

Git iş dünyamızda her zaman olacak. Linux u geliştirirken kullanıla bir versiyon sistemiydi Git

Bir tane yazılım şirketinde çalışıyoruz iki tane developer var:

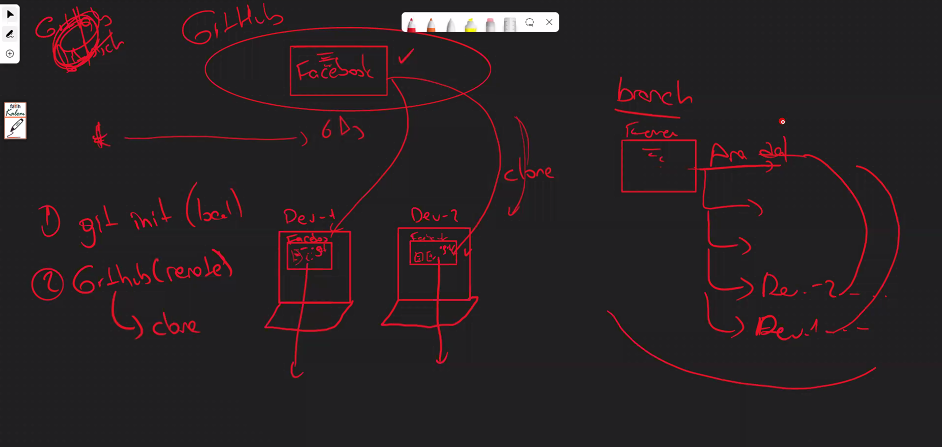
Bir yazılım geliştirecez, Facebook uygulamasını geliştirecek olsunlar. Bu süreç 6 ay ile 12 ay arasında sürecek ve binlerce satır kod yazacaklar. Birçok hata alacaklar. Bu nedenle çok iyi bir versiyonlama lazım. Bu iki developer ın ayrı ayrı yazdığı yazılımlar bir araya getirilip sorunsuz çalışması lazım. Git bunların hepsini sağlıyor.

Birden fazla developer ın çalıştığı projelerde ortak bir saklama alanı lazım. Bu da GitHub. (birçok alternatifi var.)

Bİr projeye başlayacağınız zaman önce GitHub ta bir repocery oluşturursunuz. Sonra o repocery i kendi bilgisayarınıza kopyalarsınız. Ve artık bu iki repo (local ve remote repo) arasında bir bağlantı vardır. Bir komutla local repodaki veriyi remota repoya ya da remote repodaki veriyi local repoya gönderebiliriz.

Bu kopyalamak "clone" komutuyla olur.

Peki bu iki developer aynı dosya üzerinde (app.py diyelim) farklı kodlar yazıyorlar. Bu iki farklı çalışmalar aynı dosya üzerinde büyük sorunlar oluşturur. Bu nedenle branch lar var. (dal) Yani ayrı developerlar ayrı branchlarda çalışıyorlar. Sonrasında bu branchlar ana dalda birleştiriliyor.



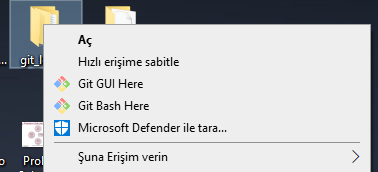
Git dediğimizde ilk aşamada öğrenmemiz gerekenler;

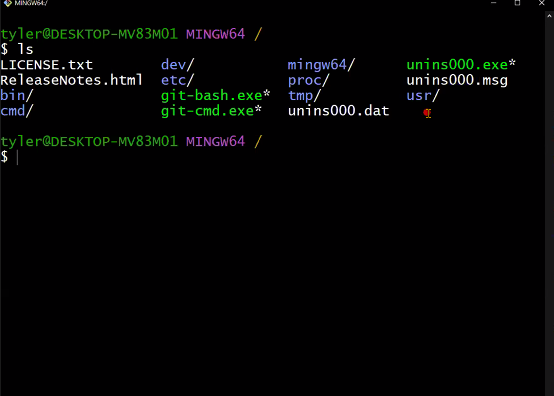
Remote Repo

Local Repo

Branch

Git Bash te bir klasörü açmak için o klasörün üzerine gidip sağ tıkladığımızda "Git Bash Here" a bastığımızda açılır:





Yukarıdaki ekranda Git Bash in içinde "root directory" klasöründeyiz. Bunu / işaretinden anlıyoruz.

Linux te / işareti root directory dir.

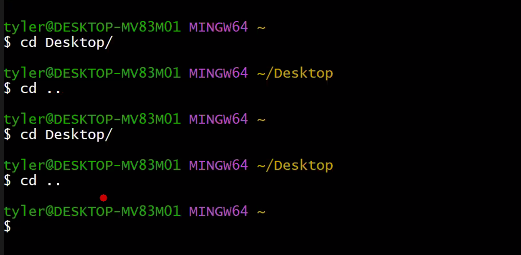
Folder Windowsta, Directory Linuxta klasör demektir.



Sadece cd komutunu girince home directory ye giriş yapmış olduk. Bunu pwd yle de görebiliyoruz.

