Makale içeriği,

- Git bash ve kullanımı.
- Github repository oluşturma, dosya yükleme, branch oluşturma ve genel bilgiler.

1- Git Nedir? Neden Kullanılır?

- Git genel tanımıyla versiyon kontrol sistemidir. Oluşturduğumuz projeleri internet ortamında(uzak bilgisayar) tutmamızı sağlayan bir sistemdir. Tanımdan anlaşıldığı üzere **git ve github aynı şey değildir.**

2- Git İş Akışı :

- Working Directory : Kendimize ait olan çalışma dizinidir (Dosya dizinimiz).
- Staging Area(Index Area) : Geçiş bölgesi, çalışma dizinimizden repositoriye aktaracağımız dosyalar için bir ara kontrol katmanıdır. Bu katman üzerinden belirli kontroller sağlanarak dosya bütünlüğü ve doğruluğu sağlanır.
- Git Repository : Git üzerindeki depolama alanıdır.

3- Git Bash Yararlı Komutlar :

- **pwd** : Mevcut dizini gösterir.
- **ls** : Mevcut dizindeki dosyaları listeler
- **ls -a** : Mevcut dizindeki gizli dosyaları listeler.
- **cd** : Dizin değiştirmeyi sağlar.
- **cd ..** : Bir önceki dizine dönmeyi sağlar.
- **clear** : Bash terminalini temizlemeyi sağlar.

4- Git Rehberi :

4.1 - Git'e Kullanıcı Bilgileri Girmek :

Bir proje üzerinde alınan sürümlerin kim tarafından alındığını bilmek için gerekli önemli bir adımdır.

git'e isim girmek :

Git'e isim girmek için aşağıdaki komut kullanılır.

```
git config --global user.name "Mahmut Özcan"
```

git'e mail girmek :

Git'e mail girmek için aşağıdaki komut kullanılır.

```
git config --global user.email "deneme@deneme.com"
```

Komutları sayesinde git üzerinde kullanıcı bilgilerimizi tanımlayabiliriz. Daha sonrasında bilgilerimizi görmek ve kontrol etmek için :

```
git config --global user.name
git config --global user.email
```

komutlarını kullanabiliriz.

4.2 - Git Projesi Oluşturma :

cd komutu ile proje dosyamızın konumuna giderek, projenin bir git projesi olmasını belirtmemiz için :

```
git init
```

komutunu yazıyoruz. Bu sayede proje dosyamızın içerisine git dosyaları yüklendi. Dosya içeriğini **ls** komutu ile kontrol ettiğinizde boş olduğunu göreceksiniz. Fakat bu dosyanın boş olduğu anlamına gelmez. Git dosyaları klasör içerisinde gizli olarak tutulur. Gizli dosyaları görmek için **ls -a** komutunu kullanabilirsiniz. Bu komutu girdikten sonra klasör içerisindeki .git dosyalarını görebilirsiniz. ".git" klasörü içerisinde git ile alakalı dosyalar bulunur.

4.3 - Projeyi Git Deposuna Ekleme :

Working directory deki dosyalarımızı verilen git iş şeması akışında git deposuna eklemek için öncellikle :

```
git add .
```

komutunu kullanıyoruz. "." şuanki dizin içerisindeki tüm dosyaları belirtir. add komutu sonrasında artık dosyalarımız "Staging Area" dediğimiz geçiş bölgesine geçer. Dosyalarımızı git deposuna eklemek için commit etmemiz gerekir :

```
git commit -m "eyyo my first commit"
```

"-m" mesajı belirtir. Girilen mesaj versiyonun niteliğini ve bilgisini içerecek şekilde olmalıdır. Şuan dosyalarımızın git tarafından bir versiyon kopyası oluşturuldu.

Alınan versiyonları listelemek için :

```
git log
```

komutunu kullanırız. Komut çıktısında ekleme tarihi, mesaj, kim tarafından eklendiği ve hash gibi bilgiler mevcuttur.

4.4 - Proje Üzerindeki Değişiklikleri Gösterme :

```
git status
```

komutu ile projede değişiklik olup olmadığı kontrolü yapılır. **"nothing to commit,working tree clean"** çıktısı commit edilecek dosyamızın olmadığını bize belirtir.

