



Wise Quarter

first class IT courses

Team :

Course : QA SDET

Date :

Subject: Git – GitHub



The future at your fingertips

+1 912 888 1630

www.wisequarter.com










/wisequarter

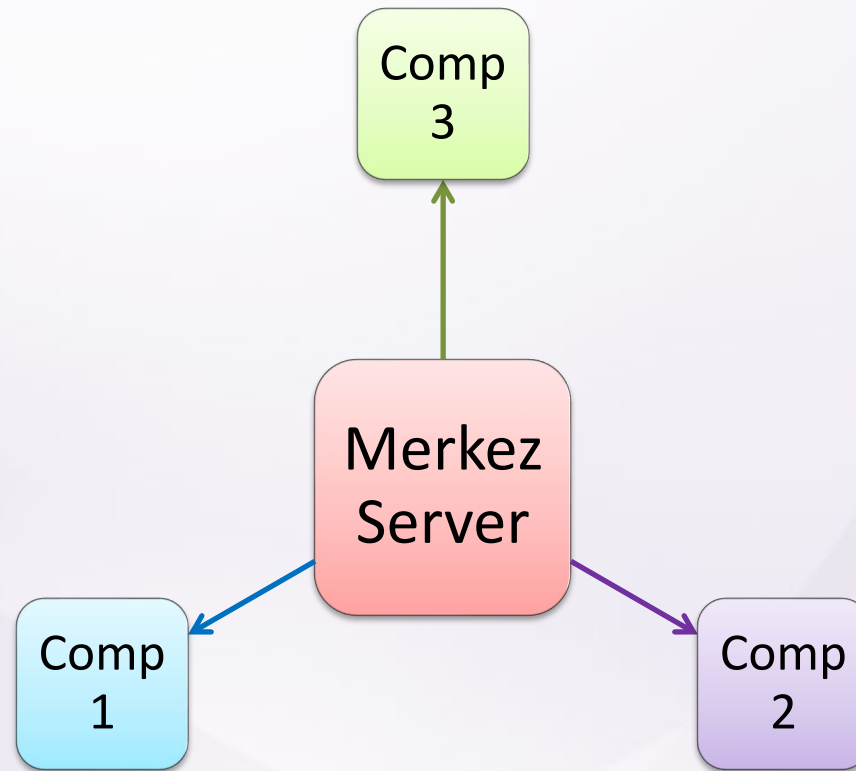
     /wisequarter

Murat Babayigit
www.wisequarter.com

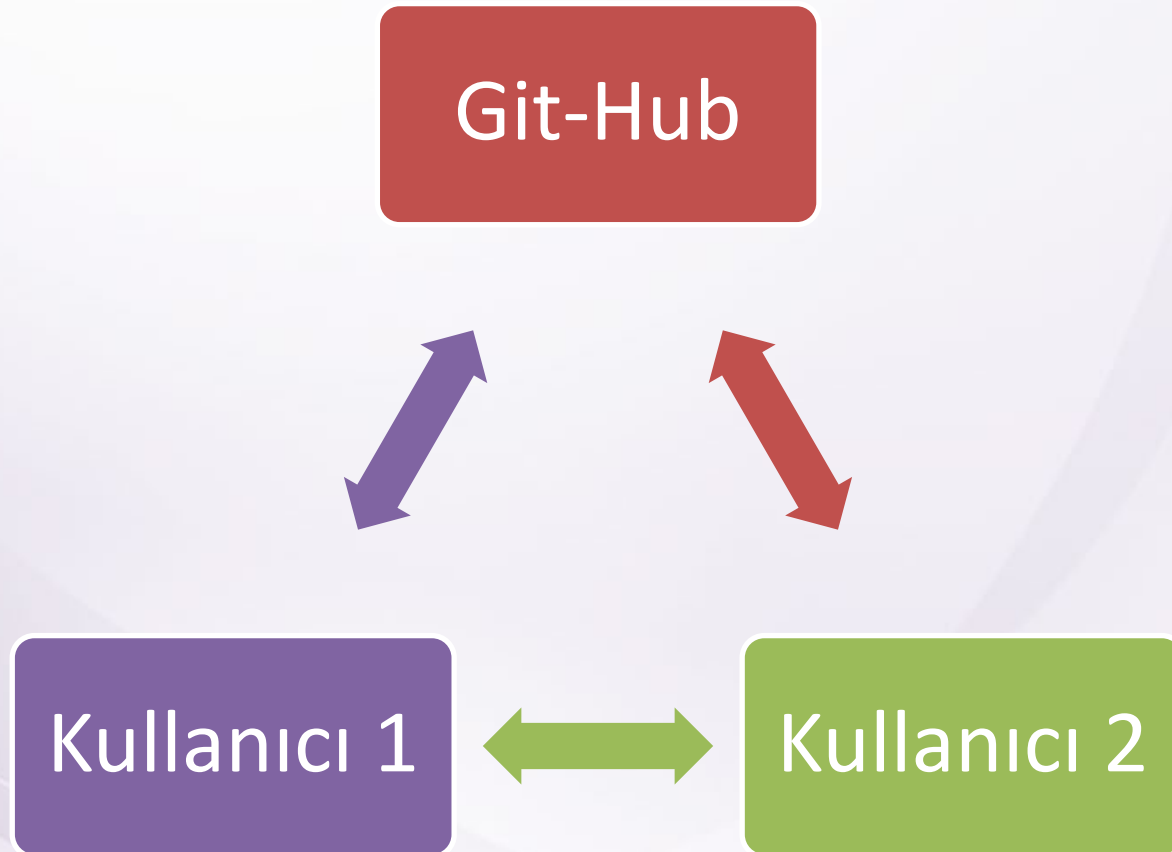
Local Versiyon Kontrol Sistemi

	GitHub_son_son_final_sonfinal_sonunsonu_bitti.pptx C: » WQ » Git-Github	29.07.2023 13:24
	GitHub_son_son_final_sonfinal_sonunsonu.pptx C: » WQ » Git-Github	29.07.2023 13:24
	GitHub_son_son_final_sonfinal.pptx C: » WQ » Git-Github	29.07.2023 13:24
	GitHub_son_son_final.pptx C: » WQ » Git-Github	29.07.2023 13:24
	GitHub_son_son.pptx C: » WQ » Git-Github	29.07.2023 13:23
	GitHub_son.pptx C: » WQ » Git-Github	29.07.2023 13:23
	GitHub Int.pptx C: » WQ » Git-Github	29.07.2023 13:23

Merkezi Versiyon Kontrol Sistemi



Dağıtık Versiyon Kontrol Sistemi





- ✂ Lokalde versiyon yönetimi yapmak
- ✂ Offline çalışabilmek
- ✂ Hataları geri alabilmek
- ✂ Versiyonlar arasında geçiş yapabilmek

