Git, versiyon kontrol sistemidir.

Bir oyun oynamak istediğimizde kaldığımız yerden devam etmek isteriz, en baştan değil işte buna git denir. Bir checkpoint, savepoint aslında.

Bir proje üzerinde yapılan değişiklikleri izlemek, yönetmek ve gerektiğinde geri alabilmek için kullanılır.

Linuxu bulan kişi tarafından bulunmuştur. Linux Torvalds.

Git, bir endüstri standardıdır.

Github, bir portaldır. Projelerimizi saklayabildiğimiz bir web sitesidir.

2 Sebepten dolayı kaynak kontrol sistemine ihtiyaç duyarız;

1. Kaynak kodunuzun değişimlerini takip etmek ve bunu yönetebilmek
2. Birlikte kod geliştirmek.

Git: git komutlarını gösterir.

Git help <komut>: komutun ne işe yaradığını gösterir.

Git –-version: git’in versiyonunu öğrenmek için.

Commit: Bir işi tamamlarız, örn, arayüz tasarlanması. Bu iş bittiğinde commit ediyoruz anlamına gelebilir.

Branch: Aynı anda aynı proje üzerinde çalışabiliriz. Örn, ben resim paylaşma fonk. Yazarım arkadaım mesajlaşma fonk. Yazar. Buna branch denilir. Tek başına da yapılabilir. Commit var ve commitin mesajı var.

Git add, yaptığımızda çalışma klasörümüzden Index-Staging klasörüne sahne klasörüne alır projeyi.

Git commit, yaptığımızda Index-Staging klasöründen Local Repositorye alır projeyi.

Çalışma klasörü - Index/Staging - Local Repository

-----🡪 -----🡪

git add git commit

Git status: güncel git durumunu göster.

Git init – initialized: git’i başlatmak için.

Master ya da main, ana branchtir. Programın başlatıldığı branchtir.

Repo: Projeyle alakalı her şey bu repo içinde yer alır.

Git add, dosyaları repoya atacağım. Commit dediğimizde atıyoruz.

Örneğin, dosya.txt dosyasını git add ile izleme listesine eklediniz ancak sonradan bu değişikliği geri almak istiyorsunuz. İşte bu durumda yapmanız gerekenler:

git reset HEAD dosya.txt

Bu komut, dosya.txt dosyasını staging area'dan çıkarır ve son yapılmış olan git add işlemi geri alınmış olur. Dosyanın mevcut durumu değişmez, sadece takip edilmeyen (untracked) bir dosya olarak kalır. Bu noktadan sonra, git status komutunu kullanarak dosyanın takip edilmediğini görebilirsiniz.

HEAD, bir Git deposundaki en son commit'i işaret eden bir referanstır. Yani bulunduğun branchin son durumunu gösterir. Dolayısıyla git reset HEAD dosya.txt komutu, HEAD referansını staging area'ya geri döndürür. Bu, son commit'e geri dönüldüğü anlamına gelir. HEAD kelimesi, mevcut durumu işaret etmek için kullanıldığından, git reset HEAD komutu, dosyanın son commit'e göre durumunu geri döndürür.

Git commit’te yer alan mesaj zorunludur.

Her commitin kendine özgü bir hash’i var. Sıkıntı olduğunda bu hash’e dön gibi şeyler yapabiliyoruz. Git log’da bunu görebiliriz.

Git add, yerine git commit –a yapılabilir.

Git add . yapmak her şeyi eklemek anlamına gelir.

Git log, projede yaptığımız commitleri görmek için. Şu tarihte bu kişi şu mesajla bunu yapmış gibi.

Git add. Yapmak bazen dezavantaj oluşturabilir.

Gitignore, git’in görmezden geleceği dosyaları belirlemek için. Ayrı bir dosyadır.

touch .gitignore şeklinde oluşturmamız lazım. Gitignore dosyası içerisine, gizlemek istediğimiz dosyaları yazarız. gizli.txt yazarız direkt.

Gitignore template python ya da java yazarsak örnekler çıkar.

Branch, dal demektir.

Git Komutları

**git config**

Bu komut git komutları ile ilişkili olan yazar adı ve e-posta yapılandırmak için kullanılır. Buradaki amaç; yapılan değişikliklerin hangi kullanıcı tarafından gerçekleştirildiğini görüntülemektir.

git config –global user.name “[name]”

git config –global user.email “[email address]”

**git init**

Yeni bir depo başlatmak amacıyla kullanılır. Bunu yaptığınız yerde .git uzantılı bir klasör oluşmaktadır.

