...Git Versiyon Kontrol Sistemi

Şub 13, 2019

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.

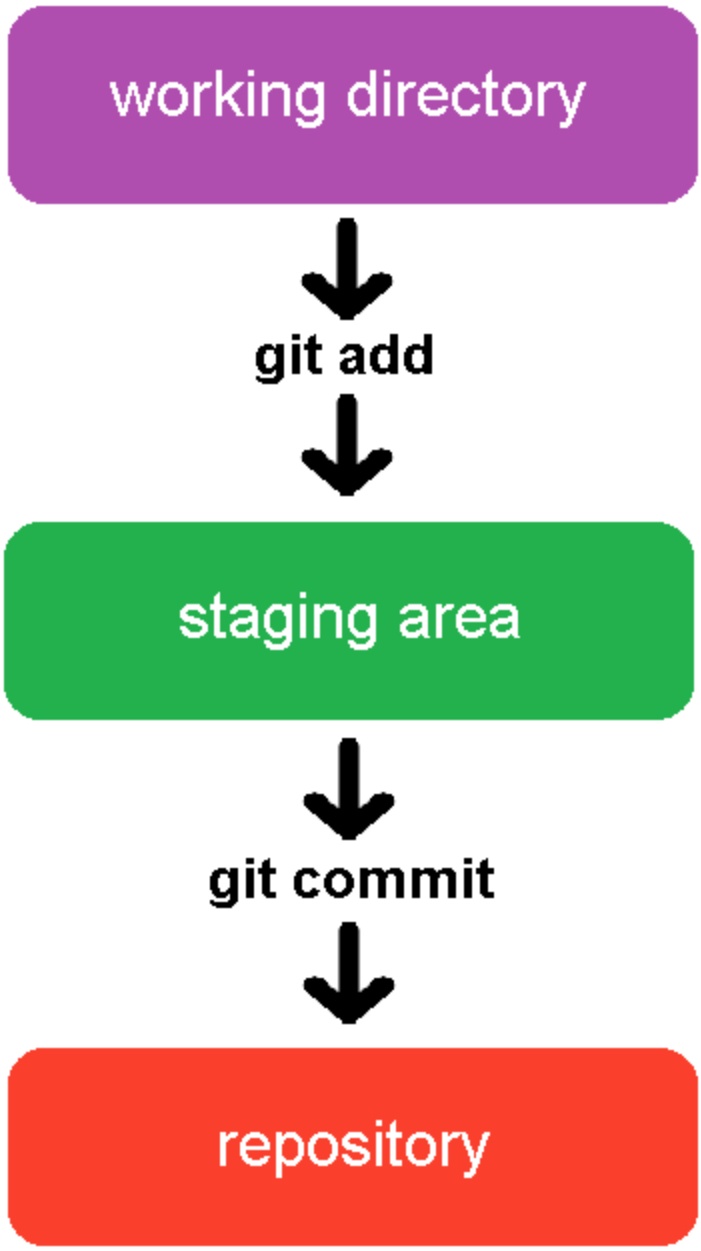
Grup çalışmasına olanak sağlaması ise diğer bir kullanılma sebebidir. Bir proje üzerinde çalışan birden fazla kişinin aynı kaynak kod üzerinde yaptıkları farklı değişiklikler tüm geliştiriciler tarafından görülebilir ve her aşama versiyonlandığı için istenilen aşamadan değişiklik yapılmaya devam edilebilir. 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

```
ITU MINGW64 ~/Desktop (master)
$ git config --global user.name kullanici_adi

ITU MINGW64 ~/Desktop (master)
$ 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.

