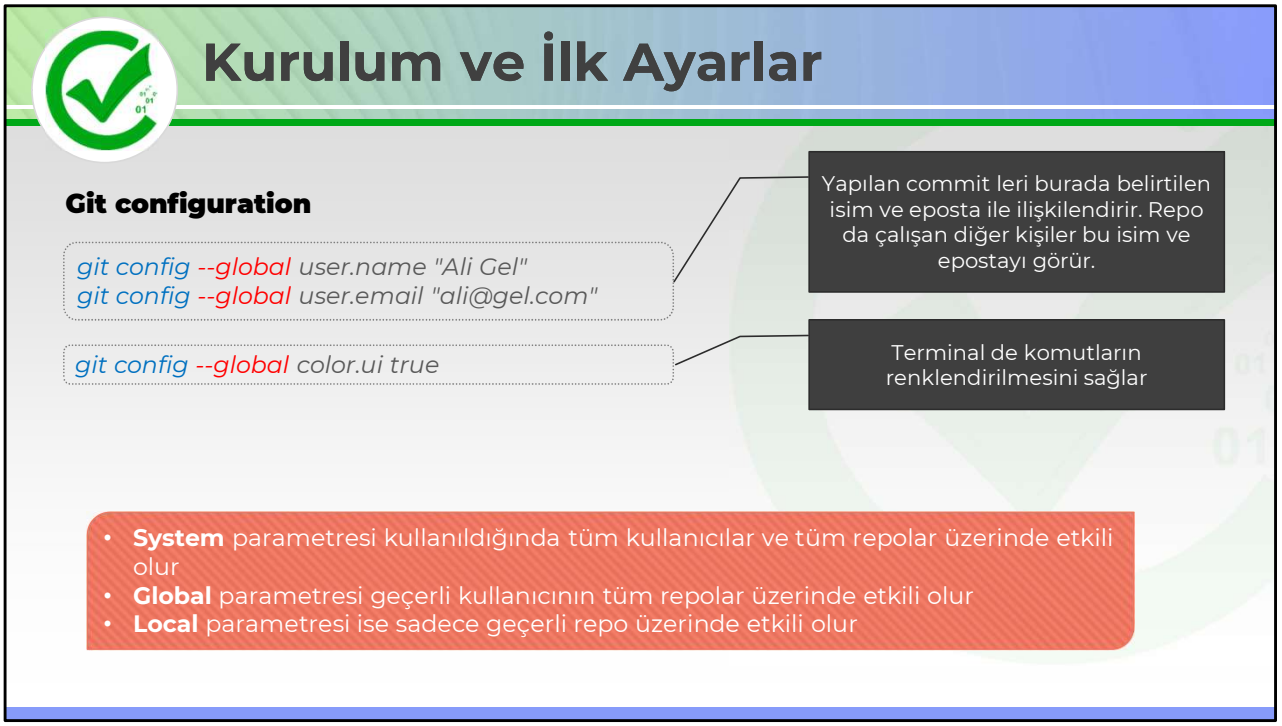
Kurulum ve İlk Ayarlar

Version Control System

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

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

- *git --version*



Git config --list: Tüm ayarları listeler
Git config user.name: istenilen ayarı lsiteler

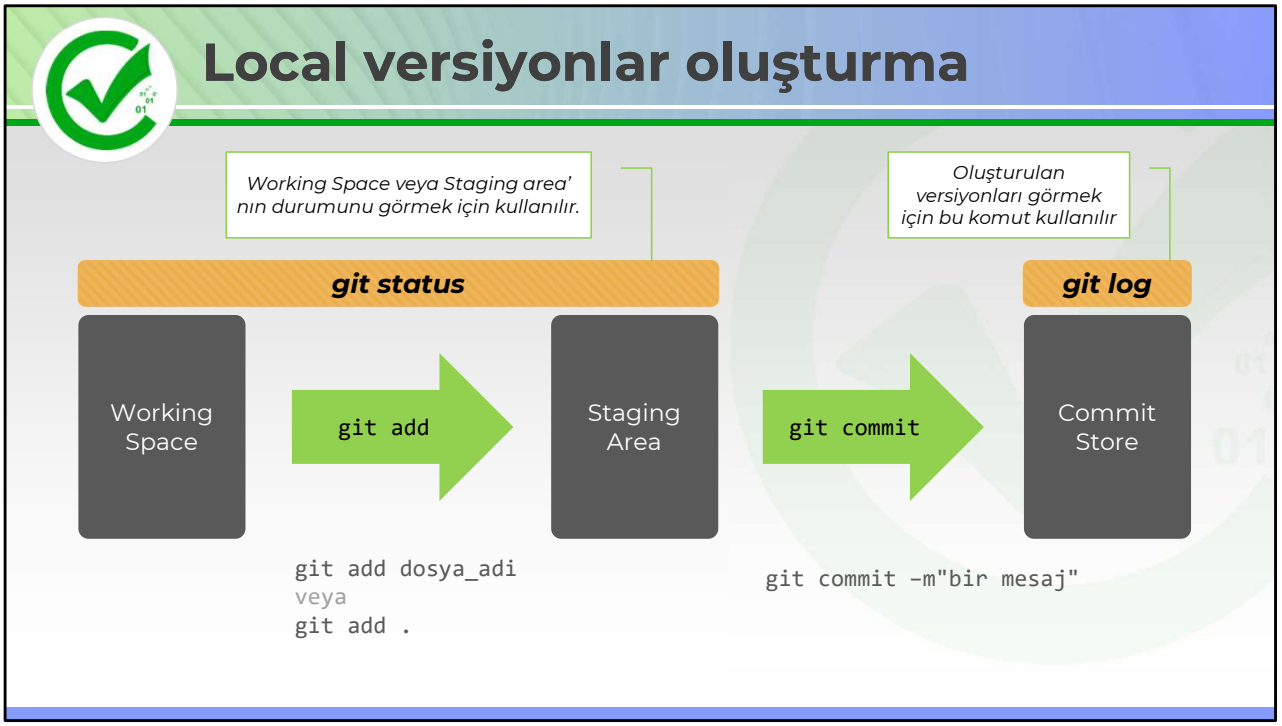

Genel Kavramlar
Repository
Versiyon kontrol ve birlikte çalışma altyapısını ayrı tutmak istediğimiz her bir bağısız yapıya **repository** denir. Genellikle her proje için ayrı bir repository tanımlanır.

