**1-)** Git bir versiyon kontrol sistemidir. Versiyon kontrol sistemi, proje dosyalarımızı kaybetmememizi ve takım arkadaşlarımızla kodlarımızı paylaşabildiğimiz bir sistemdir. İstersek kendi kodumuz ile arkadaşımızın kodlarını birleştirebiliriz. Yaptığımız her kaydetme işlemine “commit” denmektedir.

**2-)** Terminalde gezinmek için bazı komutlar bilmemiz gerekiyor.

ls : Dosyaları ve klasörleri listeler. (List)

ls -la : Gizli dosyaları ve klasörleri listeler.

pwd : Bulunduğumuz diziyi bize gösterir. (Print Working Directory)

cd klasörAdı : Klasör değiştirmek için bu komutu kullanırız. (Change directory)

cd .. : Bir önceki klasöre geri döner.

clear : Konsolu temizler.

mkdir klasörAdı : Yeni bir klasör oluşturmamızı sağlar. (Make Directory)

rmdir klasörAdı : Bir klasörü silmemizi sağlar. Silebilmemiz için o klasörün boş olması gerekiyor. (Remove Directory)

touch dosyaAdı.dosyaTürü : Yeni bir dosya oluşturmak için kullanılır. Örneğin **not** isminde ve **txt** türünde bir dosya oluşturmak için “touch not.txt” yazmalıyız.

rm dosyaAdı.dosyaTürü : Dosya silmek için kullanılır. Bu komut sadece dosyaları siler klasörleri silmez. Klasörleri silmek için başka bir kod kullanılır. (Remove)

rm -rf klasörAdi : Klasör silmek için kullanılır. İçinin dolu olup olası önemli değildir.

**3-)** Git sisteminde her kullanıcının bir kullanıcı adı ve bir e-maili vardır. Bunlar projede bir değişiklik yapıldığı zaman, o değişikliğin kim tarafından yapıldığını gösterir. Git sistemine kendi kullanıcı adımızı ve e-mailimizi eklemek için aşağıdaki kodlar kullanılır.

git config user.name : Sisteme kayıtlı kullanıcı adımızı gösterir.

git config --global user.name “Yeni İsim” : Bu komut ile kullanıcı adımızı kaydedebilir veya eski kullanıcı adımızı değiştirebiliriz.

git config user.email : Sisteme kayıtlı e-mailimizi gösterir.

git config --global user.email Yeni E-mail : Bu komut ile e-mailimizi kaydedebilir veya eski e-mailimizi değiştirebiliriz.

**4-)** Daha önce, yaptığımız her kaydetme işlemine commit adı verildiğini söylemiştik. Her commit işlemi bir mesaj içerir. Bu mesaj o commit de ne yaptığımızı belirtir. Örneğin kullanıcı arayüzü tasarlandı mesajı ile bir commit gönderirsek diğer kullanıcılar bu commit noktasında kullanıcı arayüzünün tasarlandığını anlayabilirler. Eğer biz kullanıcı arayüzünde değişiklik yapmak istersek bu noktaya geri dönebiliriz.

**5-)** Bir projede birden fazla geliştirici aynı anda çalışabilir. Örneğin birisi mesajlaşma fonksiyonunu yazarken diğeri ise fotoğraf gönderme fonksiyonunu yazabilir. Projeyi dallara ayırabilirler. İsterlerse bu dalları ilerde birleştirip projelerine öyle devam edebilirler veya bir geliştirici projenin bir bölümünü 2 farklı dala ayırıp her bir dalda farklı bir geliştirme yapabilir. Ardından bu dallardan hangisini beğenirse onun ile devam edebilir. İşte bu dallara “branch” denmektedir.

git branch : Bize bütün branch’lerin adını gösterir.

**6-)** Projede yaptığımız bir değişikliği kaydetmek istersek “git add” komutunu kullanmalıyız. Bu komut yaptığımız değişikliği “Staging area” ya alıyor. Projede yapılan her kaydetme işlemi sisteme direk olarak kaydedilmiyor. Staging area denen bir bölmede bekletiliyor. Bu bölmeye “İndex” te denmektedir. Sisteme kaydetmek için ise “git commit” komutu kullanılır.

**7-)** Bir projede git versiyon sistemini kullanabilmek için proje dosyamıza git versiyon sistemini bağlamamız(kurmamız) gerekiyor. Bunu kurmak için “git init” komutu kullanılır. Bir klasörde git versiyon sisteminin kurulu olup olmadığını görmek için “git status” komutu kullanılır.

