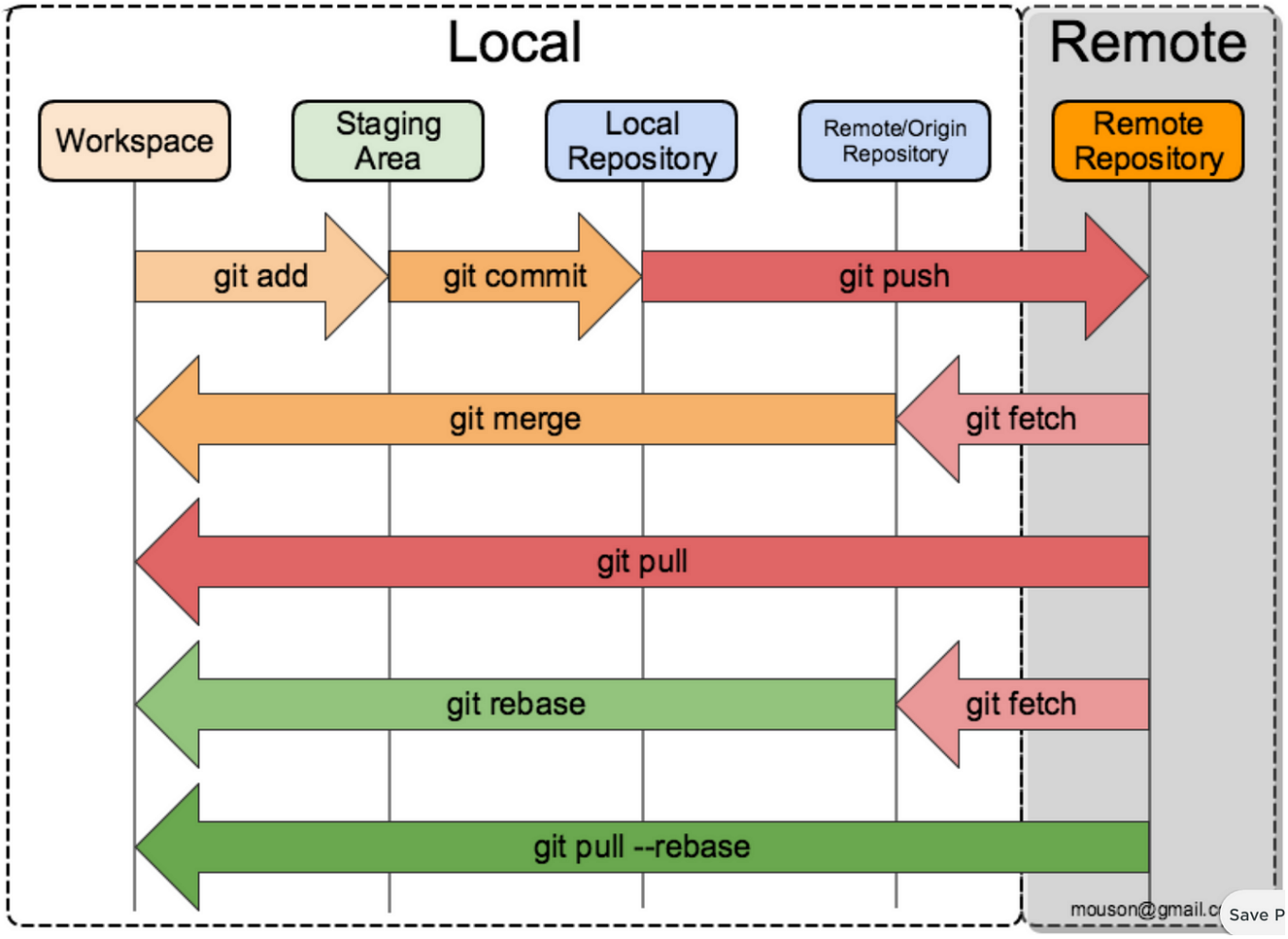
Önceki kısımlarda bir local ve remote repo oluşturmuştuk. Şimdi local repomuzda yapacağımız değişiklikleri remote repoya gönderelim. Ancek öncesinde aşağıdaki görsele göz atmak oldukça faydalı olur.