Local repo oluşturma

git init

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


Genel Kavramlar
Working Space
.git klasörünün bulunduğu çalışma alanıdır. Klasörler ve dosyalar üzerinden değişiklik burada yapılır.
Staging Area
Versiyon oluşturulacak olan dosya veya klasörlerin geçici olarak toplandığı yerdir.
Versiyon (commit) oluşturulduktan sonra otomatik olarak staging area boşaltılır
Commit Store
Git her bir commit i ayrı bir versiyon olarak tutar. Böylece yapılan çeşitli değişikliklerden sonra projede sorunlar ortaya çıkarsa bir önceki commit e geri dönülebilir.



Git add iki farklı şekilde kullanılır.
1- Belli bir dosyayı stage e göndermek için git add komutundan sonra dosyanın ismi yazılır.
2- Değişiklik yapılan tüm dosyaları stage a göndermek için nokta konulur.

Versiyon detaylarını görme

git show

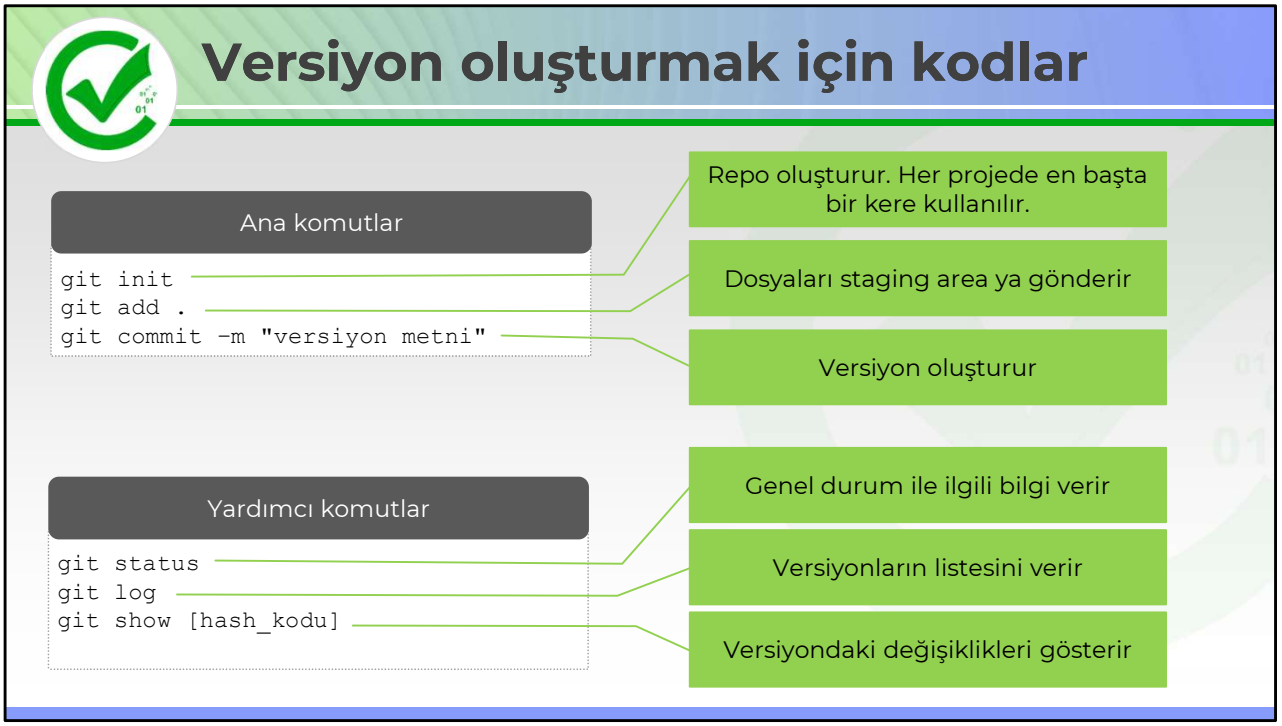
Bir versiyon içinde, hangi değişikliklerin olduğunu görmek için kullanır.

git show *[hash kodun ilk 7 karakteri]*

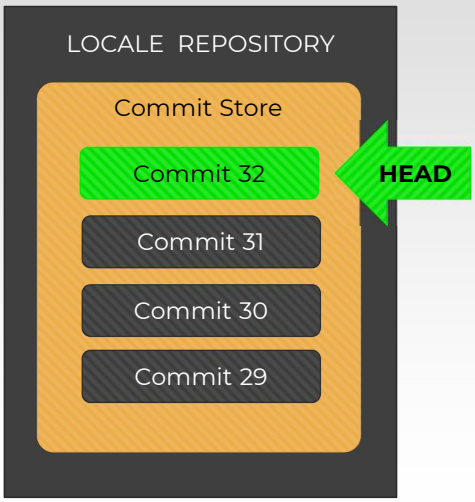
git log

Bir versiyonda hangi değişikliklerin olduğunu görmek için öncelikle git log komutu kullanılarak ilgili commit in hash kodu öğrenilir. Ardından aşağıdaki komut kullanılarak detaylara ulaşılır

