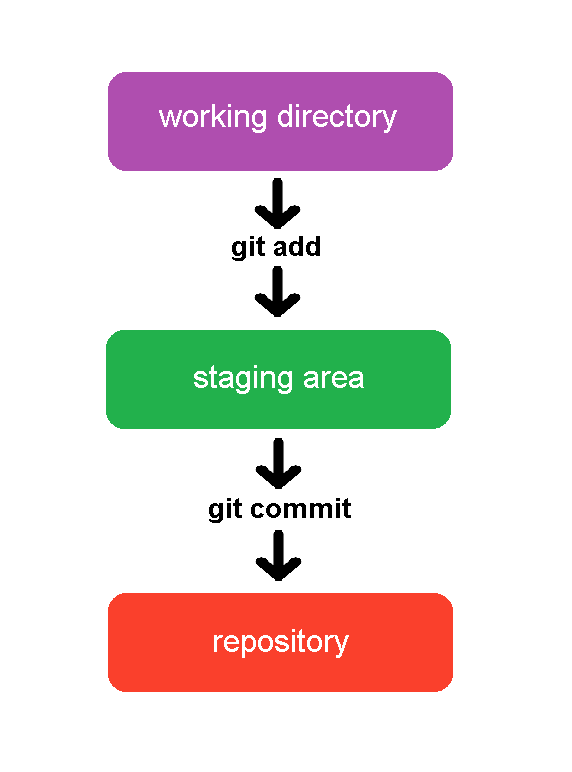
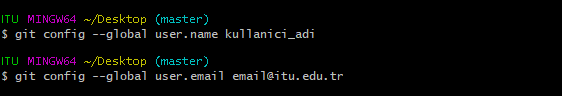
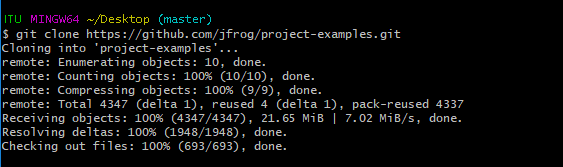
**Git Versiyon Kontrol Sistemi**

Git Versiyon Kontrol Sistemi, bir proje üzerinde birden çok kişinin çalışmasına ve her birinin kendi versiyonunu oluşturmasına, daha sonra değişiklik yapılmak istendiğinde istenilen versiyona dönülüp oradan değişiklik yapılmasına olanak veren bir kontrol sistemidir. Proje üzerinde yapılan bir değişikliğin sadece ilgili kısmını değil, projenin tamamını bir bütün halinde saklar, böylelikle projenin son halinin her geliştirici tarafından bir bütün halinde görülmesine olanak sağlar. Yerel ve uzak bilgisayarlar olmak üzere 2 ortam söz konusudur.   
  
**Git Versiyon Kontrol Sistemi’nin Tercih Edilme Sebebi**  
  
Lokal bir geliştirme ortamında Versiyon Kontrol kullanılmadan geliştirilen yazılımın diskte meydana gelecek bir sorun sonucu kaybedilme ihtimaline karşın, Versiyon Kontrol Sistemi’nin geliştirme aşamalarının tamamı farklı versiyonlar olarak kaydedilir ve olası bir soruna karşı her biri için ayrı ayrı yedekleme gerektirmeden istenildiği zaman istenilen versiyona dönülebilecek şekilde kaydedilir.  
  
Grup çalışmasına olanak sağlaması ise diğer bir kullanılma sebebidir. Bir proje üzerinde çalışan birden fazla kişinin aynı kaynak kod üzerinde yaptıkları farklı değişiklikler tüm geliştiriciler tarafından görülebilir ve her aşama versiyonlandığı için istenilen aşamadan değişiklik yapılmaya devam edilebilir. Harici disk kullanılması gibi vakit kaybına sebep olacak yöntemlere gerek kalmaksızın her geliştirici kendi istediği değişiklikleri yapabilir, diğer geliştiriciler de bu değişiklikleri görebilir ve üzerine ekleyerek devam edebilir.   
  
**Working Tree (Çalışma Ağacı) - Staging Area (Evreleme Alanı) – Local Repository (Yerel Depo)**  
Git’de üç öz alan vardır. Bunlardan ilki Working Tree olarak da bilinen Working Area’dır. Working Area, geliştiricinin o anda üzerinde çalıştığı alandır. Dosyalar bu alanda bulunur. Dosyalarda yapılan değişiklikler Working Area üzerinde görülür ve bu değişiklikler kaydedilmediği sürece kaybolur. Bunun sebebi biz Git’e dikkatini Working Area’ya vermesini söylemediğimiz sürece Git’in Working Area’daki dosyalardan ve değişikliklerden bihaber olmasıdır. Bu alanda bir geçiş yolu olmadığı için Git’in **untracked (izlenmeyen)** bölgesi olarak da bilinir. Working Tree üzerinde neler olduğunu görmek için **git status** komutu kullanılır. Bu komut ile ekranda iki farklı dosya grubu görüntülenir: Working Tree’deki dosyalar ve Staging Area’daki dosyalar.   
  
