

**BATCHS: 30-31-32-33-34**

**DAY: 3**

**Git - Github**





# Command Prompt Terminal

Command prompt veya terminal için  
komutlar



# Komutlar

	Windows	Mac
Uygulama adı	Komut İstemi	Terminal
Konum değiştirme	cd <i>klasör_ismi</i> : klasöre girilir cd .. : bir üst klasöre geri döner cd / : en üst klasöre döner	
Listeleme	dir	ls
Klasör Oluşturma	mkdir	
Klasör Silme	rmdir	
Dosya oluşturma	echo merhaba > dosya.txt	
Dosyanın içeriğini görme	more	cat
Dosya silme	del	rm
Klasör ve dosya ismi değiştirme	ren	mv
Ekran temileme	cls	clear

- Versiyon kontrol sistemi

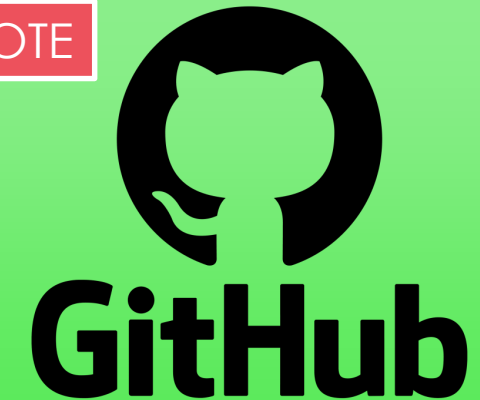
LOCALE



# git

- Git ile yönetilen repoların public veya private olarak **saklandığı** veya **paylaşıldığı** uzak sunucu
- Birden fazla kişi ile işbirliği içinde çalışma imkanı

REMOTE





Version Control System



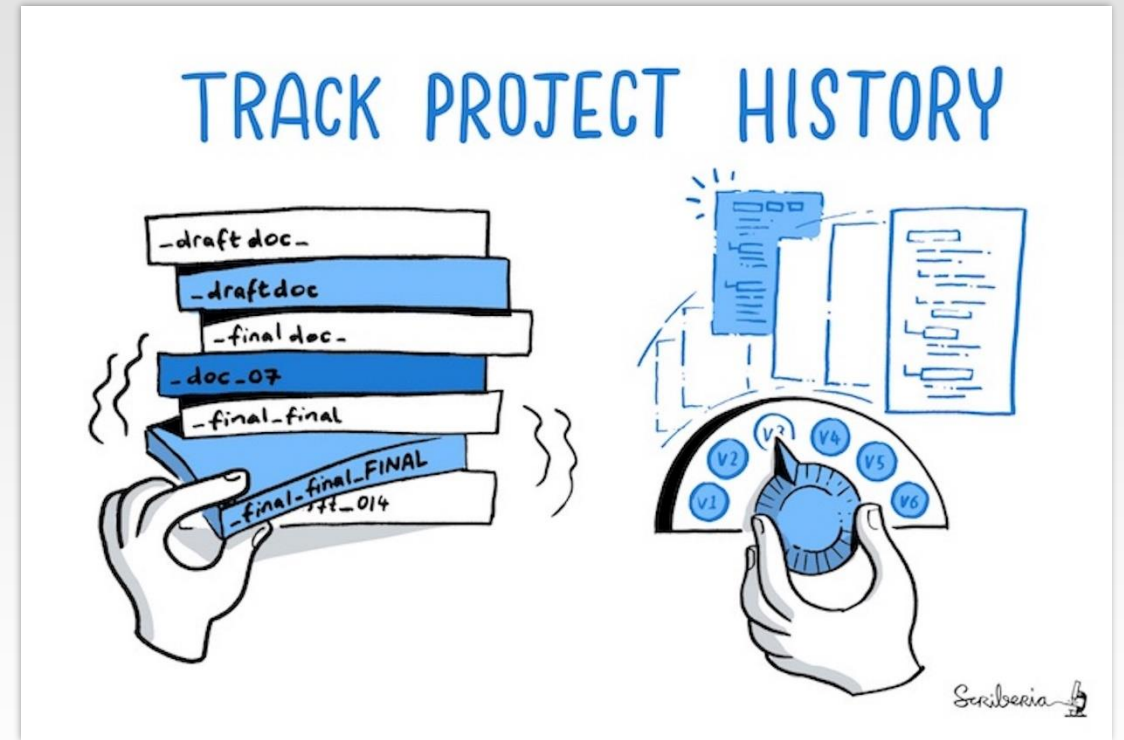
# Versiyon Kontrol Sistemi

- VKS, bir uygulamada belli değişikliklerden sonra, o ana kadar ortaya çıkan ürün ile ilgili bir versiyon oluşturulması, yeni değişikliklerin ayrı bir versiyona konulması işlemidir.
- Çoğu insanın versiyon kontrol metodu, ilgili dosyaları başka bir yere kopyalamaktır.
- Bu yaklaşım basit olduğundan çok yaygındır fakat aynı zamanda inanılmaz derecede hataya açık bir yaklaşımdır.
- Hangi dizinde bulunduğunuzu unutmak, yanlışlıkla yanlış dosya üzerine yazmak veya istemediğiniz dosyaların üzerine yazmak gibi ihtimallerin gerçekleşmesi çok olasıdır. Ve çok fazla disk tüketimine sebep olur.



# VKS nedir

Versiyon kontrol sistemi, belirli versiyonların daha sonra çağrılabilmesi için zaman içerisinde bir dosya veya dosya grubundaki değişiklikleri kaydeden bir sistemdir.

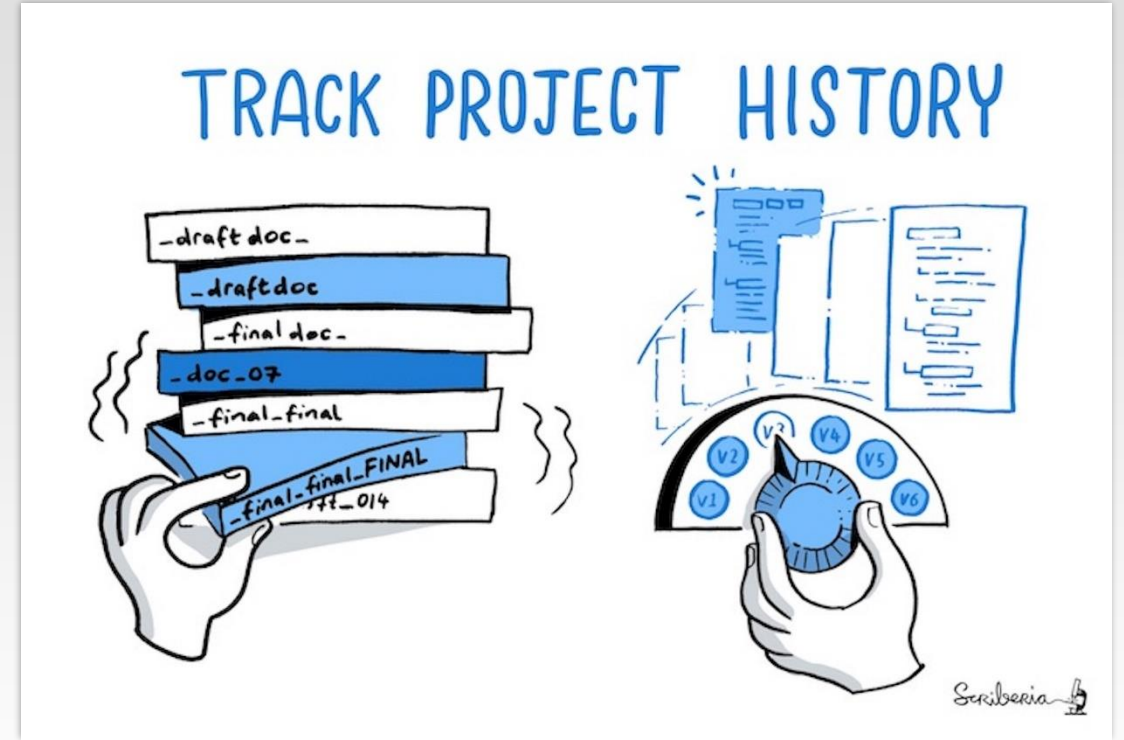




# VKS nedir

Versiyon Kontrol Sistemi, seçili dosyaların bir önceki versiyona döndürülmesi, projenin tamamının bir önceki versiyona döndürülmesi, zaman içerisinde yapılan değişikliklerin karşılaştırılması, probleme neden olabilecek değişikliklerin en son kimin tarafından yapıldığı gibi bir çok işlemin gerçekleştirilebilmesini sağlar.