<https://git-scm.com/>

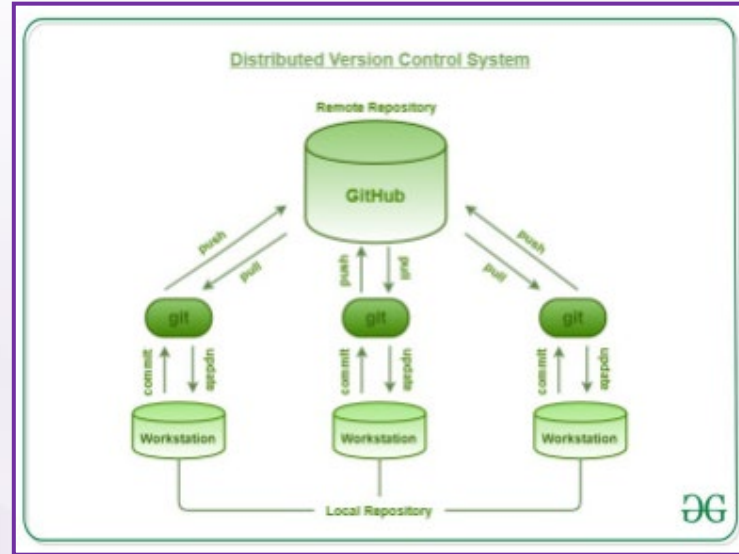


- ✂ Yedekleme (backup)
- ✂ Proje paylaşımı (share)
- ✂ Proje yayınlama (deploy)
- ✂ Ortak çalışma (collaboration)

<https://github.com/>

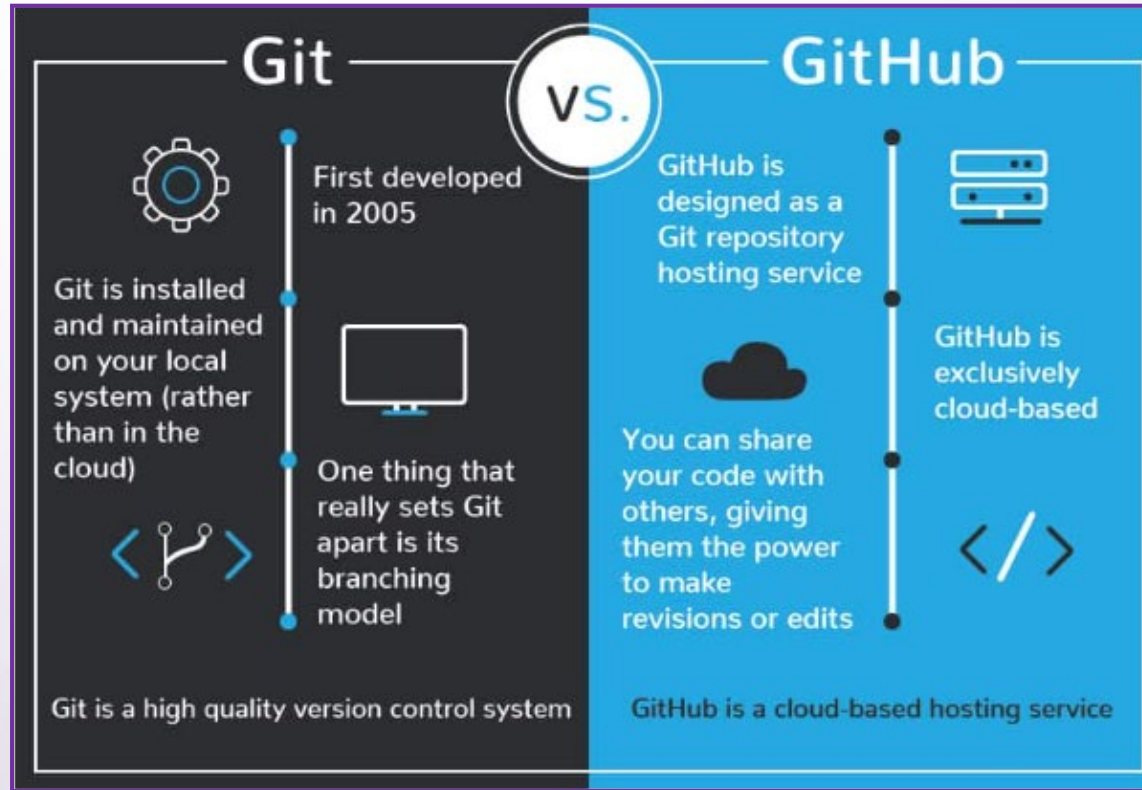
GitHub

GitHub, Git isimli bir version control system(sürüm kontrol sistemi) barındıran cloud tabanlı bir uygulamadır.



Git, bireysel küçük projelerden, ekip olarak çalışılan çok büyük projelere kadar her şeyi hızlı ve verimli bir şekilde yönetmek için tasarlanmış ücretsiz ve açık kaynaklı, **dağıtılmış bir sürüm kontrol sistemidir**(distributed version control system).

Git & GitHub



Git, kaynak kod geçmişinizi yönetmenize ve takip etmenize izin veren bir version kontrol sistemidir.

GitHub ise, Git depolarını(**repo**) yönetmenize izin veren bulut tabanlı bir barındırma(**hosting**) hizmetidir.

Temel Git İşlemleri

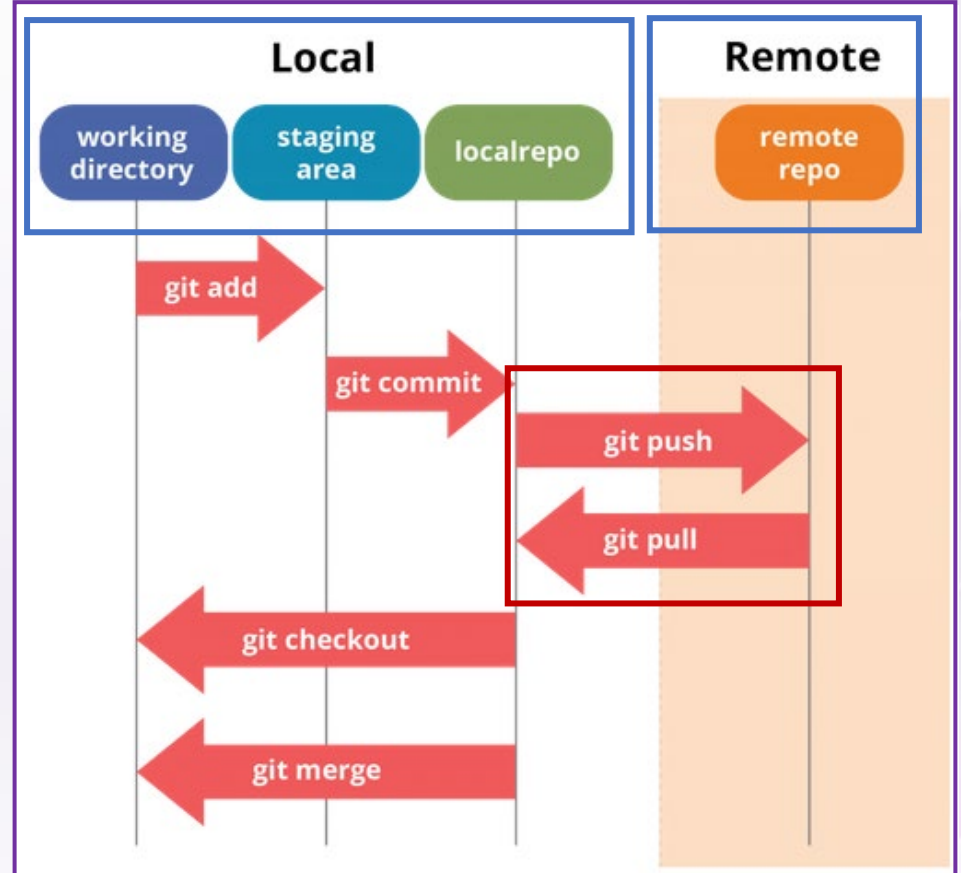


Tekli Kullanım

Tekli kullanımda her bir projemiz için GitHub'da bir repo oluşturup, bunu istediğimiz aralıklarla bilgisayarımız veya GitHub'daki hali üzerinden güncelleyebiliriz.