## Local Değişikleri Remote Repo’ya Gönderme

- localrepo/program.txt dosyasına print(“bug fix”) metnini ekleyelim. ve “bugfix” yorumu ile commitleyelim.

```
print("selam, ben başka yazılımcı")
print("bugfix")
```

- Şu an localrepo’muzda değişiklik yaptık ancak origin’in bizden haberi yok. Yaptığımız değişiklikleri remote repo’ya gönderelim.
  Aşağıda **git push** kodları başka bir repoya göndereceğimizi ifade ediyor.

**origin (veya remote)** halihazırda tanımlanmış uzak repomuzu işaret ediyor. Aslında bu kelime yerine bir url adresi de yazabiliriz.
**master** ise uzak repo'nun hangi branchine göndereceğimizi ifade ediyor.

```
git push origin master
```

Bu noktada hata alıyoruz. Çünkü remote repo'daki branch halihazırda aktif olarak kullanılıyor. Bu sorunu çözmek için remote repo, **bare repo'**ya çevirilebilir ya da remote repo üzerinde bir branch açılıp yeni branche geçilebilir. Ancak bunları uygulamayacağız çünkü zaten bu şekilde local repoda bir branch açmadan kodları göndermek de iyi bir pratik değil.

- Localdeki son yaptığımız commiti silelim.
  HEAD~1, HEAD'den bir önceki commiti ifade ediyor.
  –hard çalışma alanımızı yani localrepo/program.txt'yi de silip eski versiyonunu yükleyecek.
  Daha sonra silme işleminden detaylı olarak bahsedeceğim.

```
git reset --hard HEAD~1
```

Artık son yaptığımız commit sanki hiç yaşanmamış gibi.

- Local değişiklik yapmadan önce bir branch açalım. Yeni bir branch açmadan önce remote reponun güncel halini aldığımızdan emin olmak güzel alışkanlık.

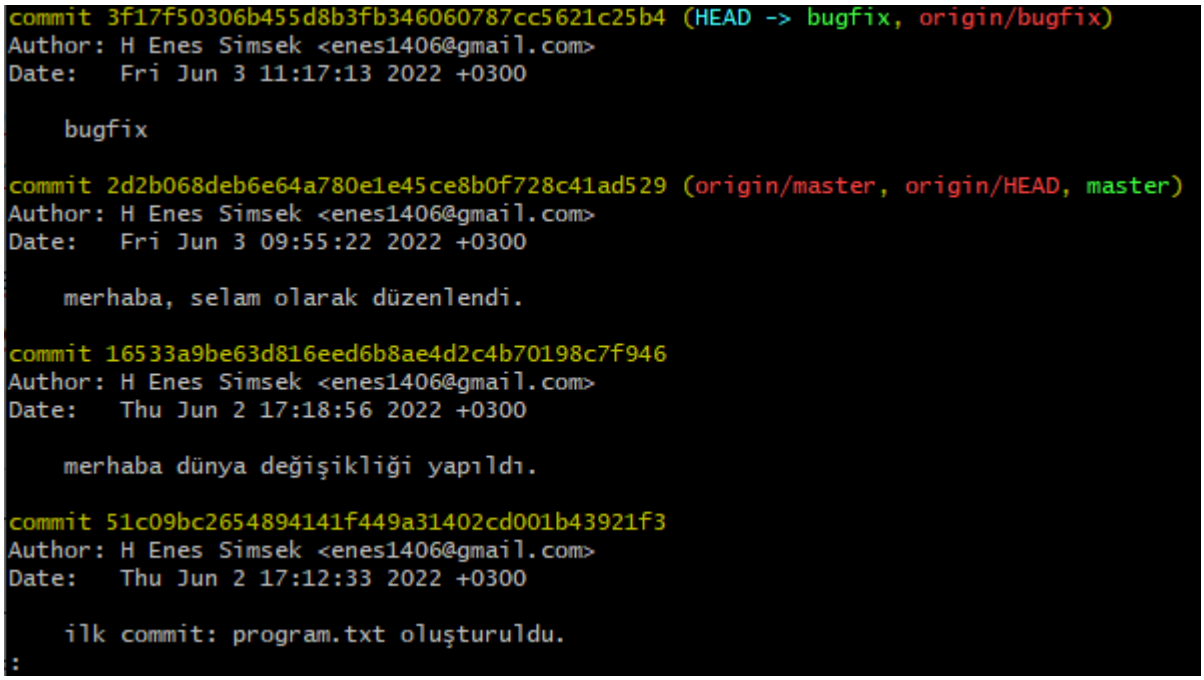
```
git fetch
git checkout -b fixbug
git log
```

fixbug isimli bir branch açtık ve master'dan yeni branch'e geçtik. Yeni branchimiz master ile aynı geçmişi bize gösteriyor.