**8-)** Kaydetme işlemini yaparken bazı komutları kullanmalıyız. Staging area’ya gönderilmeyen veya commitlenmeyen dosyaları “git status” komutu ile kontrol edebiliriz.

git add dosyaAdı : Kaydetmek istediğimiz dosyayı staging area’ya gönderir.

git add . : Kaydetmek istediğimiz bütün dosyaları staging area’ya gönderir.

git commit -m “commit mesajı” : Staging area’daki dosyaları commitlememize yarar. Mesaj girilmesi zorunludur.

git log : Yaptığımız commitleri gösterir.

git restore dosyaAdi : Bu koda girilen dosyanın en son committeki halini geri getirir.

**9-)** Projede çalışırken bazı dosyaları commitlemek istemeyebiliriz. Yani onları başka geliştiricilerin görmesini istemeyebiliriz. Bunun için “gitignore” kullanılır. Gitignore dosyasının içine göndermek istemediğimiz dosyanın adını yazıyoruz. Artık git add yapsak bile o dosya gözükmeyecektir.

touch .gitignore : Bu komut ile gitignore dosyamızı oluşturuyoruz. Eğer git yüklerken bir ide seçtiysek bu komutu girdikten sonra idemiz açılacaktır ve içerisine gözükmesini istemediğimiz dosyanın adını yazıyoruz. Bu sayede artık gözükmeyecektir. Fakat eğer bir ide girmediysek **Vim** varsayılan ide olarak atanacaktır. Bu komutu girdikten sonra konsol ekranında dosyamızın adını girmemizi isteyecek. Dosyanın adını girdikten sonra ideyi kapatmak için “q” tuşuna basmamız lazım.

**10-)** “HEAD” ifadesi bizim git içerisindeki konumumuzu gösterir.

**11-)** Branch oluşturma, değiştirme ve birleştirmede kullanılan bazı komutlar.

git branch branchAdi : Yeni bir branch açmak için kullanılır.

git switch branchAdi : Branch değiştirmemizi sağlar. Bu koda girdiğimiz branch’e döneriz.

git merge branchAdi : İçinde bulunduğumuz branch ile bu koda girdiğimiz branch’i birleştirir ve içinde bulunduğumuz branch’den devam eder.

**12-)** Bazen dosyalarımız kaybolmasın diye, master branch’de bir şey yapmayıp yeni bir branch açarak projeye bu branch’den devam edebiliriz. Daha sonra bu yaptıklarımızı commit’leyip master branch ile merge edebiliriz. Buna ”Fast Forwarding” yani ileri sarma denir.

**13-)** Dosyalarımızı merge ederken bazen çakışmalar olabilir. Bir dosyada yazılanlar silinebilir veya o dosya tamamen kaldırılabilir. Bunlara “Merge Conflict” denir. Bu sorunları çözerken yapmamız gereken, o dosyalara ne olacağını belirtmektir. Yani son halini yapıp commit etmemiz gerekiyor. Commit ettikten sonra otomatik olarak merge edecektir.

**14-)** Bazen bir branch’de projemizi geliştirirken yaptığımız değişiklikleri kaydetmeden diğer branch’lere geçmemiz gerekebilir. Bu durumda “Stash” (saklama) komutunu kullanmamız gerekir. Değişiklikleri yaptıktan sonra “git stash” yazıyoruz. Bu komut bizim değişikliğimizi bir yere saklayacak ve istediğimiz zaman bize geri getirecek. Add komutunu veya commit komutunu kullanmamıza gerek yok. Daha sonra istediğimiz branch’a gidip işimizi hallediyoruz. Bilgiyi geri getirmek istersek hangi branch’da geri getirmek istiyorsak oraya gidip “git stash pop” komutunu kullanmamız gerekiyor. Ardından add ve commit komutları ile değişikliği kaydedebiliriz.

git stash list : Saklanan değişikliklerin listesini gösterir.

**15-)** Uygulamak istediğimiz değişikliği “git stash apply” komutu ile ekleyebiliriz. Mesela alt taraftaki örnekte stash@{0} eklenmek istemiş. O halde git stash apply yazdıktan sonra stashID sini de giriyoruz. Eğer biz pop komutunu kullanmadan direk **git stash apply** yazarsak bunu listeden atmaz ve **git stash list** yazdığımızda listede gözükür başka bir yerde bunu kullanabiliriz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

git stash clear : Bütün stashlerimizi temizler.

**16-)** Önceki commit’lere dönmek için öncelikle “git log” yazarız. Daha sonra dönmek istediğimiz commit’in numarasını kopyalamamız lazım. Ardından “git checkout” komutunun sonuna kopyaladığımız numarayı yapıştırırız. Bu durumda “Detached Head” durumu oluşur. Bunun anlamı **head**’ın ve **son** **commit**’in aynı yerde olmamasıdır.