GitHub'da yaptığımız değişiklikleri her kaydettiğimizde (commit) Git yeni bir version oluşturur.

Versiyon kontrol sistemlerinin en büyük avantajı, istediğimiz bir anda versiyonlara ulaşabilmektir.



Tekli Kullanım

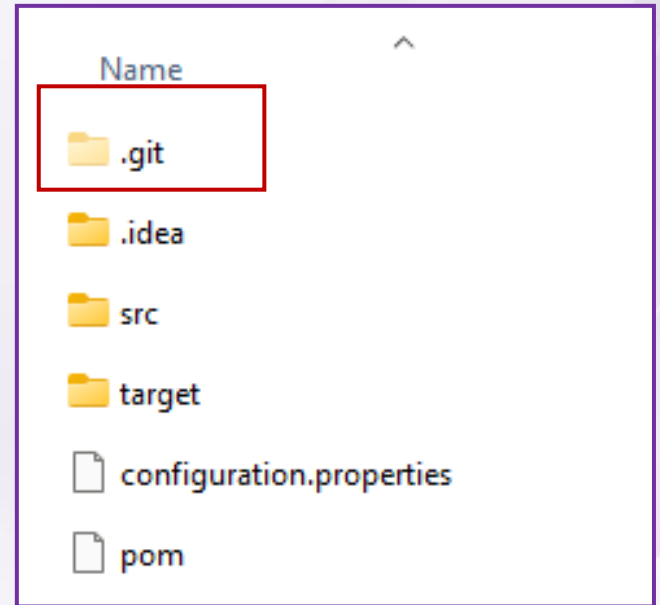
Her değişiklik yapıldığında tüm projenin bir kopyasını alıp kaydetmek çok büyük storage gerektirir.

Git her version'da projenin tamamını değil, bir önceki versiyona göre değişen kodları store eder.

Git version kontrolünü klasör üzerinden yapar, biz IntelliJ gibi bir IDE'de çalışsak da Git o projenin bilgisayarda kaydedildiği klasör üzerinden çalışır.

Git version kontrolünü yapmak için Git'e tanıttığımız (init) her klasöre .git isiminde bir klasör oluşturur ve tüm versiyonlar bu dosyada tutulur. Bu dosya silinirse tüm versiyonlar da silinir.

.git dosyasını görmek için bilgisayarınızda "Gizli dosyaları göster" seçeneğinin açık olması gerekir. (Mac için yukarı ok + command + .)



Git Komutlari

Git bilgisayarlarımızdaki windows benzeri uygulamalarla degil, bilgisayarların ilk kullanilmaya basladigi donemdeki gibi, siyah ekrana(command prompt / terminal) yazilan komutlarla calisir.

Gunumuzde GitHub Desktop gibi uygulamalar Command Prompt ihtiyaci olmadan da calismaniza izin verse de version kontrolu icin temel git komutlarini kullanmayi bilmelisiniz.

Essential git commands every developer

should know

- Git Clone
- Git branch
- Git checkout
- Git status
- Git add
- Git commit
- Git push
- Git pull
- Git revert
- Git merge
- Git remote
- Git fetch



Git ve command prompt'daki dosya kullanim komutlarini ezberlemek zorunda degilsiniz. Google ile basit bir arama yaptiginizda tum komutlar ve ne ise yaradiklari karsınıza cikacaktır.

Temel Git Komutlari

git --version : Bilgisayarlarımızda git kurulu ise versiyonunu verir

git config --global color.ui true : Terminalde kullanılan komutlari renklendirir.

git config --global user.name "John Doe": Yaptiginiz islemlerde gorunecek ismi tanimlar.

git config --global user.mail "a@b.com": Yaptiginiz islemlerde gorunecek email'i tanimlar.

NOT : --'den sonra **global** kullanilrsa o kullanicinin tum repolari icin tanimlar
system kullanilrsa, tum kullanicilar ve tum repolar icin tanimlar
local kullanilrsa, sadece o an kullanılan repo icin tanimlar

Local Repo Olusturma

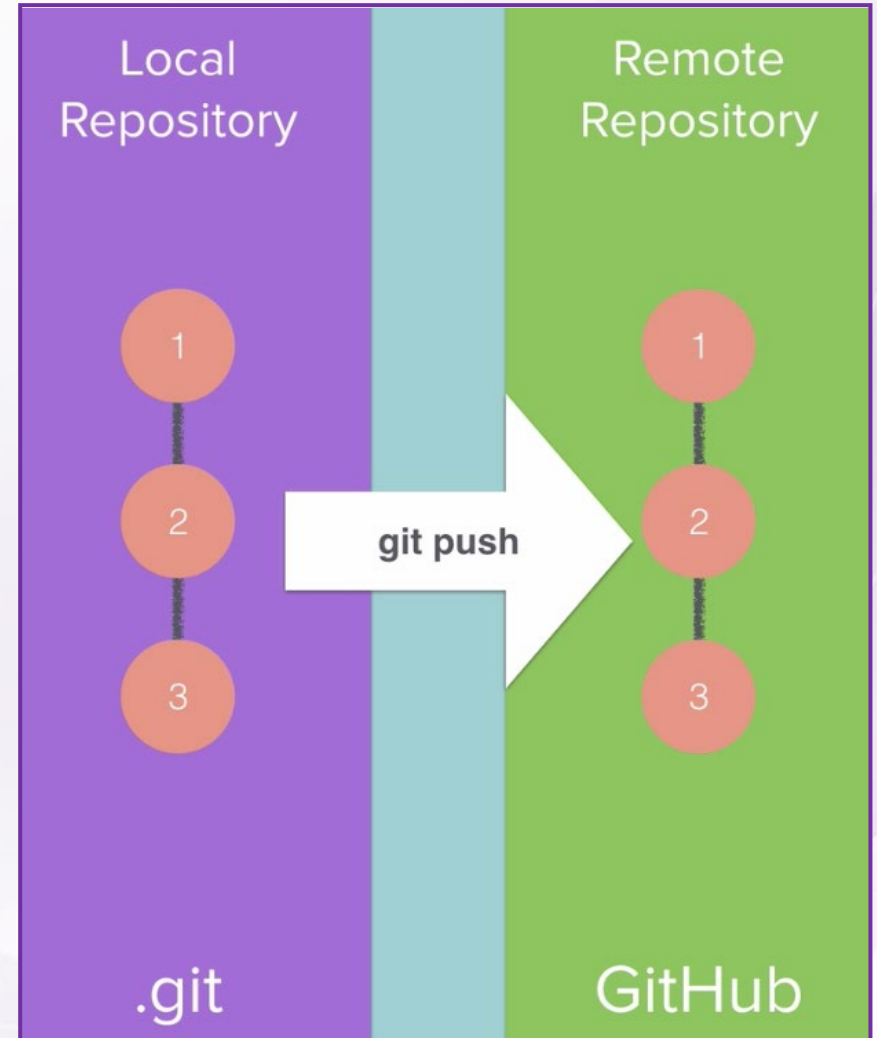
GitHub kullanmasak bile Git ile tüm projelerimizi local'de versiyon kontrol sistemi ile kullanabiliriz.