Genel olarak VKS kullanmak, değişiklik yaptığınız dosyalar üzerinde bir şeyleri berbat ettiğinizde ya da bir şeyleri kaybettiğinizde kolayca geri getirebilmeniz anlamına gelmektedir.





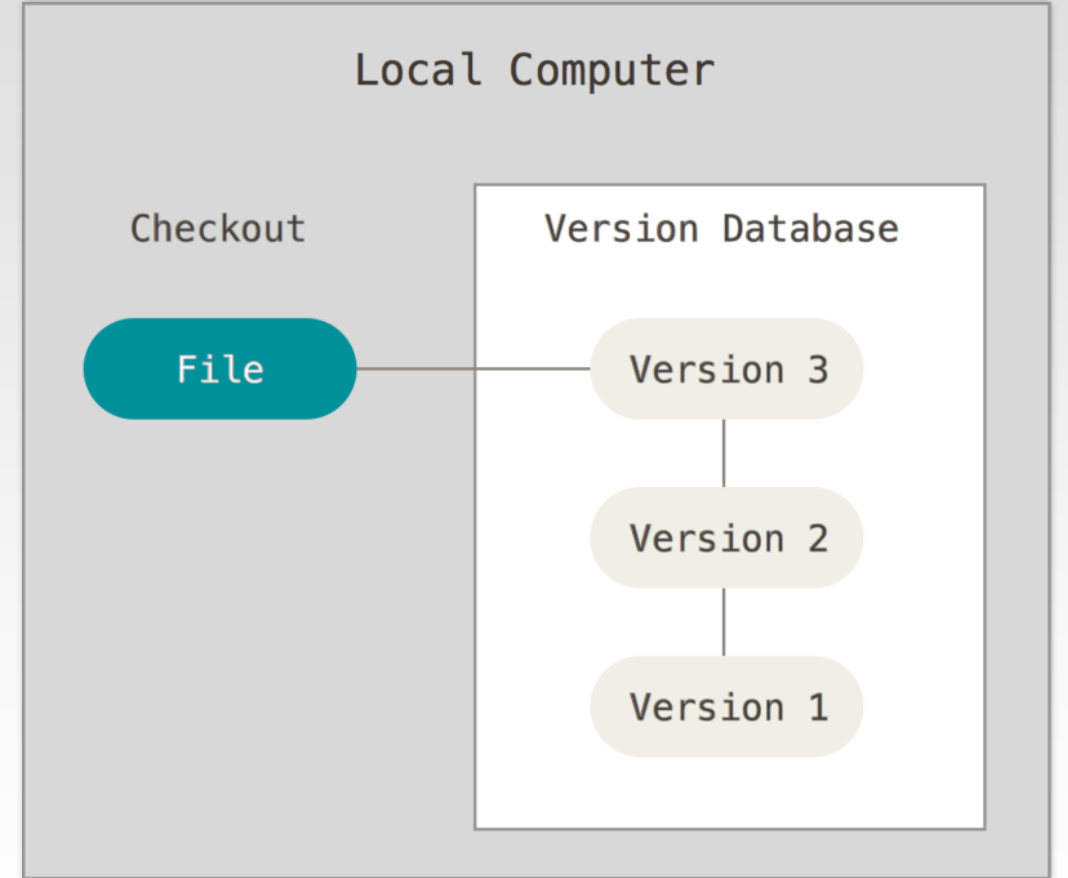


# Yerel VKS nedir

YVKS, versiyon kontrol sisteminin lokal bilgisayarda tutulduđu 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 üretmez.

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

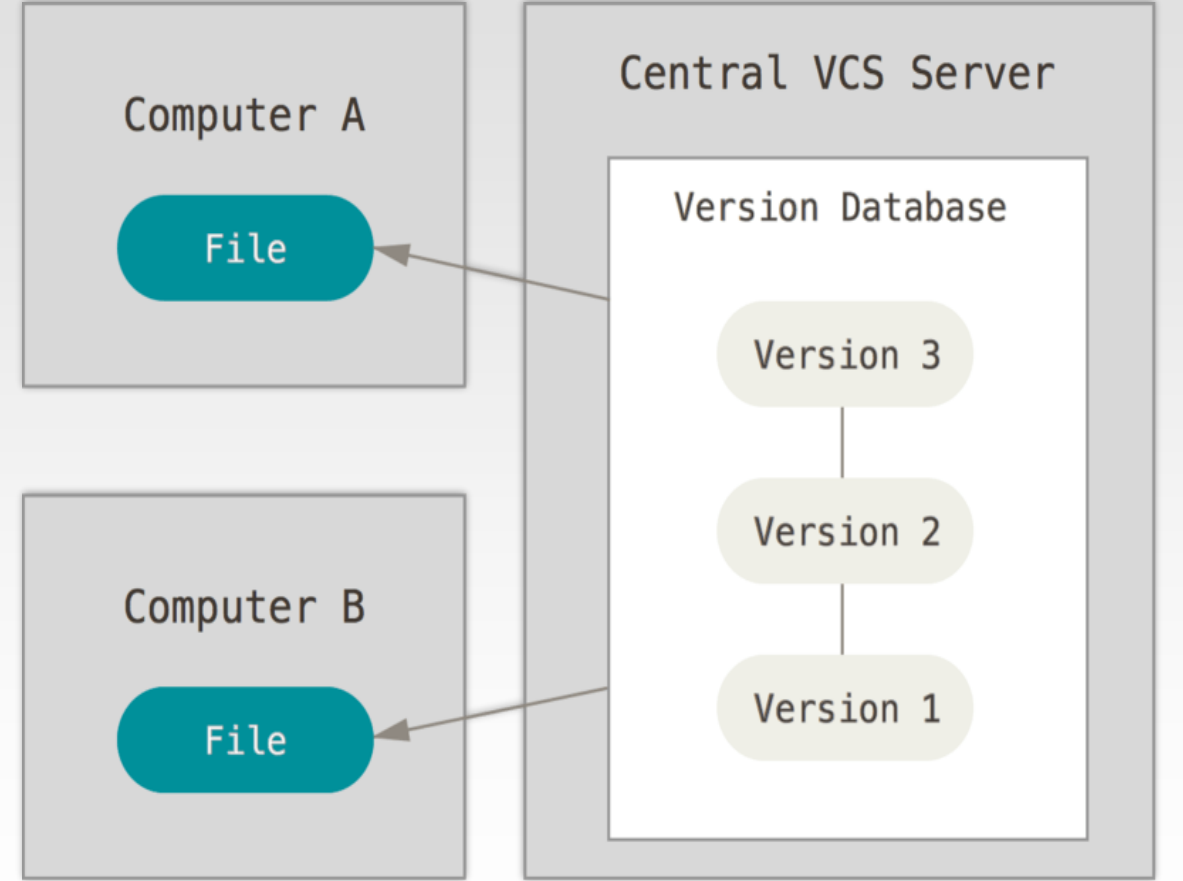




# Merkezi VKS nedir

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.