Git, dosyalardaki değişiklikleri izlemeye ve kaydetmeye başladığında Staging Area’ya geçilmiş olur. Dosyalar Staging Area’ya geçtiğinde Git artık bu dosyaları izlemeye başlamış olur ve bu dosyalardan haberdardır. Fakat bir dosya Staging Area’ya alındıktan sonra üzerinde değişiklik yapılırsa Git bunu yine görmeyecektir ve bu değişikliğin Git’e tekrar bildirilmesi gerekir. Dosyaları Staging Area’ya eklemek için **git add** komutu kullanılır. Bu komut, spesifik bir dosyayı işaret ederek kullanıldığında sadece işaret edilen dosya Staging Area’ya eklenirken, **git add .** şeklinde kullanıldığında Working Tree üzerindeki tüm dosyalar Staging Area’ya eklenir. Buradaki “.” tüm dosyaları işaret eden bir özel karakterdir.   
  
Local Repository ise .git dizinimizin içindeki her şeydir. Temel olarak tüm commit’lerimizin tutulduğu yerdir denebilir. Dosyaları ve bunlar üzerindeki değişiklikleri Staging Area’dan Local Repository’ye geçirmek için git commit komutu kullanılır. Bu komut ile Staging Area üzerindeki tüm değişiklikler  alınır, birlikte paketlenir ve Local Repository’ye yerleştirilir. Commit, son değişikliklerimizi Git’in takip etmesi için kullandığımız bir kontrol noktası olarak düşünülebilir. Commit edildikten sonra Staging Area boş kalmış olur. Local Repository’de bulunanları görmek için birkaç komut vardır. Bunlardan biri **git log** komutudur. Bu komut ile kayıt geçmişi görüntülenir. Bir kayıt ile alakalı spesifik bilgiye ulaşılmak istenirse **git show** komutu, ayrıntılı bilgi edinmek istenen commit işaret edilerek kullanılabilir.   
  
  
  
**Temel Git Komutları**  
**1. git config**   
Bu komut ile kullanıcı adı ve e-posta adresi yapılandırılır ve sonraki tüm projelerde bu kullanıcı adı ve e-posta adresi kullanılır.   
git config --global user.name kullanici\_adi  
git config --global user.email email@itu.edu.tr  
  


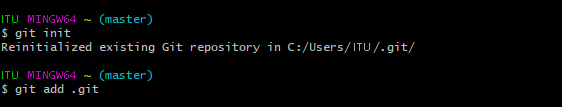
**2. git init**  
Bu komut kullanılarak bulunulan dizin boş bir git repository’si haline getirilir ve .git isimli bir dizin oluşturulur.

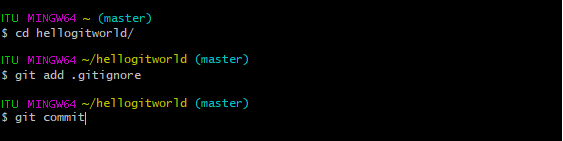
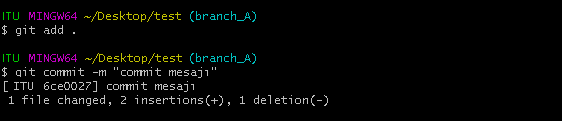
2  
  
**3. git clone**

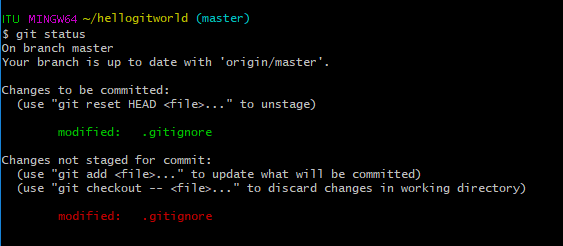
Bu komut var olan bir Git repository’sinin kopyalanması için kullanılır. Git clone komutu ile öncelikle yerel bir repository işaret edilerek bu repository’nin bir kopyası, yeni bir dizine kopyalanır. Orijinal repository lokalde olabileceği gibi uzak bir cihazda yer alıyor da olabilir.

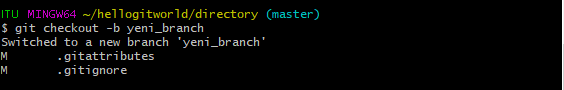


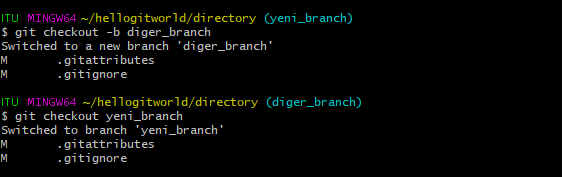
**4. git add**  
Bu komut kullanılarak işaret edilen dosya veya tüm proje, çalışılan dizine eklenir.

