-> ilk olarak git kurulmalıdır "git download"

-> Kurulum yapıldıktan sonra tüm işletim sistemleri için geçerli olan "**Git Bash**" terminali üzerinden bazı komut kullanım örnekleri

-**> Klasör veya Dosya işlemleri**

* "**ls**" komutu bulunduğun klasörün içindekileri gösterir
* "**pwd**" güncel olarak bukunduğun klasörü gösteririr
* "**cd Documents**" Documents klaösrü içine gir demektir (klasörün ilk bir kaç harfini yazıp sonrasında **TAB** tuşuna basılırsa otomatik tamamlar)
* "**cd ..**" bir önceki klasöre git demektir
* "**clear**" terminal ekranını temizler
* "**touch not.txt**" bulunuduğun klasör içinde text dosyası oluşturur
* "**rm touch.txt**" text dosyasını siler (rm sadece dosyaları siler)
* "**mkdir OrnekKlasor**" OrnekKlasor adlı klasör oluşturur
* "**rm -rf OrnekKlasor**" klasör silmek için bu komut
* **Git Commands**
* "**git**" git ile alakalı komutları gösteririr
* "**git version**" git versionu nu gösterir
* "**git status**" komutu ile o klasörün git ile bağlantısı olup olmadığı, bağlantı var ise herhangibir bekleyen işlem var mı tüm durumlar kontrol edilir
* "**git init**" ile bulunduğun klasör git'e bağlanmış olur. Bulunduğun klasörügit repository si haline getirlir
* "**git log**" atılan commtileri görmek için
* "**git add .**" yapılan değişikliği commit öncesi staged alanına taşır. Yani commite hazır hale getirir

**-> Kullanıcı Adı Soyadı ve Email bilgilerini git'e tanımlama git**

* **git config --global user.name "Emre Aktürk"** ile kulanıcı adı oluşturabilirsin
* **git config --global user.nam**e kullanıcı adını öğrenebilirsin
* **git config --global user.email "akturk\_emre@hotmail.com"**
* **git config --global user.email**

**-> Örnek Proje Açma**

* "**mkdir GitKursu**" her hangibir yerde ilk olarak klasör oluşturulur
* "**git status**" komutu ile o klasörün git ile bağlantısı olup olmadığı, bağlantı var ise herhangibir bekleyen işlem var mı tüm durumlar kontrol edilir
* "**git init**" ile bulunduğun klasör git'e bağlanmış olur. Bulunduğu klasörü git repository si oluşturur.
* "**touch test.txt**" ile text dosyası ekledik
* "**test.txt**" dosyasında herhangibr değşiklik yapıp kaydedildiğinde **unstaged** durumundadır.
* "**git add .**" yapılan değişikliği commit öncesi staged alanına taşır. Yani commite hazır hale getirir
* **Test.txt** dosyası eklenip commit ediltikten sonra yeni bir değişiklikte **Unstaged** halindeyken "**git restore test.txt**" komutunu kullanılırsa son değişiklik silinir

**Not** : Git'te "unstaged" terimi, yapılan değişikliklerin geçici alana (staging area) eklenmediği, yani commit'e dahil edilmek üzere henüz hazırlanmadığı anlamına gelir. Dosya üzerinde yapılan değişiklikler, öncelikle "working directory" (çalışma dizini) içinde bulunur ve bu değişiklikler "unstaged" olarak kabul edilir

**Working Directory (Çalışma Dizini):**

* Proje dosyalarınızın bulunduğu yerdir. Burada dosyalar üzerinde yaptığınız değişiklikler öncelikle "unstaged" olarak kabul edilir.

**Staging Area (Geçici Alan):**

* "Staged" değişikliklerin bulunduğu yerdir. Bu değişiklikler, bir sonraki commit'e dahil edilmek üzere hazırlanmıştır. git add komutuyla dosyaları geçici alana ekleyebilirsiniz.

**Repository (Depo):**

* Tüm commit'lerin bulunduğu yerdir. git commit komutuyla geçici alandaki değişiklikler depoya kaydedilir.