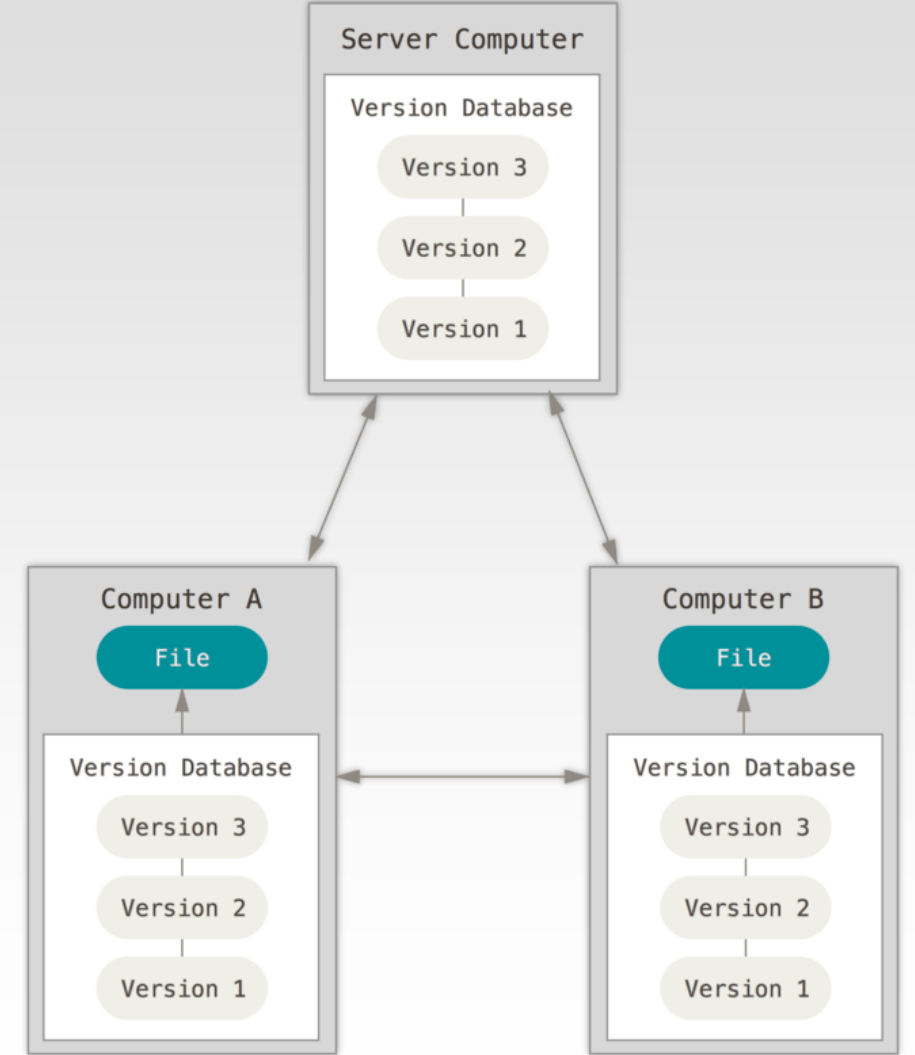




# Dağıtık VKS nedir

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.





# Kurulum ve İlk Ayarlar

- Git altyapısını oluşturmak ve git komutlarını kullanabilmek için Git kütüphanesinin kurulması gerekmektedir  
[<https://git-scm.com/downloads>]

*git -verison // git in kurulup kurulmadığını anlamak için*

- Git configuration

*git config --global color.ui true*

- 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



Repository

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



# Local repo oluşturma

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

**git init**



# 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.

**REPOSITORY**



# Local Versiyon Oluřturma

Working Space veya Staging area' nın durumunu görmek için kullanılır.

**git status**

Oluřturulan versiyonları görmek için bu komut kullanılır

**git log**

Working Space

Deęiřikliklerin Stage' e gönderilmesi

Versiyon oluřturma

**git add**

**git commit**

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

`git add dosya_adi`  
`git add .`

`git commit -m "ilk versiyon"`





# Versiyon detaylarını görme

```
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
```

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

git show *[hash kodun ilk 7 karakteri]*



# Versiyon oluşturmak için kodlar

## Ana komutlar

`git init`

(Repoda değişiklik yapılır)

`git add .`

`git commit -m "versiyon metni"`

Repo oluşturur. Her projede en başta bir kere kullanılır.