**17-)** Eğer istersek commit’leri silebiliriz. Bunu yapmak için “git reset” komutu kullanılır. Silmek istediğimiz commit’in bir önceki commit’inin numarasını kopyalıyoruz ve **git reset** yazdıktan sonra bu numarayı yapıştırıyoruz. Bu işlemi yaptıktan sonra commit’lerimiz silinecek fakat değişiklikler kalacak. Mesela 1, 2, 3 ve 4 isimli 4 tane commit attığımızı varsayalım. Eğer ben 3. ve 4. commit'lerin silinmesini istiyorsam “git reset” yazdıktan sonra 2. commit'in numarasını yazmam lazım.

**18-)** Eğer bu değişiklikleri de silmek istiyorsak “git reset --hard” komutu kullanılır. Silmek istediğimiz commit’ten bir önceki commit’in numarasını kopyalıyoruz ve **git reset --hard** yazdıktan sonra bu numarayı yapıştırıyoruz. Bu işlemi yaptıktan sonra hem commit’lerimiz hem de yapılan değişiklikler silinecektir. Mesela 1, 2, 3 ve 4 isimli 4 tane commit attığımızı varsayalım. Eğer ben 3. ve 4. commit'lerin silinmesini ve bu commit’lerde yapılan değişikliklerin silinmesini istiyorsam “git reset --hard” yazdıktan sonra 2. commit'in numarasını yazmam lazım.

**19-)** Hem branch değiştirmemek, hem commitleri silmemek hem de yapılan değişiklikleri geri almak istiyorsak “git revert” komutu kullanılır. Mesela 1, 2, 3 ve 4 isimli 4 tane commit attığımızı varsayalım. Eğer ben 3. ve 4. commit'lerin silinmemesini ve bu commit’lerde yapılan değişikliklerin geri alınmasını istiyorsam “git revert” yazdıktan sonra geri almak istediğim commit'in numarasını yazmam lazım. Bu yeni bir commit atacak ve değişiklikleri geri alacaktır.

**20-)** Yaptığımız değişiklikleri görmek için “git diff” komutu çalıştırılır. Eğer bu komutu bu şekilde kullanırsak **git add** yapmadığımız (**staging** **area**’ya almadığımız) değişiklikleri gösterir. Mesela ben bir dosyada bir değişiklik yaptım git status ile bu değişikliği görebilirim. Git add yapmadan “git diff” yazarsam değişiklikleri gösterir.

git diff HEAD : Son commit’e göre yapılan değişiklikleri gösterir.

**21-)** Eğer 2 commit’in arasındaki farkı görmek istersek “git diff” yazdıktan sonra 1. commit’in numarasını yazıyoruz ardından bir boşluk bırakıp 2. Commit’in numarasını yazıyoruz. Eğer boşluk sıkıntı çıkarıyorsa aralarına “:” da koyabiliriz.

**22-)** Eğer 2 branch arasındaki farkı görmek istersek “git diff” yazdıktan sonra branch isimlerini yazıyoruz.

**23-)** **Rebasing (Bazı değiştirme)** anlamına gelmektedir. Mülakatlarda çok fazla sorulur. Biz farklı bir branch’de çalışırken master branch’e yeni commit’ler atılabilir. Biz bu commit’leri kendi branch’imiz ile merge edip çalışmamıza öyle devam etmek isteriz. Ama bunu yaptığımız zaman “git log” ekranımız gereksiz merge commit’leri ile dolacaktır. Biz her merge edişimizde bizim branch’ımızda merge edildiğine dair bir commit oluşacaktır. “git rebase” komutu bu merge commit’lerini görmezden gelir ve master branch’deki commit’ler ile bizim branch’imizdeki commit’leri bir sıraya sokar. Fakat bu tarihleri değiştirebilir. Yani biz kendi branch’imizde commit attıktan sonra master branch’e bir commit gelebilir. Biz “git rebase” komutunu kullandığımızda sonradan atılan commit bizim commit’imizin arkasına geçecektir. Rebase komutunu **kullanmamamız** gereken önemli bir yer vardır. Eğer projede çalışan insanlar bizim branch’imizdeki commitleri kullanmaya başladıktan sonra biz “git rebase” komutunu kullanırsak onların çalışmaları darmadağın olabilir. Bu komutu şöyle kullanıyoruz. “git rebase” yazdıktan sonra hangi branch’i sıralamak istiyorsak o branch’in **adını** bu komutun sonuna ekliyoruz.