Git / GitHub 101

Cengiz C. Mataracı

Software Developer @Trendyol

cengizcmataraci

cengizcmataraci

saucecover

Git nedir?

- Git bir **versiyon kontrol sistemi**dir.
- 2005 yılında **Linux çekirdeğ**inin geliştirilmesinde kullanılması için **Linus Torvalds** tarafından oluşturulmuştur.
- Hız odaklı, **dağıtık** çalışan bir sistemdir.
- Kullanması diğer versiyon kontrol sistemlerine nazaran daha kolay ve daha hızlıdır.

Versiyon kontrol sistemleri

- **Versiyon kontrol sistemi** bir projede yaptığımız değişiklikleri adım adım kaydeden ve sonrasında bunları bir repositoryde saklamayı ve yönetmeyi sağlar.

- En önemli özelliği birden fazla insanın aynı kod üzerinde çalışabilmesini sağlamaktadır. Kod üzerinde yapılan değişiklikler isteğe bağlı senkronize edilebilir.

- Her dosyanın uzun vadeli değişiklik geçmişi tutulur. Değişimleri aktif olarak görebilirsiniz.

- Versiyon kontrol sistemlerine örnekler: **Git**, **SVN**, **ClearCase**, **Mercurial**, **TFS**, **BitKeeper**.

Git kullanmanın avantajları

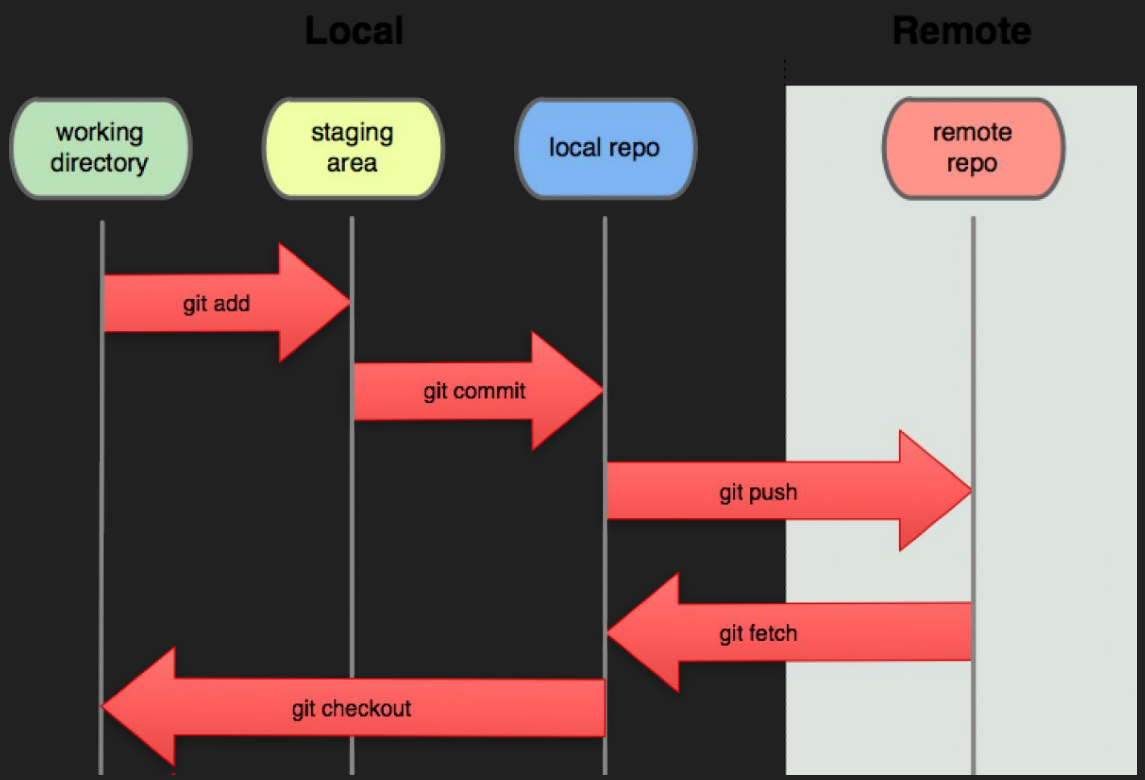
- **Dağıtık** bir sistem olduğu için kullanıcılar dosyaların en son kopyalarının yanında tüm repositoryi kopyalarlar. **Veri kaybını önlemesi**nin yanında aynı projede farklı geliştiriciler farklı iş akışları üzerinden rahatlıkla çalışabilirler.

- Dağıtık yapısı merkezileştirilmiş ürünlere göre oldukça performanslı çalışmaktadır.

- **Branching** daha kolay ve daha hızlıdır.

- Şu an kullanılan en popüler versiyon kontrol sistemidir.