Dosyaları staging area ya gönderir

Versiyon oluşturur

## Yardımcı komutlar

`git status`

`git log`

`git show [hash_kodu]`

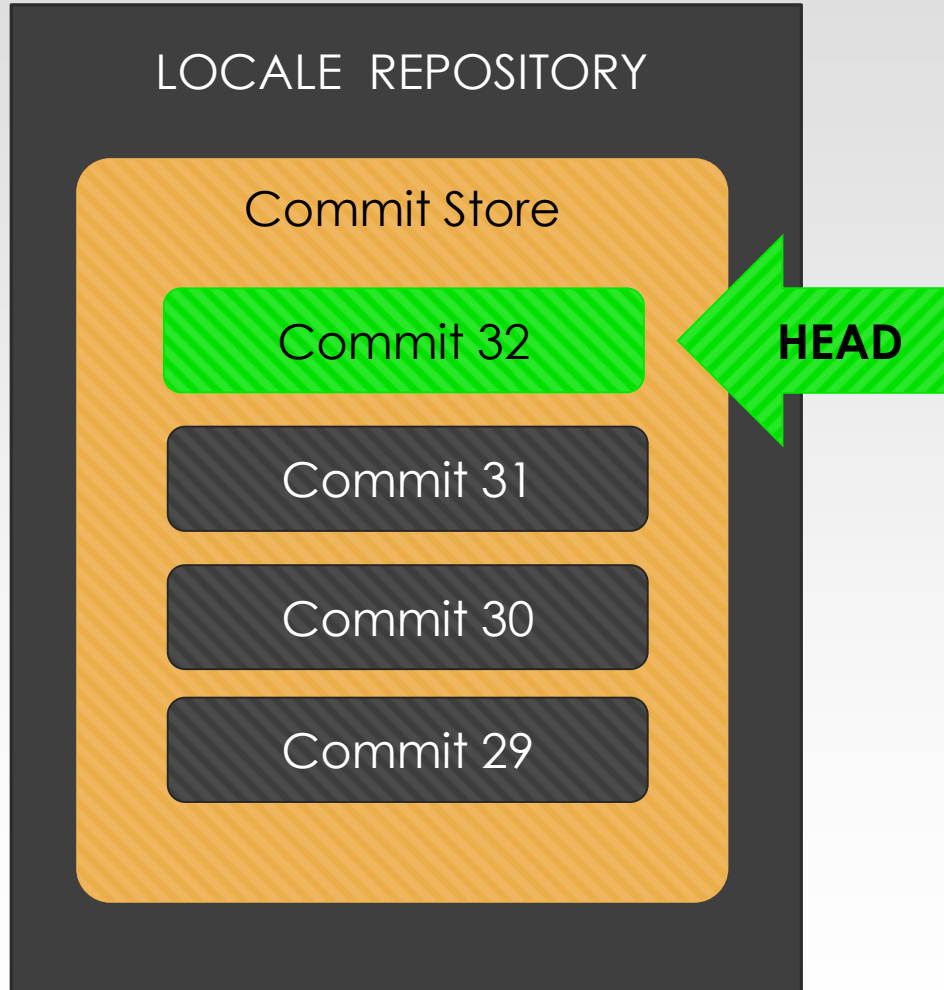
Genel durum ile ilgili bilgi verir

Versiyonların listesini verir

Versiyondaki değişiklikleri gösterir



# Commit Store



- 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
```



# Değişiklikleri geri almak

Working  
space

git restore *[dosya]*

Tek bir dosyayı iptal eder

git restore .

Tüm dosyaları iptal eder

git reset --hard

Working space deki değişiklikleri iptal eder, staging area yı boşaltır.

Stage

git restore --staged *[dosya]*

Tek bir dosyayı iptal eder

git restore --staged .

Tüm dosyaları iptal eder

Commits

git checkout *[hash]* *[dosya]*

Dosya,hash ile belirtilen versiyona döner

git checkout *[hash]* .

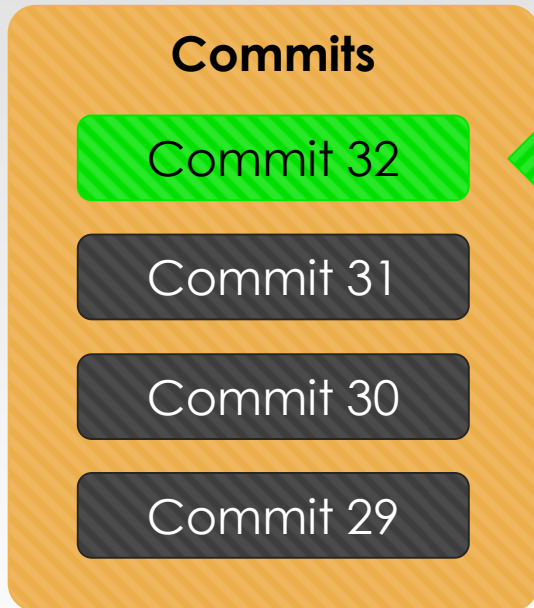
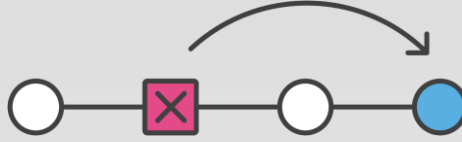
Hash değeri verilen versiyona döner

Git checkout lardan sonra değişikliklerin commit haline gelmesi için git add ve commit komutları unutulmamalıdır

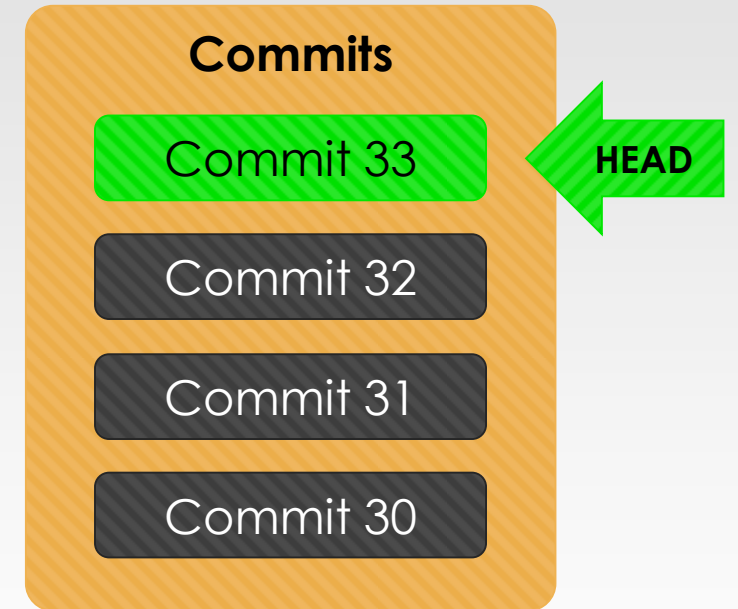


# Önceki versiyonlara dönmek

## 1.Yöntem: CHECKOUT



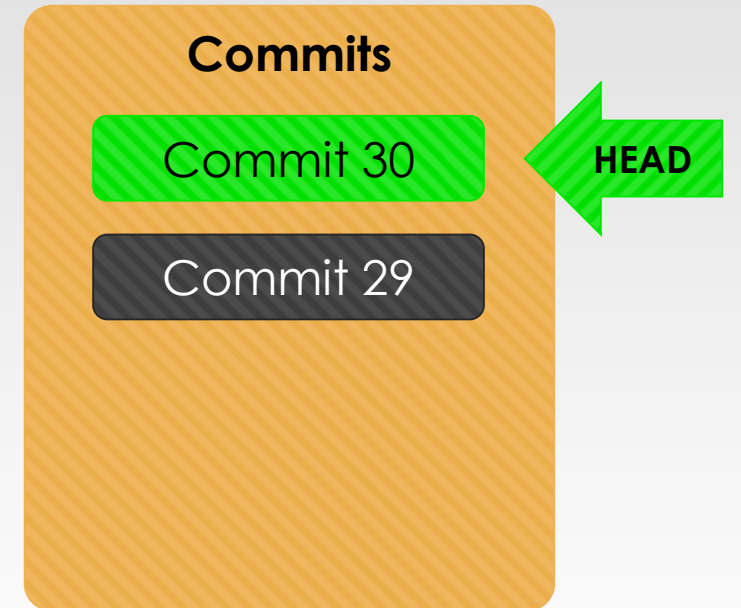
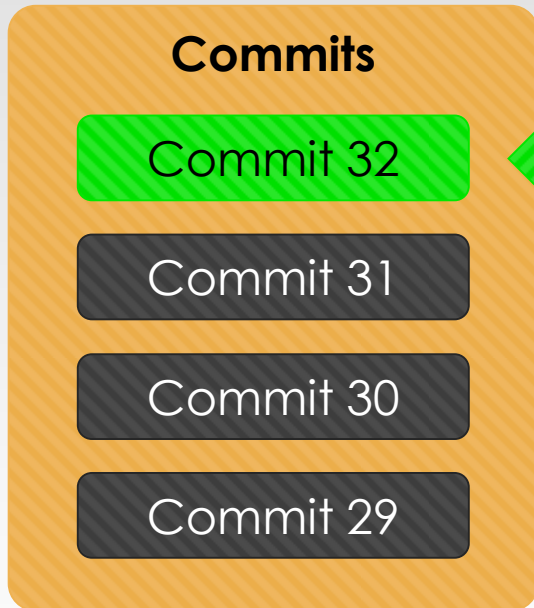
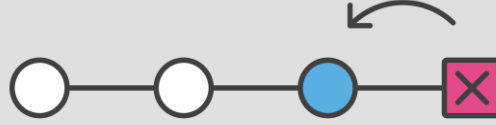
İstenilen versiyonu geri alır. Ancak bunun için sadece HEAD hareket ettirilir. Yapılan değişiklikler silinmez. Bu işlemin de bir versiyon haline gelmesi için commit oluşturmak gerekir.





# Önceki versiyonlara dönmek

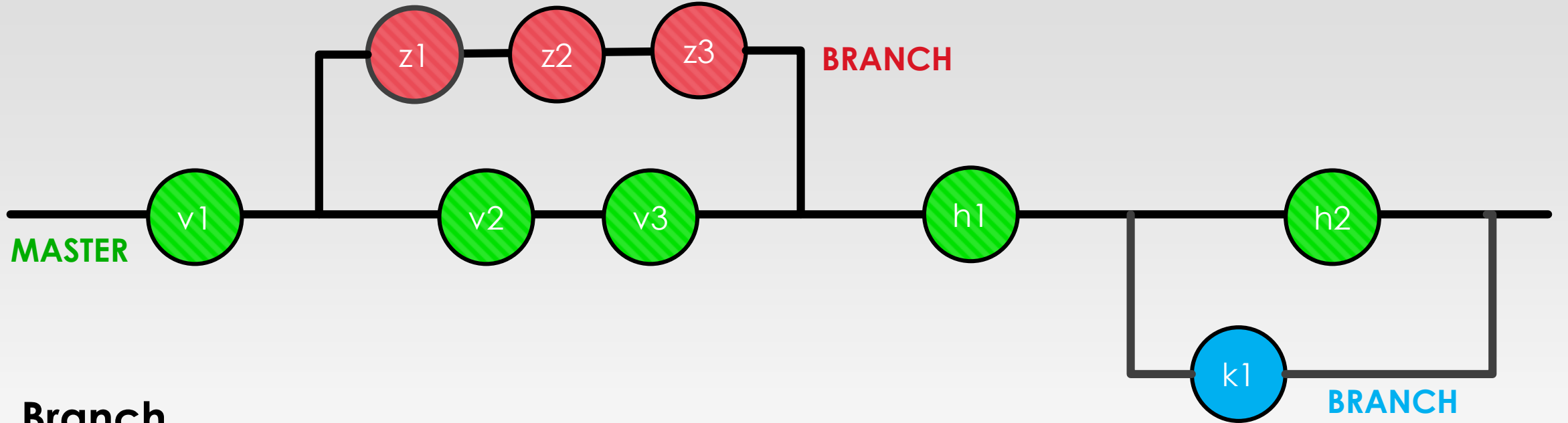
2.Yöntem: RESET



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



# Branch (Dal)

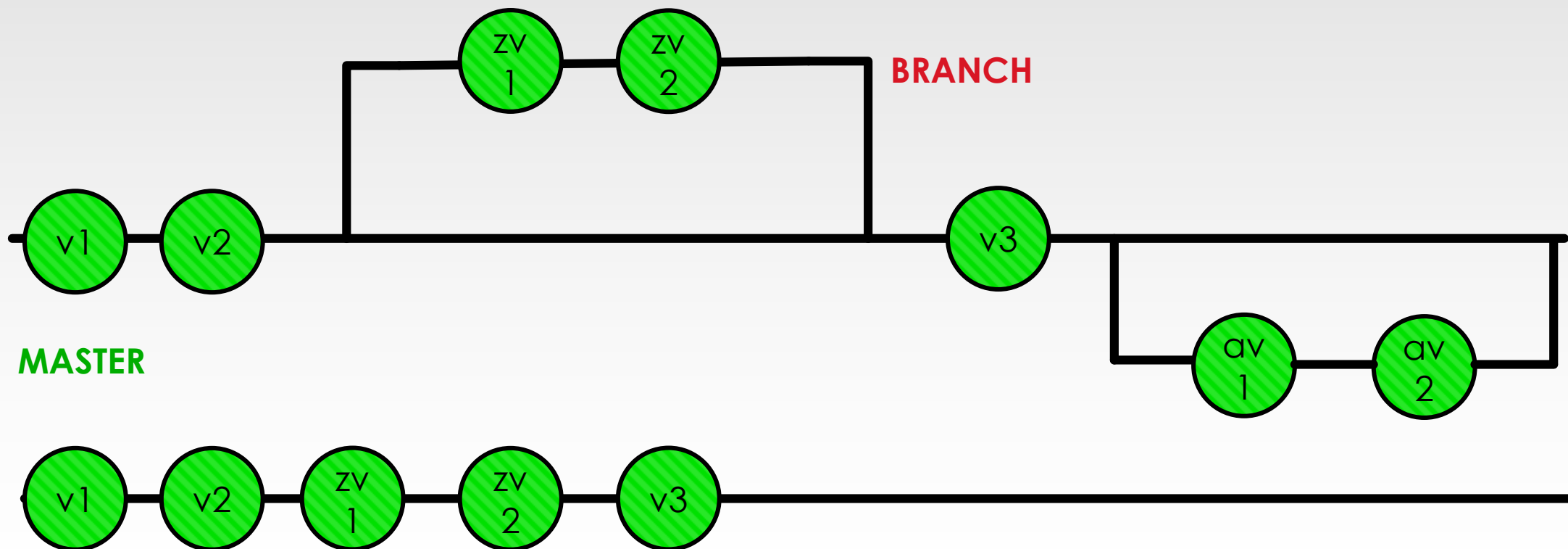