Bir klasörde git ile versiyon kontrolu yapmak istersek, terminalde o klasöre gidip

git init

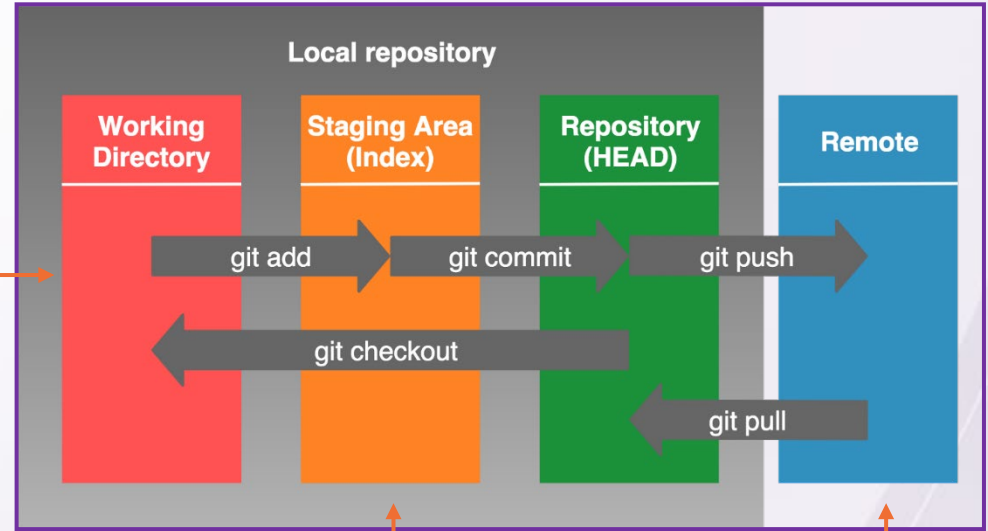
komutu çalıştırılmalıdır.

git init komutu çalıştırılınca git o klasörde hidden **.git** klasörü oluşturur ve bu klasör üzerinden versiyon kontrolu yapar.



Remote Repo'ya Yollama

Working Space (git init)
.git dosyasının bulunduğu klasordur.



Working space'de yapılan değişiklikler git add komutu ile **Staging Area**'ya yuklenir.

Staging Area'daki dosyalar git commit komutu ile gönderilmeye hazır hale getirilir

Remote repo'ya ilk yükleme için, GitHub'da remote repo oluşturulmalı ve local ile remote repo eşleştirilmelidir.

Daha önce GitHub'daki remote repo ile eşleştirilmiş olan projeler için, versiyonu remote repo'ya yollamak istediğimizde **git push** yeterli olur.

Remote Repo'ya Yollama

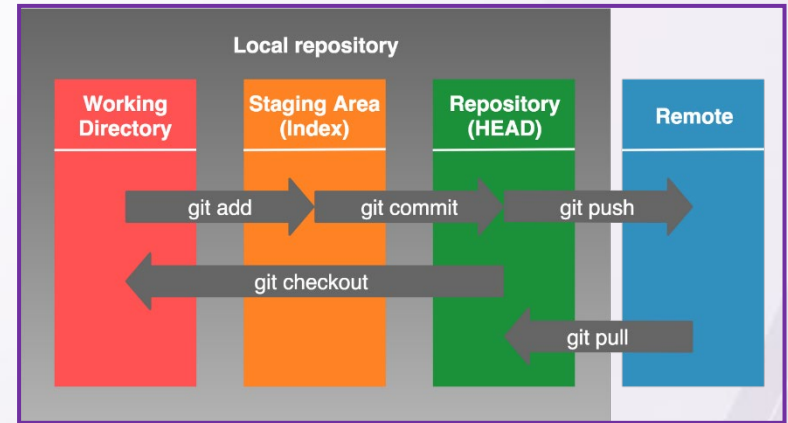
Kisisel kullanimda yapacagimiz tum islemler, kendi bilgisayarimizdaki **local repository** ile GitHub'daki **Remote Repository** arasinda **senkronizasyondan** ibarettir.

1. Bilgisayarimizdaki bir dosyayi ilk defa Git'e tanitip, Github'da olusturacagimiz bir repo'ya yukleme

- dosyalar git'e ekleyip working space olusturma **git init**
- Dosyalari staging area'ya gonderme **git add** .
- Versiyon olusturup, local repository'e gonderme **git commit -m "commit ismi"**

GitHub'da remote repo olustur.

- Remote repo'daki branch'i tanimla **git branch -M main**
- Remote repo ile local repoyu eslestirme **git remote add origin "remote repo url"**
- local repodaki dosyalari remote repoya yollama **git push -u origin main**



Cmd / Terminal Komutlari

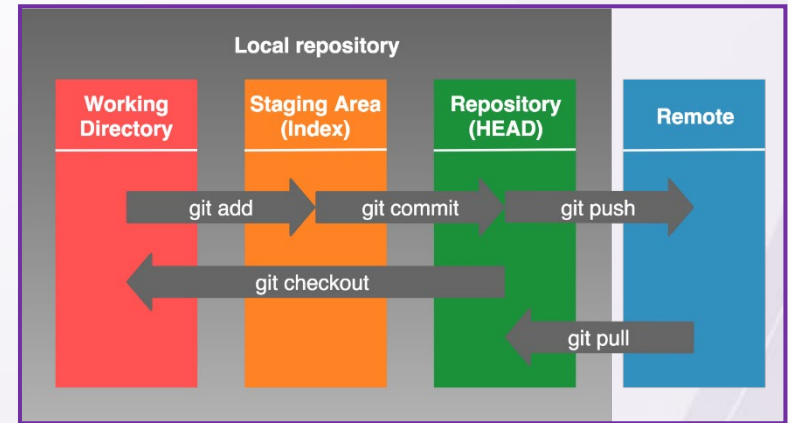
Git bilgisayarlarımızdaki windows benzeri uygulamalarla degil, bilgisayarların ilk kullanilmaya basladigi donemdeki gibi, siyah ekrana(command prompt / terminal) yazilan komutlarla calisir.

	Windows	Mac
Islem	Komut Istemi	Terminal
Klasor hareketleri	cd	
Listeleme	dir	ls
Ekran temileme	cls/clear	
Klasör Oluşturma	mkdir	
Klasör Silme	rmdir	
Dosya oluşturma	echo > dosyaAdi.uzantisi	
Dosyanın içini görme	more	cat
Dosya silme	del	rm
Klasör ve dosya ismi değiştirme	ren	mv

Remote Repo'dan Kopyalama

2. GitHub'da beğendiğimiz ve erişim yetkimiz olan bir repoyu bilgisayarımıza indirme

- Dosyaları yüklemek için bir klasör oluşturup, git'e ekleme **git init**
- GitHub'da beğendiğimiz reponun url'ini kopyalayip, remote repoyu local repoya indirmek için **git pull remoteRepoUrl**



IntelliJ'de bu işlem için new project menüsünden url ile proje açma seçilir.

NOT : Eger oluşturdugumuz local repo ile remote repoyu bundan sonra da senkronize edeceksek **git pull** yaptıktan sonra local'de commit oluşturup, remote repoya bağlayalım.

- Remote repo'daki branch'i tanımla **git branch -M main**
- Remote repo ile local repoyu eşleştirme **git remote add origin "remote repo url"**
- local repodaki dosyaları remote repoya yollama **git push -u origin main**

Local Repo'da Commit Gecmisi

Git ile yaptigimiz tum version takibi local repoda tutulmaktadır.