```
ITU MINGW64 ~/Desktop (master)
$ git init
Initialized empty Git repository in C:/Desktop/.git/
```

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.

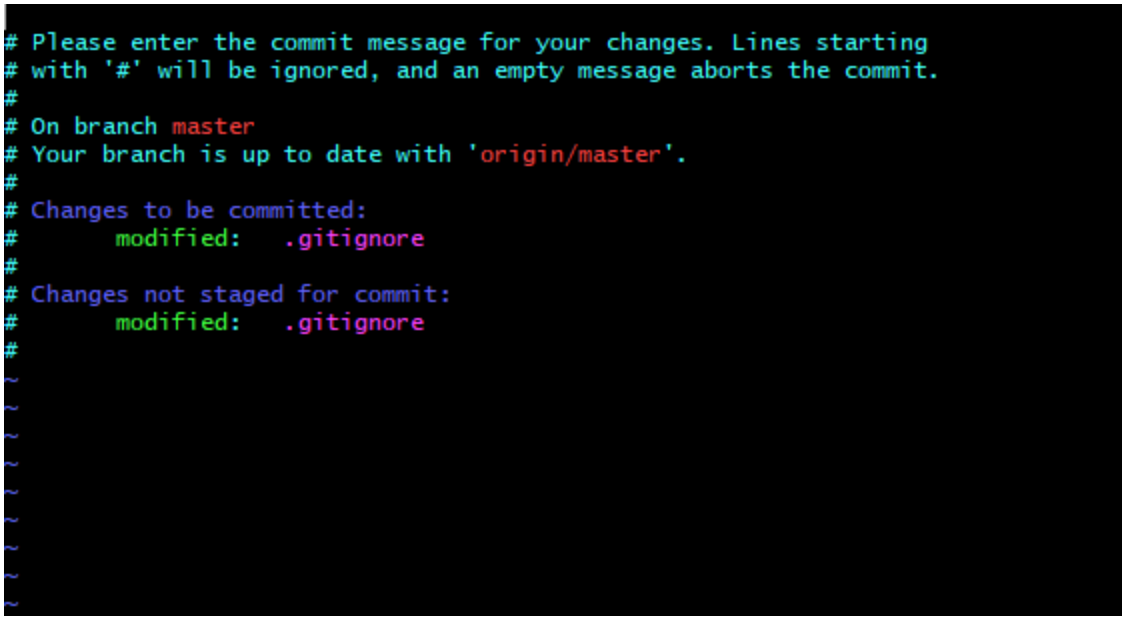
```
ITU MINGW64 ~/Desktop (master)
$ git clone https://github.com/jfrog/project-examples.git
Cloning into 'project-examples'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 4347 (delta 1), reused 4 (delta 1), pack-reused 4337
Receiving objects: 100% (4347/4347), 21.65 MiB | 7.02 MiB/s, done.
Resolving deltas: 100% (1948/1948), done.
Checking out files: 100% (693/693), done.
```

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

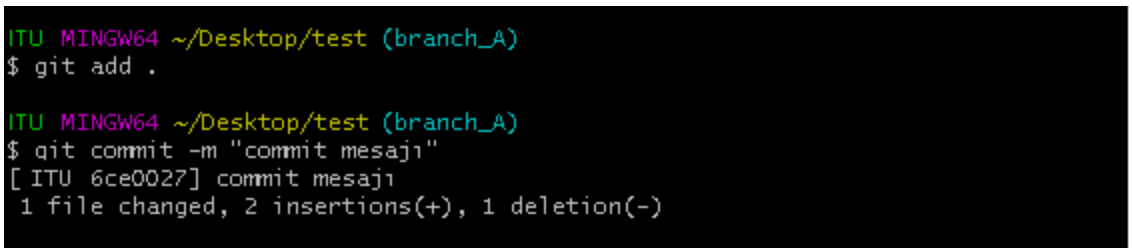
```
ITU MINGW64 ~ (master)
$ git init
Reinitialized existing Git repository in C:/Users/ITU/.git/

ITU MINGW64 ~ (master)
$ git add .git
```

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.