## Branch

Master branch, projemizin ana yapısıdır. Zaman zaman bu ana yapıyı bozmadan bazı denemeler yapmak ve gerekirse kolaylıkla bu denemeleri iptal etmek ya da master ile birleştirmek için branch ler kullanılır. Branch ler içindeki değişiklikler master dan bağımsız olarak saklanır.



# Branch (Dal)







# Branch Komutları

`git branch [isim]`

Yeni branch oluşturur

`git branch`

Mevcut branch leri listeler

`git checkout [isim]`

Branch aktif hale gelir

`git merge [isim]`

İki branch i birleştirir

`git branch -d [isim]`

Branch i siler

`git branch -m [isim]`

Branch ismini değiştirir.

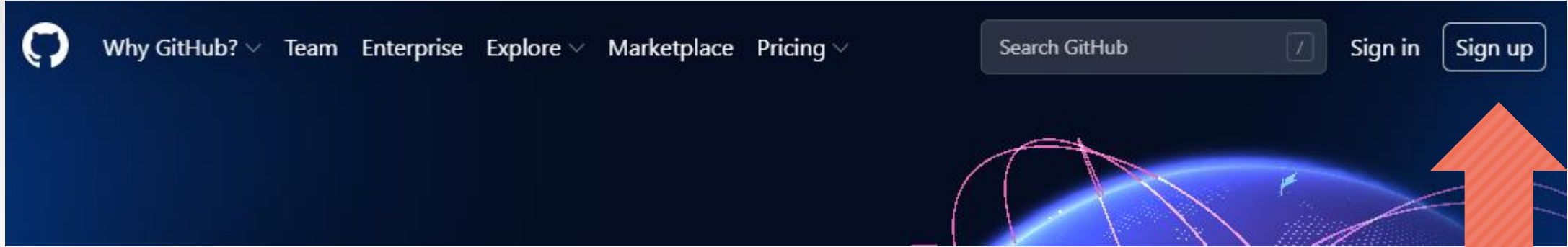




# Github hesabı oluşturma



Github.com





# 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



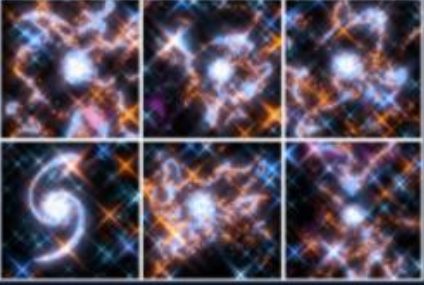
Verify your account

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 oluşturma



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

→ Enter code

Eposta adresine gönderilen  
kodu girerek işlemi  
tamamlıyoruz



# Github repo oluşturma



Pull requests Issues Marketplace Explore

Overview Repositories 15 Projects Packages

Find a repository... Type Language Sort New

1

New butonuna basılır

## 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 \*

techproeducation1 /

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

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.

Create repository

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 işlemi tek başına yapar





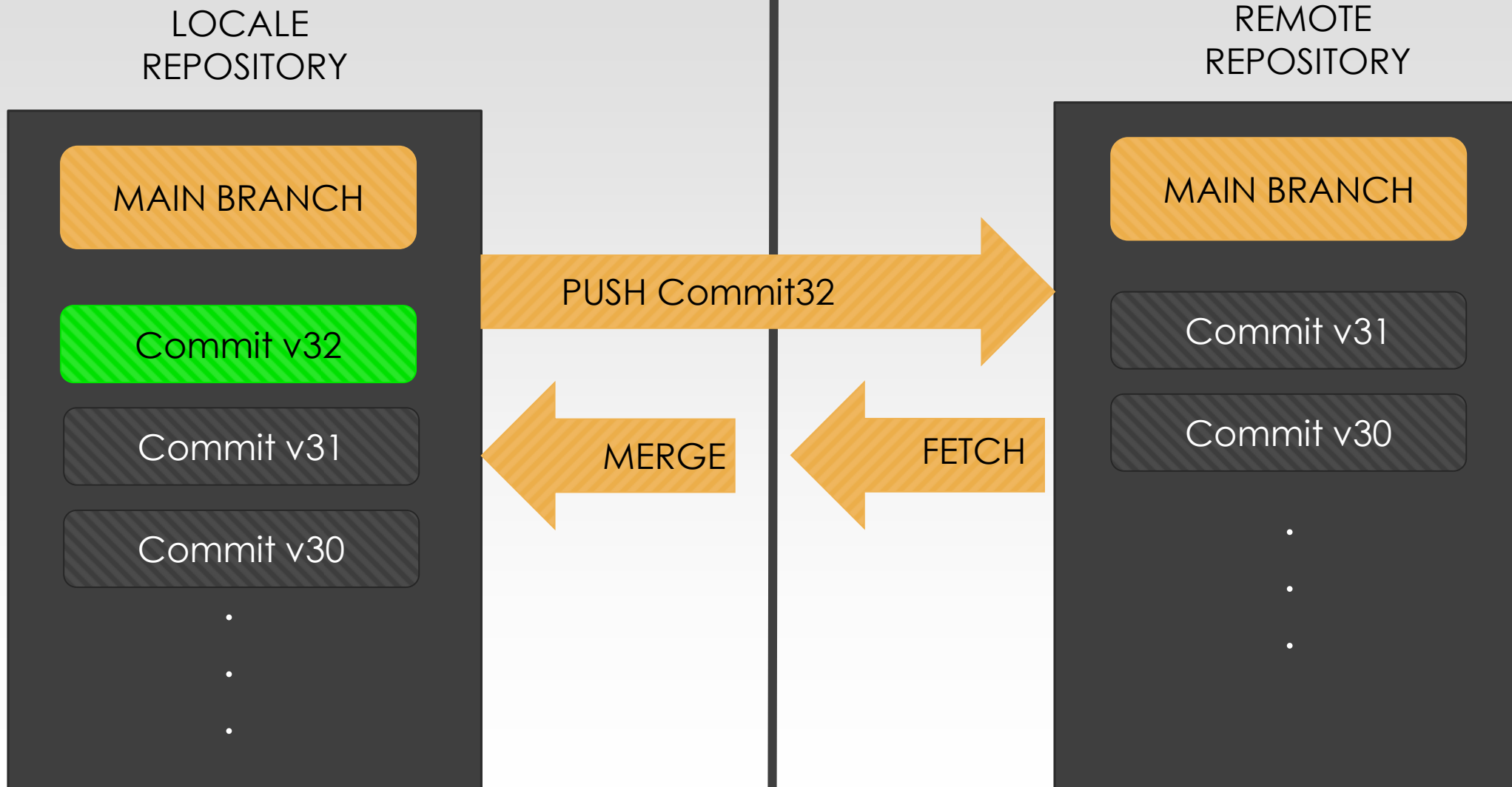
# Cloning

Github daki bir reponun lokale indirilerek geliştirilmesi için öncelikle klonlanmalıdır. Bunun için aşağıdaki komut kullanılır.