Root klasör sisteminin en köküdür. Home root un altındadır.

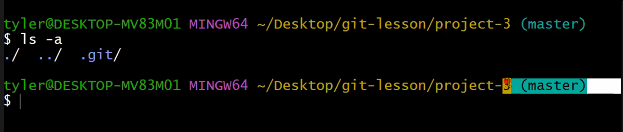
Komut girerken; örneğin desktop a gitmek için "cd des" yazıp tab tuşuna basarsam devamını "cd Desktop/" şeklinde tamamlar. Ancak bu klasörden sadece bir tane olması gerekir.



Bir üst klasöre geri gitmek için "cd .." komutunu kullanıyoruz.

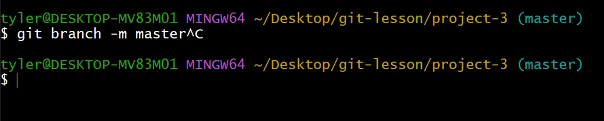
Git Bash te (Linuxta) yeni bir klasör oluşturmak için kullanacağımız komut : "mkdir klasörismi"

"ls –a" komutu gizli dosyaları gösteriyordu. "ls –al" gizli dosyaları liste halinde ve aytıntılarıyla gösterir.

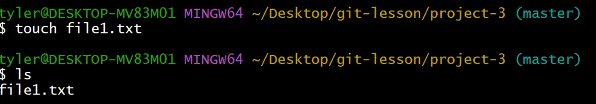


Buradaki master branch ı gösteriyor. Main de yazabilir burada. Main yazıyorsa aşağıdaki komutla master yaparız:

"git branch -m master"



Linuxte dosya oluşturmanın birçok komutur var ancak en kolayı "touch dosyaismi.dosyauzantısı"





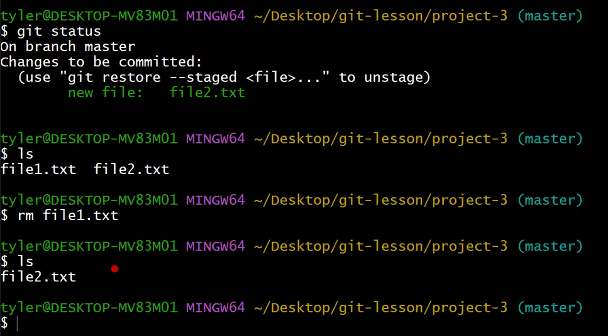
İki nokta bir üst directory i simgelerken . (tek nokta) içinde bulunduğumuz directory i simgeler."git add ." Bütün değişiklikleri staging areaya atar.

Bir dosyayı editlemek için vim editörü kullanabiliriz. "vim file1.txt" komutunu çalıştırıyoruz. Açılan kısım vim editörü. Burada yazı yazabilmem için önce i sonra enter a basmam lazım.



"merhaba" yazdım. Bunu kaydediğ çıkmak için: Esc ye basıyoruz. :wq yazıp enter a basınca çıkıyoruz. (w;write q;quit)

Vim editör dosyasından kaydetmeden çıkmak için sırasıyla; "esc" sonra ":" sonra da "q!" komutlarını girip enter'lıyoruz.



Önce file2 ekledik. Sonra file1 i silmek için rm file1 komutunu kullanıyoruz.

Bir dosyayı başka bir dosyanın  ismini değiştirme (aslında yerine taşıma) mv komutuyla oluyor.



"git log –oneline" komutu "git log" komutunda göre daha özet gösterir:

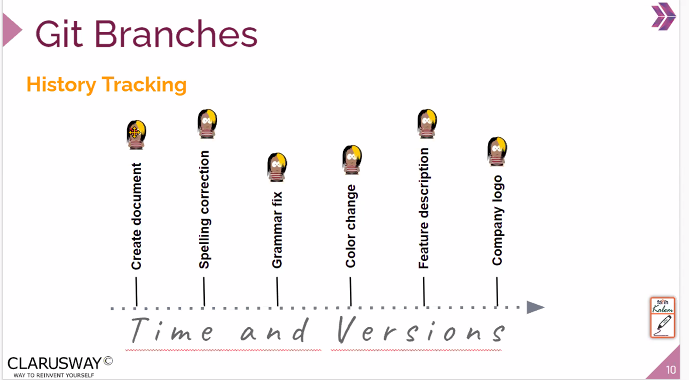


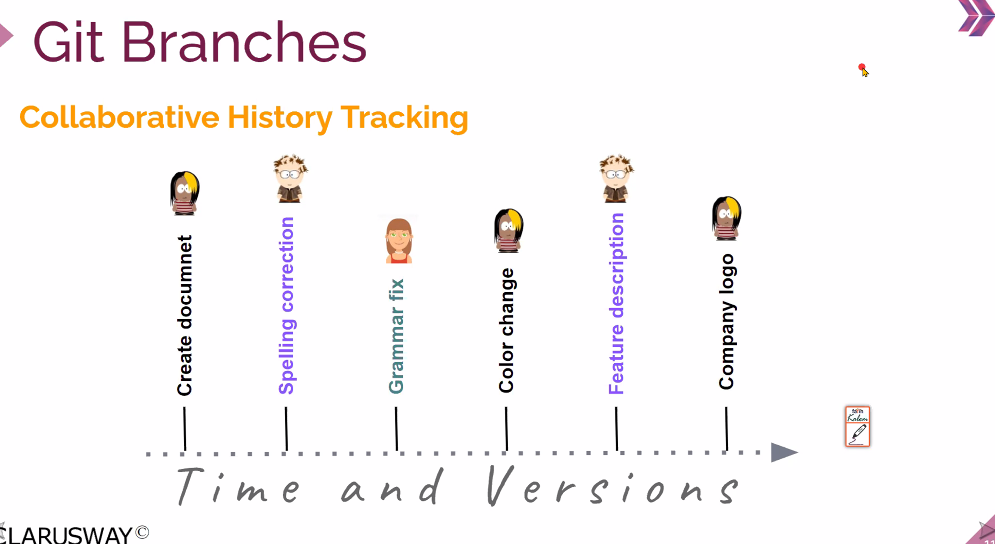
Staging aredan bir dosyayı unstage etmek için "git restore --staged <file>" komutunu kullanırız.



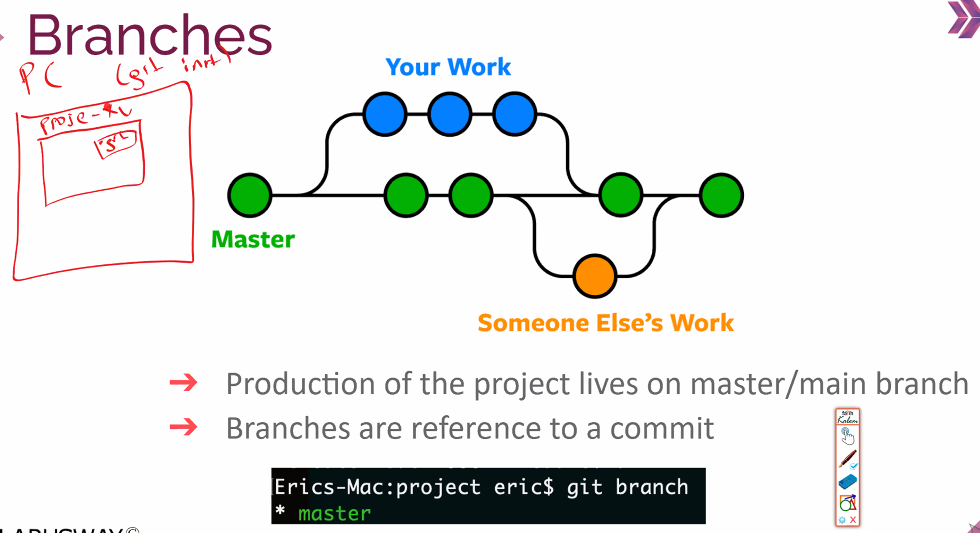


"git commit –am "…" komutu daha önce commit olan bir dosya üzerinde değişiklik yaptıktan sonra hem add hem commit komutunu aynı anda yapan bir komuttur.



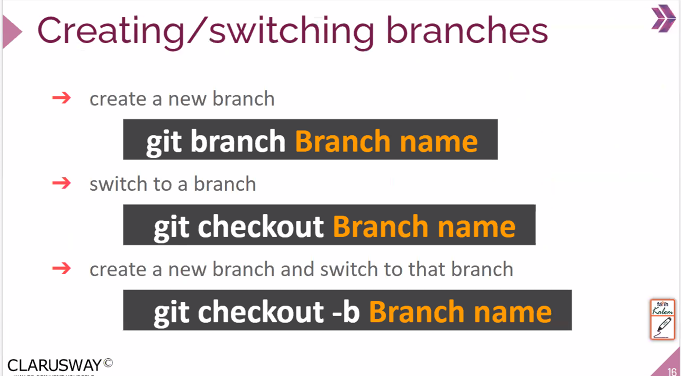


Bir kişi tek başına kod yazıyorsa çok fazla sorun yok. Birden fazla kişi aynı dosya üzerinde aynı anda çalışmaya başladığında problemler oluşuyor: Git de bunu branch larla çözüyor. Branch karmaşaya neden vermeden aynı dosya üzerinde farklı kişilerin faklı taskleri yapmasına olanak sağlıyor.



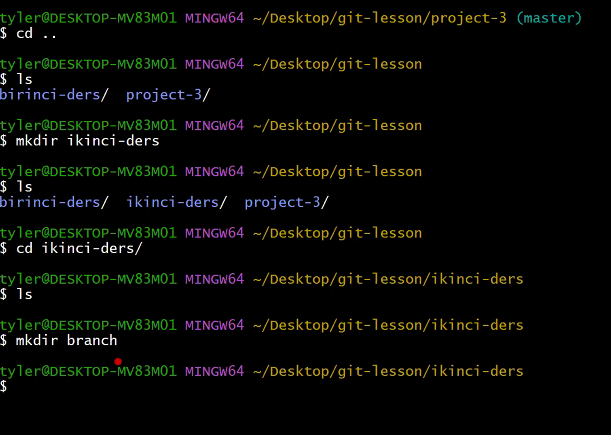
ABD deki siyahi bir vatandaşın öldürülmesinden sonra GitHub master ı main yaptı. GitHub ta oluşturulan bir reponun ana branch ı default olarak main oluyor.