Git nasıl çalışır



Kendi bilgisayarınızda çalıştığınızda **Git** bunun hemen farkında değildir. Öncelikle Git'e bu projeyi **bu dosyayı takip et** diyerek point etmek gerekir. Sonrasında çalıştığımız dizini **staging area**'ya göndeririz. Bu alan daha değişiklikleri bağladığımız (**commit**lediğimiz) durumdan daha önceki bir alandır. Değişikliklerimizi commitledikten sonra değişikliklerimiz **local repo**da tutulur. Buradan istersek **uzaktaki bir repo**ya gönderebiliriz.

Git komutları

Git config: Git kullanmaya başladığınızda uygulamanız gereken username ve mail gibi bilgilerin girildiği komuttur.

Git init: Bu komut ile çalıştığınız dizini bir git repositorysine dönüştürür ve daha sonrasında remotea gönderebileceğiniz şekilde düzenler. Bu komuttan sonra Git dizindeki değişiklikleri takip etmeye başlar.

Git clone: Uzak bir repoyu yerel dizininizde bir **klon**unu oluşturmaya yarar.

Git status: Yaptığınız değişiklikleri gözlemleyebildiğiniz ve o anki durumu öğrenmenizi sağlayan komuttur.

git add: Çalıştığınız dizindeki değişikliklerinizi **staging area**ya göndermenize yarayan komuttur. Bu aşamadan sonra değişikleriniz **commit**lemeye -bağlamaya- hazır olur.

Git commit ve commit mesajlarının önemi

- Değişikliklerimizi **staging area**ya attıktan sonra bu değişiklikleri bir mesaj ile bağlayıp **local repo**muza göndermemiz gerekmektedir. Bu işlemi git commit komutu ile yapıyoruz. **Git commit -m “değişiklikleri içeren mesaj”** şeklinde commitleyebiliriz.

- **Commit** mesajları takım çalışması ve düzenli bir repo için önemlidir. Yazdığınız mesajı karşıdaki kişi değişiklikleri incelemeden anlaması gerekmektedir. Bununla alakalı çeşitli **best practice**ler bulunmakta, bunlardan bazıları:

Git commit ve commit mesajlarının önemi

- Commit mesajlarınız “**I’m adding button**” yerine “**add button**” şeklinde olmalı, bunun nedeni “**If applied, this commit will -commit mesajı-**” şeklinde olduğunda anlamlı bir cümle oluşturacak şekilde olmasından dolayı. “**If applied, this commit will add button**” şeklinde.

- Yazdığınız commit mesajı kısa olmalı: “**change button color and button hover color on about page**” yerine “**refactor button style on about page**” olabilir.

- Yaptığınız değişikliği commit mesajının başında belirtirseniz daha açıklayıcı olabilir: **fix:...** | **feat:...** | **style:...** | **refactor:...** | gibi.

- Fazla **noktalama işareti** kullanmamaya özen gösterin ve **commit mesajını nokta ile tamamlamayın**.

- Uzun commit mesajı yazmak zorunda kalırsanız **72 karakteri** geçmemesine özen gösterin, geçtiği takdirde **bir satır altı**ndan devam edin.

Git komutları

Git push: Değişiklikleri commitledikten sonra göndermemizi sağlar. Bu değişiklikler remote bir repo ile bağlı ise oraya gönderilir. Herhangi bir yol belirtilmediği takdirde git bu değişikleri nereye göndereceğini bilemez ve “**fatal: No configured push destination.**” tarzında bir hata verir.

Git pull: **Remote repo**da olan değişiklikleri çekmenize ve bunları **local repo** ile birleştirmeyi sağlar. Takım çalışmasında bir değişiklik yapmaya başlamadan önce **git pull** komutunu çalıştırmak oldukça önemlidir.

GitHub nedir?

Versiyon kontrol sistemi olarak **Git**'i kullanan projeler için geliştirilen **depolama servisi**dir. 2008 yılında kurulduktan sonra 2018 yılında **Microsoft** tarafından satın alınmıştır.

En çok kullanılan depolama servislerinden biridir. Sadece yazılım tabanlı projeler için değil aklınıza gelebilecek her türlü projeyi depolayabileceğiniz bir platformdur.

Rakipleri: **GitLab**, **Bitbucket**, **SourceForge**, **AWS CodeCommit**, **Launchpad**

GitHub kullanmanın avantajları

- **Open Source** community bakımından en geniş kapsama sahip depolama platformu.
- Arayüzü rakiplerine nazaran daha kolay.
- Entegrasyonu kolay.
- **TFS (Team Foundation Server), HG (Mercurial) ve SVN (Subversion)** ile de çalışır.

Branch nedir ve neden önemlidir?

Branch (dallanma) projenizde çalışırken projenin bir kopyasını oluşturup bir versiyon üretmenizi sağlar. Her repositorynin oluşturulduğu anda bir **ana branch**i (**main**) vardır. Bu branch üzerinden yapacağınız geliştirmeler için ayrı ayrı branchler alabilirsiniz.