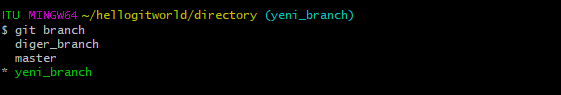
  
  
**5. git commit**  
Bu komut kullanılarak çalışılan dizinde bulunan dosyalar paketlenerek .git klasörü içindeki head isimli kısıma eklenir. Daha sonra gelen pencerede bir commit mesajı girilir. 

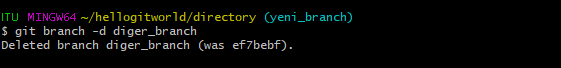
  
  
  
  
Commit mesajının ayrıca girilmesi yerine direkt olarak git commit –m komutu commit mesajı ile birlikte kullanılabilir. Bu komuttan önce mutlaka git add komutu kullanılarak commit edilmek istenen dosyaların Staging Area’ya eklenmiş olmaları gerekir.  
  
  
  
**6. git status**  
Bu komut kullanılarak üzerinde çalışılan Repository’nin o anki durumu görüntülenir. Üzerinde değişiklik yapılan dosyalar, yeni eklenmiş dosyalar ve commit komutu uygulanmamış dosyalar konsol üzerinde listelenir. 

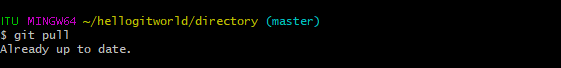
  
  
**7. git checkout –b**  
Bu komut kullanılarak yeni bir branch oluşturulur ve o branch üzerine geçilir.

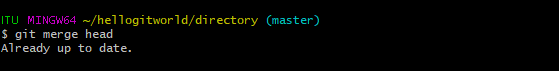
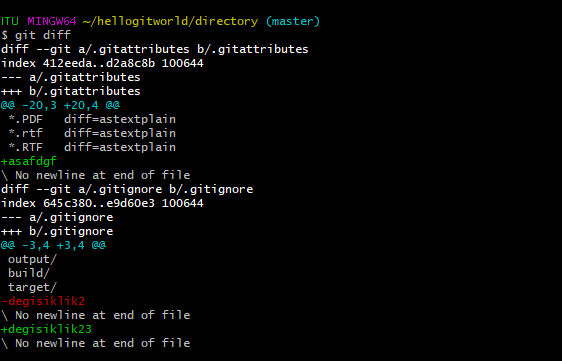
  
  
**8. git checkout**  
Bu komut kullanılarak üzerinde çalışılan branch’dan bir başka branch’a geçilir.

  
  
**9. git branch**  
Bu komut kullanılarak repository’deki tüm branch’lar listelenir ve üzerinde olunan branch görüntülenir. 

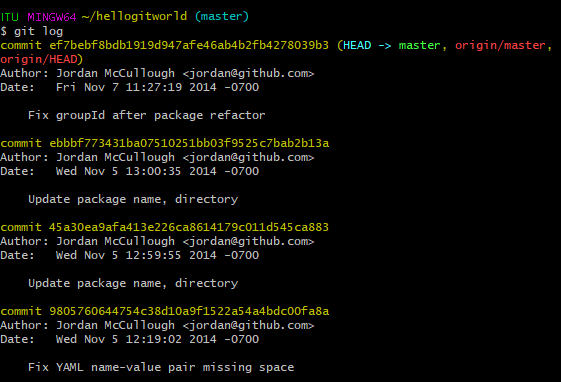
  
  
**10. git branch –d**  
Bu komut kullanılarak işaret edilen branch silinir.

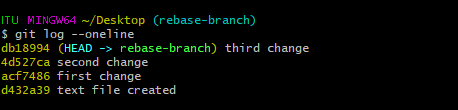
  
  
**11. git pull**  
Bu komut kullanılarak uzak bir repository’deki değişiklikler üzerinde çalışılan dizine getirilir ve bu dizin ile birleştirilir. 

