

Burak DİNAR

Salih Kerem GÜNDOĞAN

TahsinKARGİN

Git-Giriş

**Git, dağıtılmış bir sürüm kontrol sistemi olarak kullanılan güçlü ve popüler bir araçtır. Yazılım geliştirme sürecindeki dosyaları izlemek, değişiklikleri takip etmek ve farklı sürümleri yönetmek için kullanılır. Git, birden çok geliştiricinin aynı projede çalışmasını kolaylaştırır ve farklı değişiklikleri birleştirmek için etkili bir yol sağlar. Ayrıca, herhangi bir zamanda geçmişteki bir sürüme geri dönme veya hataları düzeltme gibi işlemleri de yapmanıza olanak tanır. Bu kurs, Git'in temellerini öğrenmek isteyen herkes için tasarlanmıştır. Kurs boyunca Git'in nasıl kullanılacağını, depoların nasıl oluşturulacağını, değişikliklerin nasıl kaydedileceğini ve işbirliği yapmanın nasıl gerçekleştirileceğini öğreneceksiniz. Ayrıca, dal ve birleştirme gibi gelişmiş Git kavramlarını da keşfedeceksiniz. Bu kurs, yazılım geliştirme sürecinde daha verimli ve düzenli bir çalışma deneyimi elde etmenize yardımcı olacak temel Git becerilerini edinmenizi sağlayacaktır.**

Git Kurulum

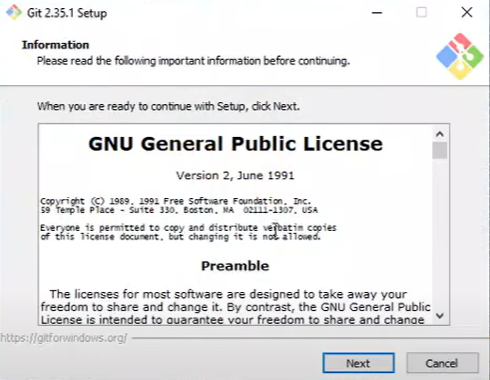
**Web tarayıcımıza git indir yazıyoruz ardından ilk siteye giriyoruz.**

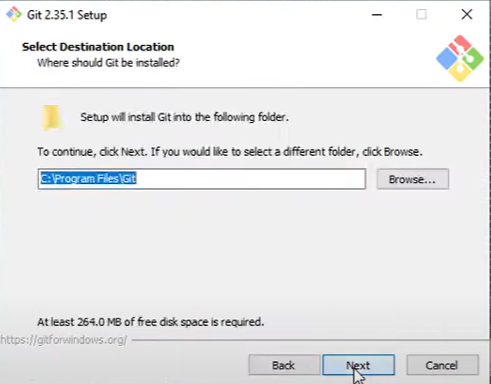
# 

**Girdiğimiz sitede işletim sistemimizi seçiyoruz.**

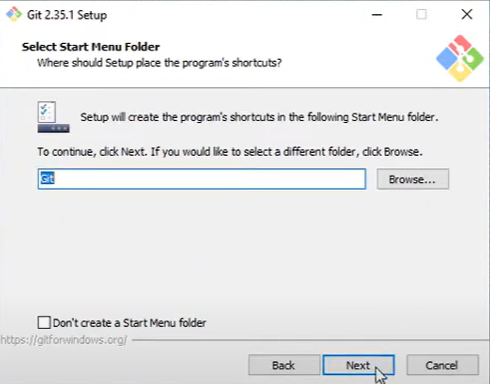
# 

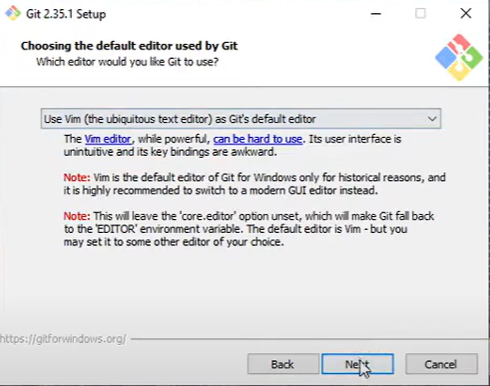
**İndirdikten sonra setup dosyasını çalıştırıyoruz.**

