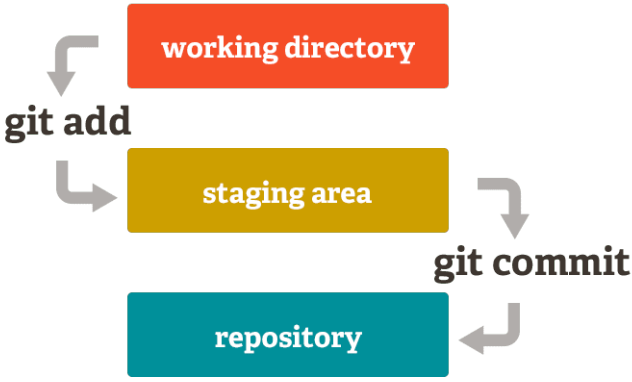
Git Nedir?

Git bir versiyon kontrol sistemidir. Versiyon kontrol sistemi nedir diyecek olursak bir projemizin olduğunu varsayalım. Projemizin gerekli anlarını kayıt altına alıp istediğimiz zaman istediğimiz kayda dönüş yapabildiğimiz,farkları gözlemleyebildiğimiz ve birden fazla kişinin çalışmasına olanak sunan bir sisteme verilen addır.

Git sistemini gözümüzde canladırmak istersek bu görseli kullanabiliriz.



Git görüldüğü üzere 3 kısımdan oluşur.Working directory dediğimiz kısım aslında proje klasörümüzdür. Staging area ise projemizdeki değişiklikleri add ettikten sonra değişikliğin bulunduğu kısımdır. Bu kısımda değişikliğin commit edilmesi beklenir. Staging areadaki işlem commit edildikten sonra repository'e aktarılır. Böylece staging area boşaltılmış olur.Projemizin bir versiyonu da kaydedilmiş olur.

1. Git Komutları

İlk başta yapacağımız değişikliklere referans olacak kullanıcı adı ve email tanımlamamız gerekir.

```
git config --global user.name " <> "
git config --global user.email <>
```

Eğer bir projede git kullanacaksak bu proje klasörünün içerisinde git repository'sinin oluşturulması gerekir. Bu komut;

```
git init
```

Eğer bir projenin anlık durumunu görüntülemek istersek;

```
git status
```

Eğer bir projedeki değişiklikleri staging area'ya aktarmak istiyorsak;

```
git add . = Tüm değişiklikleri staging area'ya aktarmayı sağlar.
git add <> = İsmi girilen dosyanın değişikliğini staging area'ya
 aktarır.
```

Eğer staging area'daki dosyaları repository'e aktarmak istiyorsak commit etmemiz gerekir.

```
git commit -m "<>"
```

Eğer bir projeyi hem add edip hem commit etmek istiyorsak;

```
git commit -a -m ""
```

Eğer projede atılan commitleri görüntülemek istersek;

```
git log
```

Eğer bir projede bir dosyamızın git tarafından takip edilmesini istemiyorsak veya bu klasörü ileride bir github gibi bir platforma push ettiğimizde bu dosyanın

görüntülenmesini istemiyorsak bu dosya ismini **.gitignore** dosyasının içerisine yazmamız gerekir. Bu dosya içerisine yazılan dosya isimleri git tarafından takip edilmez.

1.1. Branch İşlemleri

HEAD = Git içerisinde aktif olarak hangi konumda olduğumuzu gösterir.

Aktif olarak bulunduğumuz branchi ve diğer branchleri görüntülemek istiyorsak;

```
git branch
```

Uzaktaki branchleri de listelemek istiyorsak;

```
git branch -r
```

Bir branch oluşturmak istiyorsak;

```
git branch <yeni branch ismi> <çekilecek branch ismi>
```

Bir branch oluşturup direkt o branche geçmek istiyorsak;

```
git checkout -b <oluşturulacak branch ismi> <çekilecek branch ismi>
```

Bir branch silmek istiyorsak;

```
git branch -d <>
```

Uzaktaki bir branch silinmek isteniyorsa;

```
git branch -d -r <>
```

Branchimizi değiştirmek istiyorsak;

```
git switch <>
```

Eğer bir branch ile bir branchi birleştirmek istiyorsak merge işlemini yaparız.Bu komut aktif olarak bulunan branche ismi verilen branchi ekler.

```
git merge <çekilecek branch ismi> -m ""
```

1.2. Fast Forwarding

Bir branch açılıp yapılmak istenen değişiklikler bu branch üstünde yapılır. Bu branch üstünde işlem yapılırken ana branche dokunulmaz. Ana branche dokunulmayıp değişikliklerin branch üstünde yapılıp daha sonra bu branchin ana branche merge edilme işlemine **Fast Forwarding** işlemi denir.

1.3. Merge Conflict

Açılan bir branchte yapılan değişiklikle ana branchte yapılan bir değişikliğin merge işlemi sırasında çakışma durumuna verilen isimdir.Conflict'in çözülüp commit atılması gerekir. Merge etme işlemine gerek kalmaz. Otomatik olarak merge atar.

