**Git Nedir**

Git, bir sürüm/versiyon kontrol sistemidir. Yazdığımız projeleri, bilgisayarımızda ya da harici disklerde tehlike altında değil de internet üzerinde tutmamızı ve yönetmemizi sağlayan bir sistemdir.

1. **Versiyon yönetim kolaylığı**

Bilgisayarınızda, bir harici diskte ya da bir bulut sisteminde tutuyorsanız, projede bir hata yapma veya projenizin başına bir sorun gelmesi gibi olasılıkları göz önünde bulundurarak sürekli projenizi yedekleme ihtiyacı duyarsınız. Bu sebeple projenizin tüm versiyonlarını kaydetmeye başlarsınız. Projenizin büyüklüğüne göre bu işlemi yapmak oldukça karmaşık haller alabilir. Eğer Git kullanırsanız böyle bir sorununuz olmaz. İstediğiniz zaman projenizin son kaydettiğiniz haline ulaşabilir veya herhangi bir gün ki haline dönüş yapabilirsiniz.

1. **Birden fazla kişinin eş zamanlı aynı proje üzerinde çalışması**

Aynı proje üzerinde, birden fazla kişi ile eşzamanlı olarak ya da farklı zamanlarda çalışıyorsanız kodlarınızı birleştirmek ya da kod alışverişi yapmak oldukça çetrefillidir. Fakat git kullanıyorsanız bu işlemleri yapmanız hiç de zor olmayacaktır.

1. **Offline kullanılabilmesi**

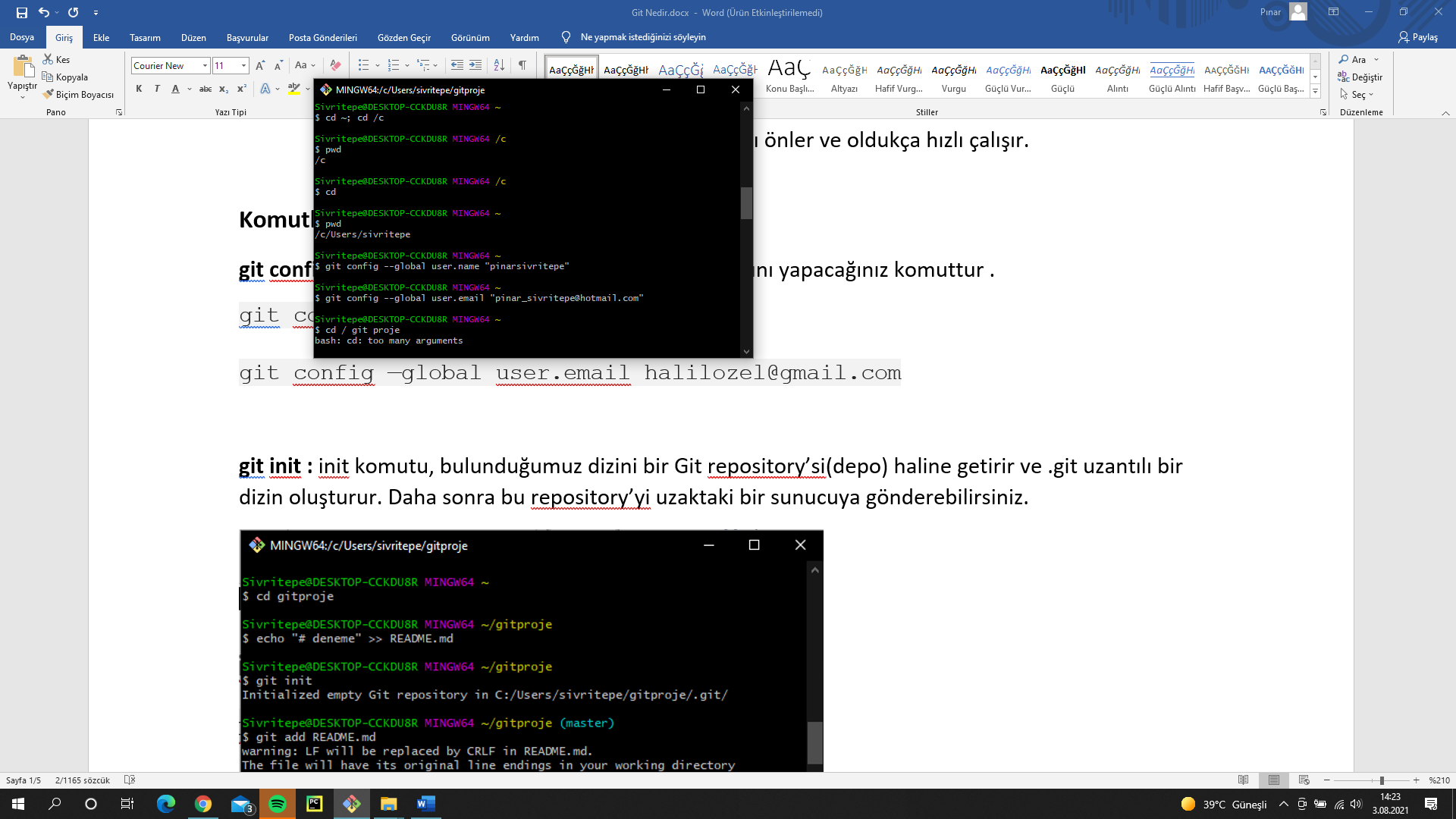
Bu özellik diğer versiyon kontrol sistemlerinden farklı olarak hiçbir ağa bağlı olmadan projenizi geliştirebilmenizi sağlar. Çoğu sistemde proje sürekli merkezi bir noktada dururken, Git‘te her kullanıcının bilgisayarında projenin bir kopyası mevcut olur ve bu sayede offline olarak da proje geliştirebilirsiniz.