Farklı developerların yaptığı çalışmalar ana branch ı etkilemesin zarar vermesin diye local branchlar açılıyor.

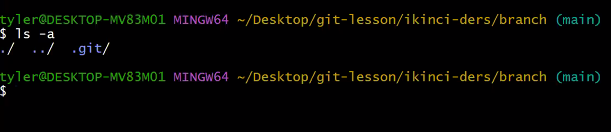


(powershell çoğu linux komutuna izin veriyor.)

Önce mkdir ikinci-ders komutuyla yeni bir klasör oluşturduk. Sonrasında bu klasörde mkdir branch komutuyla branch isimli bir klasör daha olutşurduk.



Sonra cd branch komutuyla o klasörün içine gelerek git init komutuyla .git klasörünü oluşturuyorum

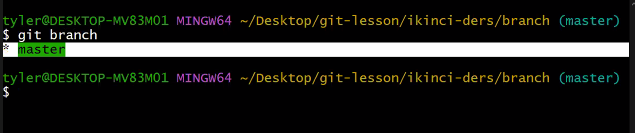


Şimdi bir tane commit yapmak için test1 isminde bir doysa oluşturuyoruz, önce staging areya atıyoruz.Daha sonra commit ediyoruz.



"ls –al" yaptığımızda çıkan listede "dr" ile başlayanlar klasör, diğerleri dosyadır.

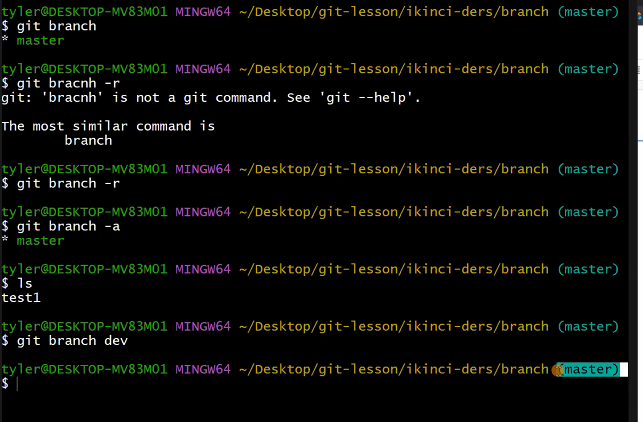
Localdeki branchları görmek için kullanacağımız komut "git branch" tır. (Hoca burada main i master a çevirdi)



Bir tane branch ımız var : master

"git branch –r" komutu remote taki (GitHub) taki branchları gösterir.

"git branch –a" ise hem local hem repodaki branchları gösterir.

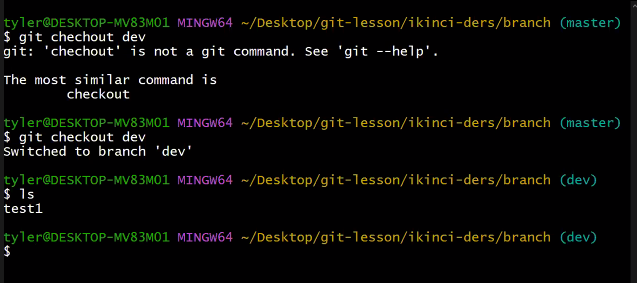


Yeni bir branch oluşturmak için "git branch isim" şeklinde komut giriyoruz. "git branch dev" yazalım:



Şimdi dev isimli bir branch ımız daha oldu.

"git switch" komutu da başka brancha gitmemizi sağlar. Biz şimdi checkout u kullanacağız. "git check out dev" diyoruz: ve ls yapıyoruz hala test1 dosyasını görüyoruz.



Sağ taraftaki master yazısı dev olarak değişti. (dev) branchındayız.

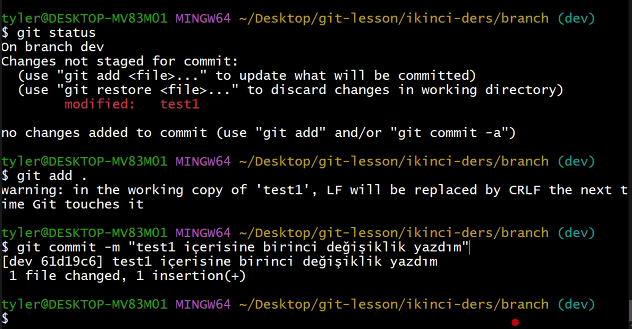


 Yukarıda dev branchının içindeyken front-end isimli yeni bir branch oluşturduk. "git branch" komutuyla tüm branchları ve yeşil yazıyla dev branchında olduğumuzu görüyoruz.



Vim editöründe test1 dosyasında birinci değişiklik metni yazaray değişiklik yaptık.