Versiyon oluşturmak için kodlar



Commit Store & Head

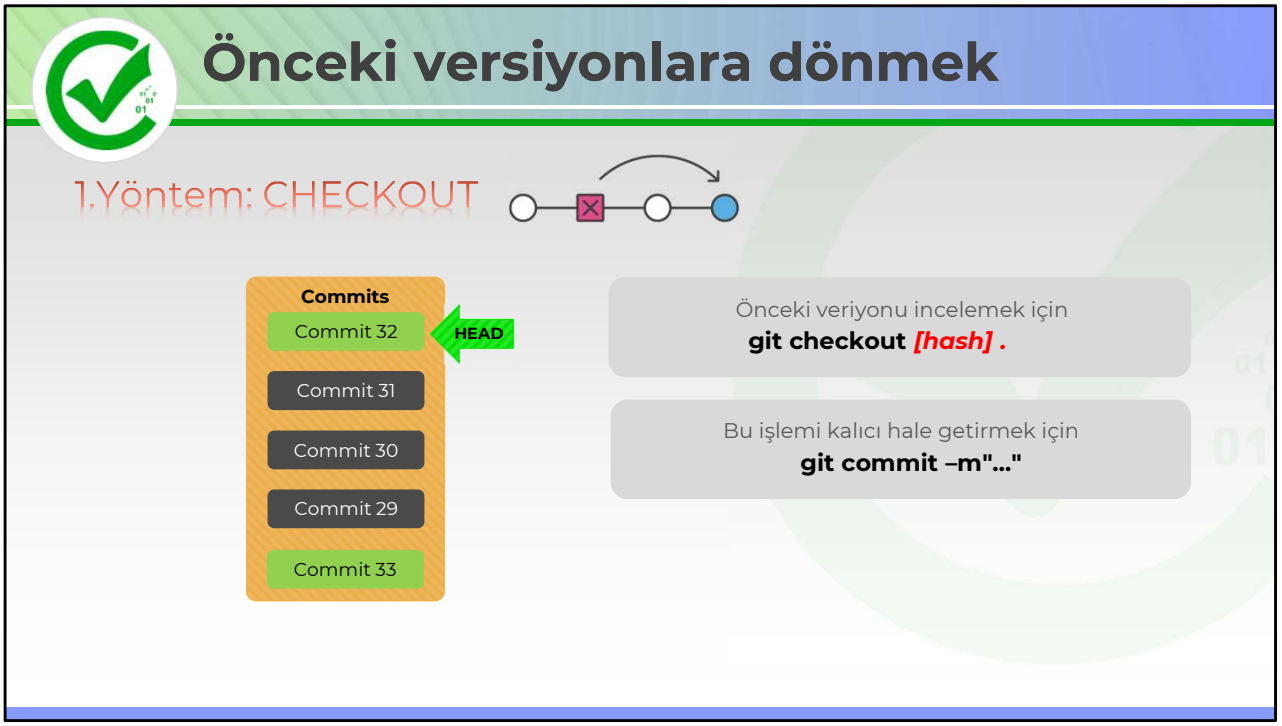


- Bir repo içinde birden fazla commit olabilir. Bunlardan en son alınan commit'e **HEAD** denir.
- Bu HEAD değiştirildiğinde önceki versiyonlara dönüş yapılabilir.

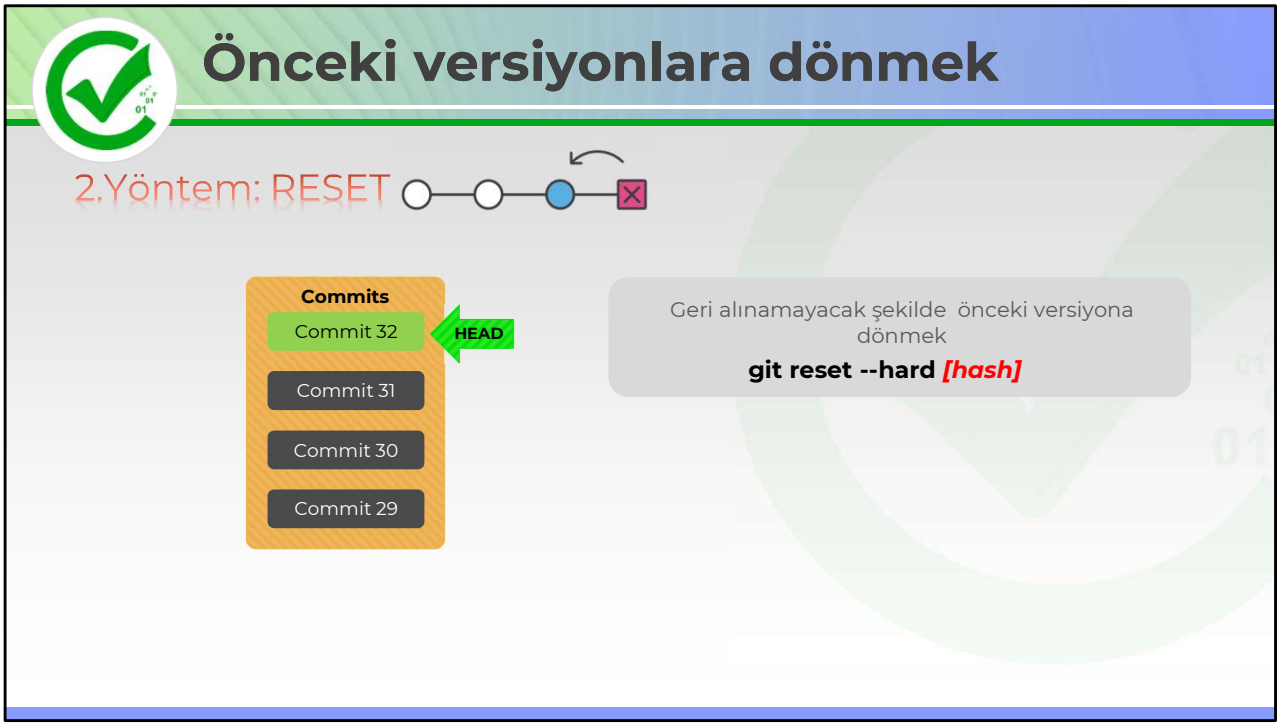
```
C:\Users\sariz\Desktop\test>git log
commit c417dfe1afa5deac505808a0a2c8ba05afc8e86d (HEAD -> master)
Author: Your Name <you@example.com>
Date:   Sat Aug 7 23:49:17 2021 +0300

    1 satır eklendi

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

    first commit
```