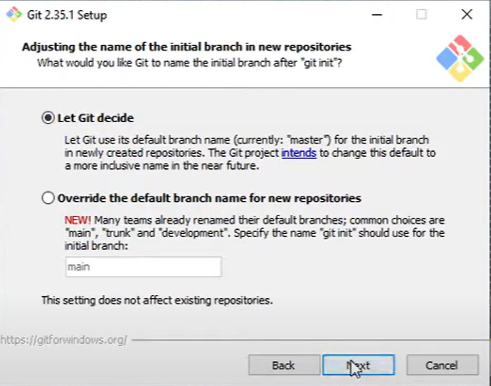


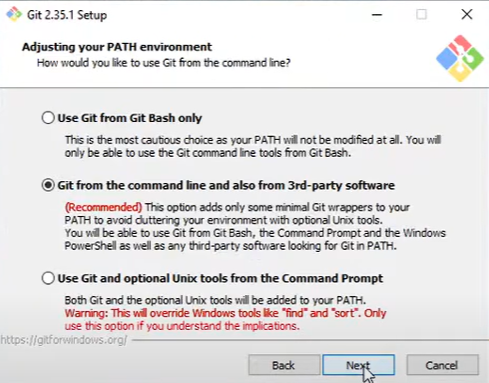


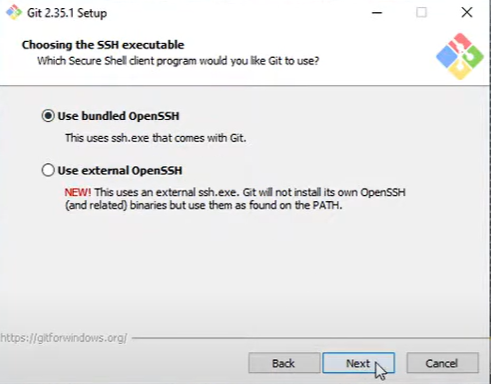
# 

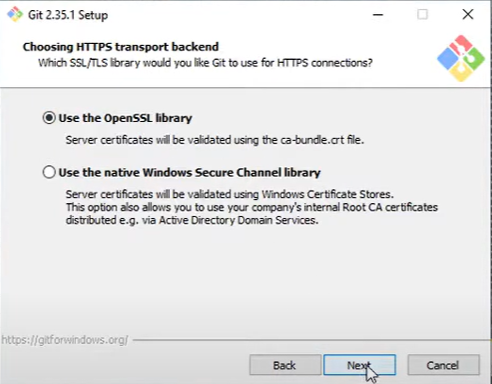


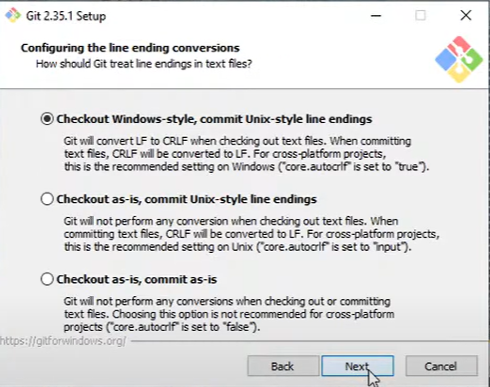


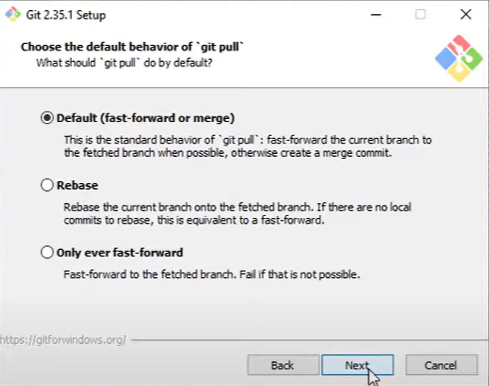


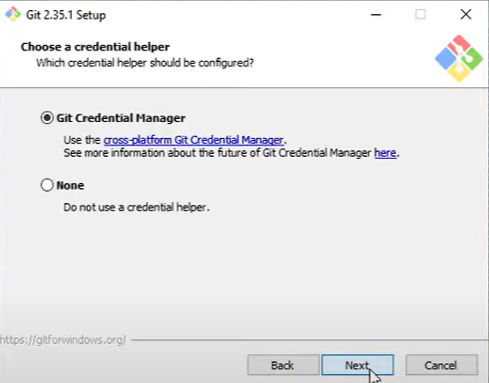


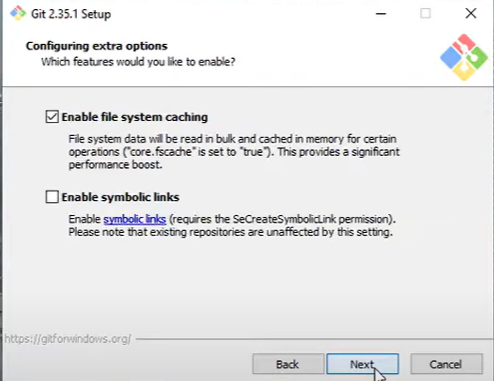


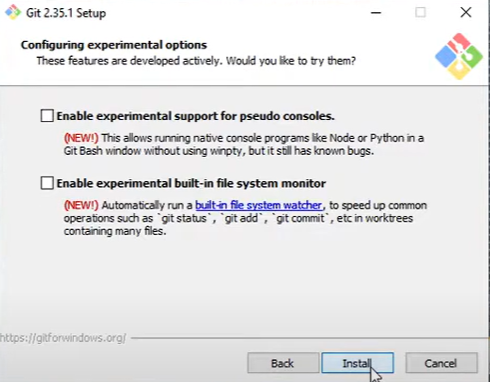


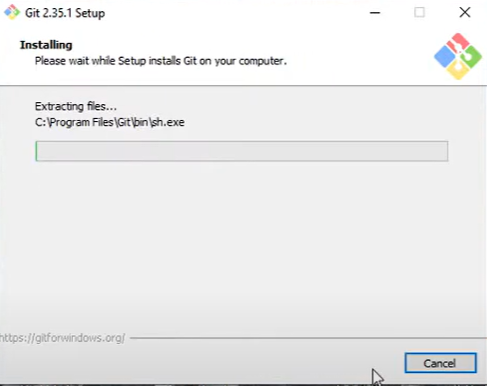


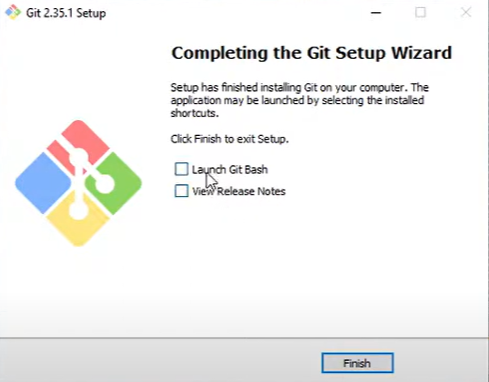












**Kurulumu bu adımları takip ederek bitiriyoruz.**

# Git Temelleri

## **“Git İnit”**

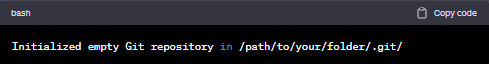
**‘git init’ komutu, oluşturulan klasörün Git’e tanımlanmasında yani Git deposu(repository) olarak kullanılmasını sağlamak amacıyla kullanılır.** **Bu komutu kullanarak, mevcut bir klasörü Git deposu olarak başlatabilirsiniz. Bu, klasörün içinde .git adında bir alt klasör oluşturur. .git klasörü, Git'in depo yönetimini gerçekleştirmek için gereken tüm verileri ve ayarları içerir.**