Commit yaptigimiz versiyonlar arasinda gezinebilir, istedigimiz commit'e gecebilir ve istersek o versiyon'u projemizin son hali sekline getirebiliriz.

- Local repoda yaptigimiz tum commit'eri listelemek icin **git log**

- Local repoda yaptigimiz tum commit'eri, her commit bir satir olacak sekilde listelemek icin **git log --oneline**

Log listesinde son version HEAD olarak belirtilir.

```
commit 01867b98e9cb888ba78dd16dd52e65abd2cf55d3 (HEAD -> main)
Author: ahmet bulutluoz <abbulutluoz@gmail.com>
Date: Sun Sep 25 07:06:03 2022 +0200

    C03

commit 03c8adde7006b96293a52990e03831af7c8bc73b
Author: ahmet bulutluoz <abbulutluoz@gmail.com>
Date: Sun Sep 25 07:05:22 2022 +0200

    C02

commit 90bd42f51e7692bfc3bd79ca3f06b3aef8d6dd57
Author: ahmet bulutluoz <abbulutluoz@gmail.com>
Date: Sun Sep 25 07:04:41 2022 +0200

    C01
```

```
670da98 (HEAD -> main) D03
3f61cc8 D2
5ad0480 (deneme) D01
01867b9 C03
03c8add C02
90bd42f C01
```

Istenen Commit'e Donme

Git ile projeyi istedigimiz bir commit'e dondurebiliriz.

Eski bir commit'e gitmek icin `git checkout 01867b9`

```
01867b9 (HEAD) C03
03c8add C02
90bd42f C01
```

```
670da98 (HEAD -> main) D03
3f61cc8 D2
5ad0480 (deneme) D01
01867b9 C03
03c8add C02
90bd42f C01
```

Eski commit'e gittikten sonra, o versiyon'u projemizin son hali yapmak icin yeniden versiyon yapmaliyiz.

`git add .`

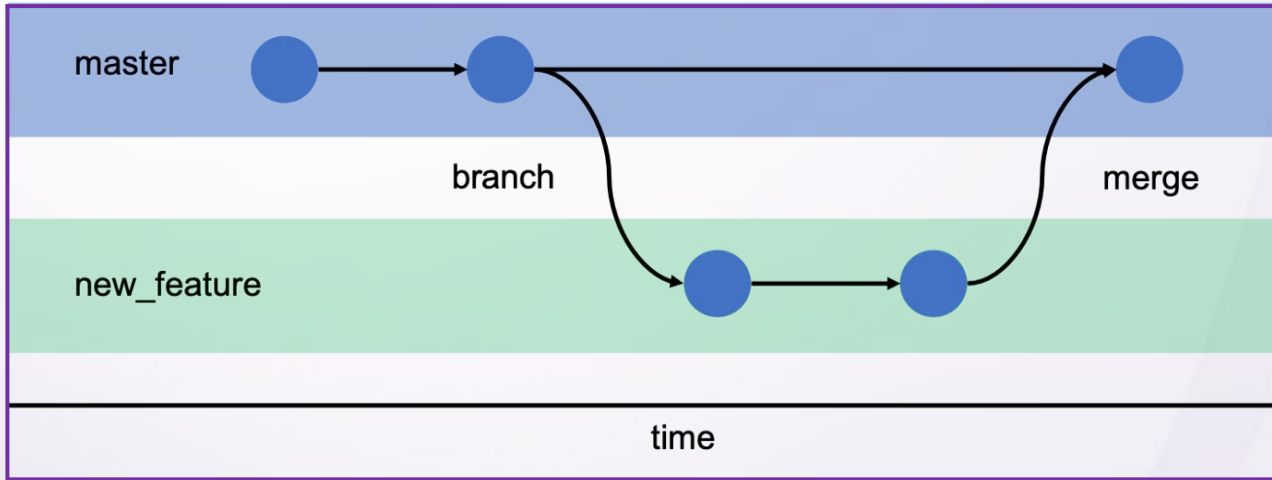
`git commit -m "commit ismi"`

Ancak git bu istegimize uyarı mesajı verecektir. Burada sorumlulugu almamiz ve eski versiyona donmek istedigimizi belirtmemiz gerekir. Bunun icin terminale

`git revert 01867b9`

komutlari yazilmalidir.

Local Repo'da Branch Kullanimi



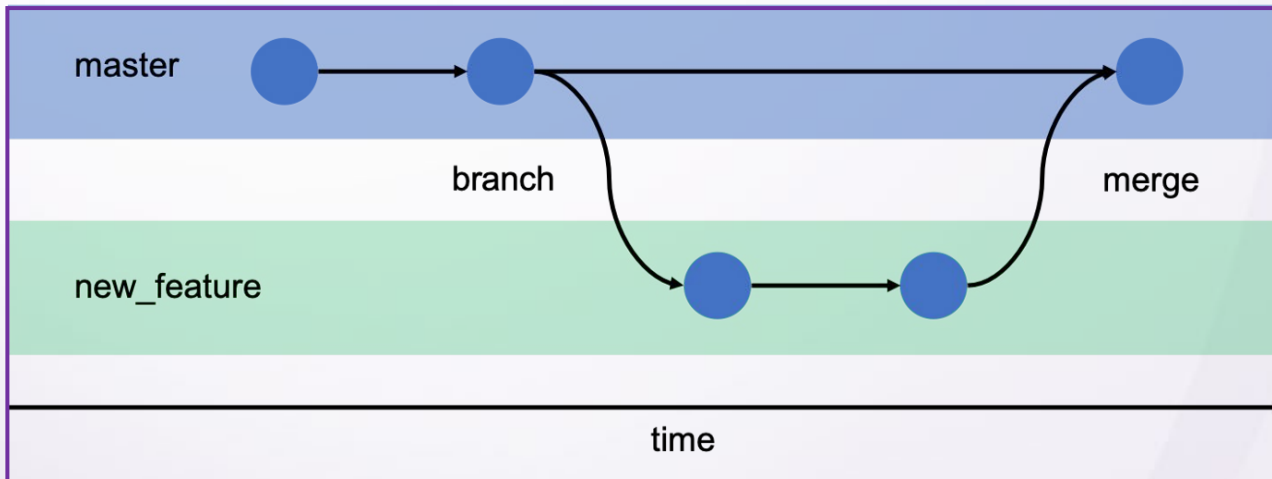
GitHub dagitilmis merkezli bir version kontrol sistemidir.

Bu ozelligi, bir projede calisan pek cok kisinin ayni anda degisiklikler yapabilmesi, kendi kodlarini ana projeye gondermeleri ve yeni kodlar ile ana projeyi guncellemelerine imkan tanir.

Branch (dal) yapisi local repo icin de mumkundur.

Projemizde sonucu belirli olmayan bir sey denemek istiyorsaniz veya projenizde calismaya devam ederken biryandan da bazi degisikliklere calismak istiyorsak main branch disinda bir branch olusturup onunla calisabiliriz.

Local Repo'da Branch Kullanımı



- Branch listesini gormek icin **git branch**
- Yeni bir branch olusturmak icin **git branch branchismi**
- Istenen branch'e gecmek icin **git checkout branchismi**
- Baska bir branch'i kullandigimiz branch ile birlestirme **git merge branchismi**

NOT : Merge yaptigimiz branchlerde ayni kod satirinda farkli kodlar varsa **conflict** olur. Git iki kodu da gosterir ve bizden manuel olarak bunlari duzenlememiz istenir.