HEAD

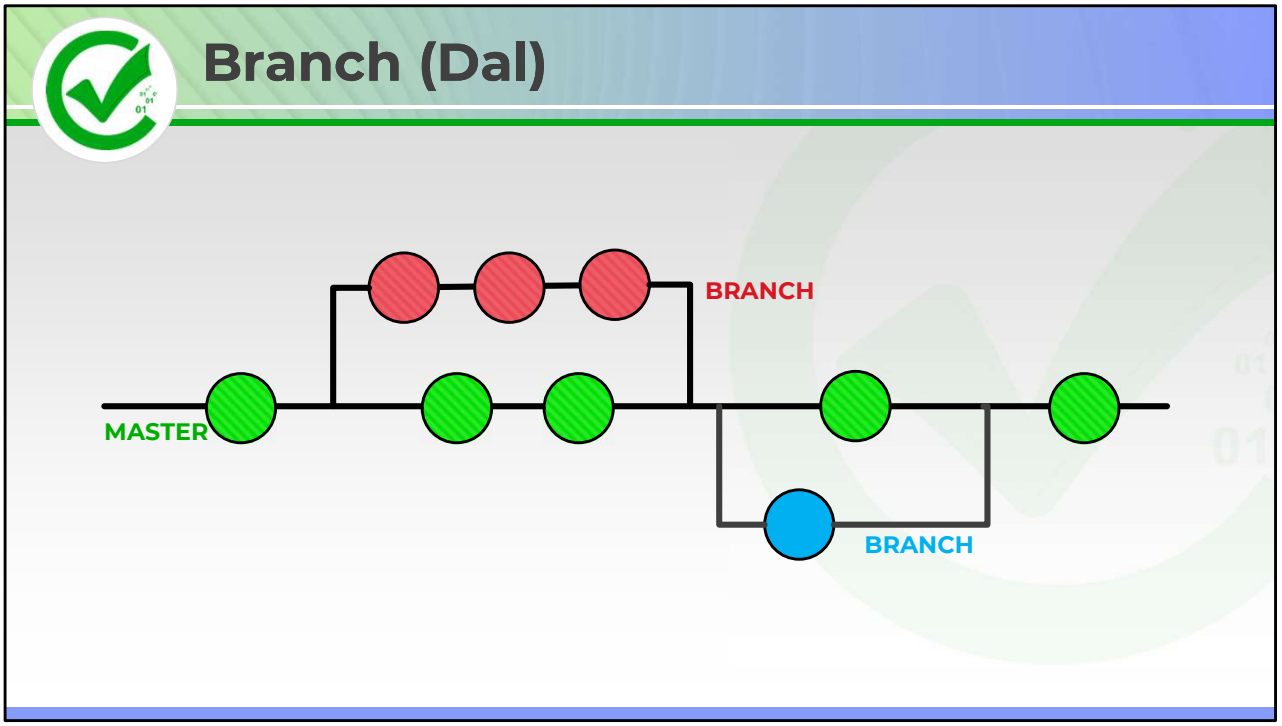
Değişiklikleri iptal etmek



git checkout [hash] komutu ile istenilen versiyona izleme modunda geçilir. Yani değişiklikler geçerli olmaz sadece HEAD hareket ettirilmiş olur. Bu durumdaki 2 seçenek vardır. Ya tekrar eski konuma geri dönmek. Ya da bu geri dönüş işlemini de bir commit yaparak geçmişe gitmeyi de kalıcı hale getirmektir. Aslında önceki bir commit e gidlip tekrar commit yapıldığında önceki commit, en tepeye yeni bir commit olarak gelecektir.

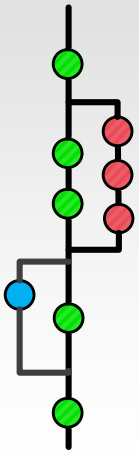


İstenilen versiyona geri döner, bu versiyondan daha sonra yapılan tüm commit ler ve içerdiği değişiklikler geri alınamayacak şekilde iptal edilir.



Master branch, projemizin ana yapısıdır. Bu ana yapıyı bozmadan proje üzerinde bazı geliştirmeler yapmak için branch ler kullanılır. Branch ler içindeki değişiklikler master dan bağımsız olarak saklanır. İstenilirse branchler birleştirilebilir. Colloboration durumunda (aynı anda birden fazla developerın çalıştığı durumlarda) hiç kimse doğrudan master üzerinde çalışmaz. Herkes kendi branch inde çalışır, branch deki geliştirme tamamlandığında master branch ile birleştirilir.

