**Git Nedir**

**Git Dağıtık Versiyon Kontrol Sistemini (Distributed Version Control System)**

* 2005 yılında ilk stabil sürümü yayınlanan Git, 2009 ve 2010 yılına kadar sadece Linux ile iç içe olan çevrelerin takip edip kullandığı dağıtık bir versiyon kontrol sistemiydi
* GitHub,Bitbucket,GitLab’ın benzer Git servislerin sağlamaya başlaması Git’in daha geniş kitlelere ulaşmasını ve farklı profillerdeki yazılım geliştiriciler tarafından yoğun bir şekilde kullanılmasını sağladı.

Versiyon kontrolünü bir dosya veya bir küme dosyadaki değişiklikleri takip edebilmek için uyguladığımız bir yöntem olarak tanımlayabiliriz. Git gibi sistemler tüm bu değişikliklerin tarihçesini ve içeriğini elektronik olarak bizim için takip ederek kayıt altına almamızı sağlayan veri tabanları olarak düşünülebilir.

* Dosyaların kayıt altına alınmış herhangi bir andaki hallerine versiyon diyoruz

**En basit hali ile :**

Versiyon kontrol sistemleri en basit anlamda dosyalarınızdaki değişikliklerin tarihçesini takip edip kayıt altında tutan sistemlerdir.

* Versiyon kontrol sistemleri değişikliklerinizi tamamlayıp **commit** etmek istediğinizde **comment** adı verilen açıklamalar girmenizi isterler. Bu comment’ler sayesinde projenizin herhangi bir versiyonundaki değişikliklerin nedenlerini de kayıt altına alıp ihtiyaç halinde geri dönüp inceleyebilirsiniz.
* Git'de commit işlemi yapılırken comment (yorum metni) girilmesi zorunludur

**Yedekleme**

Git sayesinde aynı projede çalışan herkesin kendi bilgisayarında projenin tam bir tarihçesi tutulur. Merkezi versiyon kontrol sistemi sunucusunda bir sorun oluştuğunda takımdaki herhangi birinin kendi diskindeki projeyi sunucuya geri yüklemesi yeterlidir.

# **Kısa Git Tarihçesi**

Git 2005 yılında, başta Linus Torvalds olmak üzere Linux çekirdeğini de kodlayan ekip tarafından Linux kaynak kodunu versiyon kontrolü altında tutmak ve kendi iş akışlarını düzenlemek için geliştirilmiştir

 2002 yılında Linux geliştiricileri normalde ücretli olan ancak açık kaynak projeler için ücretsiz lisanslama modeli sunan BitKeeper isimli dağıtık versiyon kontrol sistemini kullanmaya başladılar. 2005 yılında BitKeeper'ın ücretsiz sağladığı lisansı geri çekmesi üzerine Linus Torvalds ve Linux ekibi kendi dağıtık versiyon kontrol sistemini geliştirmeye karar verdiler.

Linux ekibi BitKeeper ile olan deneyimlerini de dikkate alarak öncelikli olarak aşağıdaki kriterleri sağlayan kendi yazılımlarını geliştirmeye başladılar

* Hızlı
* Kullanımı kolay
* Lineer olmayan geliştirme iş akışına uygun (branching)
* Tamamen dağıtık
* Büyük projeleri destekleyebilecek

# Git İle Çalışmaya Başlamak

Git ile çalışmak için git'in kendi **komut satırı arayüzünü** (Git Command Line Interface) veya görsel kullanıcı arayüzü olan masaüstü uygulamalar (SourceTree, Tortoise Git, Tower veya GitHub) kullanabiliriz.

## Kurulum

### Windows

İşletim sisteminiz Windows ise git ile çalışmak için "msysgit" paketini kullanabilirsiniz.

**msysgit** paketini kurmak için <http://msysgit.github.io/> adresinden kurulum uygulamasını indirip çalıştırmalısınız. Kurulum adımları sırasında karşınıza çıkacak olan ekranlarda varsayılan ayarları seçili olarak bırakarak kurulumunuzu tamamlayabilirsiniz.

### **Git Konfigürasyonu**

Git’i kurduktan sonra Git ile çalışmak için bazı ayarlar yapabiliriz. Git bunun için bize **git config** isimli bir komut sunar. Git ayarlarını bir sefer yapmamız yeterli olacaktır.

### **Kullanıcı adınızı ve email bilgisi**

Git ayarlarından en önemlisi kullanıcı adınız ve email adresinizdir. Git, ayar olarak tanımladığınız değerleri commit ve benzeri işlemler de otomatik olarak kullanır.