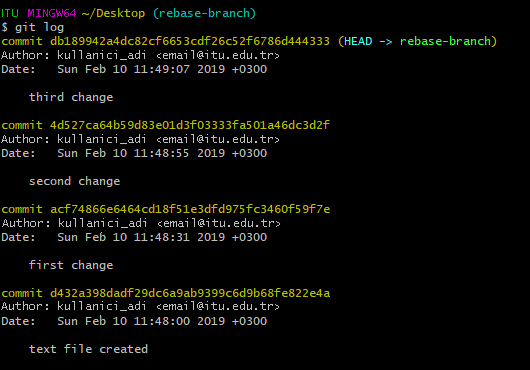
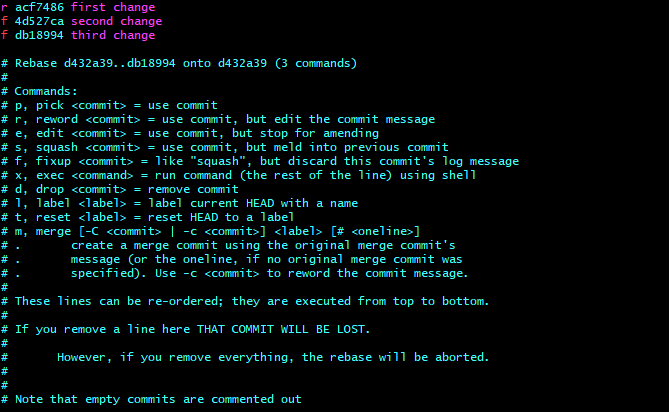
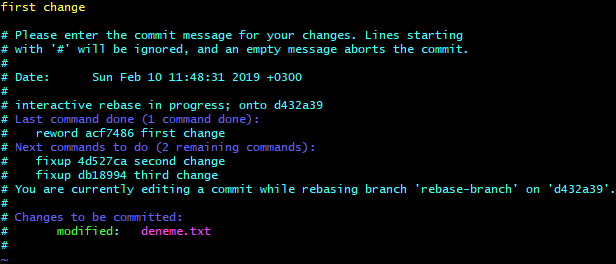
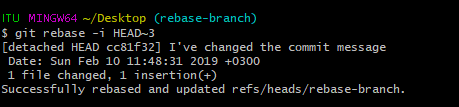
  
  
**12. git merge**  
Bu komut kullanılarak farklı bir branch, üzerinde çalışılan branch ile birleştirilir.

  
  
**13. git diff**  
Bu komut temel dosya ile olan farklılıkları gösterir.  
  


Bu komut, git diff <kaynak branch> <hedef branch> şeklinde kulanıldığında işaret edilen iki branch arasındaki farklar görüntülenir.   
  
13.2  
  
**14. git log**  
Bu komut kullanılarak proje üzerindeki kayıt geçmişi en son kayıt en üstte olacak şekilde ters kronolojik sıra ile görüntülenir. Her kayıt, o kaydı alan kişinin adı, adresi, kayıt tarihi ve kayıt mesajı ile birlikte görüntülenir. 

  
  
**15. git log --oneline**  
Bu komut kullanılarak sadece commit ID’lerinin ilk 7 karakteri ve hemen yanında commit mesajı olacak şekilde kayıt geçmişi daha temiz bir şekilde görüntülenir ve proje üzerinde yapılmış değişiklikler hakkında daha kolay bilgi edinilir.

  
  
**16. git rebase -i**  
Bir proje üzerinde uzun süre çalışılıp çok fazla değişiklik commit edildiğinde commit tarihçesi çok uzun ve karmaşık görünür. Oysa ki yapılan değişiklikler tek commit mesajı ile de açıklanabilir ve böylece projeyi inceleyen kişiler proje üzerinde yapılan değişiklikleri kısa ve öz biçimde görebilirler. Bir proje üzerinde alınan birden fazla commit, git log komutu ile görüntülenmek istendiğinde kayıt alan kişilerin ismi, adresi, kayıt tarihi ve commit mesajı ile birlikte aşağıdaki gibi görünecektir.

  
  
Tüm bu değişiklikleri tek bir commit mesajı ile belirtmek ve kayıt tarihçesini daha anlaşılır kılmak için git rebase -i HEAD komutu ile işleme başlanır. Bu komutta HEAD kelimesi branch’in en uç noktasını temsil eder ve onun yanına eklenecek sayı ile kaç commit kadar geriye gidileceği belirtilir.   
  
16.2  
  
Daha sonra gelen pencerede değişiklik yapılabilmesi için i tuşuna basılır ve INSERT (ekleme) moduna geçilir. Commit ID’lerinin başında değiştirilmek istenen commit mesajının başına r, diğer commit mesajlarının başına f yazılır. İşlem tamamlanınca ctrl ve c tuşlarına aynı anda basılır ve :wq yazılarak çıkılır.  
  
  
  
Gelen ekranda tekrar INSERT moda geçilir ve değiştirilmek istenen mesaj değiştirilerek, kayıt geçmişinde görüntülenecek ve tüm değişiklikleri açıklayacak olan tek bir mesaj belirlenip kayıt edilir.   
  
  
  
  
  
Daha sonra kayıt geçmişini görüntülemek için tekrar git log komutu kullanıldığında kayıt geçmişinde sadece belirlenen mesaj görünecektir.  
  