- İlk yaptığımız değişikliği tekrar yapalım localrepo/program.txt dosyasına print("bug fix") metnini ekleyelim. ve "bugfix" yorumu ile commitleyelim.
  Artık bugfix branch'i yeni commit'imizi içeriyor.
- Local commiti remote'a gönderelim. Fakat bu sefer dikkat edelim local/bugfix branch'ini remote/bugfix branch'ine gönderiyoruz. Remote'un bugfix isimli bir branchten haberi yok ama push sonrası yeni bir bugfix branchi açacak.

```
git push origin bugfix
```

- Şimdi **git log** ile duruma localrepo gözünden bakalım.




Halihazırda bugfix branch'i içinde olduğunuz için master branch'inin detaylarını göremiyoruz. (git log –all kullanmadık) HEAD bizim için bugfix'i ifade ediyor.
localrepo'da master önceki committe kaldı. Bugfix bir commit ileride. HEAD, bugfix'teki son commiti ifade ediyor.
remote repo'da master önceki committe kalmaya devam ediyor. bugfix isimli yeni branch'i son commiti gösteriyor.
Özetle local ve remote repolarımızın hem master'ı hem de bugfix'i birbiriyle aynı.

# Merge

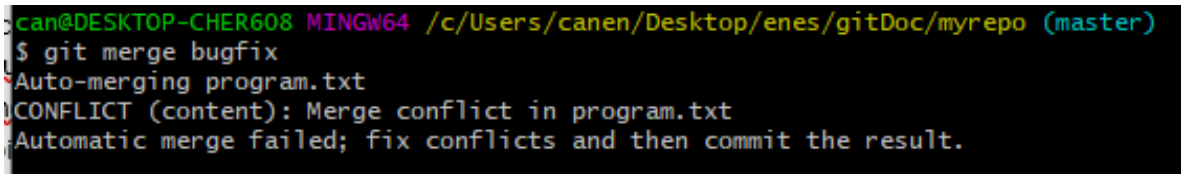
- Bu noktada oldukça muhtemel bir olay yaşandığını farz edelim. origin/master'a başka birisi başka bir commit yapsın. Yeni bir repo, yeni bir branch açıp remote'a pushlamakla uğraşmamak için git bash'i remote repo üzerinde açalım ve bir değişiklik yapıp commitleyelim.
  myrepo/program.txt içindeki şeyleri silip print("yeni dizayn") yazalım. ve commitleyelim.
- Son durumda neler olduğunu inceleyelim.
  remote repo üzerinde hem master hem de bugfix branchleri üzerinde aynı dosyada değişiklikler var. Şimdi remote repo'yu kontrol eden kişi olarak iki farklı yazılımcının bu iki farklı çalışmasını birleştirmek (**merge**) istiyoruz. Ancak **conflict** meydana geldi.

- Son durumda localrepo'daki kişi remote repo'yu yöneten kişiye **pull request** yapabilir. Yani benim çalışmalarım tamamlandı hadi artık merge yapın isteğinde bulunabilir. Ancak şu remote repo'yu da bizim yönettiğimizi düşünelim ve pull requesti unutalım.
- remote repo üzerinde bash'i açalım ve master branch'ine geçelim.

```
git checkout master
```

