Git Branch

* Branch
* Merge
* Conflict

Önce geçen dersi tekrar edelim

* Git, dağıtık bir şekilde version control sistemidir
* Ubuntu üzerinden, git init yaptığınız bundan itibaren burada yaptığınız her işlem ve değişklik traclanıyor ve kayıt altına alınıyor
* Yaptığınız her yanlış adımda diğer commitler dönebiliyorsunuz ama burada önemli olan bir dönüş noktası oluşturmanız lazım.
* Collabration, beraber aynı projede çalışmada olanak sağlayan sistem
* Kimlik ayarları, config ayarları yapmıştık burada yapılan her değişiklikte tarih saat kim tarafından yapıldı ve ne için değiştirdi bunların cevabını bulabiliyoruz.

Repostory

Working directory gözle görüp çalıştığımız yer

.git gitin mutfağı burası, bu olmazsa versiyon kontrol sistemini yapacağı görevini yapamaz

\*\*Slayttaki gibi kullanıcı adı ve mail ayarlarımızı yapıyoruz, bu bizim kimlik beyanımızdır.

Working Directory hangi commit ve hangi branch da isek orada çalıştığımız yer

Staging Area görmediğimiz yer, alışveriş sepeti gibi biz repoya atmak istediğimiz dosyaları buraya atarız,

Repostory, sonra buradan tamam deyip artık repoya göndereceğimiz yer oluyor.

Git help

Git init yeni bir klasör oluştur ve içine dosya oluştur

Git status ile durumuna bak, bilgisayar kapanınca gidecek veriler

Git add . stage ye çekebiliriz.

Git rm - - cached index.html, burada unstage yapabiliyorum, tekrar working directorye çekmiş oldum

Working directory ve stage aşamlarında bilgisayar kapanınca bilgiler gider, bizim commit yapıp repoya atmamız gerekir.

Git log comitlerimizi görebiliyoruz. Oyuna girdiğinizde kaydetip çıkıp tekrar aynı yerden başlayabiliyorsunuz.

Git checkout deneme yapalım

Biz dosyanın içeriğine bir şeyler kaydedelim.

Git status

Git add . git commit git log bundan donra dönebileceğimiz iki yer oldu

Git checkout ile diğerine dönebilirim

Git log - -all deyince tüm commitleri görebiliriz

Slayttaki taskleri yapalım

Task 1

Mkdir ile klasör onun içinde git init yap

Touch ile dosya oluştur

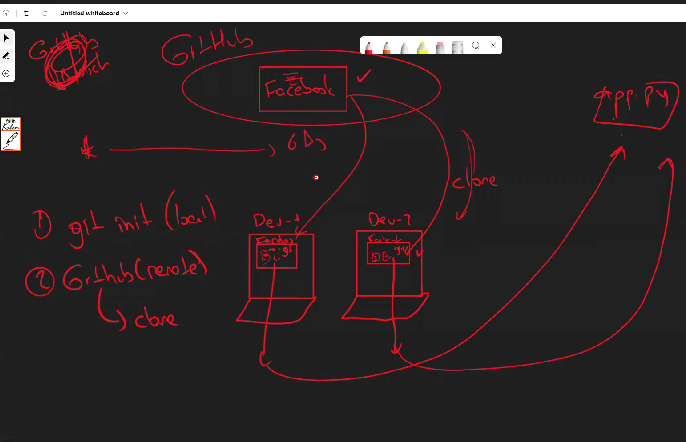
Daha sonra bu

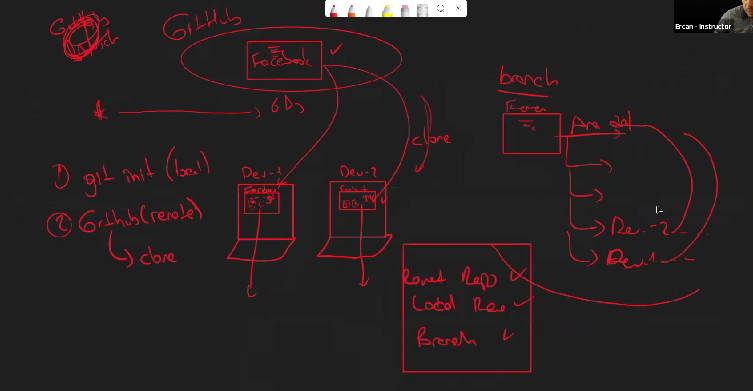
git status

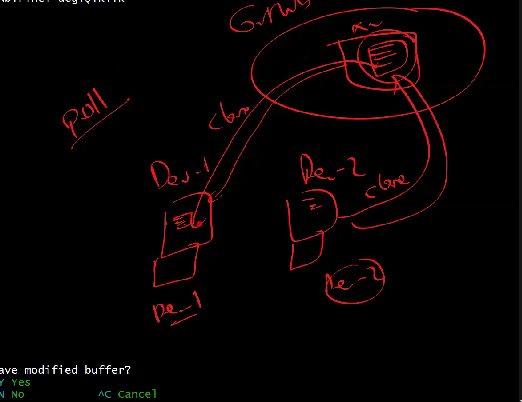
git add . git commit ………

Task 2 yapalım

Git Branch







O anda bulunduğunuz nokta commit neresiyse o commit üzerindeki dosyaların bir fotoğrafın bir kopyasını başka bir yere çekmek oluyor.