## **“Git İnit” nasıl kullanılır?**

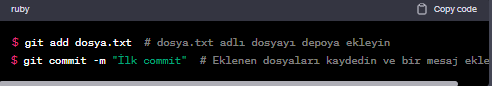
**1. Şimdi, git init komutunu kullanarak Git deposunu başlatalım:**



**2. Bu komutu çalıştırdığınızda, Git bir mesajla birlikte yeni bir Git deposu oluşturacak ve .git adında bir klasör oluşturacaktır. Örneğin:**

****

**Artık Git deposu oluşturuldu! .git klasörü, Git'in depo yönetimi için gerekli tüm verileri ve ayarları içerir. Bu noktadan itibaren, dosyalarınızı bu klasöre ekleyebilir ve Git'i kullanarak değişikliklerinizi takip edebilirsiniz.Örneğin, “git add” komutuyla dosyaları depoya ekleyebilir ve “git commit” komutuyla değişiklikleri kaydedebilirsiniz. Aşağıda bu adımların örneklerini görebilirsiniz:**

****

## 

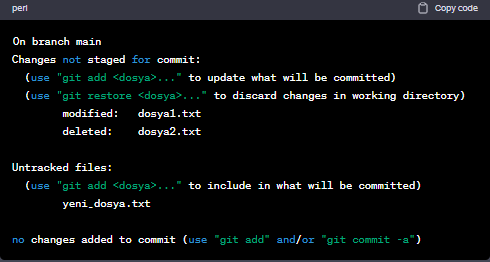
## **“Git Status”**

**‘git status’, Git deposundaki dosyaların durumunu gösteren bir komuttur. Bu komutu kullanarak, Git deposunda yapılan değişiklikleri, eklenen dosyaları veya çıkarılan dosyaları görebilirsiniz.**

**Bu komut ile aşağıdaki bilgileri öğrenebilirsiniz;**

* **Hangi dosyaların değiştirildiği veya yeni eklenmiş olduğu.**
* **Hangi dosyaların "staging area" denilen hazırlık alanında (git add komutuyla eklenmiş) olduğu.**
* **Hangi dosyaların takip edilmediği veya Git deposuna dahil edilmediği.**
* **Aktif olan branch (şube) ve commit (kayıt) bilgileri.**

**Aşağıda potansiyel bir ‘git status’ çıktısını görmektesiniz;**

****

**Yukarıdaki örnekte, dosya1.txt dosyasında yapılan değişikliklerin henüz git add komutuyla staging area'ya(genellikle Git klasörünüzde bulunan ve bir sonraki kayıt işlemine hangi değişikliklerin dahil olacağını tutan sade bir dosyadır.) eklenmediği ve dosya2.txt dosyasının ise silindiği görülmektedir. Ayrıca, yeni\_dosya.txt dosyası henüz Git tarafından takip edilmeyen (untracked) bir dosyadır.**

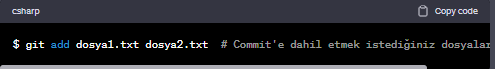
## **“Git Commit”**

**git commit, Git deposundaki değişiklikleri kalıcı olarak kaydetmek için kullanılan bir komuttur. Bu komutu kullanarak, yapılan değişiklikleri bir "commit" olarak adlandırılan bir kayda dönüştürebilirsiniz.**

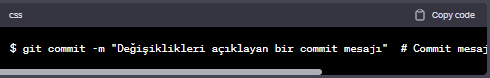
## **“Git Commit” nasıl kullanılır?**

**‘git commit’ komutunu kullanırken şu adımları izleyebilirsiniz;**

1. **Önce, ‘git add’ komutuyla değişiklikleri staging area'ya eklemeniz gerekmektedir. Staging area, commit'e dahil edilecek değişikliklerin bir ön hazırlık alanıdır. Dosyaları staging area'ya eklemek için ‘git add’ komutu kullanabilirsiniz:**

****

1. **Ardından, git commit komutunu kullanarak staging area'daki değişiklikleri kaydedebilirsiniz:**

****

**/\*Burada -m parametresi, commit mesajını belirtmek için kullanılır. Commit mesajı, yapılan değişiklikleri açıklamak için kullanılan bir metindir ve her commit için benzersiz bir mesaj olmalıdır. Mesaj, diğer geliştiricilerin ve sizin gelecekteki referanslarınızı kolaylaştırır.\*/**

1. **‘git commit’ komutunu çalıştırdığınızda, Git sizin için bir commit oluşturur ve bir commit kimliği (ID) verir. Commit kimliği, gelecekte bu kayda erişmek veya üzerinde çalışmak için kullanılabilir.**

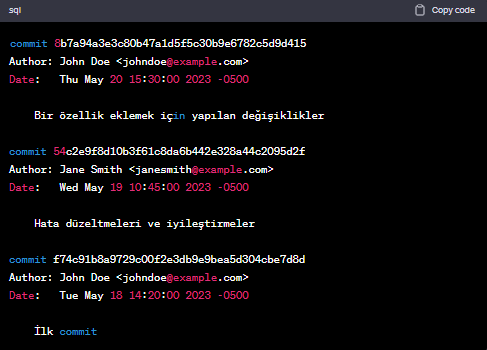
## **“Git Log”**

**‘git log’, Git deposunda yapılan commit'leri listeleyen bir komuttur. Bu komutu kullanarak, projenizin commit geçmişini görüntüleyebilirsiniz.**

**git log komutunu çalıştırdığınızda, aşağıdaki bilgileri içeren bir liste alırsınız:**

* **Her commit için benzersiz bir kimlik (ID).**
* **Commit'i gerçekleştiren kişinin adı ve e-posta adresi.**
* **Commit tarihi ve saat bilgisi.**
* **Commit mesajı (yapılan değişiklikleri açıklayan metin).**