Proje dosyamıza yeni dosyalar eklendiğinde **git status** komutu çıktı olarak commit edilmemiş dosyaları gösterir. Projeye eklediğimiz yeni dosyaları **add** komutu ile ilk önce geçiş bölgesine daha sonra **commit** komutu ile git deposuna aktarıyoruz. Tekrar **git status** komutunu çalıştırınca commit edilecek dosya olmadığını görüyoruz. **add** komutu "." ile kullanıldığında proje içindeki tüm dosyaları, spesifik olarak dosya adı verilince sadece verilen dosyayı geçiş bölgesine geçirir.

4.5 - Proje Üzerinde Değişiklik Yapma :

Proje geliştirme sürecinde dosyalarımız üzerinde değişiklik yapmamız gerekebilir. Örneğin projeye eklediğimiz hello.py dosyası içerisinde bir kaç değişiklik yapalım ve git status ile tekrar kontrol edelim.

git status komutunu çalıştırdıktan sonra hello.py nin kırmızı bir şekilde modified olarak işaretlendiğini görüyoruz. Bu bize hello.py nin en son commit edildikten sonra içeriğinin değiştirildiği anlamına gelmektedir.

4.6 - Proje Üzerinde Değişiklikleri Görme (Working Directory) :

Proje dosyalarımızdaki değişiklikleri satır satır listelemek için :

```
git diff
```

komutunu kullanırız. Dosya üzerinde ekleme yapılan kısımlar yeşil renkli ve + işareti ile satır satır gösterilir. Komut çıktısında da görüldüğü gibi sum_method() adlı yeni bir fonksiyon hello.py içerisine eklenmiş. Değişiklerin tekrar git deposuna göndermek için :

```
git add .
git commit -m "sum_method() added to hello.py"
```

komutlarını giriyoruz. Daha sonra **git status** ve **git log** komutlarını çalıştırarak her şeyin doğru olduğuna emin oluyoruz.

> Not: Dosya içerisinde silme işlemi yapıldığında terminalde çıktı olarak silinen alanlar **kırmızı** ve **"-"** işareti ile gösterilecektir.

4.7 - Proje Üzerinde Değişiklikleri Görme (Staging Area) :

git diff komutu parametresiz olarak çalıştırıldığında "Working Directory" ve "Staging Area" arasındaki değişiklikleri gösterir. Commit işlemi sonrasındaki değişiklikleri (Staging Area ve Git Repo arasındaki) görmek

için komutumuza yeni bir parametre daha ekliyoruz.

```
git diff --staged
```

4.8 - Git Dosya Silme İşlemi :

git üzerinde dosyaları silmek için 2 method bulunur.

4.8.1 - Manuel Silme :

Dosyalarınızı manuel olarak "Working Directory" üzerinde elle silebilirsiniz. Dosya silme işleminden sonra **git status** komutunu çalıştıralım. Silinen dosya kırmızı renkli ve deleted tagi ile işaretlenmiş bir şekilde gözüküyor. Sildiğimiz dosyayı geçiş bölgesine göndermek için :

```
git rm hello.py
```

komutunu kullanıyoruz. Daha sonra git deposuna silme işlemini bildirmek için tekrar **commit** işlemi uyguluyoruz. İşlemleri gerçekleştirdikten sonra **git status** komutunu yazarak her şeyin yolunda olduğunu

4.8.2 - Komut İle Silme (Önerilen Yöntem) :

Dosyalarımızı komut ile silmek daha basit ve önerilen yoldur. Komut satırına silmek istediğimiz dosyayı :

```
git rm hello.py
```

rm komutu ile giriyoruz. Ardından commit ederek işlemimizi tamamlıyoruz.

4.8.3 - BONUS - Klasör Silme :

Proje üzerinde klasör silmek için **rm** komutuna ekstra olarak yeni bir parametre ekliyoruz ve klasör adını giriyoruz :

```
git rm -r will_delete/
```

> Not: **-r** : recursive yani özyinelemeli anlamına gelir. Klasör içindeki dosyaları ve klasörü siler.

4.9 - Dosya İsimlendirme ve Taşıma İşlemleri :

Proje içerisinde dosyalarımızı isimlendirmek ve taşımak için **mv** komutunu kullanıyoruz.