Yerel git deposu oluşturmak için kullanılır.

**git clone**

Mevcut bir URL adresinden bir havuz elde etmek amacıyla kullanılır.

**git add**

Projenizi çalışma dizinine eklemek amacıyla kullanılır.

**git commit**

Commit ‘in kelime anlamı işaretlemedir. Bu komut ile hazırlama alanına aldığınız değişiklikleri yerel depoya yüklemek için kullanılır. Değişiklikleri yüklerken commit’e mesaj eklemek yararınıza olacaktır. Bu sayede üzerinden belli bir zaman geçse bile yaptığınız değişiklikleri anlamanıza ve işlemlerinizi hızlandırmanıza yardımcı olur.

**git diff**

Dosyada yapılan tüm değişiklikleri gösteren komuttur.

**git reset**

Bilgileriniz silinmeden ve yeni bir commit üretmeden değişiklikleri geri alabilirsiniz. Dosya aşamasını kaldırır fakat aynı zamanda içeriğini korur.

**git status**

Dosyaların durumunu gösteren komuttur.

**git rm**

Çalışma dizinindeki belirli bir dosyanın silinmesi için kullanılır.

**git log**

Commit kayıtlarının görüntülendiği ve listelendiği komut türüdür.

**git show**

Belirtilen commit ‘in meta verilerini ve içeriğindeki değişiklikleri gösterir.

**git tag**

Commit ‘leri etiketleme işlemidir.

**git branch**

Bu komut sayesinde projenizi dallara ayırabileceksiniz. Bir örnekle açıklamak gerekirse: Diyelim ki; yayında olan bir projeniz mevcut ve bu projede yenilikler yapmanız gerekiyor. Yayına çıkan kısımda hiç değişiklik yapmadan ekleyeceğiniz sürüm çalışıyor mu denemek istediniz.

İşte bu tarz durumlarda “branch” komutu devreye girecektir. Böylece yayında olan bir projeyi riske atmadan yeniliklerin çalışıp çalışmadığını görüntüleme şansını elde edeceksiniz.

**git checkout: REMOTE BRANCHLERE GEÇİŞTE KULLANILIR.**

Oluşturulan bir daldan diğer dala geçmek için ya da commitler arasında geçiş için kullanılır.

Commit geçişi için git chechkout (hash) şeklinde ilerleriz.

**git merge**

Belirtilen bir dalı mevcut başka bir dal ile birleştirmek için kullanılır.

**git remote**

Yerel git deponuzu uzak bir sunucuya bağlamak için kullanılır.

**git push**

Yerel commit ‘leri uzak depoya göndermek için kullanılan komuttur.

Git push –u origin master, -u bulunduğumuz klasördeki bütün dosyaları Remote Repository(uzak depo, bir versiyon kontrol sistemine ait merkezi bir depodur.) tarafına atacağımızı , “origin” adı ile daha önce ayarlanmış linke push işleminin yapılacağınıGit push origin master

Git push

Git push origin master

Git push –u origin master: -u, upstream demektir.

Üçü de push eder.

Git remote origin

Git remote add origin, yerel bir git deposuna bir uzak deposu eklemek için kullanılır.

Origin, genellikle bir uzak depo adı olarak kullanılır.

**git stash**

Geçici kaydetme işlemi

Üzerinde çalıştığınız fakat henüz commit etmediğiniz değişiklikleri geçici olarak kayıt altına alan komuttur. Böylece aktif branch ‘larınızda herhangi bir değişiklik yapmadan korursunuz. Geçici kaydetme işlemi sayesinde karışıklığa da yer vermemiş olacaksınız.

**git mv**

Bir dosyayı taşıma veya yeniden adlandırma

**git restore**

Çalışma ağacı dosyalarını geri yükle

**git switch**

Dallar arasında geçiş yapmasına olanak tanıyan bir komuttur. Git chechkout yerine tanıtıldı.

Sadece branch'ler arasında geçiş yapmak için kullanılır.

**git branch**

Var olan branchleri görmek için

Git branch –r, remote olan branchleri de görmek için

**git branch (yeni branch ismi)**

Yeni bir branch oluşturmak

**git stash**

Commit etmediğimiz klasörlerimizi, branch değiştirirsek saklamak için kullanılır.