**Örneğin, git log komutunu çalıştırdığınızda şu gibi bir çıktı alabilirsiniz:**

****

**Yukarıdaki örnekte, en yeni commit en üstte yer alır ve her bir commit'in kimliği, yazarı, tarihi ve commit mesajı görüntülenir.**

**/\*Ayrıca, commit kimliklerini kullanarak belirli bir commit'e dönme veya üzerinde çalışma gibi işlemler yapabilirsiniz. Bunu ‘git checkout’ komutu ile yapabilirsiniz\*/**

## **“Gitignore”**

**‘.gitignore’, Git tarafından takip edilmemesi gereken dosyaları veya klasörleri belirtmek için kullanılan bir dosyadır. git ignore komutu aslında bir komut değil, .gitignore adlı bir dosyanın içeriğini düzenlemenizi sağlayan bir araçtır.**

**.gitignore dosyası ise, Git deposunda bulunan dosyalardan veya belirli desenlere uyan dosyalardan Git'in dikkate almamasını istediğiniz öğeleri tanımlamanızı sağlar. Örneğin, derleme çıktıları, geçici dosyalar, gizli dosyalar veya belirli uzantılara sahip dosyalar gibi öğeleri .gitignore dosyasına ekleyebilirsiniz.**

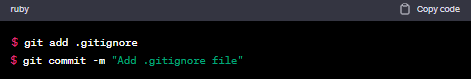
**.gitignore dosyasını kullanmak için aşağıdaki adımları izleyebilirsiniz;**

1. **Git deposunun kök dizininde bir .gitignore dosyası oluşturun (eğer henüz yoksa).**
2. **.gitignore dosyasını bir metin düzenleyiciyle açın ve Git tarafından takip edilmemesini istediğiniz dosyaları veya desenleri içine ekleyin.  
   Örnek bir .gitignore dosyası:**

****

**/\*Yukarıdaki örnekte, derleme/ klasörü, \*.log uzantılı dosyalar ve gizli/ klasörü Git tarafından takip edilmeyecek dosya ve klasörlerdir. # karakteri, yorum satırı olarak kullanılır ve .gitignore dosyasında açıklama eklemek için kullanılabilir.\*/**

**3. .gitignore dosyasını kaydedin ve Git'e dahil edin.**

****

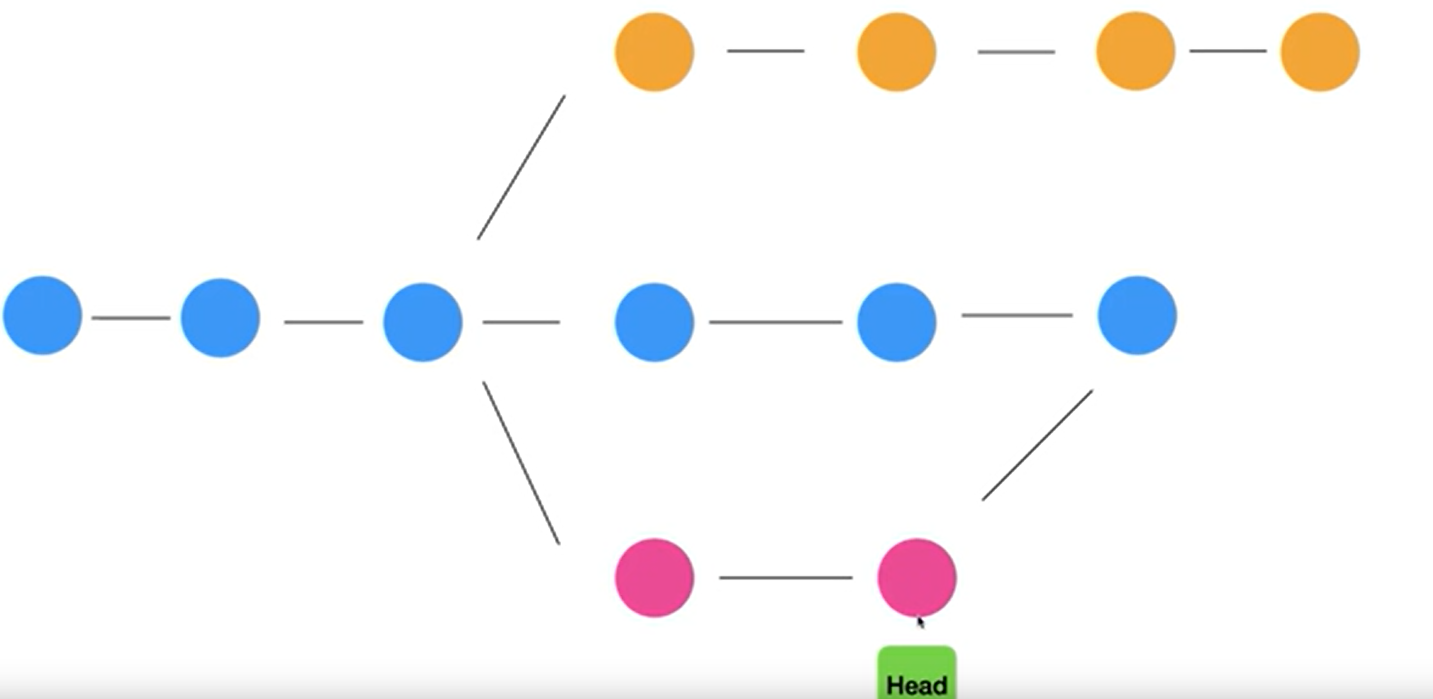
**/\* .gitignore dosyasını Git deposuna ekleyerek, diğer kullanıcılarla paylaşılabilir hale getirebilirsiniz. .gitignore dosyasını düzenledikten sonra, Git bu dosyayı kontrol ederken belirtilen öğeleri yoksayacaktır. Bu, gereksiz dosyaların ve geçici dosyaların depoya eklenmesini önler ve depodaki gereksiz karmaşıklığı azaltır. \*/**