4.9.1 - Dosya İsimlendirme :

Dosyanın adını değiştirmek için :

```
git mv hello.py new_name.py
```

komutunu kullanıyoruz. **mv** den sonra ismi değiştirilecek dosya ardından dosyanın yeni adı olacak şekilde komutumuzu yazıyoruz. Daha sonra commit işlemimizi yaparak git deposuna değişikliliği bildiriyoruz.

4.9.2 - Dosya Taşıma :

Proje dosyalarını dizin içerisindeki başka bir klasöre taşımak için :

```
git mv hello.py functions/
```

komutunu kullanıyoruz. Taşınacak dosya --> Taşınacak Klasör

4.10 - Dosyaları Geri Alma İşlemi (Working Directory) :

Proje geliştirme aşamasında hata yapma olasılığı yüksektir. Kazara yapılan hata sonucu dosyaları geri almak, kurtarmak mümkündür. Öncelikle hello.py içerisindeki sum_method() fonksiyonunu silelim (Kaza senaryomuz). Daha sonra **git status** komutunu kullanarak değişiklikleri görelim. hello.py modified tag i ile kırmızı bir şekilde gösteriliyor. **git status** çıktısında yer alan yardımcı komutlardan:

```
git restore hello.py
```

komutunu kullanarak dosyayı geri alalım.

Dosyayı geri aldık ve silinen sum_method() fonksiyonumuz da geri geldi. Böylelikle "Working Directory" üzerinde dosya kurtarma işlemini gerçekleştirdik.

4.10.1 - Dosyaları Geri Alma İşlemi (Staging Area) :

Bir önceki aşamada "Working Directory" üzerinden dosya kurtarma işlemini gerçekleştirmiştik. Şimdi geçiş bölgesinde olan dosyalarımız üzerinde dosya kurtarma işlemini görelim (**add** komutu ile geçiş bölgesine eklenmiş henüz commit edilmemiş dosyalar). Bu işlemde komutumuza aşina olduğumuz bir parametre ekliyoruz:

```
git restore --staged hello.py
```

--staged ile "Staging Area" dan kurtardığımız dosyayı daha sonrasında "Working Directory" e bildirmek için bir önceki adımda ki **restore** işlemimizi tekrar uyguluyoruz.

```
git restore hello.py
```

Bu şekilde "Staging Area" ya gönderilmiş dosyamızı "Working Directory" e bildirmiş olup, dosyamızı kurtarmış oluyoruz.

4.11 - Versiyon Değiştirme :

Projemiz üzerinde birçok versiyon bulunabilir ve ihtiyaca göre versiyonlar arasında geçiş yapmamız gerekebilir. Örneğin versiyon yapımız şu şekilde olsun :

version 1.0 --> version 2.0 --> version 3.0

version 3.0 dan version 2.0 a geçerken version 2.0 ın kopyasını oluşturmuş olup projemizi o kopya üzerinden geliştirmeye devam ediyoruz. Böylelikle yeni versiyon yapımız şu şekilde olmuş oluyor :

version 1.0 --> version 2.0 --> version 3.0 --> version 2.0-copy

Öncelikle yeni bir git projesi oluşturalım ve içerisine 3 tane dosya açalım. İlk versiyonumuza ait dosya içerisine bilgiler girelim.

Daha sonra değişikleri commit edelim ve **git log** ile versiyonlarımızı kontrol edelim.

git log komutu ile version 1.0 ı görmüş olduk. Şimdi 2 version daha alalım ve senaryomuzu gerçekleştirelim.

git log komutunu kullanarak versiyonlarımızı kontrol edelim.

Görüldüğü gibi artık 3 adet versiyonumuz var. Güncel versiyon en üstte ilk versiyon ise en altta yazmakta(yukarıda belirtilen versiyon hiyerarşisindeki gibi).

Şimdi version 3.0 dan version 2.0 a geçelim.

Versiyonlar arasında geçiş yapmak için **hash** bilgisini kullanıyoruz. **Sarı renkle yazılan commit yazısının yanındaki hash in ilk 7 hanesini** veya **hash in tamamını kullanarak** versiyonlar arası geçiş yapabiliriz :

```
git checkout 02c603d7938e4256692f8f7b480e839dce1e31ea -- .
```

komutu ile version 2.0 a dönüyoruz ("." işareti versiyondaki tüm dosyaları getirmek için kullanılır).