1. **Hızlı olması ve az yer kaplaması**

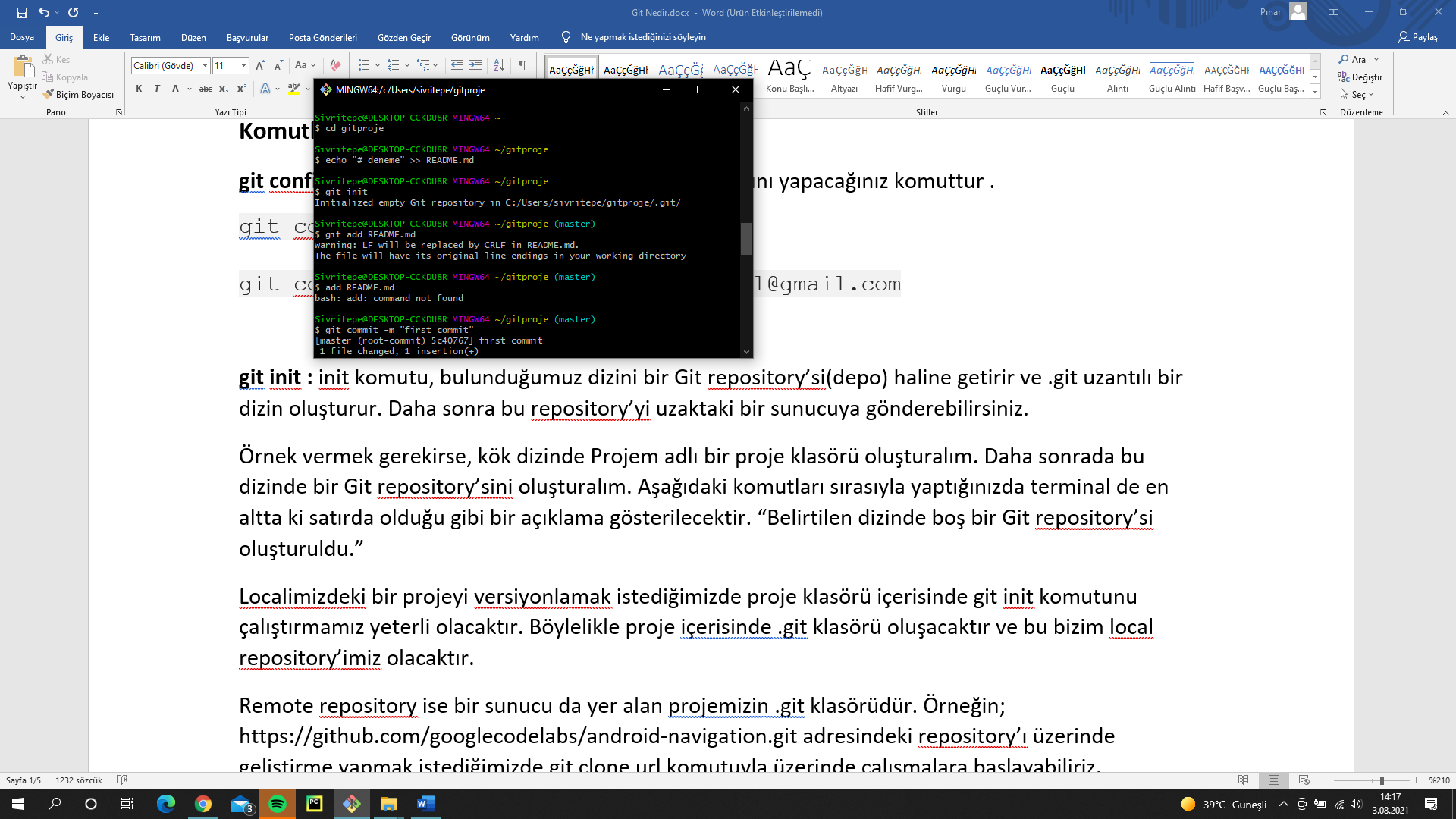
Ayrıca Git, bilgisayarınızda fazla yer kullanımını önler ve oldukça hızlı çalışır.

**Komutlar**

**git config :** Kullanıcı adı, mail gibi yapılandırma ayarlarını yapacağınız komuttur .



**git init :** init komutu, bulunduğumuz dizini bir Git repository (depo) haline getirir ve .git uzantılı bir dizin oluşturur. Daha sonra bu repository’yi uzaktaki bir sunucuya gönderebilirsiniz.



Projelerimizde görünmesini istemediğimiz dosyalar ve klasörler varsa projeye .gitignore uzantılı dosyayı eklememiz gerekiyor.

https://github.com/github/gitignore bu adresten .gitignore uzantılı dosyaların hazır templatelerini bulabilirsiniz veya ihtiyacanıza göre bu dosyaları kendinizde düzenleyebilirsiniz.

Projeye başladık ve bazı dosyaları .gitignore dosyasına eklemediğimizi farkettik. Bu gibi durumlarda .gitignore dosyasını düzenleyebiliriz. Bir başka seçenek olarak

git rm — cached “dosya uzantısı” komutunu da kullanabiliriz.

**Staging Area**

Local repository üzerinde değişiklik yaptık ve bunu sunucuya göndermek istiyoruz. Git’teki bazı komutlar yardımıyla öncelikle bu değişiklikleri staging area’ya almamız gerekir. İnternet’in olmadığı bir yerde proje geliştiriyor olabilirdik bu durumda internet bağlantısı olmadan yaptığımız tüm değişiklikleri gruplayarak staging area’ya alabiliriz. Yani staging area bize internet bağlantısı olmadan bile geliştirme yapmamızı sağlıyor.