* **Unstaged** halindeyken "**git add test.txt**" ya da "**git add .**" komutları kullanılırsa, ilgili dosya ile alakalı tüm değişiklikleri staged(commit öncesi) alanına taşır
* **Staged** durumundayken "**git restore --staged giris.txt**" komutu çalıştırılırsa **unstaged** durumuna alınır
* **Staged** durumundayken **git commit -m "ilk commit yorum**" komutu ile commit işlemi yapılmış olur

-> **Reset** ve **Revert** ve **Checkout** ve **Switch**

* **git reset <commit hash>**  komutu ile önceki commitleri history den siler ama değişikler kalır
* **git reset --hard <commit hash>**  komutu history ile birlikte değişiklikler de silinir
* **git revert <commit hash>** komutu ile commit edilen bir değişiklik geri alınır, geri alınırken yeni commit daha oluşturulur
* **ÖNEMLİ** : en son commit den itibaren revert kullanılmalı aksi durumlarda conflict çıkabilir
* **ÖNEMLİ : Reset** : historyi siler, **Revert** : history den silmez

-> **Diff**

* **git diff <Commit Hash previous> <Commit Hash last >** komutu ile commit hash lere göre farklılıkları görebilirsin
* **git diff master pilot** komutu ile commit branch lere göre farklılıkları görebilirsin

-> **Branch işlemleri**

* **git checkout <branchname>** komutu ile branch değiştirilir. NOT :karışma ihtimaline karşı branch değişkliği için **git switch <branchname>** kullanılabilir
* **git checkout <commit hash>** komutu ile bir önceki commite dönülebilir. Fakat "**HEAD detach**" şeklinde master branch ile aynı seviyede olmayan bir branch oluşturulur

Mevcut branch den kopuktur. Fakat master branch de hala o commit mevcut kalır. Yapılan değşikliği kaydetmek için yeni bir branch oluşturulabilir. Yapılan yeni değişiklikler bu yeni oluşturulan branch üzerinde tutulur. Aksi durumda yeni branch oluşturmadan tekrar master branch'e geçiş olduğunda o değişkliklerler (HEAD detach) silinir

* **git switch <branchname>** komutu ile de branch değiştirilebilir.

* "**git branch pilot**" master üzerinden yeni branch oluşturur
* "**git branch**" branch leri listeler
* "**git remote**" remote taki branchleri listeler
* "**git branch -r**" remote taki silinen branchleri de gösterir
* "**git switch pilot**" ya da "**git checkout pilot**" pilot branch ine geçer
* "**git branch -d branchName**" braanch adına göre o branch I siler

-> **Rebase**

* **git rebase master** örnek olarak pilot branch'Indeyken baştaki komutu çalıştırırsan ilk masterin commitlerini alır, sonrasında pilottaki

Commitleri, yeniden üzerine uygular.

* Yani önce master, sonrasında pilot taki commitler olarak dizilim yapıldı. Yeni bir history oluşturuldu
* Daha önceden master ve pilot merge edildi ise merge commitlerini de siler. Böylelikle gereksiz history deki loglarda silinebilir
* Mevcut dalda yapılan commit'leri alır.
* Bu commit'leri başka bir dalın en son commit'inin üzerine yeniden uygular.
* Git'te bir dalın history sini yeniden yazmak için kullanılır. Bu komut, bir dalın bazını değiştirmek için kullanılır ve daha temiz, doğrusal bir commit geçmişi oluşturmanıza yardımcı olur.

**-> Merge**

* **git merge master** örnek olarak bulunduğun branch pilot ortamı olsun. Baştaki komutu kullanarak master ile merge etmiş oldu
* -> **gitignore dosyası işlmeleri**

* "**touch .gitignore**" bulunduğun klasör içinde girignore doyası oluşur
* Gitignore dosyasına eklediğimiz dosya ismilerine göre takibi engeller
* Örnek; gizli.txt dosyası daha önceden commit edildi ise sonradan gitignore sayfasına eklediğimizde, takip işlemine yine devam eder
* Yani commit edilmeden önce gitignore sayfasına eklenmiş olması lazım

**Fast forwarding** : master branch'inin son halinden yeni bir branch alınıp, yeni geliştirmeler yapıldıktan sonra tekrardan master branch'e merge etme işlemidir.

**-> Stash işlemleri**

* " **git stash**" stash oluşturur (yapılan son değişikliği daha sonra kullanmak üzere saklamaktır.)

* **git stash save -a 'message'** yapılan son değişiklik için açıklamalı stash oluşturur

* "**git stash list"** stash'leri listeler

* "**git stash apply stash@{n}**" belli bir stash'i listeden silmeden uygular

* "**git stash pop stash@{n}**" belli bir stash'i listeden siler ve uygular

* "**git stash pop**" en sondaki stash'i siler ve uygular

* "**git stash clear**" tüm stash listesini siler

**-> Cherry Pick**

Git'te cherry-pick komutu, belirli bir commit'i bir daldan alıp başka bir dala uygulamak için kullanılır. Bu işlem, mevcut commit geçmişine dokunmadan,

yalnızca belirli değişiklikleri bir daldan diğerine aktarmayı sağlar.

* **git cherry-pick <commithash>**