Şimdi dosya içeriklerimizi tekrar kontrol edelim :

Version 2.0 a geçiş yaptık :

Görüldüğü gibi dosya içeriğimiz artık version 2.0 a dönmüş oldu. Bu şekilde projedeki versiyonlar arasında geçiş yapabiliriz.

4.12 - Github'a Dosya Gönderme :

Sıra oluşturduğumuz projeleri github repositorymize göndermede. İlk önce github üzerinden yeni bir repository oluşturuyoruz.

Daha sonra repository bağlantı linkimizi alıyoruz :

Şimdi git ile github repository arasındaki bağlantıyı oluşturalım :

```
git remote add "gitlessRepo" https://github.com/p0zn/git-guide-repository.git
```

komutu ile bağlantıyı oluşturuyoruz. **remote add** dan sonra bağlantımızı isimlendiriyoruz. Ben burada "gitlessRepo" şeklinde isimlendirdim. Siz nasıl isterseniz o şekilde isimlendirebilirsiniz.

> Not: Bu aşamada sizden github kullanıcı adı ve şifrenizi isteyebilir(daha önce yetkilendirme yapmadıysanız).

Bağlantının doğru kurulduğunu kontrol etmek için :

```
git remote
```

komutunu yazıyoruz. Eğer bağlantıya verdiğimiz isim karşımıza çıkıyorsa bağlantı doğru kurulmuş demektir 😃

Şimdi sıra dosyalarımızı github repository e yüklemede :

```
git push -u gitlessRepo master
```

komutunu kullanarak dosyalarımızı github repository imize yüklüyoruz. "-u" parametresi tüm dosyalar anlamına gelmektedir. "-u" parametresinden sonra bağlantı ismimizi ve proje dalımızı yazıyoruz(**master anadaldır.** Dallara ileride değineceğiz).

Bu şekilde dosyalarımızı github repository imize yüklemiş oluyoruz. Dosya üzerinde herhangi bir değişiklik yapınca yeniden push etmeyi unutmuyoruz.

4.13 - .gitignore nedir? nasıl oluşturulur? :

4.13.1 - .gitignore nedir? :

.gitignore git tarafından göz ardı edilmesi istenilen dosyalar için kullanılır. Örneğin projenize ait API Token bilgisi bulunan bir dosyayı herkese açık bir şekilde commit etmemek için .gitignore kullanılır.

4.13.2 - .gitignore nasıl oluşturulur? :

.gitignore dosyaları gizli dosyadır. .gitignore dosyası oluşturulurken başına "." koyulur. Nokta(.) koyulması sayesinde gizli dosya olur.

bash terminal ile .gitignore dosyası oluşturalım :

ls ile dosyalarımızı listelediğimiz zaman token.txt adlı bir dosyayı görüyoruz. Bu dosyanın git tarafından göz ardı edilmesini istiyorum. Öncelikle .gitignore dosyamızı oluşturuyoruz

```
cat >> .gitignore
```

dosya içerisine girince **git tarafından göz ardı edilmesi istenilen dosyanın adı yazılır.**

```
cat >> .gitignore
token.txt
```

CTRL + C tuşuna basılarak dosya içerisinden çıkılır.

daha sonra commit işlemleri gerçekleştirilir. Dosya üzerinde değişikilik yaptığımız için proje tekrar repository e push edilir.

Şimdi github repository imizde token.txt dosyası gözüküyor mu kontrol edelim :

Görüldüğü gibi diğer dosyalarımız gösterilirken token.txt dosyası gösterilmiyor.

4.13.3 - BONUS - .gitignore a nasıl klasör eklenir? :

Proje dosyaları içerisinde git tarafından göz ardı edilmesini istediğiniz dosyalarınızı bir klasörde topladınız. Ve bu klasörün gözükmesini istemiyorsunuz. .gitignore içerisine :

```
klasör_adı/*
```

şeklinde ekliyoruz. "*" parametresi bütün dosyalar anlamına gelmektedir. Böylelikle klasör içindeki tüm dosyalar git tarafından göz ardı edilecektir. Eğer klasör içerisindeki bir veya birkaç dosyayı istisna tutmak istiyorsanız :

```
!klasör_adı/dosya_adı
```

şeklinde belirterek dosya veya dosyalarınızı istisna edebilirsiniz.