GIT / GITHUB 2

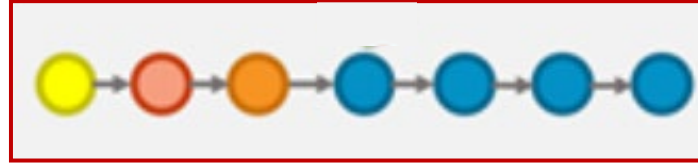
COLLABORATION

GITHUB

GitHub, Git adlı bir sürüm kontrol sistemini (VCS) barındıran bulut tabanlı bir hizmettir.

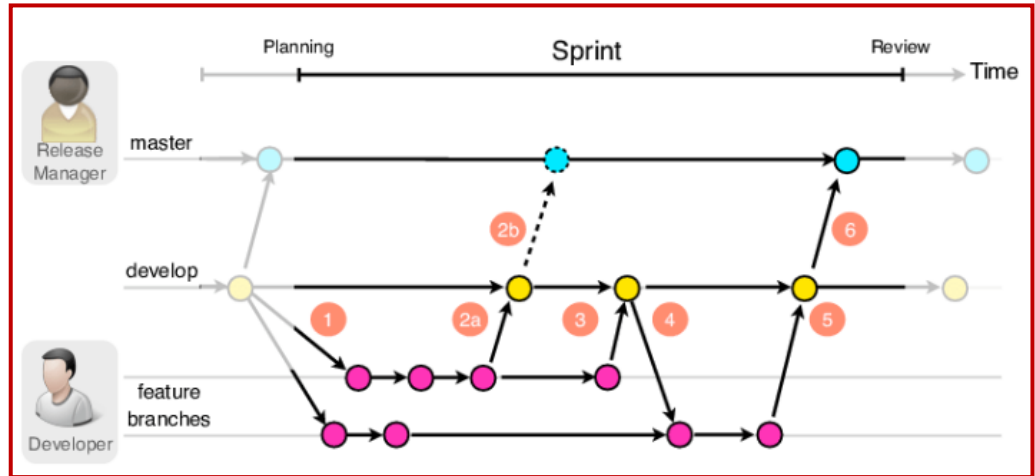
Basit bir proje version takibi

Baslangic



Bitis

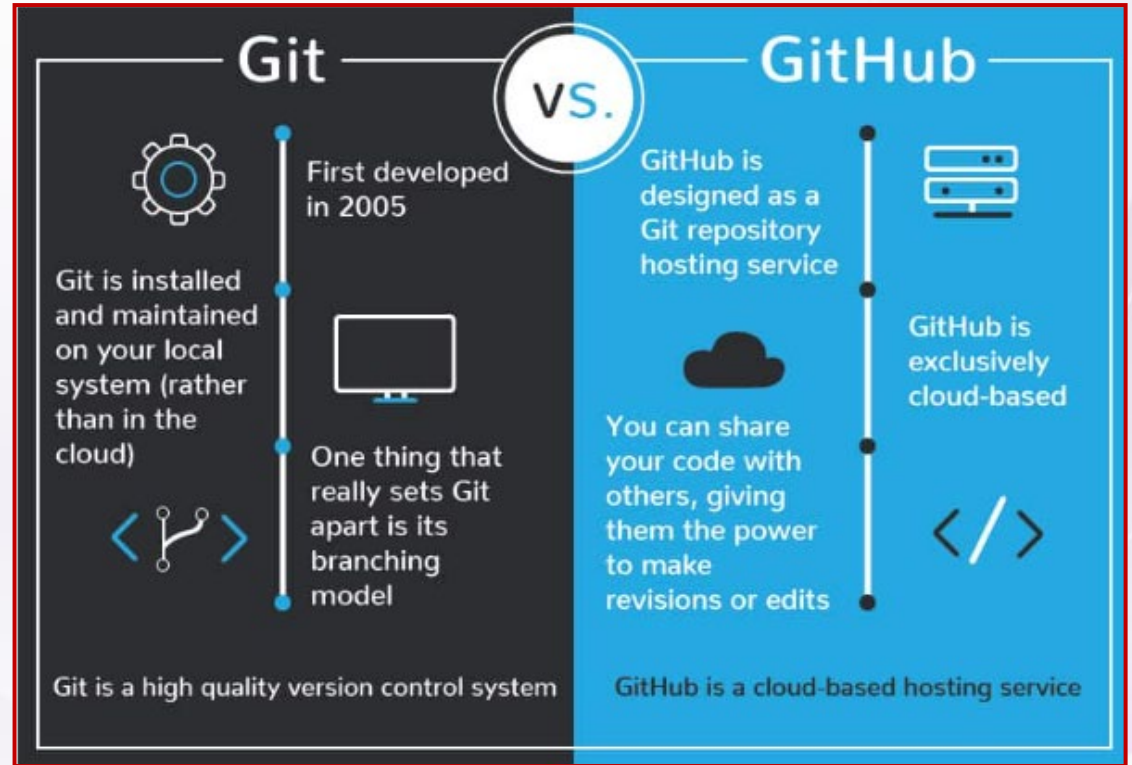
Gunumuzdeki proje takibi



GIT - GITHUB

Git, kaynak kod geçmişinizi yönetmenize ve takip etmenize izin veren bir version kontrol sistemidir.

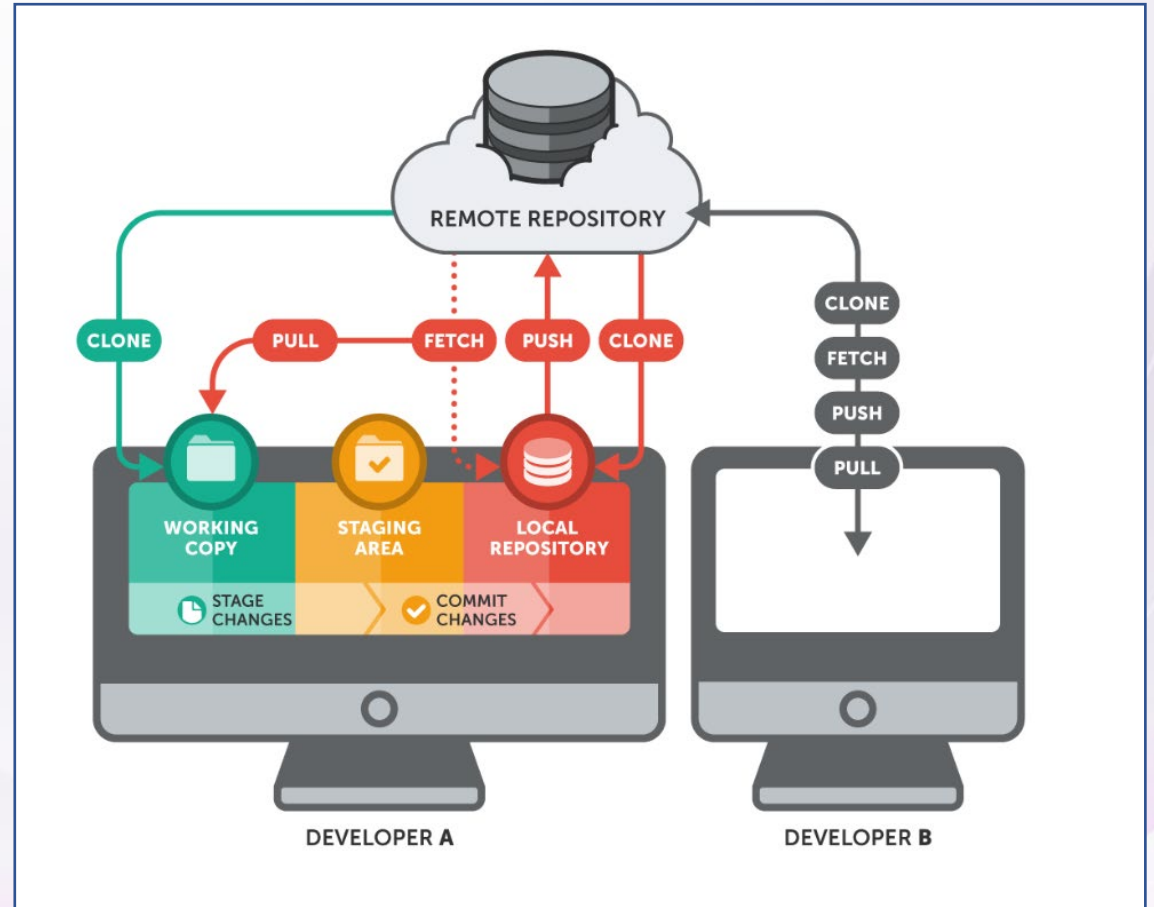
GitHub ise, Git depolarını(repo) yönetmenize izin veren bulut tabanlı bir barındırma(hosting) hizmetidir.