Şimdi test1 i staging area ya ekleyelim ve commit leyelim:



Bu değişikliği dev branchında yaptık. "ls" ve "cat test1" dediğimizde bunu görebilecez:

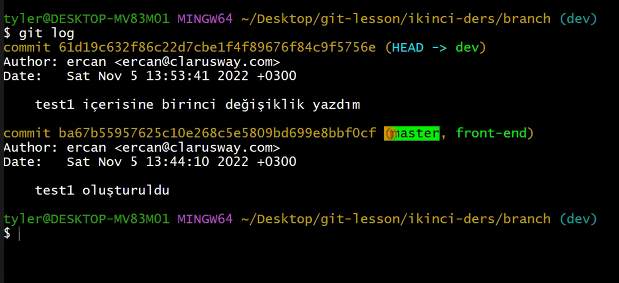


Şimdi master brancha gidip orada görebilecek miyiz ona bakalım:



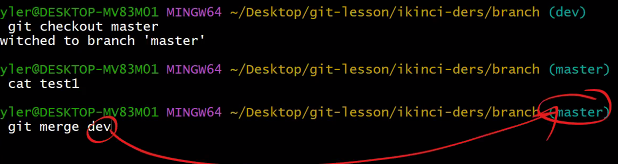
Burada master brancha gidip test1 dosyasını görüyoruz ama yapılan değişikliği göremiyoruz.

(Master brancha diğer branchları merge etme yetkisi herkeste olmaz. Bunları ileride göreceğiz.)

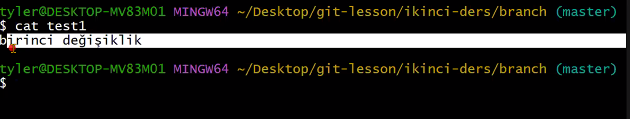


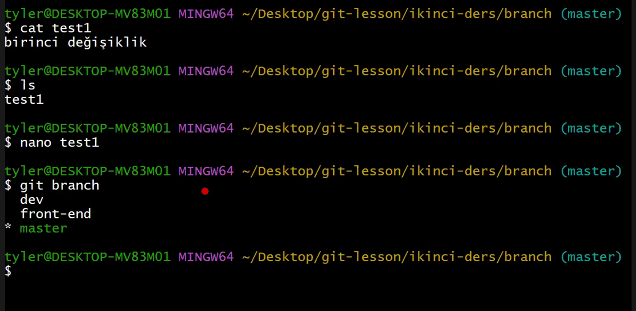
Şimdi dev brancha geri döndüğümüzde ve git log dediğimizde iki commiti de görüyoruz.

Şimdi dev branch ını master brancha merge edelim:

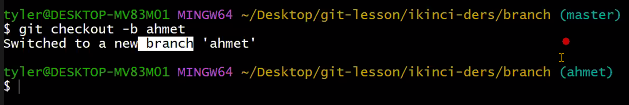


"git merge dev" komutuyla merge ediyoruz. Şimdi "cat test1" komutuyla kontrol edelim:





Master branch ın içindeyiz. "git chcekout –b ahmet" branchıyla ahet isminde branch oluşturup o brancha gidecek:



"front-end" branchını silmek istiyorum:

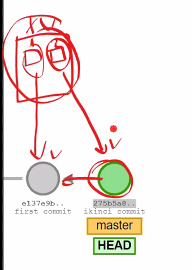


"git branch" ile kontrol ettiğimizde front-end branch ı sildiğini görüyoruz.

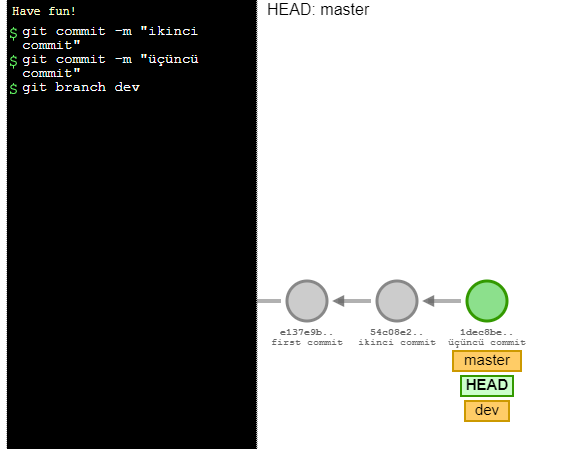
Eğer bir branchı silerken o branchta yaptığımız değişiklik başka branchta varsa küçük d ile silebiliyoruz. Ancak bir branchta bazı değişiklikler yaptık ve commit ettik. Eğer bu yapılan değişiklikleri başka bir brancha merge etmemişsek küçük d ile silemeyiz. Ancak büyük D ile silebiliriz. Küçük d temkinli bir silme metodu.