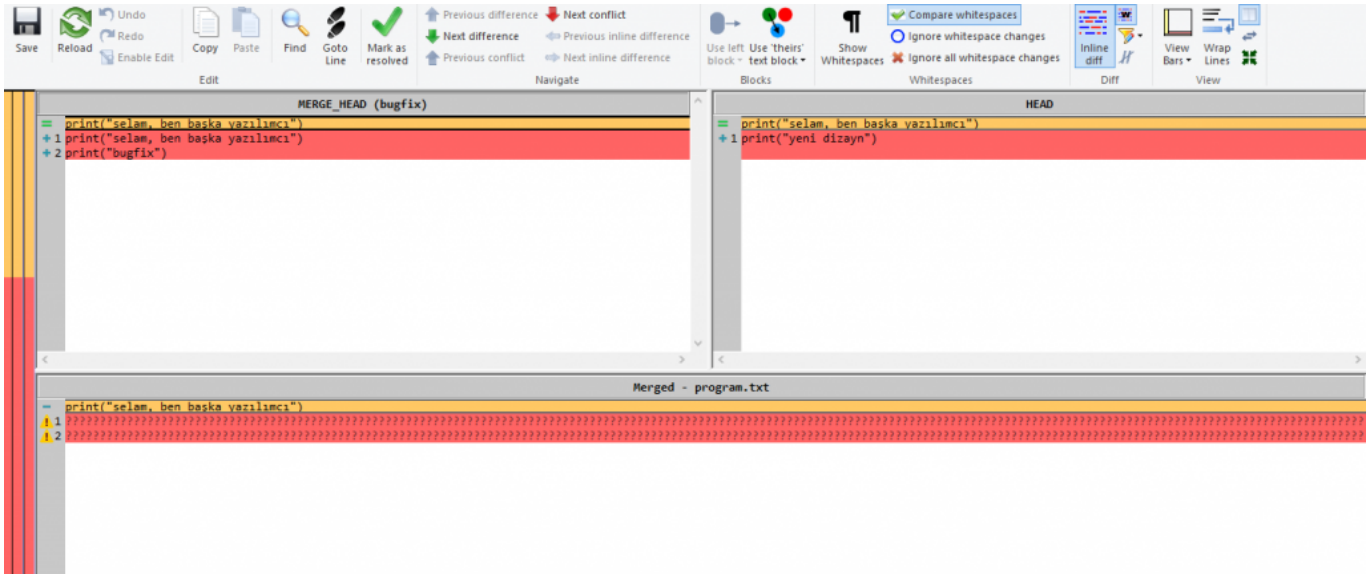
- Şimdi merge işlemini yapalım. Komutta belirteceğimiz branch'i (bugfix), hali hazırda açık olan master branch'ine merge edeceğiz.

```
git merge bugfix
```