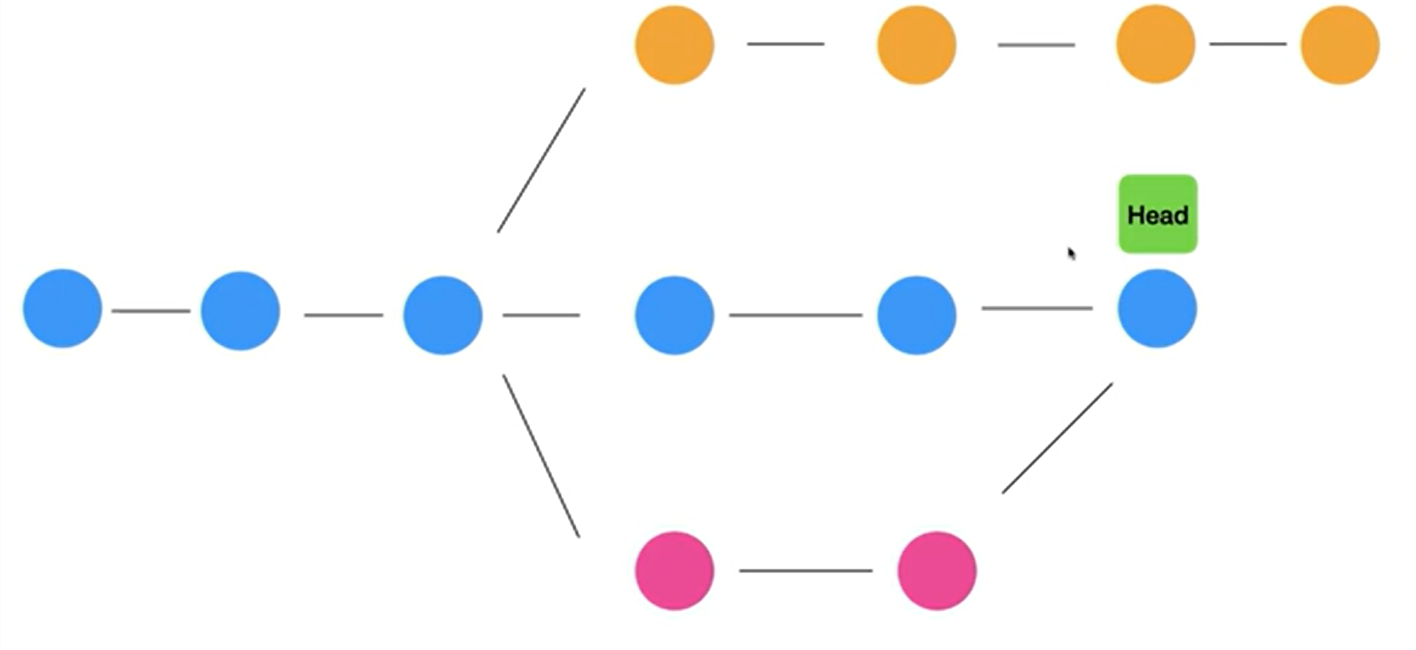
## **“Branch(Dal)” Nedir?**

**Bir işlemi dallandırmaya yarar bu şekilde birden fazla kişi aynı projede aynı anda farklı işlemler yapabilir.**

## **“Head” Nedir?**

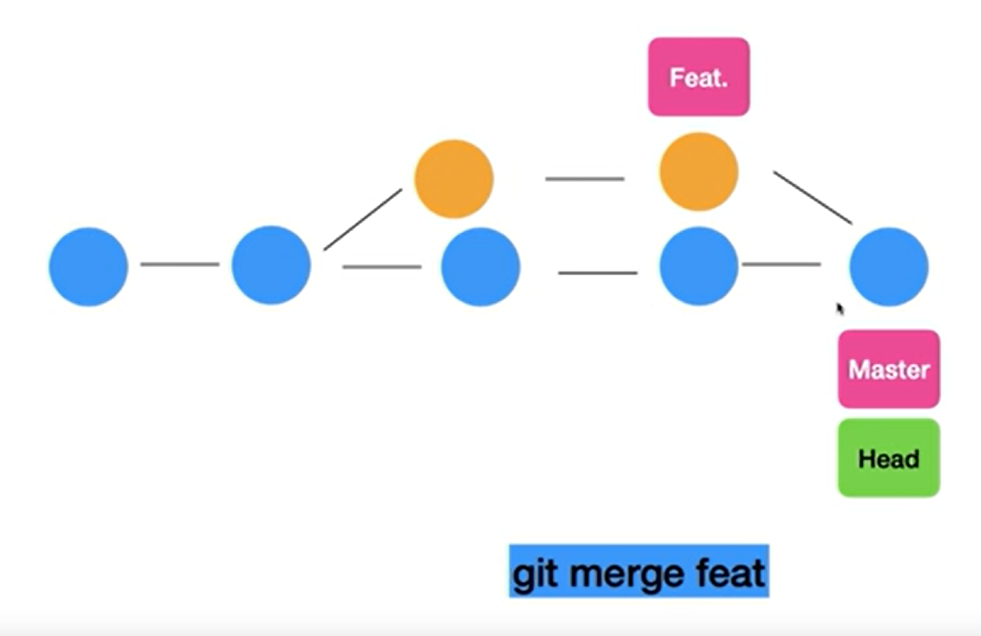
**Git içinde hangi konumda olduğumuzu gösterir. Genelde son commiti gösterir. Özet olarak hem commit hem branch olarak güncel konumumuzu verir diyebiliriz.**