**git stash pop**

Git stash’i geri almak için kullanılır.

**git reset**

Git reset (hash)

**git reset –-hard**

Git reset –-hard (hash)

**git revert**

Git revert (hash)

**git diff**

difference’dan geliyor. Commit, branch, dosya vs arasındaki farklılıkları gösteriyor.

git diff HEAD gibi

**git fetch**

fetch, al getir demek. Değişiklikleri alıp locale getirir, dosyaları değiştirmez.

git fetch origin master

Daha güvenli bir komut

**git pull**

Uzakta olan sunucudaki değişiklikleri çalışma dizinine getirerek birleştirir.

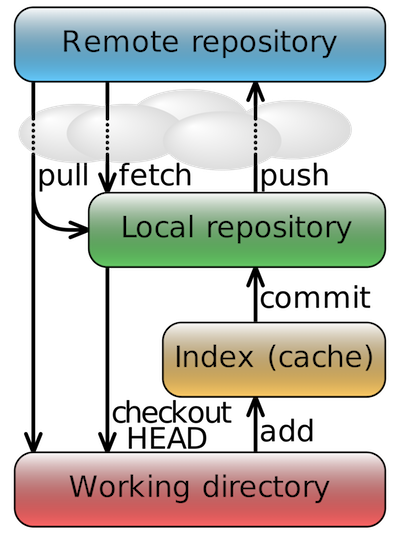
Değişiklikleri alıp locale getirir, dosyaları değiştirir.

git pull = git fetch + git merge

git pull origin master

**git clone**

Remote repoyu bilgisayarıma klonla anlamına gelir.



**Master**, bir branchtir. Projenin son halidir. Ana branchtir.

**Head**, güncel konumu verir. Güncel branchi verir.

**Merge**, farklı branchleri birleştirir.

**Fast forwarding,** Fast-forward birleştirme, temelde hedef branch'in (genellikle master) değişmeden bir alt branch'in değişikliklerini entegre etme işlemidir. Bu durumda, alt branch'deki değişiklikler, hedef branch'in son commit'iyle aynıdır veya hedef branch üzerinde yapılan değişiklikler alt branch'ten önceki bir noktaya eklenmemiştir.

Bir fast-forward birleştirme örneği şu şekildedir:

Başlangıçta, master branch'i şu şekilde görünmektedir:

A --- B (master)

Ardından, feature adında başka bir branch oluşturursunuz ve bu branch üzerinde çalışırsınız:

A --- B (master)

\

C (feature)

Son olarak, feature branch'indeki değişiklikleri master branch'ine birleştirirsiniz:

A --- B --- C (master)

\

C (feature)

Burada, feature branch'indeki değişiklikler doğrudan master branch'inin son commit'ine eklenmiştir. Bu nedenle, bir fast-forward birleştirme gerçekleşmiştir.

Fast-forward birleştirme, genellikle değişikliklerin basit olduğu ve çakışma olmadığı durumlarda gerçekleşir. Ancak, çakışma (conflict) olduğu durumlarda fast-forward birleştirme mümkün değildir ve Git, çakışmayı çözmek için birleştirme işlemi yapmanızı ister. Bu durumda, "merge commit" adı verilen yeni bir commit oluşturulur.

**Merge Conflict,** Conflict **=** çakışma. Bir dosyada, iki ayrı dalın aynı satırda düzenleme yapması veya bir daldaki dosyanın silinip diğer dalda düzenlenmesi durumunda çakışma ortaya çıkar.

**Stash,** Stash = saklama, apayrı bir yer. Commit etmediğimiz klasörlerimizi, branch değiştirirsek saklamak için kullanılır. git stash komutu kullanılır. git stash’i geri almak için, git stash pop kullanılır.

**Rebase**, bazı değiştirme. Merge commitlerden kurtulmak demektir. Master ve feat branchlerini hizalamak. Tarih hizalaması sıkıntıya girer.

**Fork**, çatallamak. Remote repoyu alıyorum, bundan sonra senden ayrılıyorum. Başka bir kişinin paylaştığı projeyi alıp, kendi repomuzda değişiklik yapabiliriz.

**Compare and Pull Request,** compare – çakışma. 2 tane branch arasındaki çakışmalara bakıyor, sıkıntı yoksa merge edebilirsin diyor.