Bir alıştırma sayfasından örneklerle devam edelim:

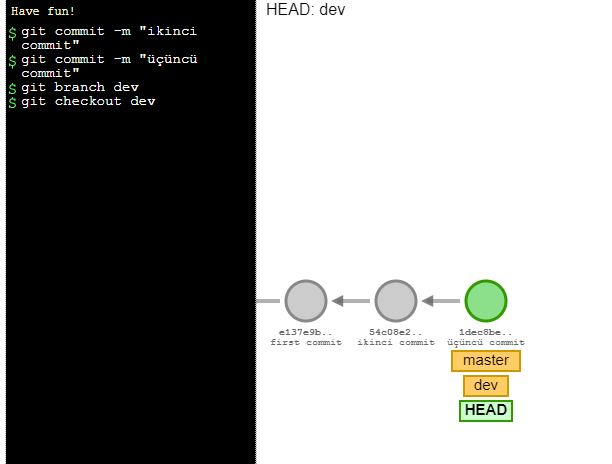
<https://git-school.github.io/visualizing-git/#free>



Burada okun sola doğru olmasının bir sebebi var. Oluşturduğumuz ikinci commit bir önceki commit i referans ediyor ve birinci commit te olan verileri oradan alıyor. Bu sayede daha az veri depolanmış oluyor. Bir commit bir önceki commiti referans eder.

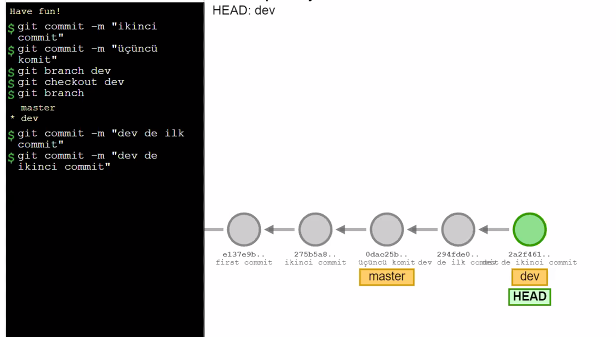


Üçüncü bir commit yaptık ve HEAD dosyası halen mater branch ında. "dev" branchına gidelim:

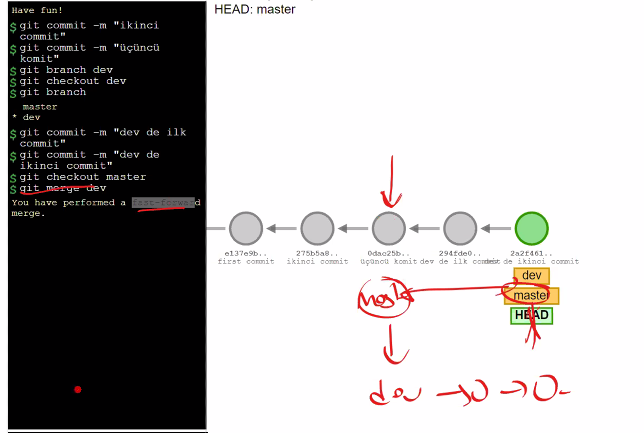


Dev branchıdayız; dev branch etiketi HEAD in üstüne geçti.

Şimdi iki commit yapalım:

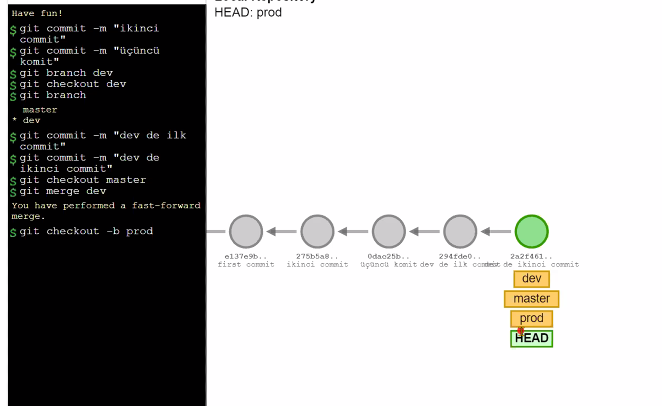


Dev branchında iki commit yaptık ve master branchı halen olduğu gibi duruyor. Şimdi dev i master a merge edince:

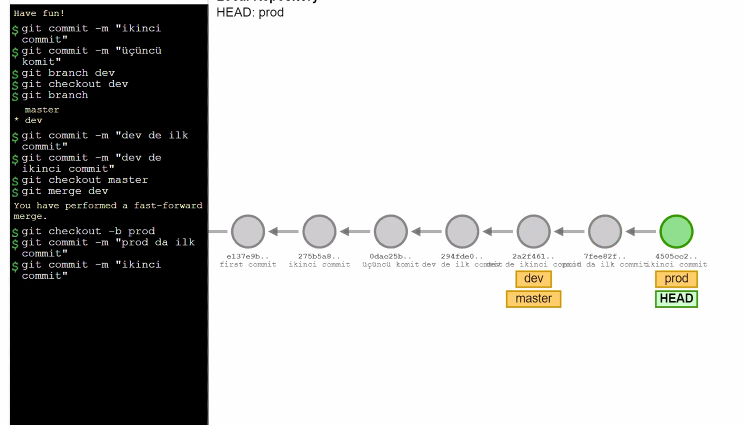


Master ı devin üstüne getiriyor.

