**GIT & GITHUB**

**Git nedir? (LOCALE de çalışır) (Yerel VKS)**

Versiyon kontrol sistemi: projede dosya ve klasör yapısındaki değişiklikleri kaydeden bir sistemdir.

* Bazı dosyaların veya projenin tamamının bir önceki versiyona döndürülmesi,
* Zaman içerisinde yapılan değişikliklerin karşılaştırılması,
* Probleme neden olabilecek değişikliklerin en son kimin tarafından yapıldığının t espiti

Git Ne amaçla kullanılır;

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

**Github nedir? (REMOTE da çalışır) (Merkezi VKS)**

Git in Cloud versiyonudur.

Github Ne amaçla kullanılır;

* Yedekleme (backup)
* Proje paylaşımı (share)
* Proje yayınlama (deploy)
* Ortak çalışma (collobration)

**Neden Git?**

* Yedekler ve versiyon versiyon kaydeder
* Ayni projede birden fazla kisi es zamanli olarak calisabilir
* Offline destegi

**Kurulum :**

1. Adim: Git uygulamasini indir ve kur (https://www.git-scm.com/)

2. Adim: GitHub hesap ve Repository olustur

3. Adim: Ornek bir proje olustur ve once Git e sonra GitHub a yedekle

**Komutlar :**

**git --version** ==> git'in kurulup kurulmadığını, versiyon bilgisini kontrol ederiz.

**git config --global user.email "email\_adresiniz"**

**git config --global user.name "isminiz"**

Yapılan commit leri burada belirtilen isim ve eposta ile ilişkilendirir. Repo da çalışan diğer kişiler bu isimi görür.

**\*\*\*** bu iki confituration 'u yapmazsak versiyon oluşturmaz.

• **System** parametresi kullanıldığında tüm kullanıcılar ve tüm repolar üzerinde etkiliolur

• **Global** parametresi geçerli kullanıcının tüm repolar üzerinde etkili olur

• **Local** parametresi ise sadece geçerli repo üzerinde etkili olur

**git cofig –global color.ui true** ==> Terminalde komutların renklendirilmesini sağlar.

**Repository**

Versiyon kontrol ve birlikte çalışma altyapısını ayrı tutmak istediğimiz her bir bağısız yapıya repository denir.

Genellikle her proje için ayrı bir repository tanımlanır.

Kişisel Kullanım : Kişisel kullanımda yapacağımız tüm işlemler, kendi bilgisayarımızdaki local repository ile GitHub’daki Remote Repository arasındaki **senkronizasyondan** ibarettir.

1. Bilgisayardaki bir dosyayı ilk defa Git’e tanıtıp, GitHub’da oluşturacağımız bir repo’ya yükleme

* Local’de repo oluşturma **git init**
* Klasörümüzdeki tüm dosyaları local’deki repoya yükleme **git add .**
* Local repoda versiyon oluşturma **git commit -m "isim”**
* Remote repo’yu GitHub’da oluşturun

GitHup hesabımız 🡺 Repositories 🡺 new 🡺 Repository name 🡺 public/ private 🡺 Creating Repository

**komut satırında yeni bir depo oluşturun**

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/idris61/deneme.git

git push -u origin main

komutları girdikten sonra senkronizasyon tamamlanır.

**Repository silme :**

Github 🡺 Repositories 🡺 Silincek repositroy 🡺 Settings 🡺 General 🡺 Denger Zone 🡺 Delete this reporistory

**Local repo oluşturma**

**git init** ==> Local'de repo oluşturma. porojemizi git ile irtibatlandırıyor. kullanılınca proje klasöründe .git klasörü oluşturulur.

Bu, local repomuzu saklayacaktır.

(Initialized empty Git repository in C:/Users/idris/IdeaProjects/Git\_Test/.git/)

**Her proje için bir kez bu komut kullanılır.**

**git add .** ==> Klasörümüzdeki tüm dosyaları local'deki repoya yükleme komutu

Working Space de yaptığımız değişikliği Staging Area'ya yollar.