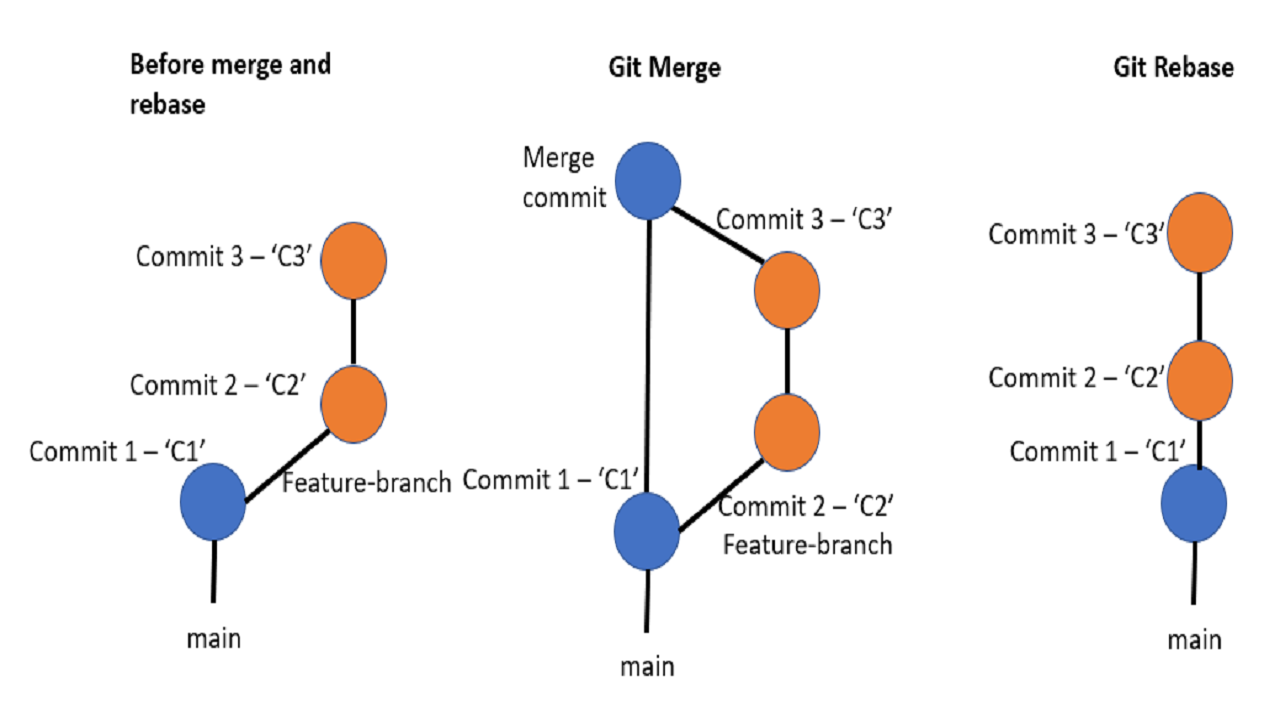
**Pull request** – branchleri merge etmeyi sadece localde yapmamıza gerek yok. Github sahibine compare olmadığını merge edilebileceğini söyleyen bir mesaj ve merge işlemidir.

**Collaborators**, işbirlikçilerimiz. Private repolarımızı görmesini istediğimiz kişileri buradan ekleyebiliriz.

.md, mark down. Html gibi. Kendi syntaxı var.

**VCS,** version control system.

**Issues(sorunlar),** yazılım geliştirme sürecinde karşılaşılan problemleri, hataları, önerileri ve görevleri takip etmek için kullanılan bir kavramdır. GitHub gibi platformlarda, "Issues" bölümü, bir projede geliştiricilerin tartıştığı ve izlediği tüm açık sorunları içerir.s



Rebase projeyi, herkesle paylaşırsak bütün herkesin kodunu darmaduman ederiz.

!!! Git push origin master komutu, yereldeki master dalındaki değişiklikleri origin adlı uzak depoya gönderir. Diğer dalları göndermek için, ilgili dala geçiş yapmalı ve git push komutunu tekrar kullanmalısınız. Genel olarak, git push komutu, sadece belirtilen dali etkiler. Bu nedenle, git push komutunu kullandığınızda, yalnızca çalıştığınız dala ait değişikliklerin uzak depoya gönderildiğini unutmayın. Diğer daları göndermek için ilgili dala geçiş yapmalı ve git push komutunu tekrar kullanmalısınız.