4.14 - Branchler :

Bir projede birden çok geliştirici bulunuyorsa ve her birinin kendine ait görevleri varsa kişilere veya görevlere göre branchler oluşturulur.

Master anadaldır. Sonrasında alt dallar yapıya eklenir. Daha sonra alt dallar birleştirilerek proje son haliyle master da toplanır.

4.14.1 - Branch oluşturma :

Github repository veya bash shell üzerinden branch oluşturulur :

branch adı verilir ve "Create branch" butonuna basılarak branch oluşturulur.

"Branches" sekmesinden branchler arasında geçiş yapılabilir. Dikkat ederseniz branchlerin her birinde dosyalar aynı olarak gözükecektir.

Şimdi oluşturduğumuz subbranch in içerisine github üzerinden bir dosya ekleyelim ve commit edelim :

Gördüğünüz gibi oluşturduğumuz "file_4.txt" dosyası subbranch e eklenmiş. Peki bu dosya master dalında bulunuyor mu kontrol edelim :

Gördüğünüz gibi master dalında dosyamız **yok**. Çünkü biz dosyamızı subbranch üzerinde oluşturduk.

Şimdi branch leri birleştirelim :

github repository > branches > All branches yoluna gidelim.

subbranch imizin yanında bulunan "New pull request" butonuna tıklıyoruz.

Ardından açılan sayfada branchlerimizin birleştirmeye uygun olduğunu gösteren "Able to merge" ifadesi ile karşılaşıyoruz.

Yorum kısmına isterseniz açıklama yapabilirsiniz.

Sayfanın en altında branchler arasındaki dosya farklılıklarını gösteren kısım bulunuyor. Bu kısımdan branchlerin birbiri üzerindeki dosya farklılığını görebilirsiniz :

Daha sonra **"Create pull request"** butonuna tıklıyoruz.

Tartışma sayfasına yönlendiriliyoruz. Bu sayfa üzerinden kendi geliştirici ekibinizle veya diğer geliştiricler ile proje hakkında soru cevap , tartışma öneri vb. faaliyetleri gerçekleştirebilirsiniz.

Branchleri birleştirmek için artık son adıma geldik : **"Merge pull request"** butonuna tıklıyoruz :

Ve artık dallarımız master dalında birleştirilmiş oldu.

Şimdi "Working Directory" üzerindeki dosyalarımıza bakalım :

Subbranch üzerindeki dosyamız bu dizinde yok.

Yapmamız gereken tek şey :

```
git pull
```

komutunu yazarak github üzerindeki dosyaları çekmek. Şimdi tekrar kontrol edelim :

Dosyalarımızı çektik. Şimdi "Working Directory" i kontrol edelim :

Ve subbranch üzerindeki dosyamızda "Working Directory" üzerine eklendi. Böylelikle branchleri birleştirmeyi görmüş olduk.

4.14.2 - BONUS - Komut İle Branch Birleştirme :

Oluşturduğumuz branchleri komut ile birleştirmek için öncelikle :

```
git branch --all
```

komutu ile repository üzerindeki branchleri listeliyoruz.

Bu örnek için yeni bir subbranch oluşturuyorum :

```
git branch subbranch_2
```

tekrar branchleri listelemek için

```
git branch
```

komutunu yazıyoruz.

Şimdi master dalından oluşturduğumuz alt dala geçiş yapalım :

```
git checkout subbranch_2
```

komutu ile oluşturduğumuz subbranch e geçiş yapıyoruz.

Proje dizini içersinde yeni bir dosya oluşturuyoruz. Daha sonra değişiklikleri commit ediyoruz.

Branchleri birleştirmek için öncelike master dalına geçiş yapıyoruz :

```
git checkout master
```

Daha sonra branchleri birleştirmek için :

```
 git merge subbranch_2
```

komutunu yazıyoruz ve branchleri birleştiriyoruz.

Son adımda github a tekrar push ederek birleştirme işlemini tamamlıyoruz.

Hazırlayan : **Mahmut ÖZCAN**