**git status** ==> Working Space veya Staging area’nın genel durumunu görmek için kullanılır.

(yardımcı komut) Klasörümüzde değişiklik yaptıktan sonra ==> git add . yapmadan, bu komutu

yazarsak yaptığımız değişiklik kırmızı renkte görünür,

git add . den sonra bu komutu yazarsak, değişikliğimiz yeşil görünür.

**git commit -m "first commit"** ==> Local repoda versiyon oluşturur

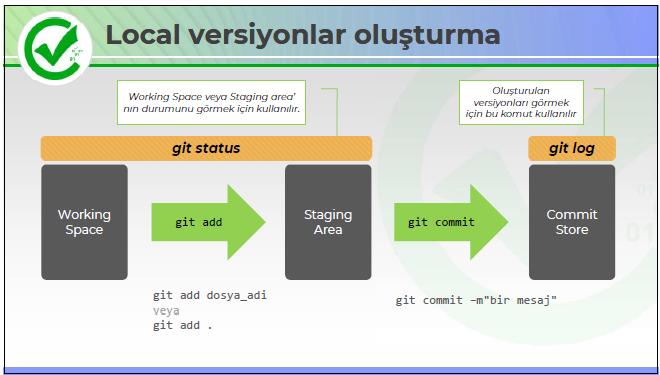
**git log** (yardımcı komut) ==> Oluşturulan versiyonların listesini görmek için bu komut kullanılır.

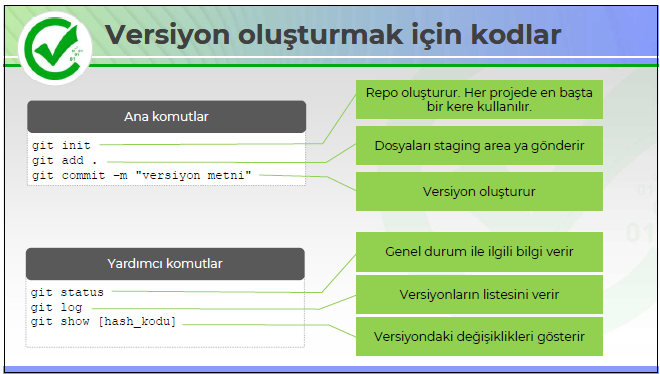
Commit store daki, yani versiyonları sakladığımız yerin listesini verir.

**git log --oneline**  ==> tek satırda özet bilgiyi gösteriyor. değişikliklerin hash kodlarını bu komutla

alırız.

**git commit -a -m "first commit" ==>** Hem add . , hemde commit komutu görevini yapar







**Değişiklikleri iptal etme Komutları :**

**git show [hash\_kodu]**  ==> Versiyondaki değişiklikleri gösterir.

**git restore .**  ==> working space de, henüz staging area'ya gönderilmemiş

(git add . yapılmamış)

ve en sonki versiyondan sonra yapılan değişiklikleri siler

**git restore [dosya]** ==> Tek bir dosyayı iptal eder.

**git restore --staged .** ==> staging area'da bulunan, henüz versiyon oluşturmadığımız değişiklikleri

temizler. Fakat working space'e dokunmuyor.

git status==> yaptığımızda değişiklikleri kırmızı olarak görmeye devam ederiz. Buradan da silmek istiyorsak git restore . yapmalıyız.

**git restore --staged [dosya]** ==> Tek bir dosyayı iptal eder.

**git reset --hard**  ==> Bu komutla; hem Stage Area, hem de Working Space deki yapmış

olduğumuz değişiklikleri siler. Bir önceki versiyona döner.