PROJE ISLEYISI

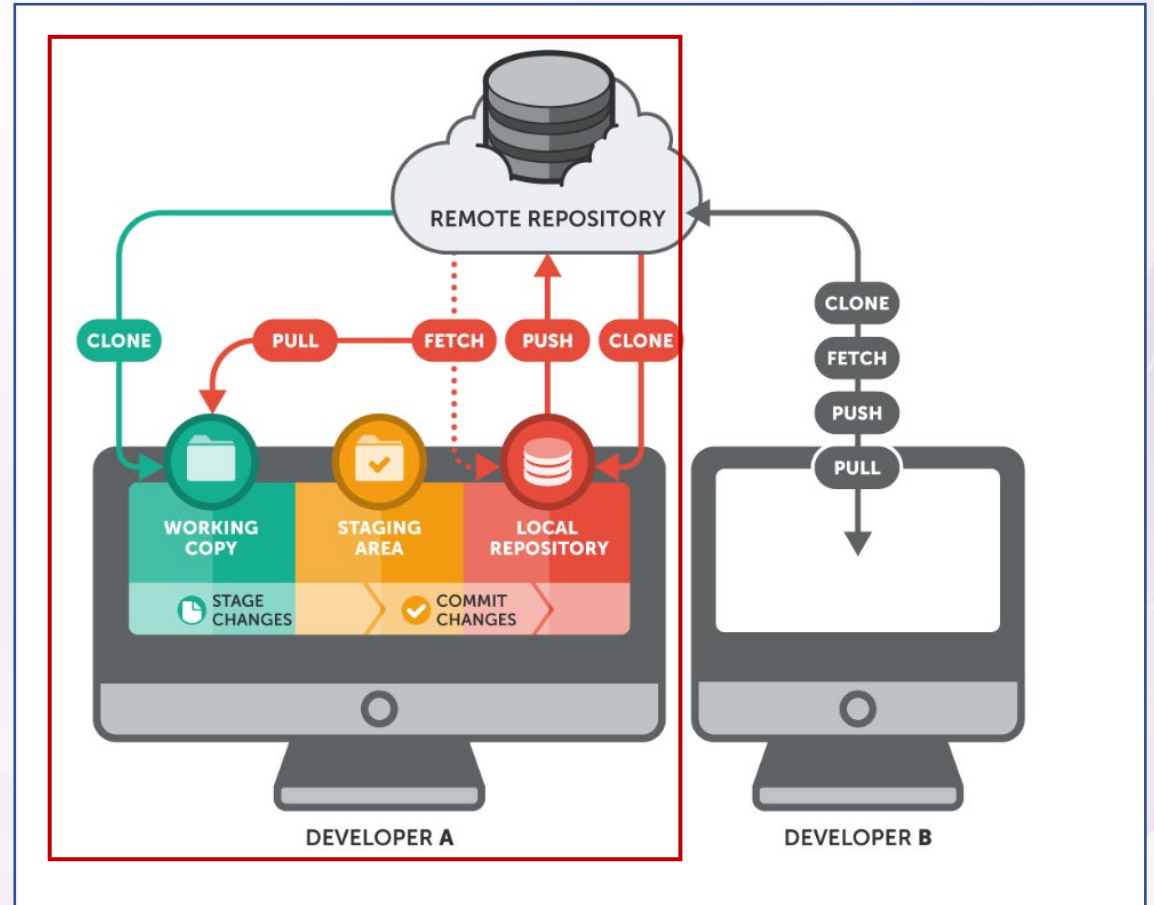
Gunumuzdeki projelerde birden fazla kisi veya team birlikte calismaktadir.

Bu durumda projeyi gelistirme isleminin yaninda projenin stabilitesini korumak, collaborator olarak rol alanlarin birbirlerinin calismasini bozmamasi da onem kazanir.