## **“Merge(Birleştirmek)” Nedir?**

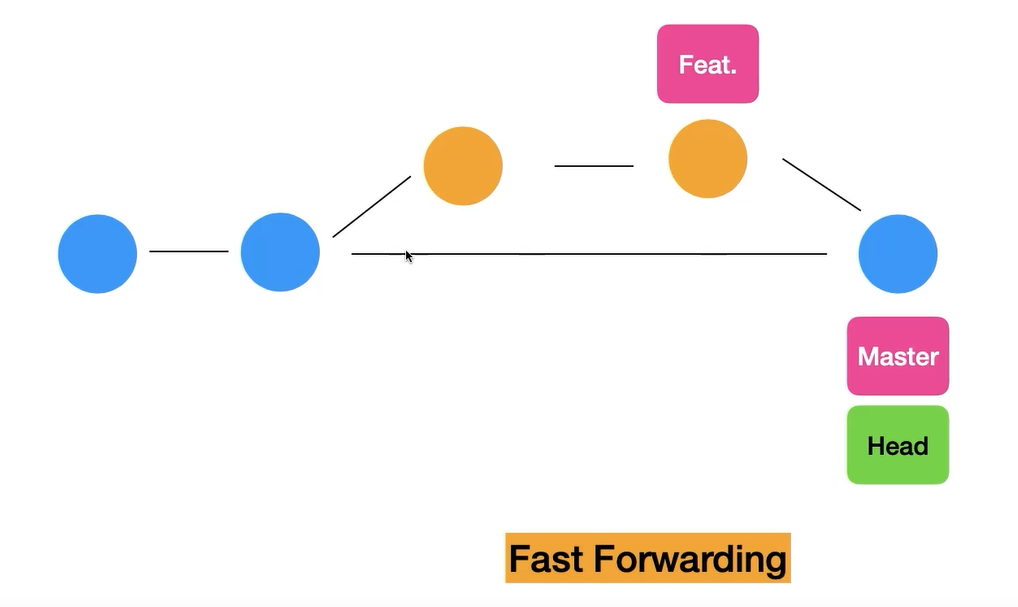
**Merge işlemi başka bir branch'deki değişiklikleri üzerinde çalıştığınız kendi branch'inize entegre etme işlemidir.**

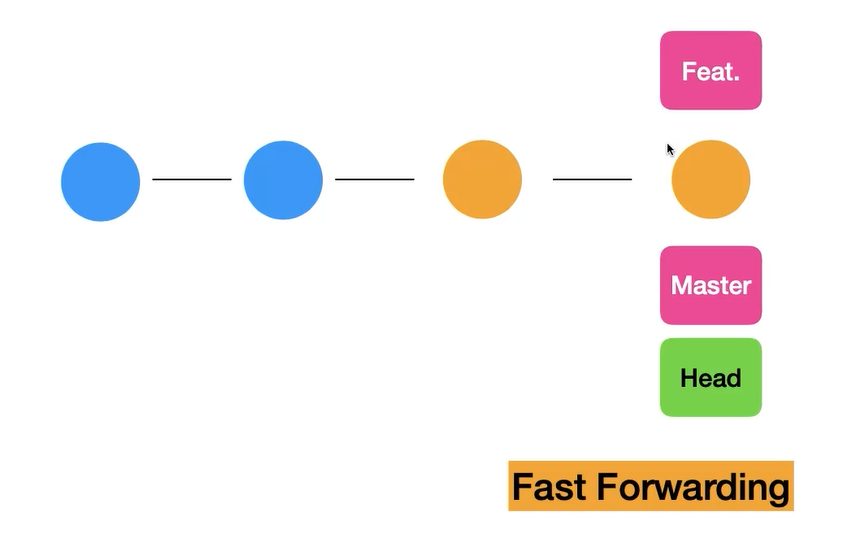


## **“Fast Forward(İleri Sarmak)” Nedir?**

**Master’da hiçbir değişiklik yapmadan Feat’te yaptığımız değişiklik ile birleştirmemizdir.**

**Feat’te yapılan değişiklikleri direk Master’a aldığı için yeni commit oluşturmadan Master’ın son commiti üzerine ekleyerek Merge işlemini yapar.**



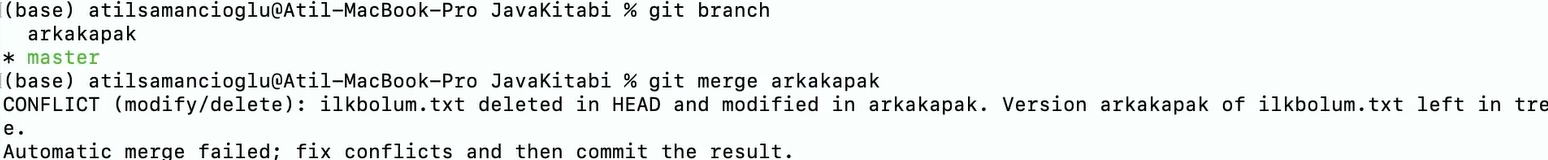


## **“Merge Conflict(Birleştirme Çakışması)” Nedir?**

**Aynı dosyanın aynı satırında yapılan çakışan değişiklikler nedeniyle Merge işlemi başarısız olur, bu olaya Merge Conflict denir.**

## **“Merge Conflict” nasıl çözülür?**

**1.Git tarafından belirtilen dosyayı açarak ordaki çakışma oluşturan değişiklikleri düzelterek çözülür.**



## **“Stash(Depolama)” Nedir?**

**Bir Branch’te işlem yaparken acil bir işimiz çıktığında işlemimizi yarım bırakıp diğer Branch’e geçtiğimizde sıkıntı çıkmaması ve kaybolmaması için yani yarım kalan değişiklikleri kayıt altına almak için kullanılır.**

