**GIT ve GITHUB**

**Git Bash** : Komut satırı üzerinden işlemleri yaptığımız sürümdür.

**Git GUI** : Grafik arayüzü olan sürümüdür.

**Git Bash**

Git ile bağlantı kurmak için Username ve Email bilgisi girilir.

**git config --global user.name** brkyyrr

**git config --global user.email** “berkayyurur@gmail.com”

Girişlerin doğru olduğunu kontrol etmek için sadece kod kısmını tekrar yazmamız yeterli olacaktır.

**Git Bash Üzerinden Proje Oluşturmak**

Oluşturduğumuz çalışma klasörüne **cd proje\_dizini** komutu ile geliyoruz. Dizin içerisinde **git init** yazarak .git adında gizli bir klasör oluşturuyoruz. **ls -a** gizli klasörleri görmemizi sağlar.

**git init**

**ls -a**

Projeyi tamamlayıp, dizin içerisine kayıt ediyoruz. **git add .**  komutu, dosyaları geçiş bölümüne ekler. Kalıcı olarak yüklemek içinde **git commit -m “**Açıklama Yazmak Zorundasın”komutu yazılarak işlem tamamlanır.

Sonraki süreçlerde **git log** komutuda projenin sürüm ve açıklamalarını listeler.

**git add .**

**git commit -m “**İlk Projem”

**git log**

Proje içerisinde değişikliğin yapıldığı dosyayı görmek için **git status** kullanılır. Sadece değişikliğin olduğu dosyayı kalıcı olarak eklemek için **git add** dosya\_adı ve git commit -m “Açıklama”

**git status**

**git add** dosya.py

**git commit -m** “Yeni Dosya Yüklendi”

Hem değişiklik yapılan dosyayı hem de içinde yapılan değişikliği görmek için **git diff** kullanılır.

**git diff**

Kod satırından dosya silmek için **git rm** dosya.py kullanılır. Ardından **commit** ile işlem tamamlanır. Klasör için de aynı işlem yapılır.

**git rm** dosya.py

**git rm -r** klasor\_adı

**git commit -m “**Dosya Silindi”

Dosyalara elle müdahalede bulunabiliriz. Sistem bunları yeni bir dosya olarak görecektir. Kod üzerinden isim değişikliği yapmak istersek **git mv** dosya\_adı yeni\_dosya\_adı . Dosyayı taşımak istersek **git mv** dosya\_adı yeni\_konumkomutu kullanılır.

**git mv** dosya\_adi yeni\_dosya\_adi

**git mv** dosya\_adi yeni\_klasor\_uzantı

Yaptığımız işlemi **commit** etmeden geri almak istiyorsak **git checkout --** dosya\_adi komutu kullanılır.

**git checkout --** dosya.py

Fakat **git add** dosya\_adi işlemini yaptıysak, geri almak için **git reset HEADB** dosya\_adi , dosyayı görmek için de **git checkout --** dosya\_adi

**git add** dosya.py

**git reset HEAD** dosya.py

**git checkout --** dosya.py

Önceye ait versiyonları **git log** ile görebiliyorduk. Listelenen **commit** kodlarına dönmek istersek **git checkout** commit\_code **-- .** yazarız. Commit\_code, **commit** kısmında bulunan değerlerin ilk 7 hanesi, nokta işareti ise projedeki tüm dosyalar demektir.

**git log**

**git checkout** 8f890db **--.**

**git commit -m** “Versiyon Dönüşü”

**GITHUB DOSYA OLUŞTURMAK ve YÜKLEMEK**

Github üzerinden **new repository** ile proje oluşturulur. Her projenin **.git** uzantılı bir adresi olur. Bilgisayarda bulunan projeyi buraya yüklemek için önce **git remote add** bağlantı\_adı uzantı\_adı yazılır. Bağlantının çalıştığını **git remote** ile kontrol ederiz.

**git remote add** baglanti\_adi <https://...../egitim.git>

**git remote**

Bağlantıyı kurduktan sonra **git push -u** bağlantı\_adı **master** yazarız. Sistem kullanıcı adı ve şifre isteyecektir. Şifre oluşturacağımız **Access Tokens** olacaktır. Oluşturmak için site üzerinden **Settings / Developer Settings / Personal Access Tokens** konumuna gidilir.

**git push -u** baglanti\_adi **master**

Projeye ait Github üzerinden oluşturduğumuz dosyaları yerele çekmek için **git pull** komutunu kullanırızç

**git pull**

**.gitignore**

Projelerimizde Git tarafından kontrol edilmesini istemediğimiz bir dosya vb. bilgiler varsa bu klasör altında barındırırız. Oluşturmak için **cat >> .gitignore**, enter yapıp, gizlenecek\_dosya\_adi yazılır. Gizlenen dosyayı görmek için de **cat.gitignore** yazarız.

Dikkat edilmesi gereken husus **add**  işlemi yapmadan önce dosyayı oluşturmaktır. Dosyayı oluşturduktan sonra **add - commit - push** işlemleri yapılır.

**cat >> .gitignore**

**cat .gitignore**

**BRANCHES (DALLAR)**

Branch projenin kopyasıdır. Ekip çalışmalarında kullanılır ve en son master üzerinde birleştirilir. Hem terminal hem de site üzerinden oluşturabiliriz.

Localde bulunan dalları görmek için **git branch** , uzak sistemdeki dalları listelemek için de **git branch --all** komutunu kullanırız.

**git branch**

**git branch --all**

Branch oluşturmak için **git branch** branch\_isim, oluşturduğumuz branch’a **master** üzerinden geçiş yapmak için de **git checkout** branch\_isim komutunu yazarız.

**git branch** branch\_isim

**git checkout** branch\_isim

Bulunduğumuz dalı görmek için **git branch** komutu kullanılır.

**git branch**

Dallar üzerindeki işlemlerimiz bittikten sonra, **git add .** - **git commit** işlemleri tamamlanır. Ardından dalları **master**  üzerinde birleştirmemiz gerekir.

**git checkout master** ile master a geçiş yapıp,  **git diff master** branch\_isim ile aralarındaki farkı gözlemleriz. Birleştirme işlemi için **git merge** branch\_isim , sonrasında da Github sayfasına iletmek için **git push -u** baglanti\_adi **master** komutunu yazarız.

**git checkout** branch\_isim

**git add .**

**git commit -m “**Branch oluştu ve yüklendi”

**git checkout master**

**git diff master** branch\_isim

**git merge** branch\_isim

**git push -u** baglanti\_adi **master**

**Site üzerinden** birleştirme yapmak için **Branches / All branches / New puss request**’ e tıklanır. Karşılaştırma sonunda sorun yoksa, **Create pull request**  denir. **Merge pull** ve **confirm merge**  ile işlem tamamlanır.

**Değişiklikleri Geçici Olarak Kayıt Etmek**

**git stash** ile üzerinde çalıştığınız ancak henüz commit etmediğiniz değişikliklerin geçici olarak Git tarafından kayıt altına alınmasını ve aktif branch'inizin herhangi bir değişikliğin olmadığı temiz bir duruma getirilmesini sağlarsınız.

**git stash list** komutunu kullanarak aktif branch'inizde geçici olarak kayıt altına aldığınız değişikliklerin listelenmesini sağlayabilirsiniz.