Öncelikle commitlemek istediğim değişiklikleri belirleyip **git add “dosya uzantısı”** komutu ile staging area’ya alıyorum. Tüm değişiklikleri staging area’ya almak istersem kısaca **git add \*** komutunu kullanabilirim. Staging area’ya alma işlemi bittikten sonra değişikliklerimi **git commit -m “commit mesajı”** komutu ile commitleyebilirim.

**git commit :** Çalışma dizinimize eklenen dosyaları head(baş) denilen .git uzantılı klasöre ekler. Commit işlemi yerelde gerçekleşen bir işlemdir. Uzak sunucudaki repository’de bir değişiklik olmaz. commit komutuyla projemizde yapmış olduğumuz değişiklikleri belirteceğimiz kısa bir açıklamadır.

**git pull :** Uzak sunucudaki repository’de değişikliğe uğramış dosyalar varsa bunları indirir ve yereldeki repository ile birleştirerek dosyalarımızı günceller.

**git push :** Commit’lenmiş değişikliklerimizi uzak sunucudaki repository’ye göndermemizi sağlayan bir komuttur.

git push master origin

**git clone :** clone komutuyla local(yerel)’de ya da uzak bir sunucuda(server) yer alan bir repository(depo)’nin çalışan kopyasını bulunduğumuz dizine kopyalarız.

git clone /path/to/repository

git cloneusername@host:/path/to/repository

**1.** satırdaki komut ile localde ki dosyalarının kopyasını alırız.

**2.**satırdaki komut ile de uzaktaki bir sunucunun dosyalarının kopyasını alırız.

**git reset HEAD “dosya uzantısı”**

komutu ile staging area’dan çıkartabilirim. Yaptığım işlemleri geri almak için bunun dışında birçok git komutu bulunmaktadır. Genel olarak kullandıklarımı kısaca açıklamaya çalışacağım.

**git rm :** Belirttiğimiz bir dosyayı çalışma dizininden siler.

**git rm -f “dosya uzantısı”** Dosyayı projeden tamamen silmiş olacağız.

**git commit -amend -m “commit mesajı”** Son commit mesajını değiştirebiliriz.

**git checkout -“dosya uzantısı”** Dosya uzantısının staging area’yada olmayan değişikliklerini silip ve dosyayı commit edilen son noktaya geri döndürebiliriz.

**git reset -hard** Projede yapılan tüm değişiklikleri silip ve son commit edilen noktaya geri getirebiliriz.

**git reset [hash değerinin ilk 7 karakteri]** Commitlediğimiz değişiklikleri geri almamızı sağlar. Bu işlemi yeni bir commit oluşturmadan yapar.

**git revert [hash değerinin ilk 7 karakteri]** Commitlediğimiz bir değişikliği geri alırken projemizin tarihçesinden atmış olduğumuz commit’i silmez ve değiştirme işlemini yeni bir commit atarak gerçekleştirir.

**git status** Proje üzerinde değişiklikler yaptık ve tüm değişiklikleri liste halinde görmek istiyoruz bu durumda git status komutuna kullanabiliriz. Bu komut belli gruplar halinde tüm değişiklikleri dönecektir.

* **Untracked files:** Git tarafından takip edilmeyen tüm dosyaları burada görebiliriz.
* **Changes not staged for commit:** Bu listedeki dosyalarda değişiklik yapmışız fakat henüz staging area’ya bu değişiklikleri bildirmemişiz.
* **Changes to be committed:** Staging area’da olan commitlenmeyi bekleyen tüm dosyaları burada görebiliriz.

