**Git (Globel information tracker)**

**Version Control Systems**

Uzerinde calistigimiz projemizi **lokalda** olusturacagimiz **GIT Repository**  de Versiyon Degisikliklerini konrtkontrol etmek

**Adim Adim.**

Git Surum kontrol sistemini kullanabilmek icin (**git-scm.com/downloads**) windows veya mac isletim sistemine ait kurulum dosyalarini indirip klasik kurulum islemi yapilir. Linux ve turevleri icin kurulum terminal ile yapilabilir.

Git Bash te Git zaten kurulu halde iken, linux ve turevlerinde ornegin linux/ubuntuda asagidaki komut kullanilir

**sudo apt-get install git**

Kurulum tamamlandiktan sonra Windows, Linux ve turevleri icin terminal ekranimiza kurulumun basarili olup olmadigini ve surum bilgisi kontrolu yapilir. Zaten sistemimizde Git kurulu ise yine asagidaki **git – version** komut ile bunun kontrolunu yapabiliriz.

**git -- version**

Git kurulumu yapildiktan sonra **Git ayarlari** yapilir. Git ayarlari **config** komutu ile yapilir. Bunun icin **kullanici adimizi** ve **e-posta** adresimizi kayit etmemiz gerekecektir. Bu islem guncellemelerin kimin tarafindan yapildigi bilgisi icin gereklidir. Bu islem su sekilde yapilir.

**Kullanici adini degistirmek ic**in

**git config --global user.name** **"Ahmet KAYA"**

**Kullanici e posta adini degistirmek** icin, ise asagidaki komutu kullaniyoruz.

**git config --global user.email "ahmetkaya[@mail.com](mailto:yusufsezer@mail.com)"**

Simdi **git config –list** komutu ile kullanici bilgilerimizin kayit edilip edilmedigini kontrol edelim.

**git config –list**

Simdi Git te calismalarimizi yuruturken hangi editoru (**nano**, **vim**) kullanacagimizi kaydedelim bunun icin **git config --global core.editor "nano"** veya g**it config --global core.editor "vim"** komutunukullaniyoruz

**git config --global core.editor "nano"**

Ayarlarimizi tekrar kontrol etmek icin **git config –list** komutunu calistiralim.

**git config –list**

Ayarlarimizi yaptigimiza gore simdi bilgisayarimizda herhangi bir yerde Projelerimizi saklayacagimiz bir **klasor olusturalim** oncelikle hangi dizin altinda oldugumuzu gormek icin **pwd** komutu ile yerimizi kontrol edelim.

**pwd**

Ben calisma projelerimi **masa ustu** uzerinde olusturacagim “**GIT\_Calismalar**” adli klasorde saklamak istiyorum. Bunun icin **pwd** komutu ile konumumu kontrol ettikten sonra **masa ustu (Mustafa@DESKTOP-GQ7O2AR** MINGW64 ~$**)** konumuna gelebilmek icin **cd** ve **cd..** komutlarini kullaniyorum.

**cd**

**cd ..**

**Mustafa@DESKTOP-GQ7O2AR** MINGW64 ~$ masa ustu konumunda iken “**GIT\_Calismalar**” adli klasorumu asagidaki komut ile olusturuyorum.

**Mkdir GIT\_Calismalar**

**GIT\_Calismalar** Klasorumun dizin seviyesine cikabilmek icin **cd GIT\_Calismalar**  komutunu kullaniyorum.

**cd GIT\_Calismalar**

[**mustafa@ubuntu**](mailto:mustafa@ubuntu)**:~/Desktop/GIT\_Calismalar$** dizin seviyesinde iken **GIT\_Calismalar** Klasorumun icerisine girip bakmak istersem bunun icin uc farkli komuttan birini kullanabilirim. Bunlar **ls, ls -a** ve ls **-al** . Simdi ls -al komutu ile klasor icerisindeki dosyalari detaylari ile gorelim.

**ls -al**

Bu komut calistirildiktan sonra asagidaki ciktiyi aliyorum.