```
git clone [git url]
```



# Github Çalışma Prensibi





# Local Repo nun Github a Yükleneceği



Localde en az 1 commit oluşturduktan sonra, her proje için bir kereye mahsus olmak üzere aşağıdaki komutlar kullanılır. Böylece local repo ile github ilişkilendirilmiş olur.

1

`git remote add origin [REMOTE_URL]`

Remote url, github sitesi üzerinden ilgili repo sayfasında code bölümünden elde edilir

2

`git push -u origin master`

Local master branch, remote repo ya gönderilmiş olur.



## ***Git push origin master***

komutu 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.



# Local Repo nun Github a Yükleneesi



İlk yüklemeden sonra yapılacak yeni local commit leri github a göndermek için aşağıdaki komutun kullanılması yeterlidir.

```
git push
```



# Github dan commit çekme



Github üzerinden bir commit ile local repo güncellenmek istenirse aşağıdaki komutlar kullanılır

`git fetch`

Değişiklikleri remote'dan local'e indirir

VEYA

`git pull`

fetch & merge

`git merge`

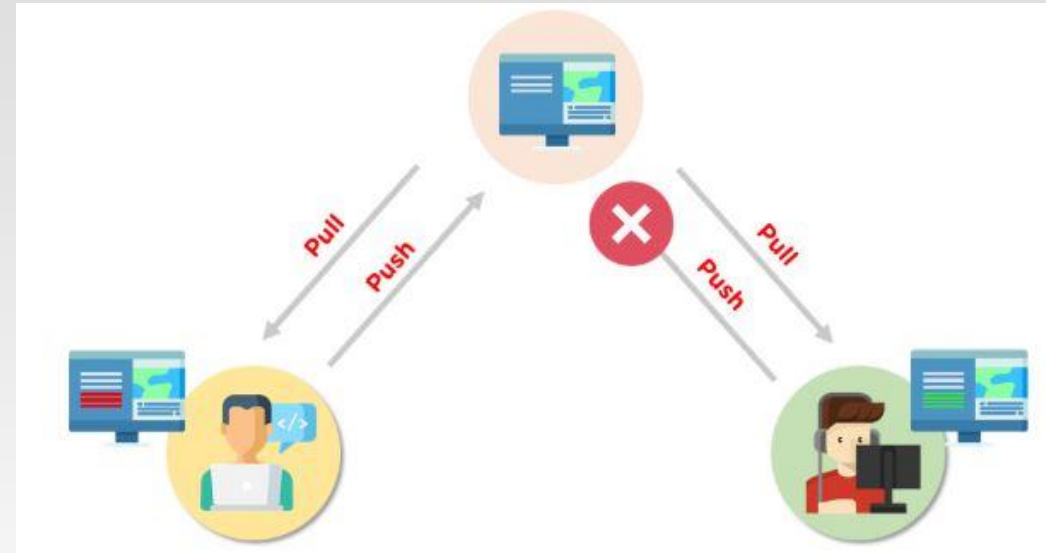
İndirilen değişiklikleri local repoya uygular



# Merge Conflict

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.