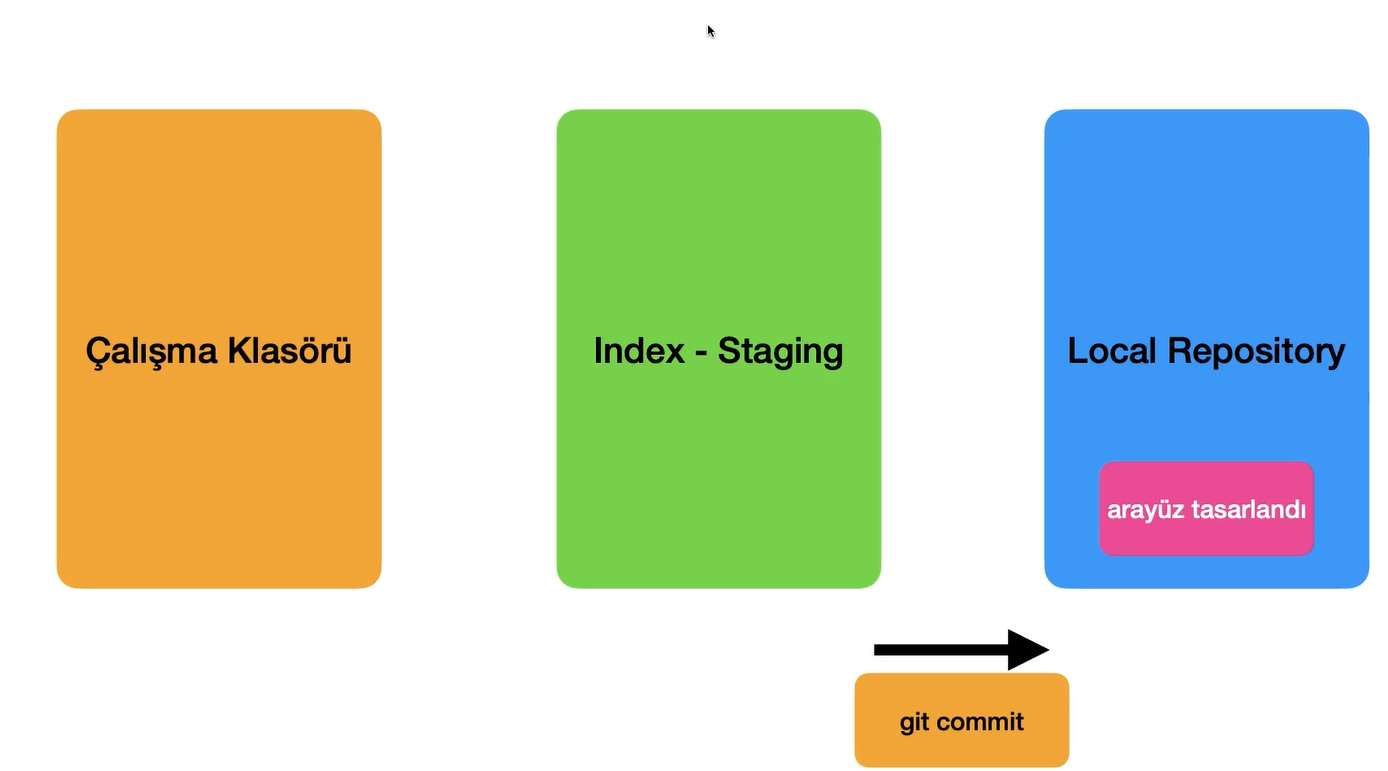
**Bilginin karışmaması veya kaybolmaması için yapılabilecekler:**

**1.Commit etmek.**

**2.Yaptığımız değişiklikten vazgeçmek =ctrl+z veya git restore ile geri dönmek .**

**3.Stash(Depolama) yapmak.**

**Git Stash’i bu ana başlıkların haricinde bir yerde saklıyor.**



## **“Pop” Nedir?**

**Git stash komutuyla depoladığımız değişiklikleri git stash pop komutuyla geri getirebiliyoruz . Stash listemizde bir den çok Stash varsa listenin en üstündeki Stash’i getirir.**

**Git stash clear yaparak stashlerimi sıfırlayabilirim.**

**Değişiklik yapıp git stash diyerek arka arkaya stash ekleyebilirim.**

**Git stash apply <stash ismi> diyerek stashleri ekleyebilirim.**

**Git stash list ile listelenen stashleri görebilirim.**

## **“Git Checkout”**

**‘git checkout’, Git deposunda farklı branch'ler arasında geçiş yapmak, commit'leri geri almak veya belirli bir dosyanın veya klasörün durumunu değiştirmek için kullanılan bir komuttur.**

**1.Branch değiştirme:**



**/\*Bu komut, belirtilen branch\_adi isimli branch'e geçiş yapmanızı sağlar. Projenizin mevcut durumu, seçtiğiniz branch'deki en son commit durumuna getirilir. \*/**

**2. Commite geri dönme:**

****

**/\* Bu komut, belirtilen commit\_ID ile tanımlanan bir commit'e geri dönmenizi sağlar. Projenizin durumu, seçtiğiniz commit'in durumuna geri alınır. Bu durumda, "detached HEAD" durumuna geçersiniz ve yeni değişiklikleri yapamazsınız.\*/**

**3.Dosya veya klasör durumunu değiştirme:**

****

**/\*Bu komut, belirtilen dosya\_yolu ile tanımlanan dosyanın veya klasörün durumunu değiştirir. Bu, dosyayı son commit durumuna geri alır ve yapılan değişiklikler kaybedilir.\*/**

**Dikkat!!**

**git checkout komutunu kullanırken dikkatli olmanız gereken noktalar vardır. Özellikle, mevcut değişikliklerinizi kaybetmeden branch veya commit'lere geçiş yapmak için öncelikle çalışma alanınızı temizlemek veya değişikliklerinizi kaydetmek önemlidir. Ayrıca, dikkatli olmadığınızda geçmiş durumları geri getirirken veya dosyaları değiştirirken var olan değişikliklerinizi kaybedebilirsiniz.**