**Ayar Komutları:**

git config --global user.name "ramazan"

git config --global user.email "rmzns1515@gmail.com"

* **--global** seçeneği ile Git'e global ayarları düzenlediğinizi söylüyoruz
* Global seviyede tüm ayarları listelemek için; **git config --global –l**
* Global seviyede tek bir ayar'ın değerini öğrenmek için örneğin kullanıcı adını öğrenmek için; **git config --global user.name**

# Basit Anlamda Versiyon Kontrolü İş Akışı

Versiyom kontrol sisteminin en temel bileşeni **repository** dir. **Repository**, dosyalarınızdaki tüm değişiklikleri ve bu değişiklikler ile ilgili ilave bilgileri (değişikliği kim, ne zaman yaptı ve değişiklik ile ilgili girilen açıklamalar) ayrı birer **versiyon** olarak kayıt altında tutan bir veri tabanıdır.Git tüm bu bilgileri .git klasörünün içinde tutar.

Yukarıda bahsettiğimiz **repository**'yi kendi bilgisayarınızda oluşturmak için iki yöntem kullanabilirsiniz.

* Henüz versiyon kontrolünde olmayan bir projeniz varsa **git init** komutu ile projenizi tüm klasör ve dosyaları ile birlikte versiyon kontrolüne alabilirsiniz
* Projeniz uzaktaki veya şirket ağınızdaki bir Git sunucusunda versiyon kontrolü altında tutuluyorsa projeyi kendi bilgisayarınıza **git clone** komutu ile indirebilirsiniz.
* Repository oluşturduktan sonra dosyalarınız üzerinde istediğiniz değişiklikleri yapabilirsiniz.
* Yaptığınız değişikler istediğiniz noktaya geldiğin de veya sorunu giderdikten sonra versiyon kontrolü bakışı ile değerlemdirmeniz gerekir yani değişikleri **commit** adı verilen komut ile projenizin yeni bir versiyonunu oluşturursunuz.

Yaptığımız değişikliği commit ettik peki nereye kaydoldu bu değişikler ?

**Local (Yerel) & Remote (Uzak) Repository'ler**

* Local repository, kendi bilgisayarınızda proje klasörünüzün altında bulunan **.git** klasörüdür. Bu repository üzerinde sadece siz çalışabilirsiniz ve değişiklikler yerel diskinize kaydedilir.
* Remote repository'ler ise genellikle uzaktaki bir sunucuda yer alırlar ve bu sunucudaki **.git**klasöründen ibarettirler. Takım çalışması söz konusu ise takımdaki kişiler değişikliklerini bu uzaktaki repository üzerinden paylaşırlar.

## NOT:

## Projenizi ister local bir proje olarak oluşturmuş olun isterseniz remote bir repository'yi klonlamış olun tüm değişiklikleriniz yerel diskinizde gerçekleşecek ve **commit’leriniz** ile oluşturacağınız tüm versiyonlar git tarafından yerel diskinizdeki .git klasöründe takip edilecektir. İlerleyen bölümlerde ayrıntılı olarak ele alacağımız **git push** komutunu çalıştırmadığınız sürece yaptığınız değişiklikler sadece yerel diskinizde kayıt altına alınır.

## Dosya Durumları

Git'de dosyalarınız genel olarak iki durumda olabilir

* Untracked (Takip Edilmeyen): Bu dosyalar versiyon kontrolü altında olmayan veya sizin henüz versiyon kontrolü yapmak için git'e eklemediğiniz dosyalardır. Bu dosyalardaki değişiklikler siz dosyaları git'e eklemediğiniz sürece versiyon kontrolüne tabi değildir
* Tracked (Takip Edilen): Bu dosyalar ise git'in versiyon kontrolü takibi altında olan dosyalardır. Bu dosyalar üzerinde yapacağınız tüm değişiklikler git tarafından takip edilmektedir.

## Staging Area

Çoğu versiyon kontrol sisteminde değişiklikleriniz iki yerde kaydedilir

* Yerel diskinizdeki çalışma klasörünüz (working folder) veya
* Versiyon kontrol sisteminin veri tabanı

Ancak git'de değişikliklerinizin kayıt altına alındığı üçüncü bir alan daha vardır ki buna **Staging Area** denir ve git'in en temel kavramlarından birisidir. Staging Area'yı, proje dosyalarımızdaki bir dizi değişikliği remote repository'ye göndermeden önce kayıt altında tuttuğunuz veri tabanı/alan olarak tanımlayabiliriz.

GİT KOMUTLARINA GİRİŞ

**Banch**

Git ağaç mantığı ile çalışmaktadır. Bir projeyi versiyonlamak istediğimizde git otomatik olarak master branch’ini oluşturur ve biz bu branch üzerinde çalışmaya başlamış oluruz.