```
$ cat merge.txt
<<<<<<< HEAD
this is some content to mess with
content to append
=====
totally different content to merge later
>>>>>>> new_branch_to_merge_later
```



# Merge Conflict

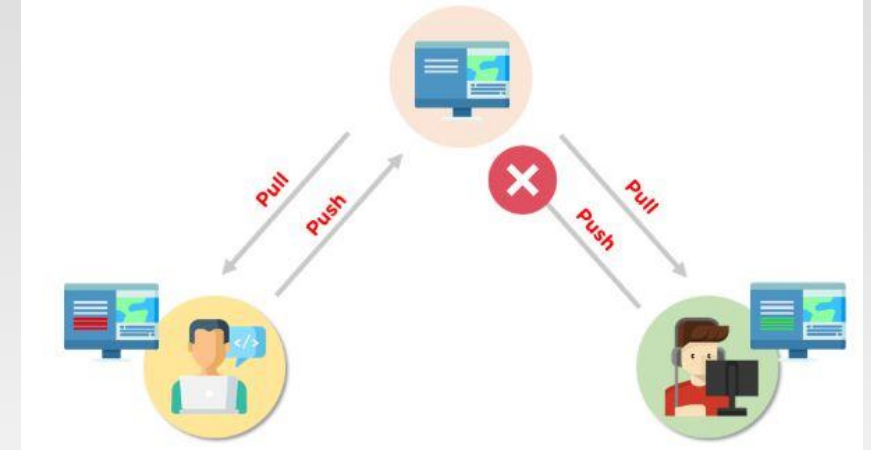
Merge conflict olduğunda neler yapılabilir ?

1. Çakışmanın olduğu 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

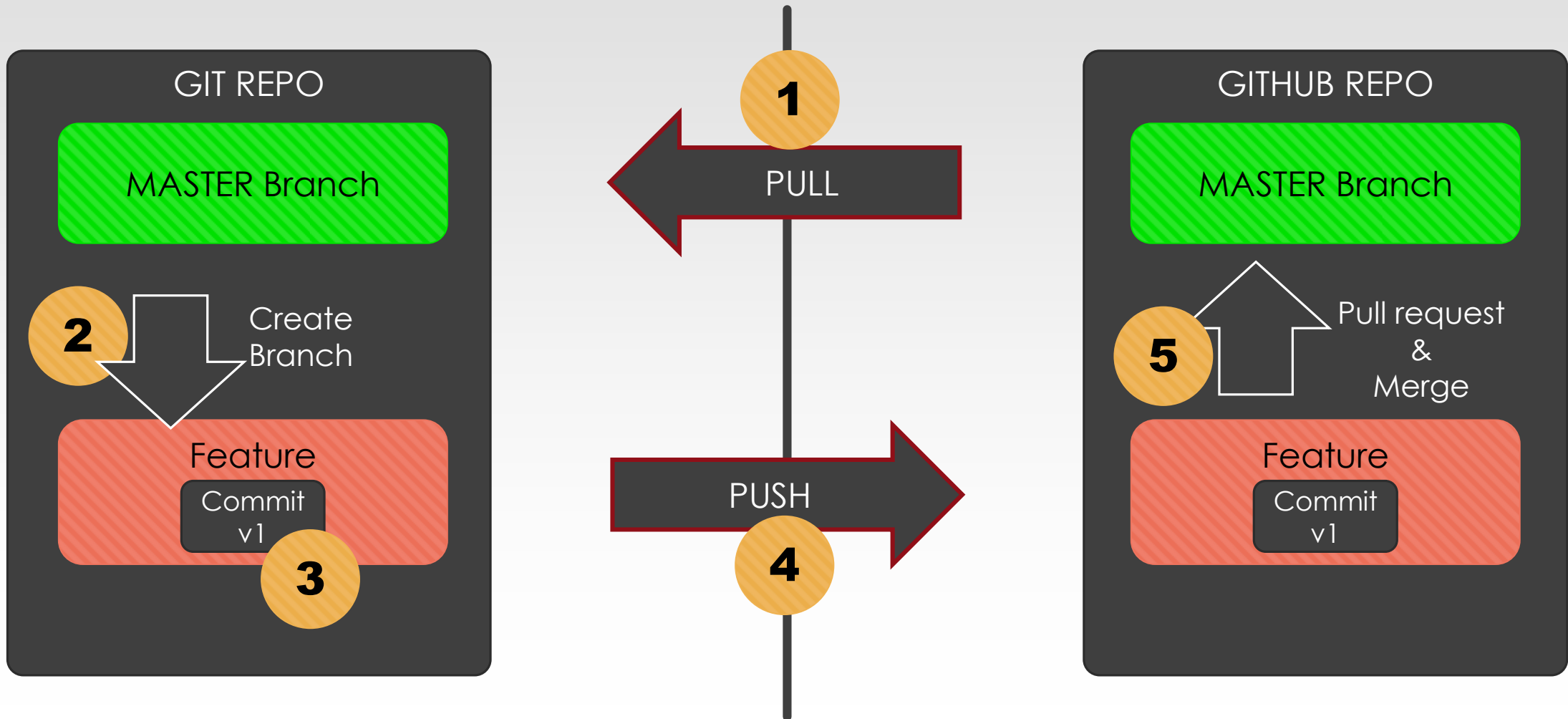
`git merge -abort`





# Git – Github Çalışma döngüsü

BEST PRACTISE

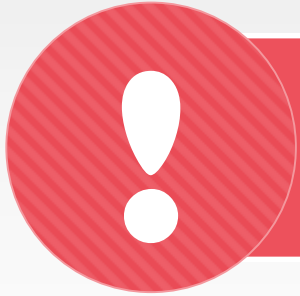




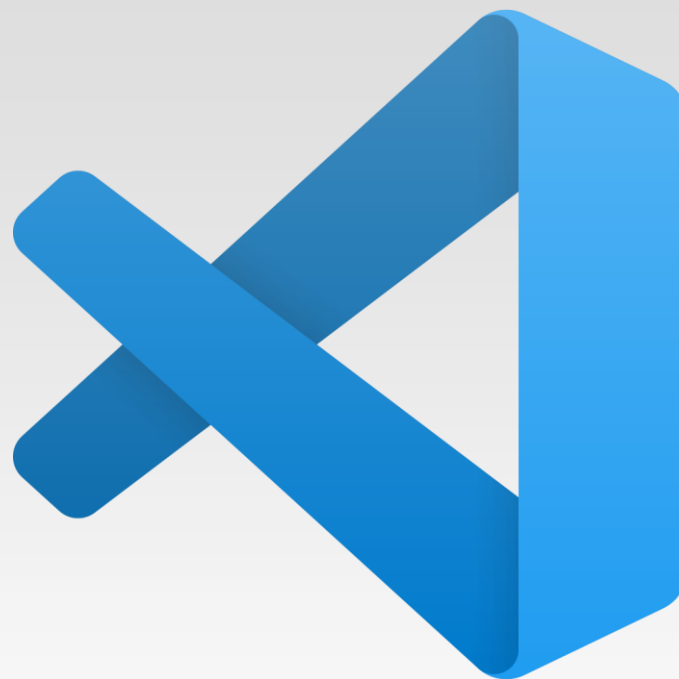


# Git – Github Çalışma döngüsü

- 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 önlenabilir.



Birden fazla colloborator ile çalışılan repo larda feature branch i push yapmadan önce mutlaka master pull yapılmalı ve olabilecek conflict ler düzeltilmelidir.





# VS Code & Git-Github

## VS Code' da Git in Aktif Hale Getirilmesi

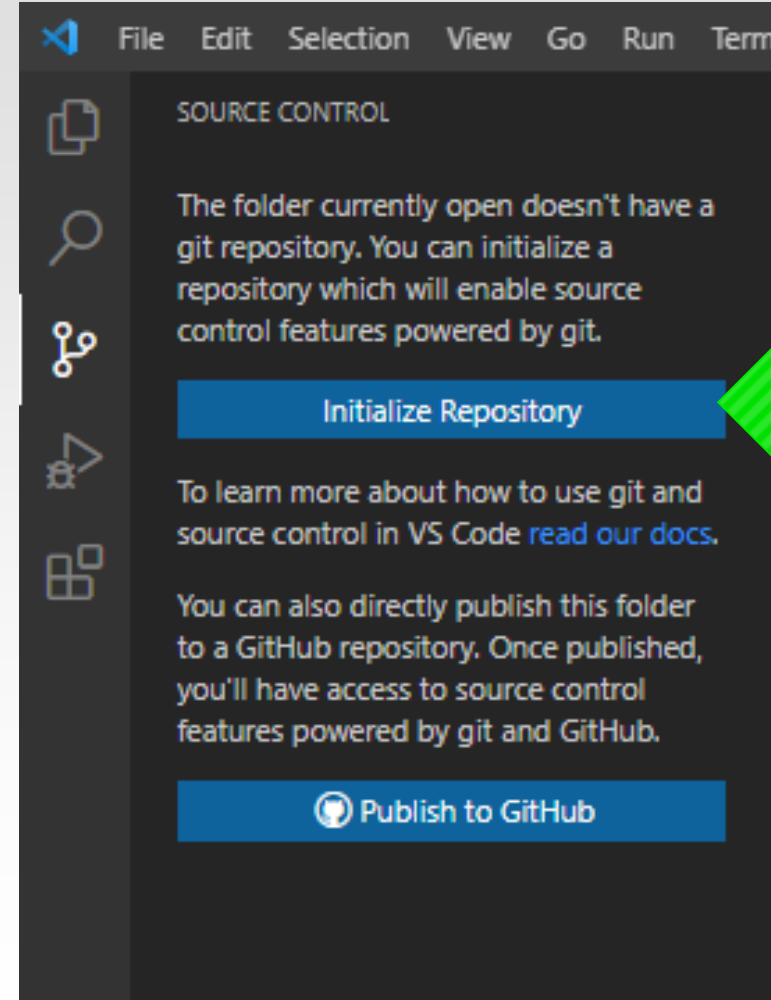
- 1- Ayarlar simgesine tıklanır
- 2- Settings seçilir
- 3- Arama bölümüne «git enabled» yazılır
- 4- Alt tarafa gelen ayarlardaki Git Enabled kutusu işaretlenir.

The image shows two screenshots of the VS Code interface. The left screenshot shows the Command Palette with 'Settings' selected, indicated by a green arrow and a red circle with the number 2. A green arrow and a red circle with the number 1 point to the gear icon in the bottom left corner. The right screenshot shows the 'git enabled' search results in the Settings window, indicated by a green arrow and a red circle with the number 3. A green arrow and a red circle with the number 4 point to the 'Git: Enabled' checkbox, which is checked.