Branch lerin faydaları



- Orginal 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

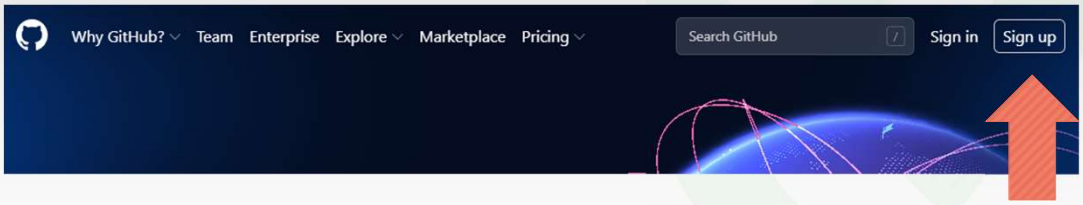
Branch Komutları

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**

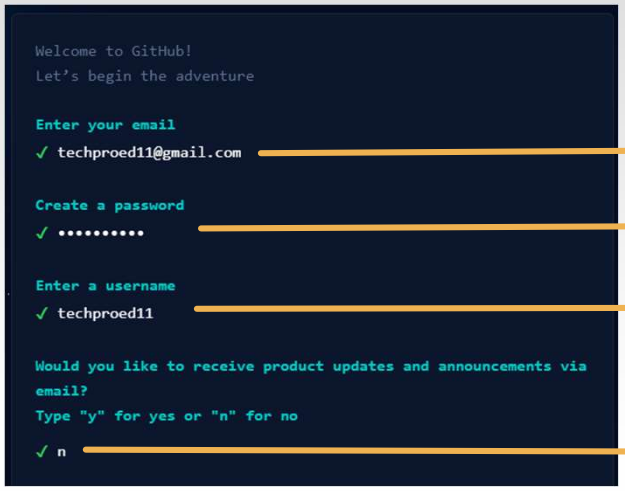

Git - Github
GitHub
Hesap oluşturma
Repo oluşturma
Genel kavramlar
Github çalışma prensibi
Clone
Push & Pull
Gitignore
Merge & Conflicts

Github hesabı oluşturma

Github.com



Github hesabı oluşturma



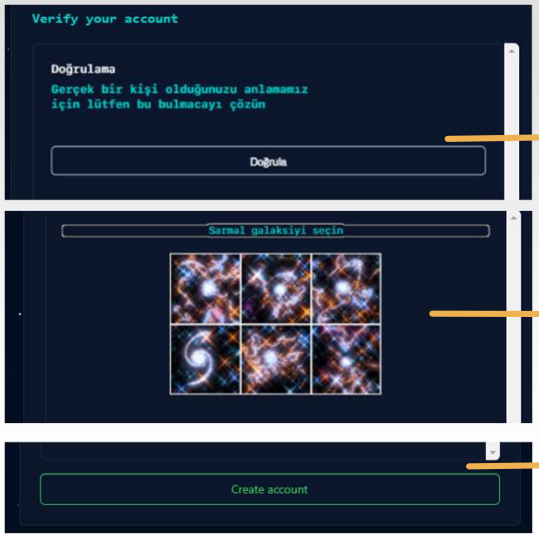
Eposta adresinizi giriniz

Şifre belirleyiniz

Bir kullanıcı adı belirliyoruz

Ürün güncelleştirmeleri ve tanıtımlardan email yoluyla haberdar olmak istemiyorsak n yazıyoruz