Geliştirme yaparken branch tabanlı geliştirme yapmak önemlidir, yapacağınız değişiklikleri takip etmeniz, herhangi bir hata durumunda ana branchin bozulmaması ve yaptığınız değişiklikleri daha sonra ana branch'e **pull request** açarak **code review** yapmanıza olanak sağlaması gibi avantajları vardır.

5 kişinin ortak çalıştığı bir proje düşünelim, bir ana branch üzerinden çeşitli taskları bölüşüp 5 ayrı branch oluşturup herkes kendi branchi üzerinde çalışır. Böylece hiç kimse o taskı tamamen bitirmeden ana branche atmamış olur, bir task tamamen bittiğinde ana branch'e pull request açılıp o task kapsamında yapılan değişiklikleri diğer 4 kişinin -ya da kaç kişi isterseniz- incelemesi istenir ve eğer herkes onaylarsa bu branch ile ana branch birleştirilir. Bu işleme de mergeleme denir.

Terminalde komut ile branch üretmek için **git branch** komutu kullanılır. **Git branch -branch adı-** şeklinde bir branch oluşturabilirsiniz. Başka bir branche geçmek için **git switch -c -branch adı-** komutunu kullanabilirsiniz. **Git**'in **cheat sheet**inde yazmayan ama uzun süredir kullandığım **git checkout -branch adı-** komutunu da kullanabilirsiniz. Bu komutu **git checkout -b -feature/newButton-** şeklinde yaparsanız "**feature/newButton**" adında bir branch oluşturur. **Git branch -d -branch adı-** ile de branchi silebilirsiniz.

Şimdi **pull request** ve **merge** kavramları üzerine konuşalım.

Pull request nedir?

Ana branchten aldığınız bir **branch**te -ya da **fork**ladığınız bir projede- yaptığınız değişiklikleri ana branche, başka bir branche ya da direkt repositorye göndermek istenildiğinde **pull request - çekme isteği-** atılır. Bu işlem projenin sahibi ya da diğer geliştiricilerin yaptığınız değişiklikleri incelemesini -**code review**- ve bunun sonrasında duruma göre onaylama, değişiklik isteme ya da reddetme gibi seçenekleri barındırır.

Temelinde açık kaynak projelere katkı sunmak için projenin kendi hesabınıza kopyalanıp -**fork**lanıp- sonrasında yapılan değişiklikleri proje sahibine göndermeniz ve böylece açık kaynak projelere katkı sunmanızı -**contribute**- sağlar.

Mergeleme işlemi

Pull request açtıktan sonra değişikliklerinizin onaylandığı durumda çalıştığınız branch ile ana branchin **birleştirilmesi** gerekir ki değişiklikleriniz projeye eklenip etki edebilsin. Bunun için iki branchin birleştirilmesi yani **merge**lenmesi gerekir.

Mergeleme işlemi **pull request** olmadan da yapılabilir fakat bu mergeleme işlemi **merging branch** olarak geçmektedir yani branchleri birleştirme olarak düşünülebilir. Üzerinde çalıştığınız projede iki branchi birleştirebilirsiniz fakat bir ekip çalışması içerisinde olduğunuz projelerde kesinlikle pr açmadan mergeleme gibi bir aksiyon almamalısınız. Zaten team lead ya da proje sahipleri -CTO vs- bu işlemi önleyecek şekilde aksiyonlar almışlardır.

Diğer git komutları

Git stash: Yaptığınız değişikleri **commit**lemeden saklamanızı sağlar. Örneğin bir projede çalışıyorsunuz ve değişiklik yaptığınız branchten commit atmadan başka bir branche geçmek istiyorsunuz. Değişikliklerinizi stashleyerek daha sonra da kullanabilirsiniz. G**it stash -stash mesajı-** şeklinde değişiklikleri kaydettikten sonra g**it stash list** diyerek stashlerinizi listeleyebilirsiniz. **Git stash pop** komutu son kaydettiğiniz stashi getirir, **git stash drop** ise en son kaydettiğiniz stashi siler.

Git log: Üzerinde çalıştığınız branchin sürüm geçmişini listeler. Atılan **commit**ler, açılan **pr**'lar dahil detaylı bir liste getirir.

Diğer git komutları

Git cherry-pick: Bir branchi diğerine eşitlerken -senkronize yaparken- kullanılan komuttur. Bir branchte geliştirme yapıp bu geliştirmeyi başka bir branche almak isterseniz git cherry-pick -commit numarası- şeklinde kullanılarak bu işlemi yapabiliriz.

Git rebase: Rebase komutu kullandığımızda ise ile A dalındaki her bir commit B dalına sanki commit işlemi B dalında yapılmış gibi yeniden yazılır. Bu sayede B dalının commit tarihçesi sanki tüm değişiklikler bu dalda olmuş gibi düz ve kesintisiz görünür. Örn: git rebase master


ONE DOES NOT SIMPLY
UNDERSTAND GIT
MemesHappen


GIT PUSH --FORCE
PROBLEM SOLVED
makeameme.org