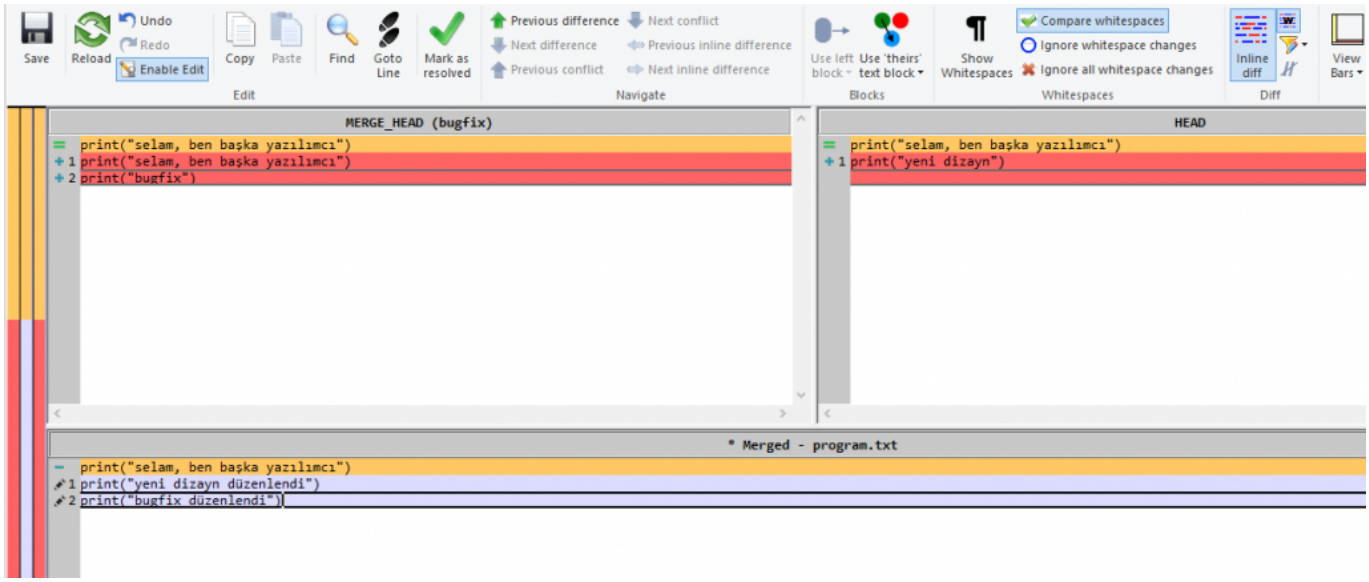

Otomatik merge, conflict yaşandığı için gerçekleşemedi. (Eğer mesela remote repo/master üzerinde commit olmasaydı conflict yaşanmayacağı için bugfixteki değişiklik master'a yeni bir commit olarak eklenecekti)
Şimdi manuel olarak bu durumu çözmeliyiz. Bunun için çeşitli **merge tool'lar** git ile entegre halde geliştirilmişler. Bu yazıda anlatmayacağım ancak bir merge tool (gui'si olanlardan kullanmanızı tavsiye ediyorum) yükleyip kullanabilirsiniz. link üzerinden birini konfigure edebilirsiniz. Ben **Git Tortoise** kullanıyorum

Git Tortoise üzerinden diff kısmını program.txt için açıyorum. Aşağıdaki gibi bir arayüz belirecek.




Burda soldaki kısım remote/bugfix’teki son commitimizi, sağdaki kısım ise açık olan branch’i yani master’daki son commit’i gösteriyor. Aşağıdaki kısım ise merge sonrası oluşacak program.txt dosyası olacak. Bu aracı kullanarak aşağıdaki gibi yeni dosyamı oluşturuyorum.

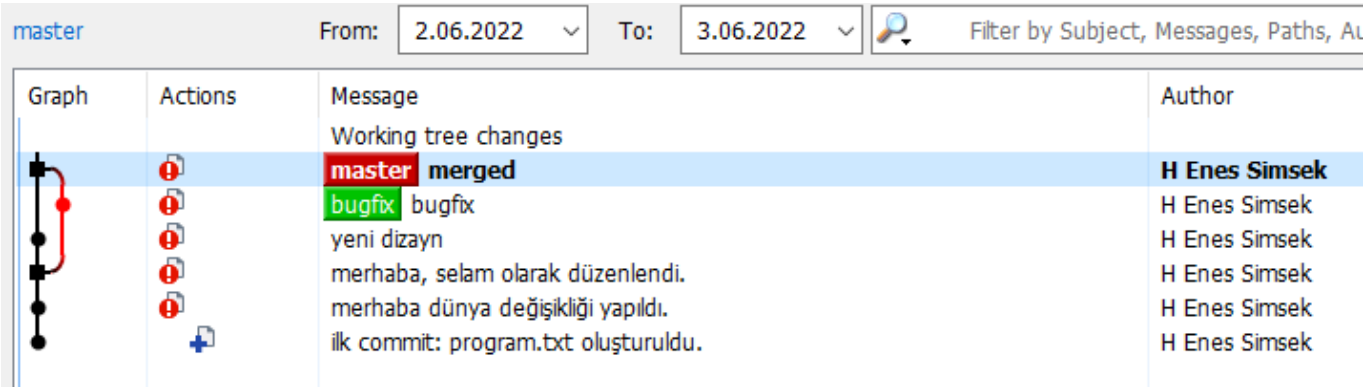
- Merge işlemimi tamamlayıp, mark as resolved butonuna basıyorum ve merge dosyam artık hazır. Bu dosya için conflict çözüldü ancak merge bitmedi.




- Yeni bir commit ile mergeleyebiliriz.

```
git commit -m “merged”
```

Tortoise üzerinde git log’a benzer işlem ile yapılan işlemleri şöyle görüyoruz.




- Merge işlemini remote üzerinde yaptığımız için local repo’da git pull ile merge edilmiş yeni versiyonu almayı unutmayalım.
- Eğer bugfix ile işimiz bittiyse bu branch’i hem local’den hem de remote’tan silebiliriz.