\*\*Aslında sizin en değerli branch nız master oluyor, ana branch nız master olarak şimdi bunu main olarak değiştirdiler, main diyoruz artık buna. Her zaman buraya bir eklemeyi direk yapmayız.

\*Normalde bu branch ihtiyacı nereden çıkıyor ?

Siz tek bir developer bile olsanız bir branch a ihtiyaç duyarsınız ama developer sayısı arttıkça bunların neler yaptığı nereye eklediği gibi karışıklıklar oluyor.

Şirkette kafanıza göre branch açamazsnız, siz yetkilere göre işlem yaparsınız.

Bizler farklı kişilerin aynı task üzerinde birlikte çalışabilme imkanı tanır.

Slayttaki şekil üzerinden açıklayalım

Mkdir new\_project içine dosya oluştur ve repoya at,

Git branch olanları listeler

Git branch -r remote dakileri gösterir

Git branch -a tümünü gösterir

Git branch development yapınca yeni bir branch oluşturabiliriz

Git checkout dev main den dev branch ına gitmenizi sağlar

Git status

Git log

Git branch test && git checkout test bunun yerine alttaki komutla da yapabiliriz

Git branch -b test hem oluşturur hem de içine girer.

Branch silme işlemi

Git branch -d test

Git branch -D test arasındaki fark; içindekileri merge etmeden branch ı silmek istersek ne olursa olsun sil demek

\*\*\*

Denemeler yapalım

Yeni bir klasör oluşturup içinde branch lar oluşturup merge işlemleri yapalım

\*\*\*Slaytlardan anlatalım ve siteye gidip oradan anlatalım, git komutlarını anlayabilmek için bir similatör

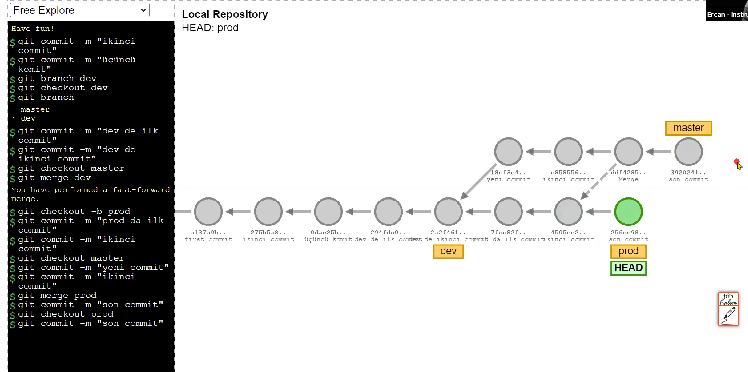
<https://git-school.github.io/visualizing-git/#free>

<https://learngitbranching.js.org/>

metin, ekran görüntüsü, diyagram, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Commit yapılınca sadece değişen yeri alıyor, diğerlerinin fotoğrafını çekmiyor yoksa şişme olur.



metin, ekran görüntüsü, yazı tipi, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Slaytlardan da anlatalım

Fast ve 3-way merge işlemlerine bakalım

Conflict merge de çakışma olunca ne yaparız;

Main ve diğer dev branch ında değişklikler yapıyoruz aynı dosya üzerinde daha sonra main de dev i merge işlemi yapınca çakışma olacak.

Bunun için dosya içine girmek lazım ve hangi değişikliği istiyorsak el ile düzeltiyoruz manuel olarak.

Git merge dev

Nano file ile dosyanın içine girerek hatayı düzeltiyoruz

Git add .

Git commit……

Daha sonra merge işlemi tamamlanmış olacak

\*\*\*

Yeni bir handson\_proje oluşturalım

Git init yapalım daha sonra index.html dosyası yapalım

Git add . git commit -m “proje başlatıldı”

Yeni hotfilx branch oluşturacağız.

Git branch hotfilx ve içine gir

Html dosyası içerisindeki title yi değiştir, Clarusway yap sonra git add . ve git commit yap

\*bu yapılan değişikliği ana, master(main) branch ına merge etmemiz lazım

Git checkout master yapıp oraya geçiyoruz,

Sonra git merge hotfilx yapıyoruz

\*git log - -all yapıp ne yaptığımızı görelim

\*yeni dev branch ı oluşturacağız.

Git checkout -b develop bunun içinde yeni bir dosya oluştur (dockerfile)

Git commit ile gönder

Yine index.html üzerinde değişiklik yap bunu da commit ile gönder

Şimdi develop branch ında iki tane nokta oluştu

Git log - -all deyip hepsine bakabiliriz.

\*\*\*\*Merge den sonra problem oluyorsa onu nano ile girip düzeltiyoruz, sonra git add ve commit yapıyoruz.

Token nasıl alınır ?

Windows da credential manager de token e kaydediliyor, sizin kimliğinizi yansıtan şey.

Sağda avatar resmi var oraya tıklıyoruz

Settings

Sol altta developer settings

Pasword Access tokens

Token

Generate new token------clasics------isim seçelim her hangi bir isim olabilir

Süresini seçiyoruz, süresiz olsun

Alttaki tiklerin hepsini tıklayarak tüm yetkileri veriyoruz.

Token üretildi bunu bir defa gösteriliyor, bunu not defterine kopyalayın ve güvenli bir yere kaydedin.