**total 8**

**drwxrwxr-x 2 mustafa mustafa 4096 Jul 1 01:05 .**

**drwxr - xr-x 7 mustafa mustafa 4096 Jul 1 01:17 ..**

Buradaki ilk karakter ("d") şimdilik bir kenara bırakırsak, diğer karakterler 3'erli olarak (drwxr-xr-x ) sırasıyla dosya sahibinin, grubunun ve diğer kullanıcıların yetkilerini göstermektedir. Kısacası rwx yazan kısım mustafa kullanıcısının yetkilerini, r-x yazan kısım mustafa grubunun yetkilerini, r-x yazan kısım için sistemde geriye kalan bütün kullanıcıların yetkilerini göstermektedir. Bu yetkiler çeşitli anlamların kısaltmalarıdır. r harfi okuma yetkisi (read), w yazma yetkisi (write) ve x çalıştırma yetkisi (execute) anlamına gelmektedir.

**GIT\_Calismalar** klasoru icerisine 3 farkli klasor daha olusturmak istiyorum **Odev1, Odev2 ve Odev3** klasorleri

**Mkdir GIT\_Odev1**

**Mkdir GIT\_Odev2**

**Mkdir GIT\_Odev3**

Simdi calistigimiz lokal makinamizda Odev1 klasoru icerisinde repo (repository) olusturmak istiyorum bunun icin Odev1 konumunda iken **git init** komutunu kullaniyorum. Calistirdigimiz lokal makinamizda repo olusturuyoruz.

**git init**

Bu komuttan sonra **Odev1** klasorum icerisinde **.git** adinda gizli gorulmeyen repo dosyasi olusturmus oldum. Bu komutla birlikte yapacagimiz islemler master dali bransi uzerinde oldugu icin komut satirimda (master) ifadesinin yazdigini goruyorum. Durumumu kontrol etmek icin **git status** komutunu kullaniyorum. Burada yazan ‘**On branch master**’ yazisi master bransinda oldugumu ‘**No commits yet**’ yazisi ise henuz hic bir projeyi kommit etmedigimi gosteriyor.

**git status**

Odev1 klasorumun icerisine kommit etmek icin **dizinler.py** adinda bir paython dosyasi olusturmak istiyorum. Bunun icin **touch** komutunu kullaniyorum

**touch dizinler.py**

olusturdugum **dizinler.py** dosyasini ve durumumu kontrol etmek icin tekrar **git status** komutunu kullaniyorum. **git status -s** komutu ile de daha ozet bir bilgi almak mumkundur.

**git status**

Bu asamadan sonra **dizinler.py** dosyasinin kirmizi renkte yani komit edilmedigi uyarisini asagidaki ifade bicimi ile aliyorum. (use "git add <file>..." to include in what will be committed)

**dizinler.py**

Simdi bu **dizinler.py** dosyamizi **git add** komutu ile .git dosyamiza ekleyelim.

**Git add dizinler.py**

dosyamizi ekledikten sonra durumunu **git status** komutu ile kontrol ettigimizde **dizinler.py** dosyamizin yeni halinin asagidaki gibi **dizinler.py** seklinde yesil renkli sekilde ifade edildigini goruyoruz.

(use "git rm --cached <file>..." to unstage)

**new file: dizinler.py**

dosyamizda yaptigimiz degisiklikler hakkinda not dusebilmek icin **git commit** komutunu kullanabiliriz bu komut bizi daha onceden ayarlamis oldugumuz **nano** editor ekranina tasiyacaktir. Buraya “bu kommit benim bu proje hakkindaki ilk kommitimdir” seklinde hatirlatici bir not yazdiktan sonra **nano** yekranindaki calismalarimi kaydetmek icin **ctrl+o** kombinasyonunu kullanip enter yapiyorum. Bu **nano** ekranindan cikmak ve **bash ekranima** donmek icinde **ctrl+x** ve enter yapiyorum.

**git commit**

Tekrar bash ekranina geldigimde ‘bu kommit benim bu proje hakkindaki ilk kommitimdir” seklindeki yazinin yazilmis oldugu goruluyor.

Tekrar **touch** komutu ile **about.txt**  ve **index.html** adlarinda iki dosya daha olusturmak istiyorum. Istersek yine **git status** komutu ile bu dosyalarin durumunu kontrol edebiliriz. Her iki **about.txt**  ve **index.html** dosyamizinda kirmizi konumda oldugu gorulecektir. Her iki dosyamizi da git e eklemek icin **git add .** Komutunu kullaniyoruz.