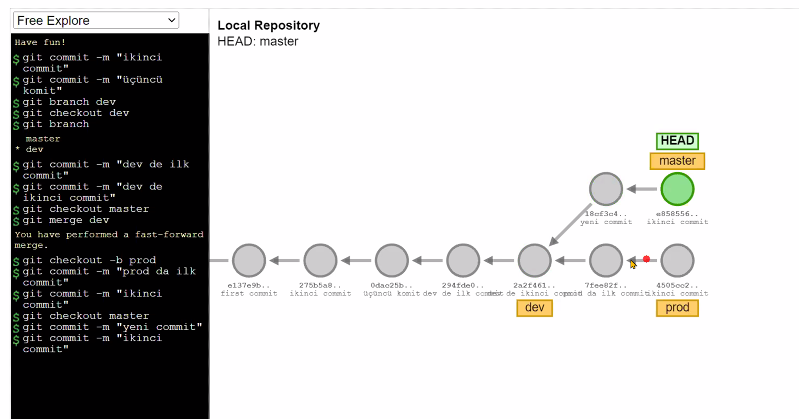
Şimdi prod isimli yeni bir branch yapalım:



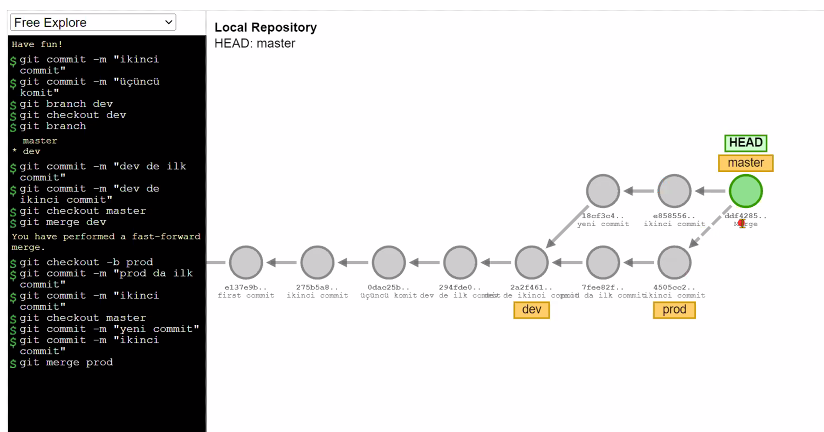
Şimdi iki tane commit yapalım:



Yaptığımız iki commiti prod branchında yapıyor. Tabi bu esnada başkaları da master branchında commitler yapıyor:

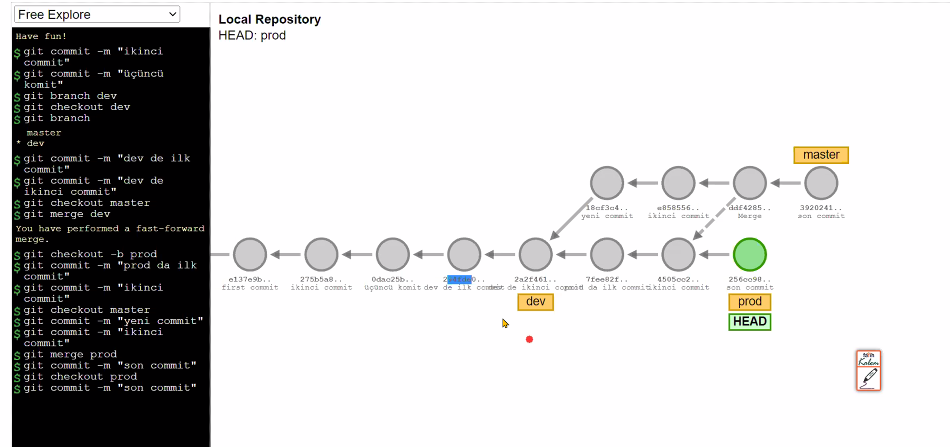


Şimdi three way merge olacak: prod u master a merge ettiğimizde:



Yeni bir commit oluşturdu ve bu commit hem prod hem master daki eski commitlere referans ediyor. Neden yeni bir commit oluşturdu çünkü daha önceki dev i master a merge ederken master da değişiklik yoktu. Ancak şimdi master da değişiklik var, commit var. Bu kez merge ederken yeni bir commit oluşturdu.

Birisi de diğer yandan prod branchında çalışıyor ve yeni bir commit oluşturuyor:

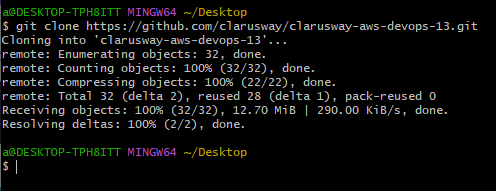


Bunların tekrarını yapacaz daha sonra.

"Git Bash te merge işlemi yaptık burayı not alamadım daha sonra eklemeye çalışacağım"

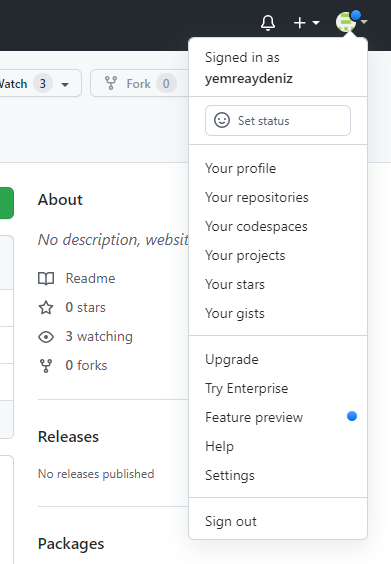
Git ile GitHub aynı şey değil!