**"Meta veri",** "Meta" kelimesi, "öz" anlamına gelir ve "veri" ise bilgi anlamına gelir. Bir bilgi parçasının kendisi değil, o bilgi parçası hakkında bilgi veren başka bilgiler olarak düşünülebilir.

Bir dosyanın meta verileri:

Bir dosyanın meta verileri, dosyanın boyutu, oluşturulma tarihi, son değiştirilme tarihi, izinler, dosya türü gibi dosya hakkındaki bilgileri içerir.

**GİT MÜLAKAT SORULARI**

**1.1. Git Nedir?**

Git, dağıtılmış bir versiyon kontrol sistemidir. Yazılım geliştirme projelerinin yönetiminde kullanılır.

**1.2. "Bare repository" ve "working directory" arasındaki fark nedir?**

"Bare repository", çalışma alanı olmayan bir Git deposudur, yalnızca Git verilerini içerir. "Working directory" ise kullanıcının çalışma alanıdır, dosyaların değiştirilebilir olduğu ve üzerinde çalışılabildiği alandır.

**1.3. "Fork" ve "branch" arasındaki fark nedir?**

"Fork", bir projenin kopyasını almak ve onu kendi hesabınızda devam ettirmek anlamına gelir. "Branch" ise bir projenin ana geliştirme hattından ayrılan ve üzerinde değişiklikler yapılan bir dalı ifade eder.

**1.4. Commit oluşturup yayınladınız. Ancak hala değişiklik yapılması gereken şeyler olduğunu fark ettiniz. Commit'in bu aşamasında yapabilir misiniz? Evetse nasıl?**

Evet, yapabilirsiniz. Değişiklikleri yapın, ardından git add ve git commit --amend komutlarını kullanarak önceki commit'e yeni değişiklikleri ekleyebilirsiniz.

**1.5. "Cherry-picking" nedir?**

"Cherry-picking", belirli bir commit'in değişikliklerini alıp başka bir dala uygulama işlemidir.

**1.6. "Stash" nedir?**

"Stash", üzerinde çalıştığınız ancak henüz commit etmek istemediğiniz değişiklikleri geçici olarak saklamak için kullanılan bir Git komutudur.

**1.7. Git "stash drop" nedir?**

"Stash drop", "stash" içindeki geçici değişikliklerden birini kaldırmak veya silmek için kullanılan bir Git komutudur.

**1.8. Git'te "conflict"leri nasıl çözersiniz?**

Conflict'leri çözmek için git mergetool kullanabilir veya manuel olarak dosyaları düzenleyip sonrasında git add ve git commit yapabilirsiniz.

**1.9. Git'te kullanılan dil hangisidir?**

Git'in temel kodlama dili C'dir.

**1.10. "Pull request" nedir?**

"Pull request", bir projede yaptığınız değişiklikleri ana projeye entegre etmek için diğer katılımcılardan inceleme ve onay istemenizdir.

**1.11. Kötü commit bulmanın en etkin yolu nedir?**

Kötü commit'leri bulmanın en etkin yolu, projeye katkıda bulunan diğer geliştiricilerin geri bildirimlerini göz önünde bulundurmak ve commit geçmişini incelemektir.

**1.12. Git ve GitHub arasındaki farkı biliyor musunuz?**

Evet, Git, bir versiyon kontrol sistemidir; GitHub ise Git tabanlı projelerin barındırıldığı bir platformdur.

**1.13. Diğer Git repository hosting hizmetlerini adlandırabilir misiniz?**

GitLab, Bitbucket, SourceForge, Azure Repos gibi birçok alternatif Git repository hosting hizmeti bulunmaktadır.

**1.14. "Commit object" ne barındırır biliyor musunuz?**

"Commit object", bir commit'in içeriğini ve meta bilgilerini barındırır, yani değiştirilen dosyaların durumunu, kimin yaptığını, ne zaman yapıldığını vb. bilgileri.

**1.15. Git kullanmanın herhangi bir avantajını görüyor musunuz?**

Evet, Git, paralel geliştirme, versiyon kontrolü, işbirliği ve proje yönetimi gibi birçok avantaj sağlar.