**git add .**

Bu komuttan sonra Her iki **about.txt** ve **index.html** dosyamizinda yesile donustugunu goruyoruz.

**git add .**  komutu g**it add about.txt** komutundan farkli olarak tekbir dosyayi degil tum dosyalari **git** e ekler.

Simdi ekledigimiz dosyalara not dusmek icin **git commit -m “”** komutunu kullaniyoruz. Titnak ici “” ifadelere notumu “*bu benim ikinci kommitimdir hayirli ugurlu olsun*” seklinde yazmak istiyorum

**git commit -m “”**

Bu git **commit -m “….”** komutunun **git commit** komutundan farki **nano** ekranina gitmemize gerek birakmadan komit yazabiliyor olmamizdir. Butun kommitlerimizi gormek icin **git log** komutunu kullanabiliriz.

**git log**

Bu komutu calistirdiktan sonra asagidaki sekilde bir cikti aliriz. Degisiklikleri , degisiklikleri yapan kullanicinin adini ve email adresini , degisiklikleri ne zaman yaptigini ve hangi notu yazdigini gorebiliriz.

commit e6f39da8a9a4b30b22e7468283ea5991b318dcc2

Author: mustafa <ekicimustafa113@gmail.com>

Date: Wed Jul 1 03:24:08 2020 -0700

bu kommit benim bu proje hakkindaki ilk kommitimdir

Buraya kadar olusturdugumuz bir projemizi Lokalde **GIT** e **Repo** icerisine atmayi adim adim gormus olduk.

Bunun disinda kullanilan komutlardan bahsetmek gerekirse **git config** komutu ile hangi komutlari kullanabilecegimiz hakkinda bir dizi komut dokumu almis oluruz.

**git config**

yine komutlar hakkinda bilgi alabilmek icin **git status -help** komutunu kullanabiliriz.

**Working Directory - Staging Area** - **Repository**

**git commit -am** “…” komutu ile -**Staging Area**- (**Working Directory**, **Repository**) alanimizi kullanmadan Repository alanimiza atma yontemi vardi.

Touch komutu ile **question.html** adli bir dosya olusturup ekleyelim sonra bu dosyamizi **git add .** komutu ile **stage** edelim. **git status** ile baktigimizda

on branch master

Changes to be committed:

(use “git restore –staged <file>…” to unstage)

new file: questions.html

seklinde bir gorunum elde ederiz. Bu gorunumden dosyamizin stage edildigini goruyoruz. **git commit -am** Komutumuz ile kisa yoldan **Staging Area** alanimizi kullanmadan **Repository** alanimiza alabilmek icin ilk basta tracklamamiz commit etmemiz lazim, tracklamadan sifirdan yaptigimizda hata aliriz

simdi bu dosyamizi **git commit -m “….”** komutu ile mesaji bu ekranda “bu bizim ucuncu komitimiz……..” seklinde yazmak sureti ile notumuzu dustuk entere bastik kommitimizi yaptik. **git status** ile kontrol ettigimizde

on branch master

nothing to commit, working tree clean

seklinde gorunum elde ederiz. Buradan da goruldugu uzere working tree alanimizin temizlendigi anlasiliyor.

### Simdi tek seferde 5 yeni dosya ekleyelim bunun icin touch file {1..5} .txt komutunu kullaniyoruz. ls komutu verdigimizde file1.txt file2.txt file3.txt file4.txt file5.txt seklinde dosyalarimizin olustugunu goruruz.

### git add file1.txt

Komutunu verdiğimizde dosyalardan bir tanesinin eklendiğini eklenen dosya **file1.txt** yeşil renkte ve **staged** ifadesi ile gösterilir. Diğerlerinin eklenmediğini yani track edilmediğini goruyoruz. Bu durumda dosyalar kirmizi renkte görünürler. **Untracked** **file**: seklinde ifade edilirler

**Git add .**

Komutu ile butun dosyalari **trackla** demiş oluyoruz. Dosyalarin drumunu kontral etmek için **git status** komutunu verdiğimizde butun dosyalarin ortadaki sahada yani **Staging Area** da olduğunu goruyoruz