**Branch**

Git ağaç mantığı ile çalışmaktadır. Bir projeyi versiyonlamak istediğimizde git otomatik olarak master branch’ini oluşturur ve biz bu branch üzerinde çalışmaya başlamış oluruz. Yetkili olduğunuz bir projede değişiklik yapmak istediğinizde direk master branch’i üzerinde değişiklik yaptığınızı ve tüm ekibinde bu branch üzerinde çalıştığını düşünün olay çok karmaşık ve yönetilemez bir hal alacaktır. Bir diğer neden ise iki kişi farklı konular üzerinde çalışıyorlarsa farklı branchlerde çalıştıkları için birbirlerini etkilemeyecektir. Yaptığımız tüm değişiklikler sadece bulunduğumuz branch’i etkileyeceği için başka branch açarak farklı bir konuda da geliştirme yapabiliriz ve bunlar birbirlerini etkilemeyeceği için kodu // gibi yorum satırına almaktan veya silmek zorunda kalmaktan kurtulmuş olacağız.

**git branch branchadı** Bulunduğumuz branch’den yeni bir branch çıkabiliriz.

**git branch** Projedeki local branchleri görebiliriz.

**git branch -v** Projedeki local branchleri son commit bilgisi ile görebiliriz.

**git branch -va** Local’deki ve remote’daki branchleri liste halinde son commit bilgileri ile görebiliriz.

**git checkout branchadı** komutuyla branch’ler arasında geçiş yapabiliriz.

Oluşturduğumuz branch’lerle işimiz bittiğinde veya başka bir nedenden dolayı bu branch’leri silmek isteyebiliriz. Local’deki bir branch’i silmek istersek

**git branch -d branchadı** komutunu kullanabiliriz.

Silmek istediğimiz branch remote repository de ise

**git branch -dr branchadı**

**git push origin :branchadı** komutları yardımıyla bu branch’i remote repository’den silmiş olacağız.

**git stash** Bir branch üzerinde çalışırken tam olarak bitmeyen tüm değişiklikleri commitlemeden kayıt altına alıp branch’i ilk durumuna döndürmek isteyebiliriz. Bu durumda tam olarak git stash komutunu kullanmamız gerekir. Bu komut commitlenmemiş tüm değişikliklerimizi kayıt altına alıp branch’i ilk haline çevirir. git stash list komutu ile de geçici olarak kayıt altına alınan değişikliklerin listesini görebiliriz.

**git stash pop** Stash list’de olan en üstteki değişikliği geri yükler ve değişiklikleri silebiliriz.

**git stash apply stash@{listId}** Stash list’deki değişikliği silmeden geri yükleyebiliriz.

**git stash drop** Değişikliği geri yüklemeden stash list’den siler. Bu birazcık tehlikeli bir komut gibi duruyor

**git commit sonrası**

Bulunduğumuz branch’de commit işlemlerini tamamladık ve artık branch üzerinde yaptığımız tüm değişiklikleri remote repository’de görmek istiyoruz.

Bu branch remote repository’de ilk kez oluşturulacaksa, git push origin branchadı komutunu kullanarak branch’i remote repository’e pushlayabiliriz. Ama bu branch remote repository’de varsa ve başka geliştiricilerde bu branch üzerinde çalışıyorlarsa

**git fetch**

komutunu kullanarak başka biri tarafından değişiklik yapılıp yapılmadığına bakmamız gerekecektir. Projede aynı branch üzerinde çalıştığımız takım arkadaşlarımız var ise geliştirme yapmaya başlamadan önce bu komut yardımıyla remote branch’deki tüm değişiklikleri local branch’imize almamız gerekir. Bu iki branch arasında farklılıklar bulunuyor ise branch’ler arasında çakışmalar meydana gelebilir.

**git diff** komutu ile iki versiyon arasındaki farkları inceleyebilirim.

Farkları inceleyerek remote branch’deki tüm değişiklikleri local branch’ime alarak local branch’imi çalışır en güncel hale getirmeliyim. Bence bunun nedeni ise local branch’imi push’ladığımda hem kendi geliştirmelerim hemde remote’daki tüm geliştirmeleri içeriyor olması gerektiğidir. Tüm çakışmaları giderirsek artık local’imizdeki branch’i güncel versiyon olarak push’layabiliriz.