Github hesabı oluşturma

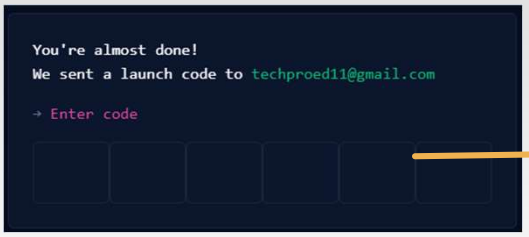


Doğrula butonuna basıyoruz

Doğrulama adımlarını geçiyoruz

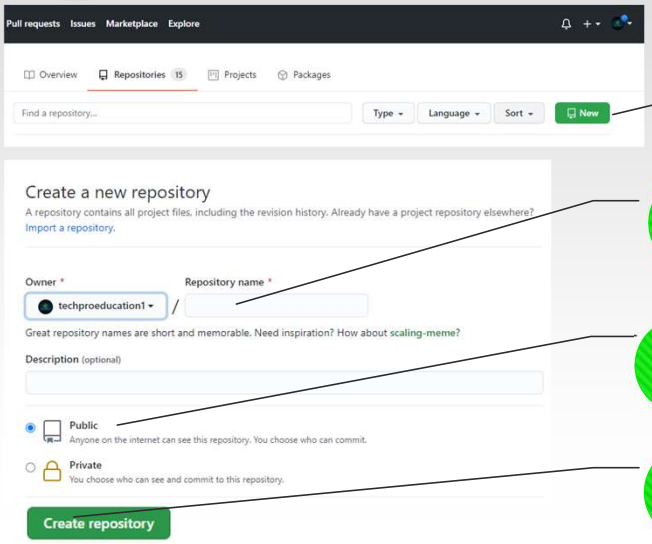
Create Account butonuna basıyoruz

Github hesabı oluşturma



Eposta adresine gönderilen kodu girerek işlemi tamamlıyoruz

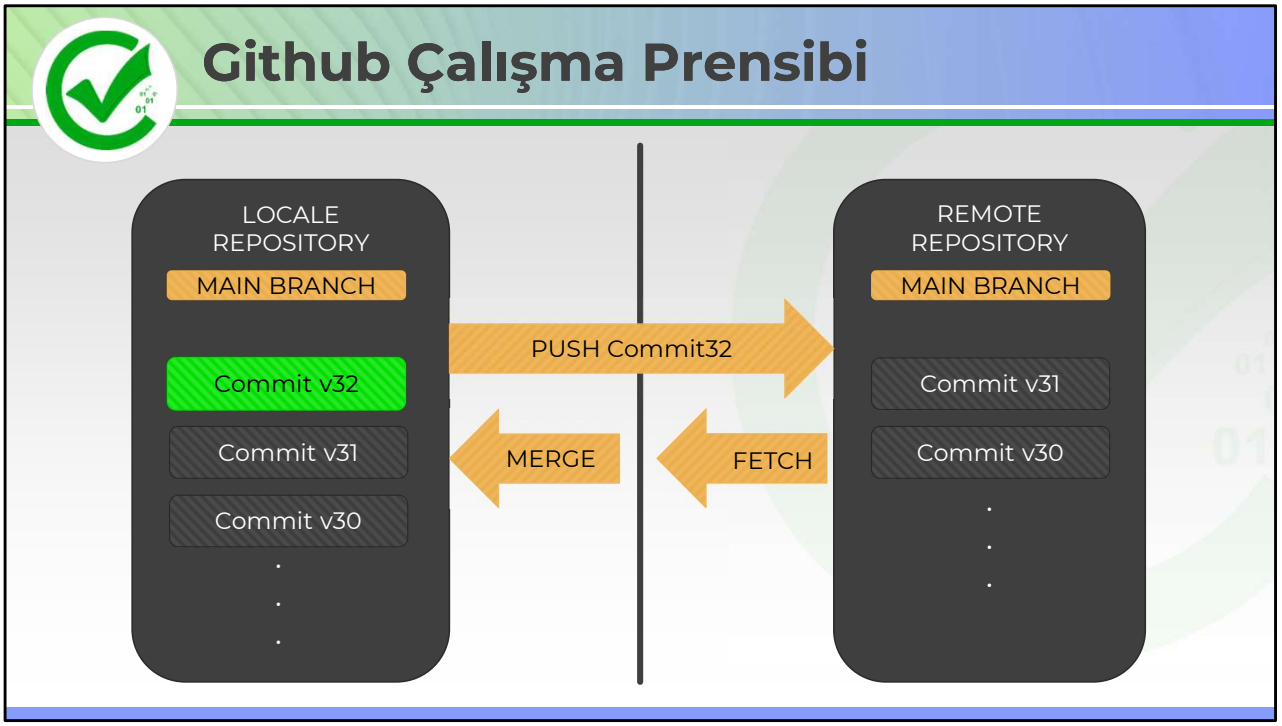
Github repo oluşturma



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



Lokal repo da bir commit oluşturulduktan sonra, bu commit i Github a göndermek için push işlemi yapılır, github da bulunan değişiklikleri lokale çekmek için fetch ve merge işlemi yapılır.

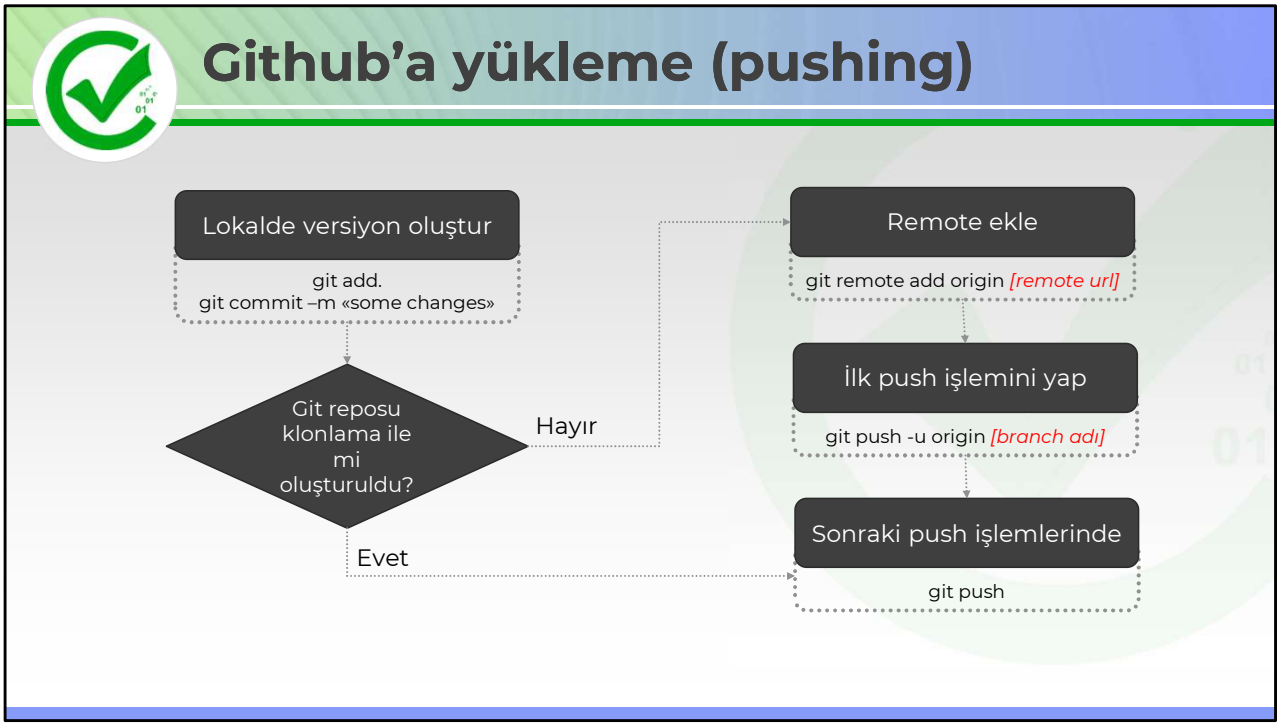
Cloning

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-batchs/B-71-FED-TR.git



Localde en az 1 commit oluşturduktan sonra, her proje için bir kereye mahsus olmak üzere yukarıdaki komutlar kullanılır. Böylece local repo ile github ilişkilendirilmiş olur.
git init ile sıfırdan açılan repolarda bu komutlar kullanılmalıdır. Eğer repo lokale git clone ile alınmışsa remote add komutunun kullanımına gerek yoktur. Sadece git push –u origin master komutu yeterlidir.
–u origin master komutu sadece ilk push işleminde yapılır. Sonrakilerde sadece git push yeterlidir.