**Önceki versiyonlara dönme Komutları :**

**git checkout [hash] ==>** Önceki versiyonu incelemek için yazılır.

**git commit -m“….” ==>** Gidilen versiyonu kalıcı hale getirmek için bu komut kullanılr.

**git reset --hard [hash] ==>** Geri alınamayacak şekilde hash kondunu yazdığımız versiyona döndürür.

**----------------------------------------------------------------**

**git push --force**

**git push orgin idris -- force**

**git push** [**https://github**](https://github)**......... --force**

**----------------------------------------------------------------**

**Branch (Dal) Komutları :**

 Orginal kodların güvenliği sağlanır

 Her developer kendi bölümünden sorumlu olur

 Daha hızlı geliştirme yapılır

 Daha az hata oluşur

 Sorunlar daha hızlı düzeltilir.

 Organize kod yapısı sağlanır

 Kaos olmaz

**git branch [isim] ==>** Yeni branch oluşturur.

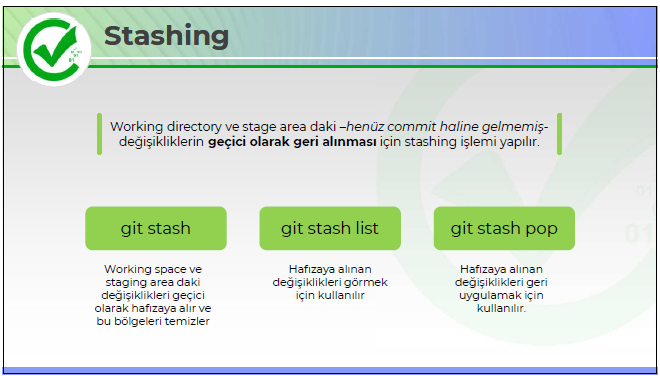
**git branch ==>** Mevcut branch’leri listeler

**git checkout [isim] ==>** “isim” deki branch’i aktif hale getirir.

**git merge [isim] ==>** iki branch’i birleştirir.

**git branch -m[isim] ==>** Branch ismini değiştirir.

**git branch -d[isim] ==>** Branche’i siler.

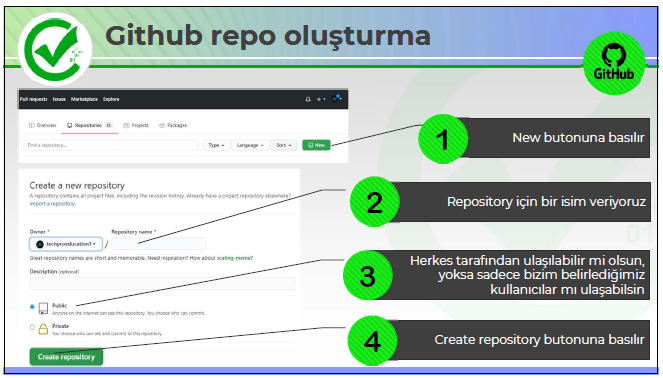
****

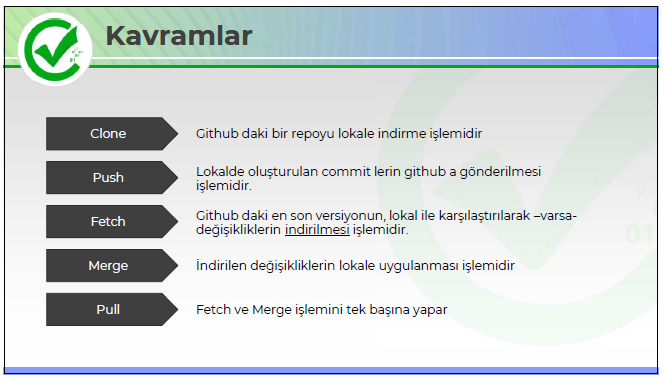
**git stash ==>** Working space ve staging area daki değişiklikleri geçici olarak hafızaya alır ve bu bölgeleri