```
ITU MINGW64 ~ (master)
$ cd hellogitworld/

ITU MINGW64 ~/hellogitworld (master)
$ git add .gitignore

ITU MINGW64 ~/hellogitworld (master)
$ git commit
```

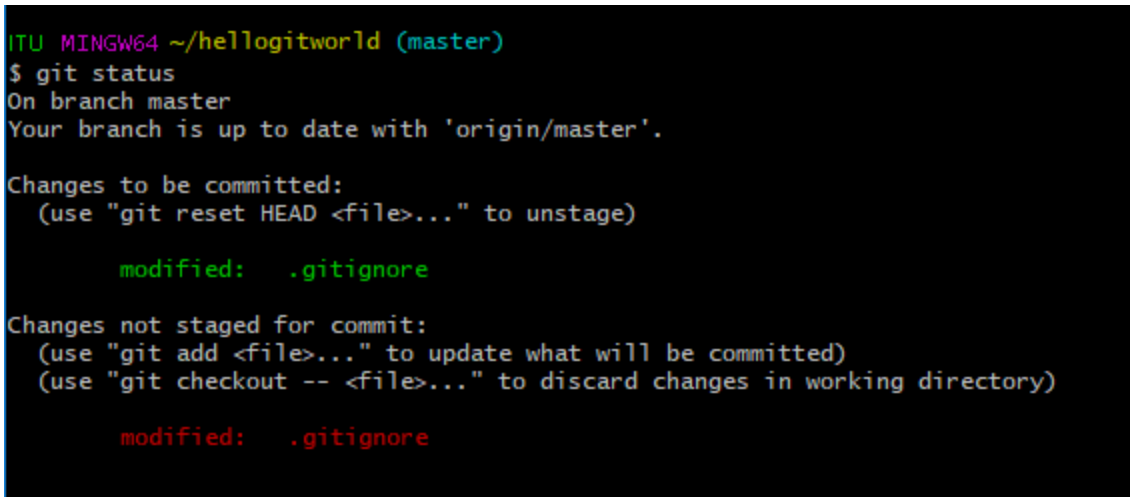


Commit mesajının ayrıca girilmesi yerine direkt olarak git commit –m komutu commit mesajı ile birlikte kullanılabilir. Bu komuttan önce mutlaka git add komutu kullanılarak commit edilmek istenen dosyaların Staging Area'ya eklenmiş olmaları gerekir.



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.

```
ITU MINGW64 ~/hellogitworld/directory (master)
$ git checkout -b yeni_branch
Switched to a new branch 'yeni_branch'
M       .gitattributes
M       .gitignore
```

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

```
ITU MINGW64 ~/hellogitworld/directory (yeni_branch)
$ git checkout -b diger_branch
Switched to a new branch 'diger_branch'
M       .gitattributes
M       .gitignore

ITU MINGW64 ~/hellogitworld/directory (diger_branch)
$ git checkout yeni_branch
Switched to branch 'yeni_branch'
M       .gitattributes
M       .gitignore
```

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

```
ITU MINGW64 ~/hellogitworld/directory (yeni_branch)
$ git branch
  diger_branch
  master
* yeni_branch
```

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

```
ITU MINGW64 ~/hellogitworld/directory (yeni_branch)
$ git branch -d diger_branch
Deleted branch diger_branch (was ef7bebf).
```

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

```
ITU MINGW64 ~/hellogitworld/directory (master)
$ git pull
Already up to date.
```

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

```
ITU MINGW64 ~/hellogitworld/directory (master)
$ git merge head
Already up to date.
```

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

```
ITU MINGW64 ~/hellogitworld/directory (master)
$ git diff
diff --git a/.gitattributes b/.gitattributes
index 412eeda..d2a8c8b 100644
--- a/.gitattributes
+++ b/.gitattributes
@@ -20,3 +20,4 @@
 *.PDF   diff=astextplain
 *.rtf   diff=astextplain
 *.RTF   diff=astextplain
+asafdgf
\ No newline at end of file
diff --git a/.gitignore b/.gitignore
index 645c380..e9d60e3 100644
--- a/.gitignore
+++ b/.gitignore
@@ -3,4 +3,4 @@
 output/
 build/
 target/
-degisiklik2
\ No newline at end of file
+degisiklik23
\ No newline at end of file
```

Bu komut, git diff <kaynak branch> <hedef branch> şeklinde kulanıldığında işaret edilen iki branch arasındaki farklar görüntülenir.

```
ITU MINGW64 ~/hellogitworld/directory (master)
$ git diff branch1 yeni_branch
```

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.

```
ITU MINGW64 ~/hellogitworld (master)
$ git log
commit ef7bebf8bdb1919d947afe46ab4b2fb4278039b3 (HEAD -> master, origin/master,
origin/HEAD)
Author: Jordan McCullough <jordan@github.com>
Date:   Fri Nov 7 11:27:19 2014 -0700

    Fix groupId after package refactor

commit ebbbf773431ba07510251bb03f9525c7bab2b13a
Author: Jordan McCullough <jordan@github.com>
Date:   Wed Nov 5 13:00:35 2014 -0700

    Update package name, directory

commit 45a30ea9afa413e226ca8614179c011d545ca883
Author: Jordan McCullough <jordan@github.com>
Date:   Wed Nov 5 12:59:55 2014 -0700

    Update package name, directory

commit 9805760644754c38d10a9f1522a54a4bdc00fa8a
Author: Jordan McCullough <jordan@github.com>
Date:   Wed Nov 5 12:19:02 2014 -0700

    Fix YAML name-value pair missing space
```

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.

```
ITU MINGW64 ~/Desktop (rebase-branch)
$ git log --oneline
db18994 (HEAD -> rebase-branch) third change
4d527ca second change
acf7486 first change
d432a39 text file created
```

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.