Git push –u origin master komutu ilk kullanıldığında eğer daha önceden github hesabına login olunmamışsa login ekranı açılabilir veya kullanıcı adı şifre istenebilir. Girilen bu bilgiler windows ta «Denetim Masası/Kullanıcı Hesapları/Kimlik Bilgileri» bölümünde, MAC te ise Keychain uygulamasında saklanır.

Github dan commit çekme

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

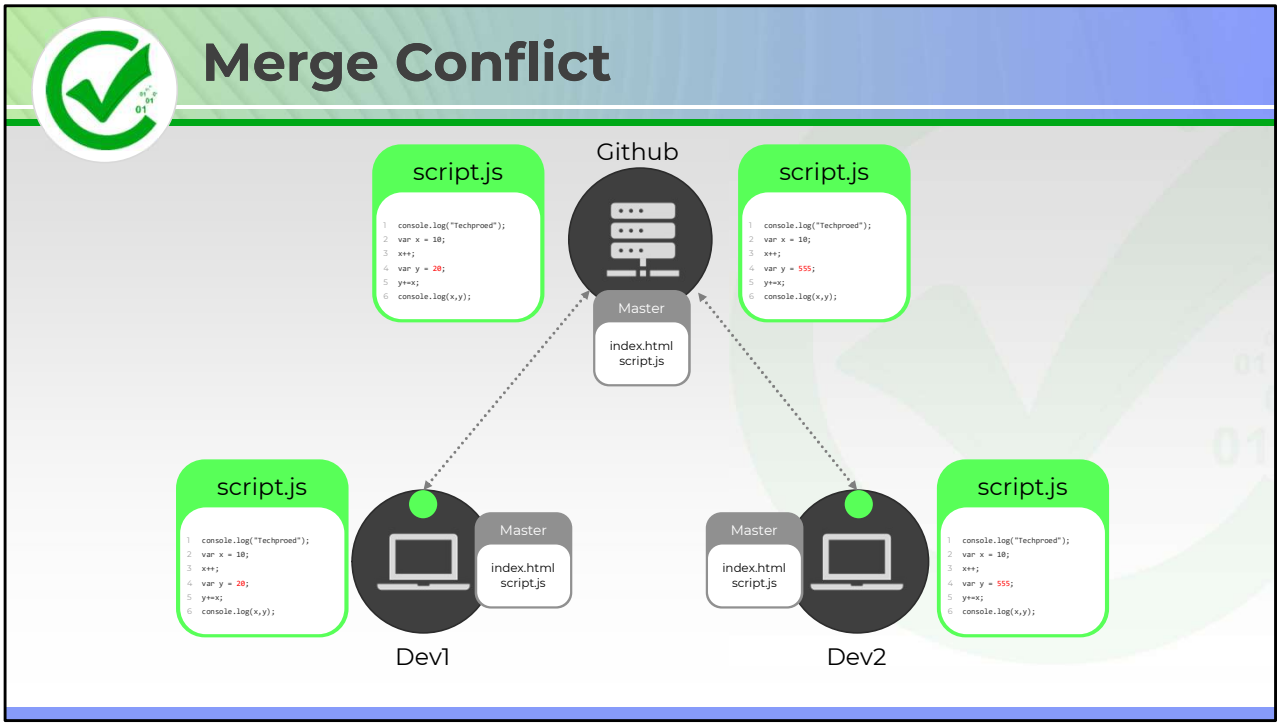
fetch & merge

.gitignore

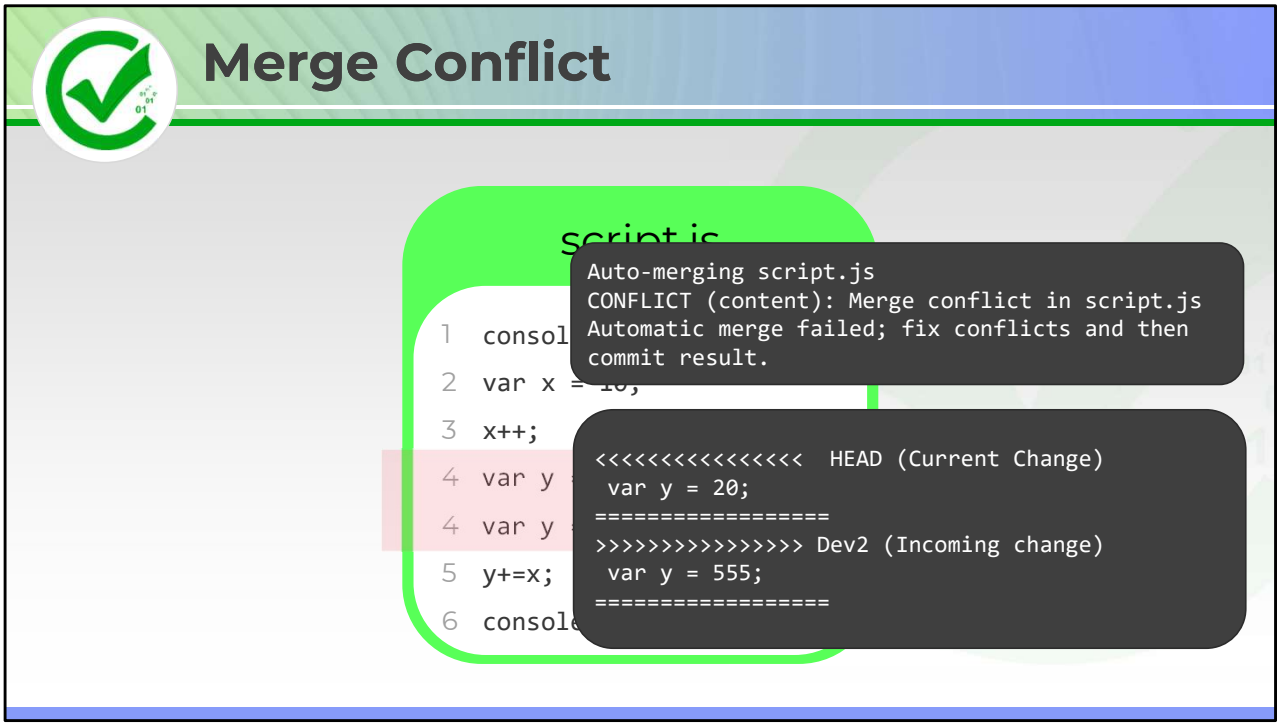
Staging area' ya gitmesini istemediğimiz, yani versiyon kontrol sistemine dahil etmek istemediğimiz dosya ve klasörlerimizi tanımladığımız özel bir dosyadır.