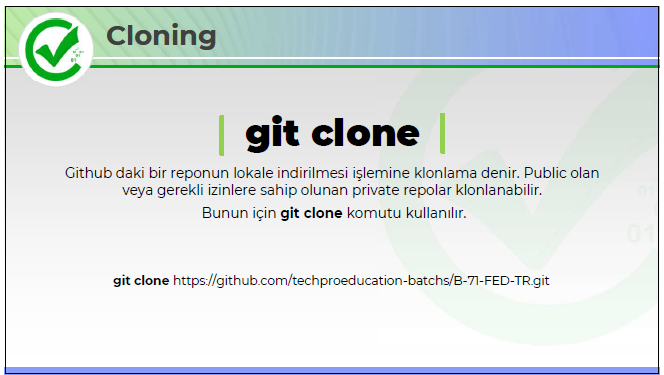
temizler.

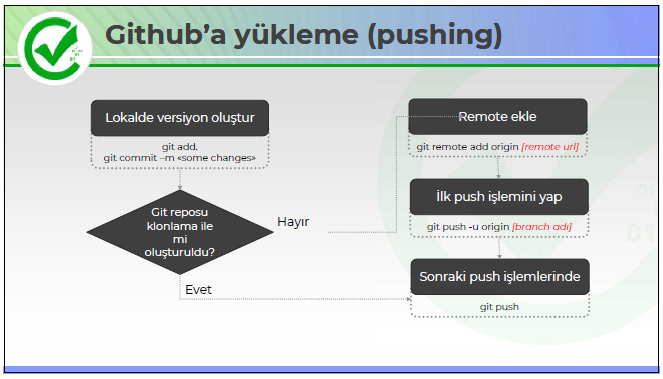
**git stash list ==>** Hafızaya alınan değişiklikleri görmek için kullanılır.

**git stash pop ==>** Hafızaya alınan değişiklikleri geri uygulamak için kullanılır.

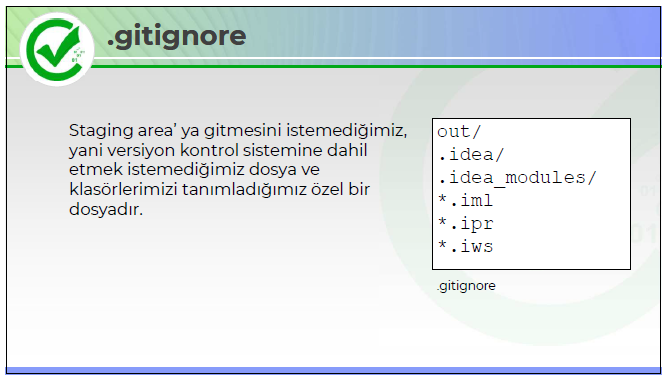












### Proje Alma ve Oluşturma

| **Komut** | **Açıklama** |
| --- | --- |
| git init | Yerel bir Git reposunu başlat |
| git clone ssh://git@github.com/[username]/[repository-name].git | Uzak bir reponun yerel bir kopyasını oluştur |

### Temel Anlık Görüntü

| **Komut** | **Açıklama** |
| --- | --- |
| git status | Durumu kontrol et |
| git add [file-name.txt] | Hazırlama alanına bir dosya ekle |
| git add -A | Tüm yeni ve değiştirilen dosyaları hazırlama alanına ekleyin |
| git commit -m "[commit message]" | Değişiklikleri commit et (açıkla) |
| git rm -r [file-name.txt] | Dosyayı (ya da dizini) sil |