GitHUb ta clarusway private öğrenci reposunu locale klonlamak için GitHub ta url sini kopyalayıp Git Bash te kod ile giriyoruz:



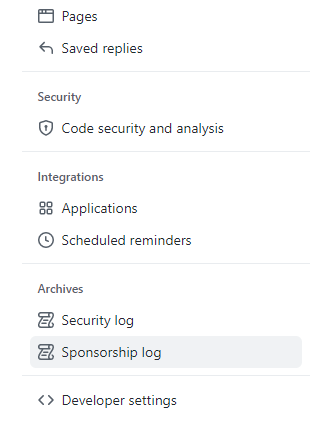
Bu işlemi yapabilmemiz için kullanıcı adı ve şifre sorar. Bunu bir kez girdiğimizde windows bunu credential manager isimli bir yere kaydediyor. Aynı account için tekrar tekrar sormamaısnı sağlıyor. Ancak gerçek bir srverda böyle bir özellik yok.

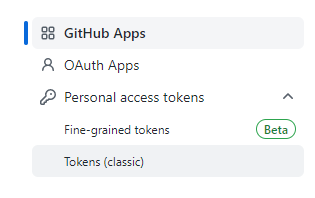
İkinci yöntem token la klonlamadır.



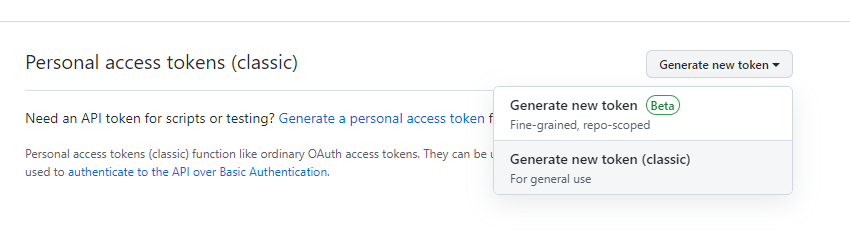
En sağda bir avatar var. Oradaki ok işaretinden settings e geliyoruz.

Sol altta developer settingse basıyoruz:

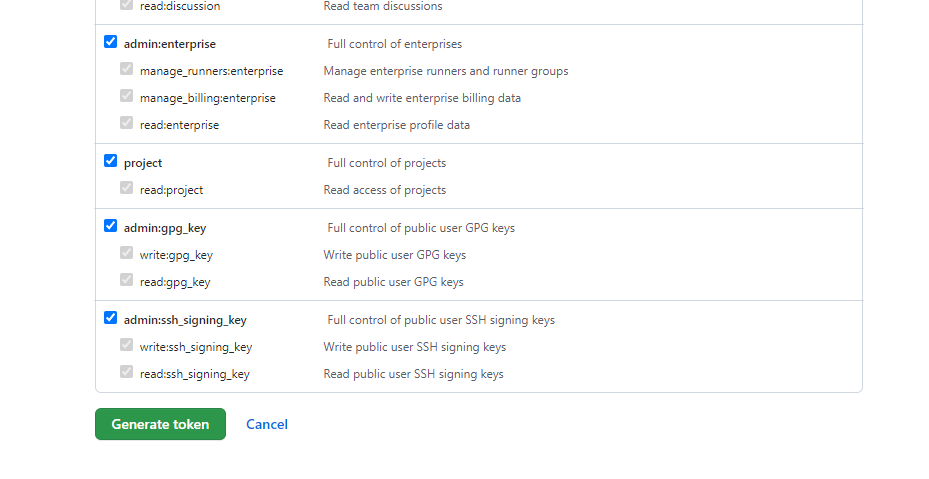




Bu sekmeye tıklıyoruz ve Sağ taraftan generate new token diyoruz.

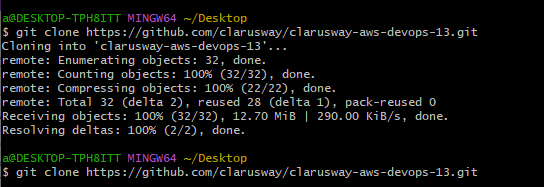


Bu token çok önemli. Bu token birinin eline geçerse sizin hesabınızdan ulaşabildiğiniz her bilgiye ulaşabilir. İsim verip son kullama tarihi girmeyecez şimdilik:



Bütün çentikleri işaretleyip Generate token diyoruz.  Ve token oluşturuluyor.

Token la klonlamak için:



Imleci [\\](NULL) ile github arasına getirip token ımızı buraya yapıştıracaz ve @ simgesini koyacağız. Sonra enter a basınca token ile klonlama yapacaktır.

Haftaya tekrar ve GitHub üzerinden merge yapacaz.