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

.gitignore



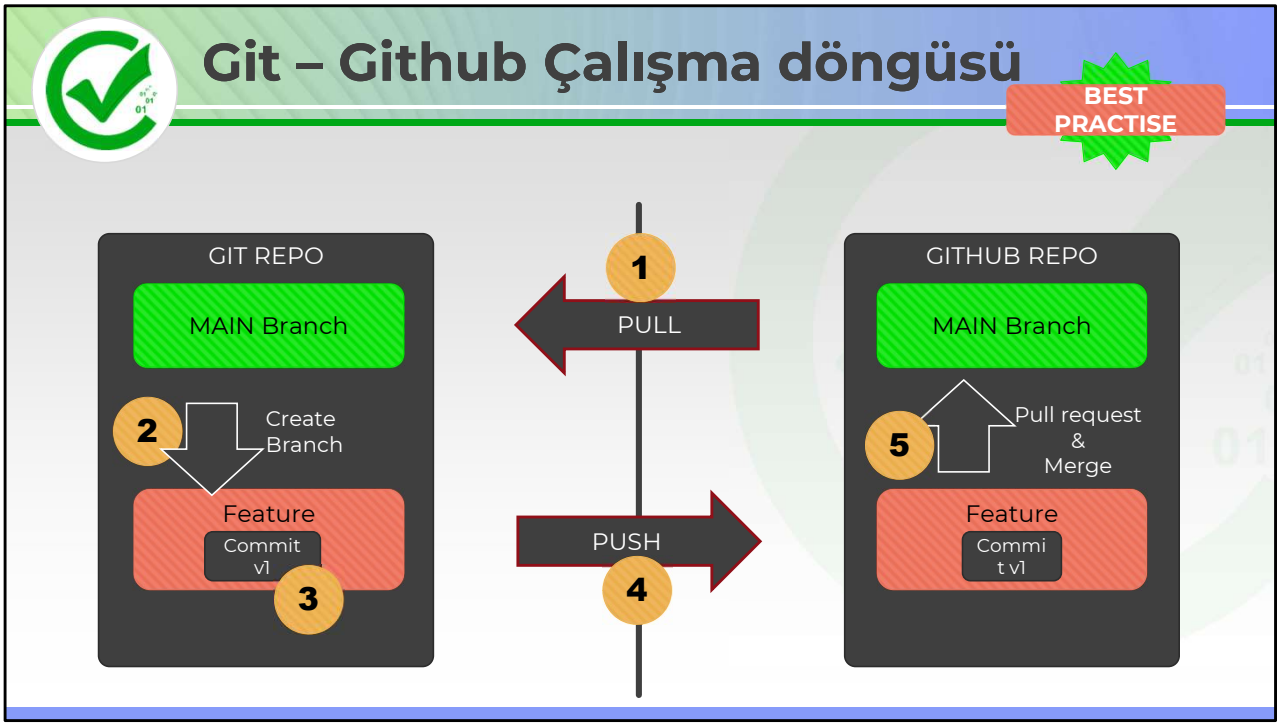
Birleştirilecek commitlerde, aynı dosyanın aynı satırında birbirinden farklı değişiklikler varsa bu durumda merge işlemi sırasında çakışma oluşur. Buna **merge conflict** denir. Merge conflict, remote-local birleştirmelerinde veya branch birleştirmelerinde gerçekleşebilir.



Bu durumda Gtihub aynı dosyada ve aynı satırda farklı kullanıcılar tarafından yapılmış değişikliklerde hangisini alacağına kendi karar veremeyeceği için conflict oluşturur ve merge işlemini yapan kişiye bu dosyayı düzenlemesi gerektiğini belirtir.

Merge conflict olduğunda neler yapılabilir ?

1. Çakışmanın oluştuğu dosya açılır ve manuel olarak çakışma giderilir. Dosya kaydedilir ve ardından tekrar bir commit oluşturulur.
   git commit –m "conflict çözüldü"
2. Ya da birleştirme işleminden vazgeçilebilir
   git merge –abort



Bu modelde Local Master da güncelleme yapılmaz. Local Master sürekli remote master ile beslenerek diğer developer ların ne gibi güncellemeler yaptıkları izlenerek local branch üzerinde oluşabilecek conflict ler önlenebilir.
Birden fazla colloborator ile çalışılan repo larda feature branch i push yapmadan önce mutlaka master pull yapılmalı ve olabilecek conflict ler düzeltilmelidir.