### Dallanma ve Birleşme

| **Komut** | **Açıklama** |
| --- | --- |
| git branch | Bracnh'leri listeleyin (yıldız işareti mevcut branch'i gösterir) |
| git branch -a | Tüm brach'leri listele (yerel ve uzak) |
| git branch [branch name] | Yeni bir branch oluştur |
| git branch -d [branch name] | Branch'i sil |
| git push origin --delete [branch name] | Uzak branch'i sil |
| git checkout -b [branch name] | Yeni bir branch oluştur ve ona geçiş yap |
| git checkout -b [branch name] origin/[branch name] | Uzak branch'i klonla ve ona geçiş yap |
| git branch -m [old branch name] [new branch name] | Yerel branch'i yeniden adlandır |
| git checkout [branch name] | Belirtilen branch'e geçiş yap |
| git checkout - | Son kontrol edilen branch'e geç |
| git checkout -- [file-name.txt] | Bir dosyadaki değişiklikleri sil |
| git merge [branch name] | Bir branch'i aktif branch ile birleştir |
| git merge [source branch] [target branch] | Branch'i hedef branch ile birleştir |
| git stash | Yarım kalan bir çalışma dizinindeki değişiklikleri saklayın |
| git stash clear | Saklanan tüm girişleri kaldır |

### Projeleri Paylaşma ve Güncelleme

| **Komut** | **Açıklama** |
| --- | --- |
| git push origin [branch name] | Uzak repoya bir branch gönder |
| git push -u origin [branch name] | Değişiklikleri uzak repoya aktarın (ve branch'i hatırla) |
| git push | Değişiklikleri uzak repoya aktarın (ve branch'i hatırla) |
| git push origin --delete [branch name] | Uzak branch'i sil |
| git pull | Yerel repoyu en yeni commit'e güncelleyin |
| git pull origin [branch name] | Değişiklikleri uzak repodan çek |
| git remote add origin ssh://git@github.com/[username]/[repository-name].git | Uzak repo ekle |
| git remote set-url origin ssh://git@github.com/[username]/[repository-name].git | Bir reponun başlangıç branch'ini SSH olarak ayarla |

### İnceleme ve Karşılaştırma

| **Komut** | **Açıklama** |
| --- | --- |
| git log | Değişiklikleri görüntüle |
| git log --summary | Değişiklikleri görüntüle (detaylı) |
| git log --oneline | Değişiklikleri görüntüle (kısaca) |
| git diff [source branch] [target branch] | Birleştirmeden önce değişiklikleri önizle |

İlk defa PUSH yapacaksanız bir projeyi ;

1 - Önce GitHub a gidip New Repository i oluşturun

2- Oluşturduğunuz repository de size verilen <https://github.com/.............git> adresini kopyalayın

Şimdi IntelliJ ye gelip TERMINAL ı açın;

3- git init yazın

4- git add . yazın

5- git commit -m"first commit" (first commit yerine ilke versiyon, ilk ders veya istediğiniz bir şeyi yazın)

6- git branch -M main

7- git remote add origin <https://github.com/.............git> (yukarıda2. satırda kopyalağınız adresi buraya yapıştırın)

8- git push -u origin main

SONRASINDA Aynı PROJE için ;

her yeni eklediğiniz şeyden sonra GitHub'a yüklemek için

9- git add .

10 - git commit -m"YENI NUMARA VEYA YAZI YAZ"

11 - git push

\* IntelliJ'de proje ismi üzerine sağ tıklayıp ==> open in ==> explorer tuşlayınca projenin bulunduğu dosyaya gidip .git klasörünü görebiliriz.

**Genel Kavramlar**

Working Space : .git klasörünün bulunduğu çalışma alanıdır. Klasörler ve dosyalar üzerinden değişiklik burada yapılır.

Staging Area : Versiyon oluşturulacak olan dosya veya klasörlerin geçici olarak toplandığı yerdir. Versiyon (commit) oluşturulduktan sonra otomatik olarak staging area boşaltılır.

Commit Store : Git her bir commit i ayrı bir versiyon olarak tutar. Böylece yapılan çeşitli değişikliklerden sonra projede sorunlar ortaya çıkarsa bir önceki commit e geri dönülebilir.

**GithUb da target cakismamasi icin:**

**1- Git ignore eklemek**

**2- Target dosyasini silmek**

**3- Terminal'de mvn clean**