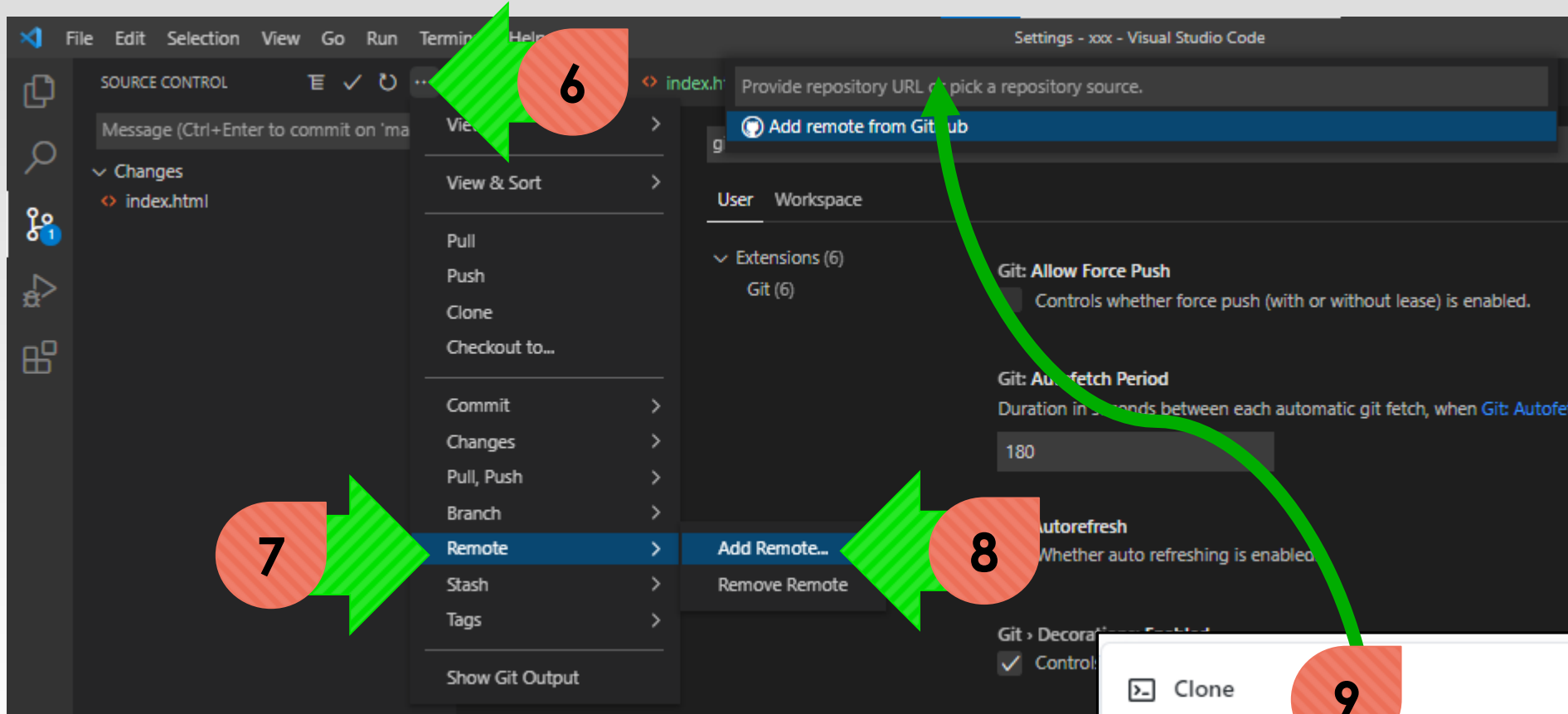
# VS Code & Git-Github

5- Bir proje VS Code ile açıldıktan sonra Initialize Repository ile repository oluşturulur. Burada VS Code **git init** komutunu çalıştırır. Bu aşamada local version kontrol sistemi kurulmuş olur.



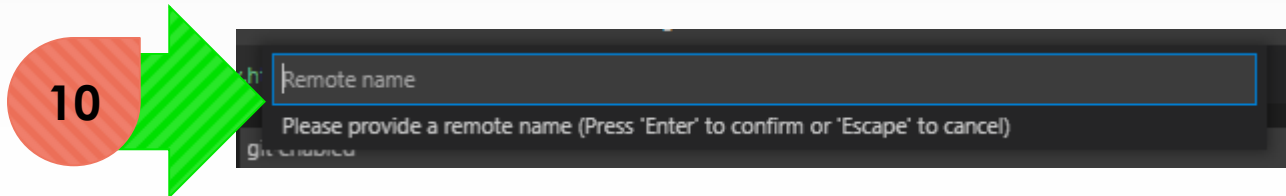
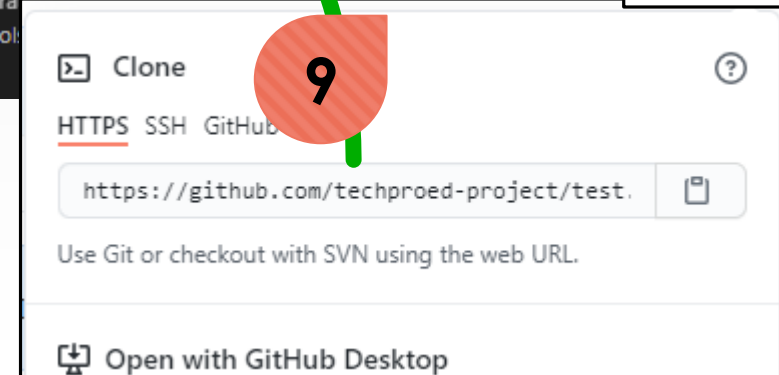


# VS Code & Git-Github



## Github bağlantısı

- 6- Üç noktaya basılarak menü açılır
- 7- Remote seçilir
- 8- Add remote seçilir.
- 9- Açılan kutuya github hesabından alınan repo url adresi yapıştırılır
- 10- Reponun ismi yazılır ve enter a basılır

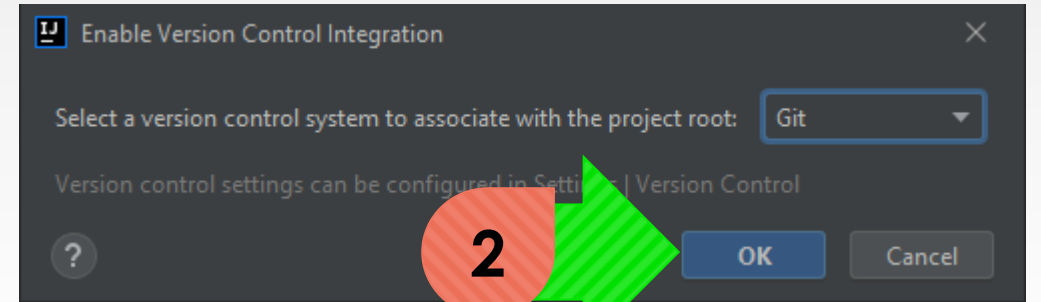
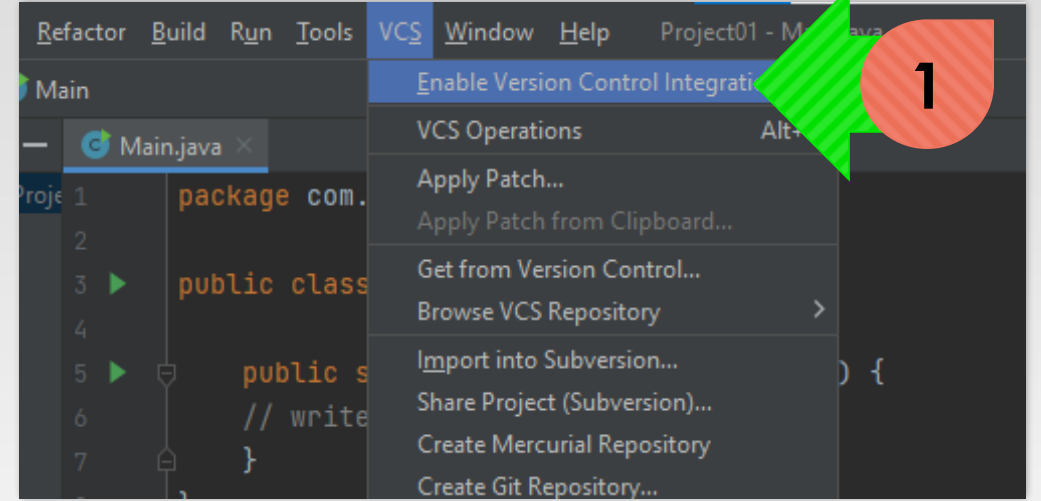




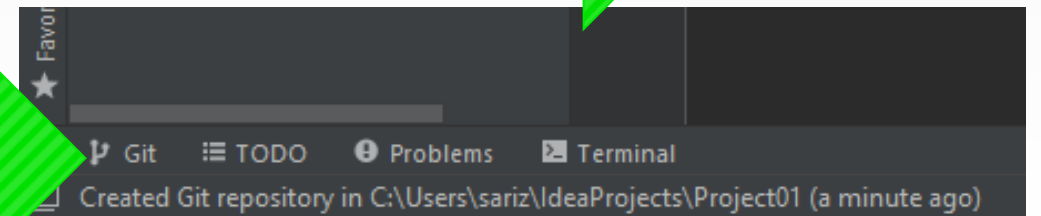


# Intellj ile Git Github

- 1- IntelliJ üzerinde VCS kullanmak için VCS menüsündeki «Enable Version Control Integration» komutu seçilir.
- 2- Açılan pencerede Git seçilerek OK butonuna basılır
- 3- Böylece sol alt tarafta Git sekmesi ve üst tarafta Git menüsü görülecektir. (Bu aşamada git init çalıştırılmış oldu)



IntelliJ, üzerinde git ile ilgili oluşabilecek sorunlarda File/Invalidate Caches komutu kullanılabilir





# Intellj ile Git Github

## Github bağlantısı

4- Git menüsüne tıklanır

5- Manage Remotes seçilir

6- Açılan pencerede + butonuna basılır

7- Github adresi yapıştırılır ve OK butonuna basılır

Ardından github hesabı için authorization işlemleri başlayacaktır.