1.4. Stash

Bir projemizin olduğunu varsayalım. Bu projeden bir branch açıp branch üzerinde değişiklikler yaptık. Fakat bu branch üstündeki değişiklikleri commit etmeden terkar ana branche döndük. Git burada bu branchte yapılan değişiklikleri de ana branche getirir. Bu da ana branchteki dosyaların bozulmasına neden olur. Eğer bunu istemiyorsak branchten ana branche geçmeden önce

```
git stash
```

Komutu yazılması gerekir. Bu komut yapılan değişikleri saklamayı sağlar. Eğer değişiklikler saklanırsa bu değişiklikler branch değiştirilse bile taşınmaz. Böylece sorun ortadan kalkmış olur.

Eğer tekrardan branche gelip sakladığımız değişikliği ortaya çıkarmak istiyorsak;

```
git stash pop
```

2. Commitler arası Dolaşma / Geri Döndürme

Yapılan değişiklikleri staging area'ya taşıyıp geri çekmek istiyorsak veya direkt yaptığımız değişiklikleri geri çekmek istiyorsak;

```
git restore <>
```

Eğer bir commite geri dönülmek istenirse bu komut kullanılır. Fakat bu komutta dikkat edilmesi gereken bir nokta vardır. Belirtilen commite gidildiğinde değişiklik yapılması önerilmez. Eğer bir değişiklik yapılmak istenirse en sağlıklı yöntem bir branch açıp bu branch üstünde işlem yapmamızdır.

```
git checkout <commit hashi>
```

Eğer bir commite geri dönülüp o commite geri dönülürkenki tüm commitleri silmek istersek fakat bu commitlerin yaptıkları değişiklikler silinmeyecekse;

```
git reset <commit hashi>
```

Eğer bir commite geri dönmek istersek ve commite dönerkenki tüm commitleri silmek istersek ayrıca bu commitlerin yaptıkları değişiklikleri de silmek istersek;

```
git reset --hard <commit hashi>
```

Eğer bir commiti geri almak istersek ve bunların loglarını tutmak istersek;

```
git revert <commit hashi>
```

3. Farkları Gözlemleme

Anlık olarak yapılan değişiklikleri gözlemlemek istersek;

```
git diff
```

Güncel commit ile şu anki hal arasındaki farkları gözlemlemek istersek;

```
git diff HEAD
```

İki commit arasındaki farkları gözlemlemek istersek;

```
git diff <commit hashi> <commit hashi>
```

İki branch arasında değişiklikleri gözlemlemek istersek;

```
git diff <branch ismi> <branch ismi>
```

4. Rebase İşlemi

İsmi verilen branchin commitlerini en alttan sıralamaya başlar. Daha sonra bulunulan branchteki commitleri sırasıyla yazar. Böylece log dosyası düzenlenmiş olur.

```
git rebase <branch ismi>
```

5. Github nedir?

GitHub; kalabalık bir ekibin sürüm kontrol sisteminde bir yazılım geliştirirken kullanabilecekleri internet tabanlı bir depolama servisidir. GitHub, Git yönetim sistemi ile geliştirilen yazılımlarda kullanılır. Açık kaynak kodlu bir servis olması nedeniyle pek çok yazılım ekibi tarafından kullanılmakta ve geliştirilmektedir.

Github'ta bir repository oluşturup buraya uzaktan erişmek istiyorsak öncelikle bu repositorynin linkini bir kısaltma altına almamız gerekir.

```
git remote add <remote repo ismi> <repo linki>
```

Eğer projemizdeki bir değişikliği githubtaki repoya aktarmak istiyorsak;

```
git push -u <remote repo ismi> <branch ismi>
```

Eğer uzak repodaki proje ile kendi proje arasında fark var mı diye kontrol etmek istiyorsak;

```
git fetch <remote repo ismi> <branch ismi>
```

Eğer remote repodaki bir branche bağlanmak istiyorsak;

```
git checkout <branch ismi>
```

Eğer remote repodaki değişiklikleri localimizdeki projeye çekmek istiyorsak;

```
git pull <remote repo ismi> <branch ismi>
```

Eğer githubta bulduğumuz bir projenin bir kopyasını kendi localimize çekmek istersek clone etmemiz gerekir.

```
git clone <remote repodaki .git uzantılı link>
```

Clone edilen dosyalar pull edilerek güncel hali tekrar kendi localimize çekilebilir.

6. Eğer githubtaki bir projeye katkıda bulunmak istiyorsak

Eğer githubtaki bir projeye katkıda bulunmak istiyorsak bu proje sahibine yaptığımız değişikliği pull request oluşturarak onun takdirine bırakabiliriz. Aşamaları izleyecek olursak;

1) Proje fork edilir.

2) Proje fork edildikten sonra kendi localimize clone ederiz.

3) Clone ettikten sonra eklemek istediğimiz değişiklikleri ekleyip commit edip push ederiz.

4) Artık ana proje klasörüne giderek pull request oluşturabiliriz.