```
ITU MINGW64 ~/Desktop (rebase-branch)
$ git log
commit db189942a4dc82cf6653cdf26c52f6786d444333 (HEAD -> rebase-branch)
Author: kullanici_adi <email@itu.edu.tr>
Date:   Sun Feb 10 11:49:07 2019 +0300

    third change

commit 4d527ca64b59d83e01d3f03333fa501a46dc3d2f
Author: kullanici_adi <email@itu.edu.tr>
Date:   Sun Feb 10 11:48:55 2019 +0300

    second change

commit acf74866e6464cd18f51e3dfd975fc3460f59f7e
Author: kullanici_adi <email@itu.edu.tr>
Date:   Sun Feb 10 11:48:31 2019 +0300

    first change

commit d432a398dadf29dc6a9ab9399c6d9b68fe822e4a
Author: kullanici_adi <email@itu.edu.tr>
Date:   Sun Feb 10 11:48:00 2019 +0300

    text file created
```

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.

```
ITU MINGW64 ~/Desktop (rebase-branch)
$ git rebase -i HEAD~3
```

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.

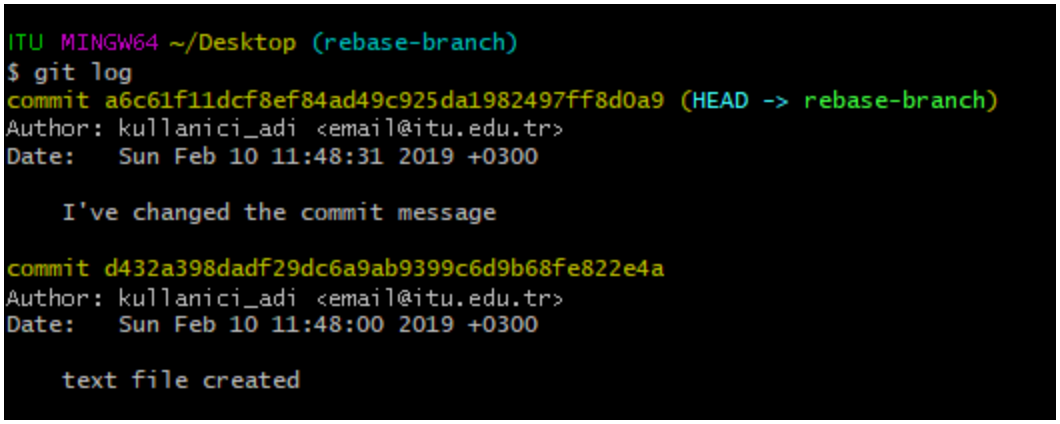
```
r acf7486 first change
f 4d527ca second change
f db18994 third change

# Rebase d432a39..db18994 onto d432a39 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
#       However, if you remove everything, the rebase will be aborted.
#
#
# Note that empty commits are commented out
```

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.

```
first change

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun Feb 10 11:48:31 2019 +0300
#
# interactive rebase in progress; onto d432a39
# Last command done (1 command done):
#    reword acf7486 first change
# Next commands to do (2 remaining commands):
#    fixup 4d527ca second change
#    fixup db18994 third change
# You are currently editing a commit while rebasing branch 'rebase-branch' on 'd432a39'.
#
# Changes to be committed:
#       modified:   deneme.txt
#
~
```

```
ITU MINGW64 ~/Desktop (rebase-branch)
$ git rebase -i HEAD~3
[detached HEAD cc81f32] I've changed the commit message
 Date: Sun Feb 10 11:48:31 2019 +0300
 1 file changed, 1 insertion(+)
Successfully rebased and updated refs/heads/rebase-branch.
```

Daha sonra kayıt geçmişini görüntülemek için tekrar git log komutu kullanıldığında kayıt geçmişinde sadece belirlenen mesaj görünecektir.


ITU MINGW64 ~/Desktop (rebase-branch)
$ git log
commit a6c61f11dcf8ef84ad49c925da1982497ff8d0a9 (HEAD -> rebase-branch)
Author: kullanici_adi <email@itu.edu.tr>
Date: Sun Feb 10 11:48:31 2019 +0300
I've changed the commit message
commit d432a398dadf29dc6a9ab9399c6d9b68fe822e4a
Author: kullanici_adi <email@itu.edu.tr>
Date: Sun Feb 10 11:48:00 2019 +0300
text file created