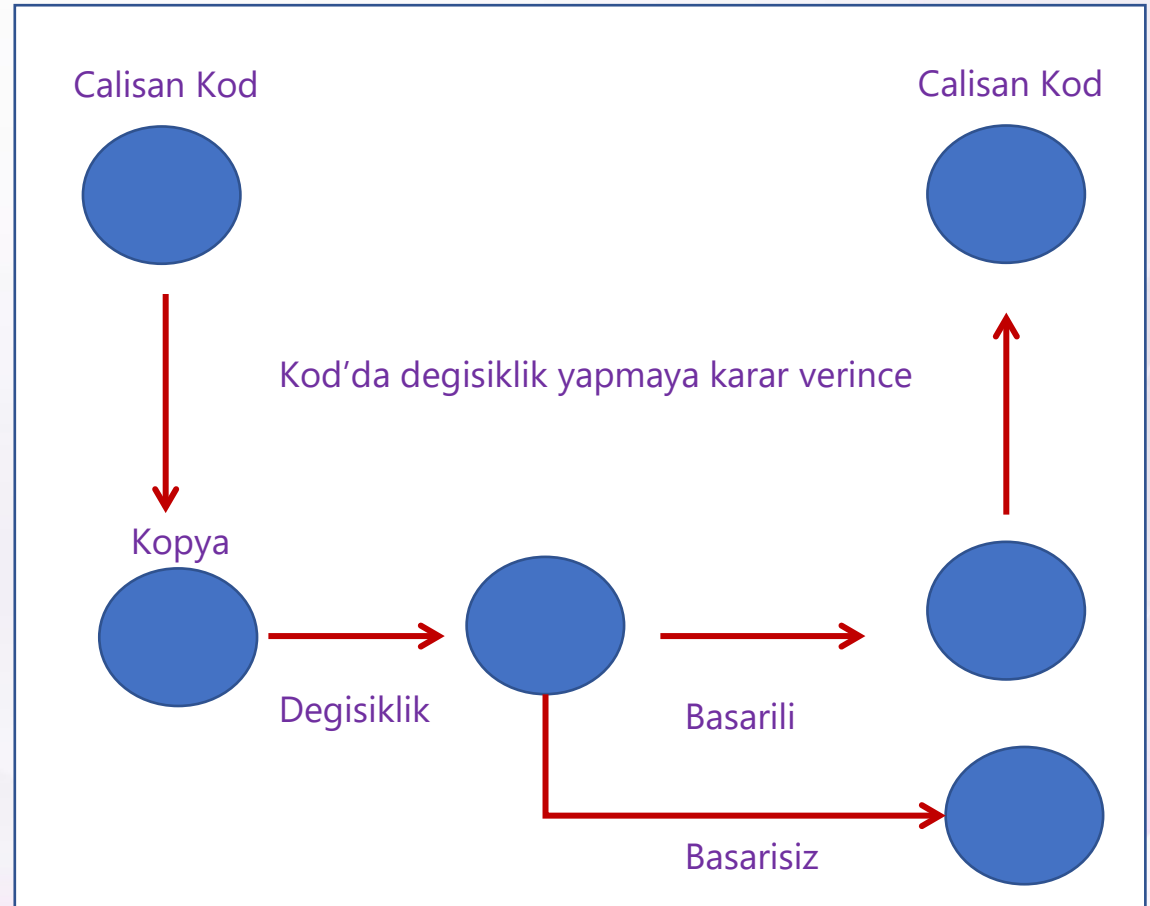
KISISEL SUREC

Calismaya devam eden bir uygulamadan son halini alip, bize verilen gorevleri yapip yeniden ana uygulamaya yollayincaya kadar, diger kullanicilar tarafından yapilan bir cok degisiklik olabilir. Ana projeye yapilacak her ekleme mutlaka system admin tarafindan control edilmelidir.



LOCAL BRANCH KULLANIMI

Biz oncelikle kisisel kullanimda branch yapisinin nasil calistigini ogrenelim.



LOCAL BRANCH KULLANIMI

1. Bilgisayarimizdaki bir dosyanın kopyasını alıp, geliştirmeleri yapma, başarılı ise ana dosyayı güncelleme

I- Biz oluşturduğumuz branch'da çalışırken, main branch'da hiç değişiklik yapılmıyorsa

A. Local'de branch oluşturma **git branch deneme**

(projedeki branch listesini ve hangi branch'de olduğunuzu görmek isterseniz **git branch**)

B. Oluşturduğumuz deneme branch'ına geçme **git checkout deneme**

Artık oluşturduğunuz ve geçiş yaptığınız branch'de istediğiniz eklemeleri yapabilirsiniz.

Eklemeler bittikten sonra kodumuz çalışıyorsa, yeni branch'de commit oluşturmalsınız.

C. Projenin yeni halini local repoya ekleme : **git add .**

D. Commit oluşturma : **git commit -m "commit ismi"**

E. Ana branch'a geçme : **git checkout main**

F. Eski kodlara dokunmadığımız için yeni eklenen kodları direk main branch'a ekleme : **git merge deneme**

LOCAL BRANCH KULLANIMI

1. Bilgisayarimizdaki bir dosyanın kopyasını alıp, geliştirmeleri yapma, başarılı ise ana dosyayı güncelleme

II- Biz oluşturduğumuz branch'da çalışırken, main branch'da değişiklik yapılıyorsa, son basamakta sorun çıkabilir.

Merge edeceğimiz branch'larda aynı satırda değişiklik yapıldıysa Git kodlardan hangisini tercih edeceğimizi bize soracaktır.

Merge yaptıktan sonra Git çakışan her iki kodu açıklamalarıyla getirip, bizden düzenleme yapmamızı bekleyecektir. Düzenleme yapılmazsa kod hata verecektir.

Manuel düzeltme yaptıktan sonra commit yapmamız gerekir.

A. Local'de branch oluşturma **git branch deneme**

B. Oluşturduğumuz deneme branch'ına geçme **git checkout deneme**

C. Projenin yeni halini local repoya ekleme : **git add .**

D. Commit oluşturma : **git commit -m "commit ismi"**

E. Ana branch'a geçme : **git checkout main**

F. Eski kodlara dokunmadığımız için yeni eklenen kodları direkt main branch'a ekleme : **git merge deneme**

REMOTE BRANCH KULLANIMI

2. Remote repo'da da branch olusturmak mumkundur.

Remote branch kullanimi kisisel kullanimda cok gerekli olmasa da, ekip calismalarinda **MUTLAKA** yapilmasi gereken bir islemdir.

Ekip calismalarinda, calisan (canli) kodlara zarar verilmemesi cok onemlidir.

Calisan kodun bozulmaması için herkes kendi branch'ında çalışır ve işi bittiginde birleştirme talebi "pull request" oluşturur

Sistem yöneticisi (team lead, development team lead) pull request talebini görür, inceler ve uygun görürse çalışan kod ile birleştirir.

Local'imizde çalışmaya başlamadan önce ve değişikliklerimizi remote repo'ya göndermeden önce çalışan kod bilgisayarımıza pull edilip, yaptığımız değişikliklerin projenin son hali ile karşılaştırılması **DAHA SAĞLIKLI** olacaktır.

REMOTE BRANCH KULLANIMI

2. Remote repo'da da branch olusturmak mumkundur.

Remote branch kullanimi kisisel kullanimda cok gerekli olmasa da, ekip calismalarinda MUTLAKA yapilmasi gereken bir islemdir.

Local'de main branch'da oldugunuzu kontrol edin, farkli bir branch'da iseniz main'e gecin.

1. Local'de calismaya baslamadan once projenin son halini alma
git pull

2. Local'da calismak icin yeni bir branch olusturun **git branch deneme**

3. Deneme branch'ina gecip(**git checkout deneme**) istediginiz degisiklikleri yapin ve sonra commit olusturun : **git add . , git commit -m "commitismi"**

4. Siz local'de deneme branch'inda calisirken remote calisan kod'da baskalarinin yaptigi degisiklikler olabilir. Oncelikle kendi degisikliklerinizin calisan kodun guncel haliyle karsilastirilmasinda fayda var. Bu durumda local'de main branch'a gecin (**git checkout main**),calisan kodun son halini pull edin (**git pull**). Boylece local main ile remote main ayni olur

5. Local'de deneme branch'ini main branch ile merge yapin. **git merge main**

Cakisma varsa duzeltin, boylece sizin kodlariniz gonderilmeye hazir olacaktır.

REMOTE BRANCH KULLANIMI

2. Remote repo'da da branch oluşturmak mümkündür.

Remote branch kullanımı kişisel kullanımda çok gerekli olmasa da, ekip çalışmalarında MUTLAKA yapılması gereken bir işlemdir.

Su ana kadar local'de farklı bir branch oluşturup değişiklikleri yaptık, sonra projenin son halini local main branch'a indirip, değişiklik yaptığımız branch'da merge ettik.

Yani degistirdigimiz kisimlarin ana proje ile cakisip cakismadigini gorduk ve cakismalar varsa bunlarla ilgili yapmamiz gerekenleri yaptik.

Simdi yaptigimiz projeleri ana projeye gonderme zamani

1. Local'de calisip son haline getirdigimiz branch'ta oldugunuzdan emin olun, yoksa gecis yapin **git branch deneme**
2. Local'da son haline getirdigimiz branch'i ayni isimle uzak masastune gonderin **git push --set-upstream origin deneme**
3. Github'da yaptiginiz degisikliklerin calisan koda eklenmesi icin **pull request** olusturun
4. Sistem yoneticisi olusturdugunuz **pull request** group inceleyecek, cakismalar varsa giderip, calisan koda ekleyecektir.

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