**git commit -m “….”**

Simdi Komutu ile komit edip mesajimizi yazıyoruz entera basıyoruz biz bunu yaptigimizda **Staging Area** alanindaki dosyalarimizin Repository alanimizda yeni bir **obje** oluşturduğunu goruyoruz. Bu yeni oluşan obje daha önceki commit ettiğimiz dosyalarin objeleri ile birlikte korunmaktadır. Bu objeler **.git** gizli klasörü içerisinde (.git gizli klasörü içerisinde hooks, info, logs, objects, refs klasörleri bulunmaktadır. ) objects klasörü içerisinde oluşmuştur. (object klasörü içerisinde her kommitten sonra 1b, 2c, 7a, …. Gibi klasörler oluşur klasör içlerinde ise 440ce7a07f5047d6972429d8bdf828c22cd173 seklinde objeler olusur)

**git status**

İle baktigimizda working tree alanimizin temizlendigini dosyalarimizin Repository alanina alindigini anlıyoruz.

Simdi yapilan tum kommitleri görebilmek için **git log** komutumuzu kullaniyoruz .

**git log**

bu komutu kullandiktan sonra yaptigimiz tum kommitleri en yukarida en son yaptigimiz kommit olmak uzere sirali sekilde tum kommitleri goruyoruz. Asagida tek bir komite ait gorunum görülmektedir.

$ git log

commit e01256be94e92a1176b97230e613453ff408a0dc (HEAD -> master)

Author: Mustafa <ekicimustafa113@gmail.com>

Date: Sat Jul 4 19:25:59 2020 +0200

bu bizim dorduncu komitimiz burada 5 yeni dosya olusturduk

burada commit karsisindaki **e0**1256be94e92a1176b97230e613453ff408a0dc rakam ve sayilardan oluşan dizinin .git gizli klasörü içerisinde oluşan obje klasörü içerisindeki obje olduğunu anlıyoruz. Bu karekterler dizisinin ilk iki karekteri obje klasörü içerisindeki klasör isimlerini oluşturuyor **e0,** 1b, 2c, 7a, …. gibi bu obje klasörleri içerisinde de objeler kommit ID (**e0**1256be94e92a1176b97230e613453ff408a0dc) bulunuyor.

Time trawel (zamanda yolculuk)

Simdi oluşturduğumuz 4 kommitten yani değişikliklerden ikinci komitimizin olduğu noktaya gitmek istiyorum yani son uc ve dorduncu değişiklikler yapılmadan önceki hallerine dönmek istiyorum bunun için git checkout komutumu kullaniyorum. **git checkout** yazdıktan sonra bu komutun onune gitmek istedigim 2.kommitin object numarasini kopyalayıp yapistirarak enter yapmam gerekiyor.

$ git checkout e01256be94e92a1176b97230e613453ff408a0dc

Komutunu verip enter yaptiktan sonra .git klasorumun bulundugu Repository alaninda 2.kommitimdeki durum ne ise o hale donmus oluyorum yani 2. Kommitten sonra oluşturmuş olduğum dosyalar bu ekranda gözükmüyor. Zaman olarak 2. Kommitin olstugu zamanda var olan dosyalarimi görebiliyorum. Burada 8 dosya vardi simdi sadece 2 dosya goruluyor cunku biz ikinci kommite geri donduk. Bu dizelere cha1 (sau diye okunuyor) kommit ID denilmektedir.

$ git checkout e01256be94e92a1176b97230e613453ff408a0dc

Tekrar **git checkout** komutu ile 4. Komitimizi verdiğimiz zamana donuyoruz. Burada leapyear.py disyamizi visual studio cod programinda acim leapyea ile ilgili assigntment yapip visual studio cod editorumuzde kayd ediyoruz. Daha sonra git status komutu ile kontrol ettiğimizde

**git status**

Bash ekranimizda **leapyear.py** dosyasinin degistigini ve leapyear.py dosya adinin kirmizi renkte olduğunu goruyoruz.

Bu dosyamizi tekrar **git add .** komutu ile stage alanimiza aliyoruz daha sonra git **commit -m “leapyear dosyasini değiştirdik method1 ekledik”** komutu ile yaptigimiz değişiklikleri mesaj yazarak kommit ediyoruz. Git status ile bakip working alanimizin temiz olduğunu gorduk. Tekrar **git log** ile baktigimizda 5. Kommitimizin yazili olduğunu gorduk.