* **git branch deneme**,komutu ile testBranch adında yeni bir branch oluştururuz ve o anda master branch’in kopyası yeni branch’cimize kopyalanmış olur.
* **git branch ile tüm branch leri listeleye biliriz.**

**Branch’ler arası geçiş**

**deneme** branch’ini oluşturduk ama bu branch’ e geçmedik hayla **master** branch deyiz. **Deneme** branch’ine geçmek için **git checkout deneme** komutu ile deneme branc’ine geçtik.

**Branch’de Değişikler Yapma**

Deneme branch’inde değişikler yaptınız xyz.html, login.html ve abc.js dosyaların da değişikler yaptınız ve bu değişikleri commit etmek istiyorsunuz fakat xyz.html dosyasını commit’e eklemek istemiyorsunu bunun için ;

**git add login.html abc.js** diyerek sadece belli dosyalarınızı eklediniz.

**Commit**

Yaptığınız eklmeden sonra commit için ;

**git commit –m”login.html sayfası oluşturuldu.”** Şeklin de ilk commiti gönderdik.

**PUSH**

Commit ettik peki bu nereye kaydedildi dersek kendi localimizde .git klasörüne kaydettik bunu. Bu commit’i uzak sunucuya branch’e göndermek için **git push** komutu ile göndeririz.

# **Değişiklikleri Merge Etmek**

Merging en basit anlamda herhangi bir brach'de yaptığımız değişiklikleri **master** branch'imiz ile birleştirme veya **master** branch'e entegre etme işlemidir.Bu sadece **master** branch’i olmak zorunda değil başka bir branch ile de birleştire bilirdik.

Git'de merge işlemi çok basit iki adımda yapılır.

1. **git checkout** komutu ile değişikliklerin aktarılacağı hedef branch'inizi aktif (HEAD) hale getirirsiniz.
2. **git merge**komutu ile kaynak branch'deki commit edilmiş değişiklikler HEAD'e entegre edilir.

**Git Init**

init komutu, bulunduğumuz dizini bir Git repository’si haline getirir ve .git uzantılı bir dizin oluşturur. Daha sonra bu repository’yi uzaktaki bir sunucuya gönderebilirsiniz. Komut: **git init .**

**GİT LOG**

. Projeniz için oluşturduğunuz commit'lerin tarihçesini incelemek için **git log** komutunu kullanabilirsiniz.

# **Git Status**

O andaki branch de yapılan tüm değişikleri gösterir. Komut: **git status**

**Git Stash**

O andaki barnch de yaptığınız değişikleri saklamak için kullanılır.

Niiçin saklarız peki?

Ana branch deki değişikleri kendi brancmize aldığımız zaman çakışmaması için veya değişikler kaybolmaması için saklarız.

**Git Stash Apply**

Git stash ile saklanan değişikleri geri getirmek için kullanılır.

**Git Fetch**

remote'dan güncelleme bilgilerini indir. Komut: **git fetch**

**Git Diff**

remote ve local arasındaki farkları incele.Komut: **git diff**

**Git Pull**

Uzak sunucudaki repository’de değişikliğe uğramış dosyalar varsa bunları indirir ve yereldeki repository ile birleştirerek dosyalarımızı günceller. Komut: **git pull**

# **Branch'leri Silmek**

Bir branch'i kendi bilgisayarımızdan silmek için **git branch -d silinecekBranch** komutunu, remote repository'den silmek için de **git branch -dr silinecekBranch** komutunu kullanabiliriz.

## Commit Edilen Bir Değişikliği Geri Almak

Hatalı bir düzenleme yaptığınızda veya geliştirdiğiniz bir özelliğin artık gerekli olmadığına karar verildiğinde yaptığınız değişikliği geri almanız gerekecektir.

**git revert** komutu commit ettiğiniz herhangi bir değişikliği geri almak için kullanılır. Bu komut ile commit işleminizin kendisi veya bilgileri silinmez sadece commit işleminizdeki değişiklik geri alınır.

Komut: **git revert 55b57b** ,  Bu komutun en önemli parametresi geri almak istediğimiz commit'in hash değeri (hash'in ilk yedi karakterini kullanabiliriz). Bu komutu çalıştırdıktan sonra git'in otomatik olarak bir commit oluşturduğunu ve bu commit'in bilgilerinde hangi değişikliğin geri alındığına dair ayrıntıların yer aldığını görürüz.

# **Çakışmaları Gidermek**

Versiyon kontrolünde en büyük sorun merge işleminde oluşan çakışmalardır.

## Çakışma Nasıl Oluşur?

## Aynı satırda yapılan değişiklikler veya takımdaki bir kişinin bir satırı silmesi durumunda sizin bu değişikliği kendi branch'inize nasıl entegre edileceğine karar vermeniz gerekir. Bu durumda Git dosyanızı conflicted (çakışmalı) olarak işaretler ve sizin çalışmanıza devam edebilmeniz için bu çakışmayı çözmeniz gerekir.