## **“Git Reset ve Git Revert”**

**git reset:**

**‘git reset’ , Git deposunda yapılan değişiklikleri geri almanın bir yoludur. Bu komut, commit geçmişini değiştirir ve geri alınan commit'e kadar olan commitleri ortadan kaldırır veya geçici olarak "geri sarar". git reset komutunu kullanırken üç farklı mod seçeneği bulunur:**

* + **--soft: Bu modda, geri alınan commit'e kadar olan commitler silinmez, ancak bu commitlere geri dönülür ve değişiklikleriniz geçici olarak staging area'ya geri eklenir. Yani, geri alınan commit'in üzerine yeni bir commit oluşturmak için hazırsınız.**
  + **--mixed (varsayılan): Bu modda, geri alınan commit'e kadar olan commitler silinmez, ancak bu commitlere geri dönülür ve değişiklikleriniz staging area'dan kaldırılır. Yani, değişikliklerinizi tekrar staging area'ya eklemeniz gerekecektir.**
  + **--hard: Bu modda, geri alınan commit'e kadar olan commitler tamamen silinir ve geri dönülmez. Değişiklikleriniz tamamen kaybolur. Bu modu kullanırken dikkatli olmanız gerektiğini unutmayın, çünkü geri alınamayan değişikliklerinizi kaybedebilirsiniz.**

**git revert:**

**‘git revert’, Git deposunda yapılan değişiklikleri geri almanın başka bir yoludur. Bu komut, geri alınan commit'in tam tersini yaparak bir "geri alma commiti" oluşturur. Yani, geri alınan commit'in yaptığı değişiklikler geri alınır, ancak bu geri alma commiti commit geçmişine eklenir.**

**‘git revert’ komutu kullanıldığında, bir commit kimliği belirtilir ve bu commit geri alınır. Bir geri alma commiti oluşturulur ve yapılan değişiklikler bu committe geri alınır. Bu şekilde, geri alma commiti ile projenin geçmişi tutulur ve geri alınan değişiklikler geri alınır.**

## **“Git Diff”**

**‘git diff’, Git deposundaki dosyalar arasındaki farkları gösteren bir komuttur. Bu komut, değişiklik yapılan dosyaların içeriğini karşılaştırır ve farklılıkları görsel veya metin tabanlı olarak görüntüler.**

**Bu komut ile farklı durumları kontrol edebilirsiniz;**

1. **Çalışma Alanı ile Staging Alanı Arasındaki Farkları Görmek:**

****

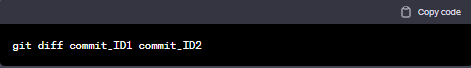
**/\*Bu kullanım şekli, çalışma alanındaki (yani yapılan son değişikliklerin olduğu yer) ve staging alanındaki (yani git add komutuyla eklenen dosyaların olduğu yer) değişiklikleri karşılaştırır. Bu şekilde, dosyaların değiştirilen satırlarını, eklenen veya silinen içerikleri görüntüleyebilirsiniz.\*/**

1. **Staging Alanı ile Son Commit Arasındaki Farkları Görmek:**

****

**/\*Bu haliyle, staging alanındaki değişiklikleri (yani git add ile eklenen dosyaları) ve son commit arasındaki farkları gösterir. Bu, son commit'e eklemek üzere staging alanına eklediğiniz değişiklikleri kontrol etmenizi sağlar.\*/**

1. **İki Farklı Commit Arasındaki Farkları Görmek:**

****

**/\*Bu şekilde kullanıldığında ise, belirtilen commit\_ID1 ve commit\_ID2 ile tanımlanan iki farklı commit arasındaki farkları gösterir. Bu şekilde, belirli iki commit arasındaki değişiklikleri kontrol edebilirsiniz.\*/**

## **“Git Rebase”**

**‘git rebase’ , Git'te branch'leri birleştirmek veya branch'in temelini değiştirmek için kullanılan bir komuttur. git rebase, commit geçmiğini değiştirir ve daha düzenli ve anlamlı bir commit geçmişi elde etmeyi sağlar.**

**git rebase komutunu kullanarak aşağıdaki işlemleri gerçekleştirebilirsiniz:**

1. **Branch’leri birleştirmek:**

****

**/\*Bu komutlar, hedef\_branch adlı branch'e geçiş yapar ve sonra kaynak\_branch adlı branch'i hedef\_branch üzerine birleştirir. git rebase komutu, kaynak\_branch üzerindeki commitleri hedef\_branch üzerine uygular ve daha düzgün bir commit geçmişi oluşturur. Sonuç olarak, hedef\_branch üzerinde birleştirilmiş ve güncellenmiş bir commit geçmişi elde edersiniz.\*/**

1. **Commitleri düzenlemek:**

****

**/\*Bu komut, son n commit'i interaktif bir şekilde düzenlemenizi sağlar. HEAD~n ifadesi, mevcut HEAD commitinden geriye doğru n adet commiti temsil eder. git rebase -i HEAD~n komutunu çalıştırdığınızda, bir metin düzenleyici açılır ve belirtilen commitleri yeniden düzenlemeniz veya birleştirmeniz için size talimatlar sunar.\*/**

1. **Branch'in Temelini Değiştirmek:**

****

**/\*Burada ise, branch\_adi adlı branch'e geçiş yapar ve sonra yeni\_temel\_branch adlı branch'i temel olarak belirler. git rebase komutu, branch\_adi üzerindeki commitleri yeni\_temel\_branch üzerine uygular. Bu, branch\_adi'nin yeni temel branch ile aynı değişiklikleri içeren bir commit geçmişi almasını sağlar.\*/**

# Git altyapısını kullanan platformlar:

**1.GitHub**

**2.GitLab**

**3.Bitbucket**

**4.Azure DevOps**

**5.GitKraken**