## Çakışmaları Nasıl Çözeriz

## ****git status**** komutunu çalıştırdığınızda Git size branch'inizde entegre edilmemiş dosyalar olduğunu söyler.

## Çakışma olduğun da ve çakışan dosyayı açtığımız da karşımıza:

* **<<<<<<<<< HEAD** ile başlayan ve **============**kadar devam eden kısım dosyanın bizim branch'imizde olan versiyonuna ait
* **================**belirtecinden sonraki kısım da değişiklikleri entegre etmek istediğiniz branch'de yer alan dosyanın içeriğini gösterir.

## Bunu gördükten sonra hangi değişikliğin kalacağına karar verditen sonra kaydettikten sonra normal bir commit işlemi ile çakışmayı çözme işlemini tamamlıyoruz.

## 1-)Çakışma gerçekleşmeden önce

* Çakışma gerçekleşmeden önce bunu önlemenin en iyi yolu, üzerinde çalıştığımız özellik dalını sıklıkla master üzerinden rebase komutuyla güncelleştirmektir.

*Deneme diye bir branch oluşturduk. Master branch'inin kopyasını aldık ve değişikler yaptık dosyalar oluşturduk. Biz bunları oluşturur iken diğer takım arkadaşlarımız da master da değişikler yaptılar artık deneme branch'miz master dan çok farklı bunu eşitlemek için ;*

* *git add -A*
* *git stash*
* *git checkout master*
* *git pull*
* *git checkout* ***deneme***
* *git rebase master*
* *git stash apply*

**git add -A** : Üzerinde çalıştığım tüm dosyaları stash edeceğimi bildiriyorum.

**git stash**: Çalıştığımız dosyaları daha sonra kullanılmak üzere saklıyor.

**git checkout master**: master’a geçiş yaptık.

**git pull**: Şu anda master üzerinde olduğumuz için master’ı son haline güncelliyoruz.

**git checkout deneme** : Üzerinde çalıştığımız özellik dalına tekrar geçiş yaptık. Rebase işlemine hazırız.

**git rebase master:**master ve deneme'yi eşitliyoruz bu komutla. Normalde rebase korkulu bir komuttur ve git’e aşina olmayanlar bu komuttan kaçınır. Aslında ben de pek sevmezdim. Ancak ikinci adımda “stash” kullanmamızın sebebi rebase’in çakışmalarının önüne geçmek (aslında ertelemek). stash kullandığımda zaten tüm değişikliklerimi kaldırmıştım. Dolayısıyla bu adımda hiçbir çakışma gerçekleşmeyecek. Böyle bir durum varsa eğer son adımda gerçekleşecek.

**git stash apply**: Şu adıma kadar master ve deneme branchlerim birbirine eşit hale geldi. Ve değişiklerimi ekledim.

**DİKKAT**: stash apply işleminden sonra çakışma gerçekleşebilir. Bu durumda çakışma olan dosyayı elle düzeltmemiz gerekiyor.

## 2-) Çakışma gerçekleştikten sonra

* Master branch'inden pull çektiniz değişiklik yaptığınız dosyalarda başka bir arkadaşınız da değişiklik yaptı ve çakışma oldu bunu çözmek için ek komutlara ihtiyacım olacaktır.
* **git reset HEAD~1 (** Burda 1 attığınız commit sayısı ne kadar commit atılmış ise onu belirtiyoruz.**)**
* Bu işlem ile yaptığım değişiklikleri geri alıyorum.
* Daha sonra yukarıda çakışma gerçekleşmeden önce uyguladığım tüm adımları tek tek tekrar uyguluyorum. Daha sonra:
* ***git add -A***
* ***git commit -m “Çakışma giderildi.”***
* ***git push --force***

* Force ilk bakışta biraz ürkütücü gelebilir ancak eğer ne yaptığınızı biliyorsanız. Reset komutunu kullandıktan sonra üzerinde çalıştığımız yerel kod, sunucudaki kodun 1 commit gerisinde kaldı. Git sizin “pull” etmenizi bekliyor. Ancak pull ile bu kodu çekersek edersek tekrar başa dönmüş olacağız. “Pull” etmeden “push” komutunu kullanmaya kalkarsak da git buna izin vermeyecek. Bunun için ***git push --force*** ile bunu önüne geçiyoruz ve çakışmayı bu şekilde giderdik yapmamız gereken tek şey master'a deneme branch'ni merge'lemek.

* ***git checkout master***
* ***git merge deneme***

## Merge İşlemini Nasıl Geri Alabiliriz?

Dosyanızın merge işlemine başlamadan önceki haline istediğiniz zaman geri dönebilirsiniz. Bunun için yapmanız gereken tek şey **git merge --abort** komutunu çalıştırmak.

Kaynak:: <https://aliozgur.gitbooks.io/git101/>