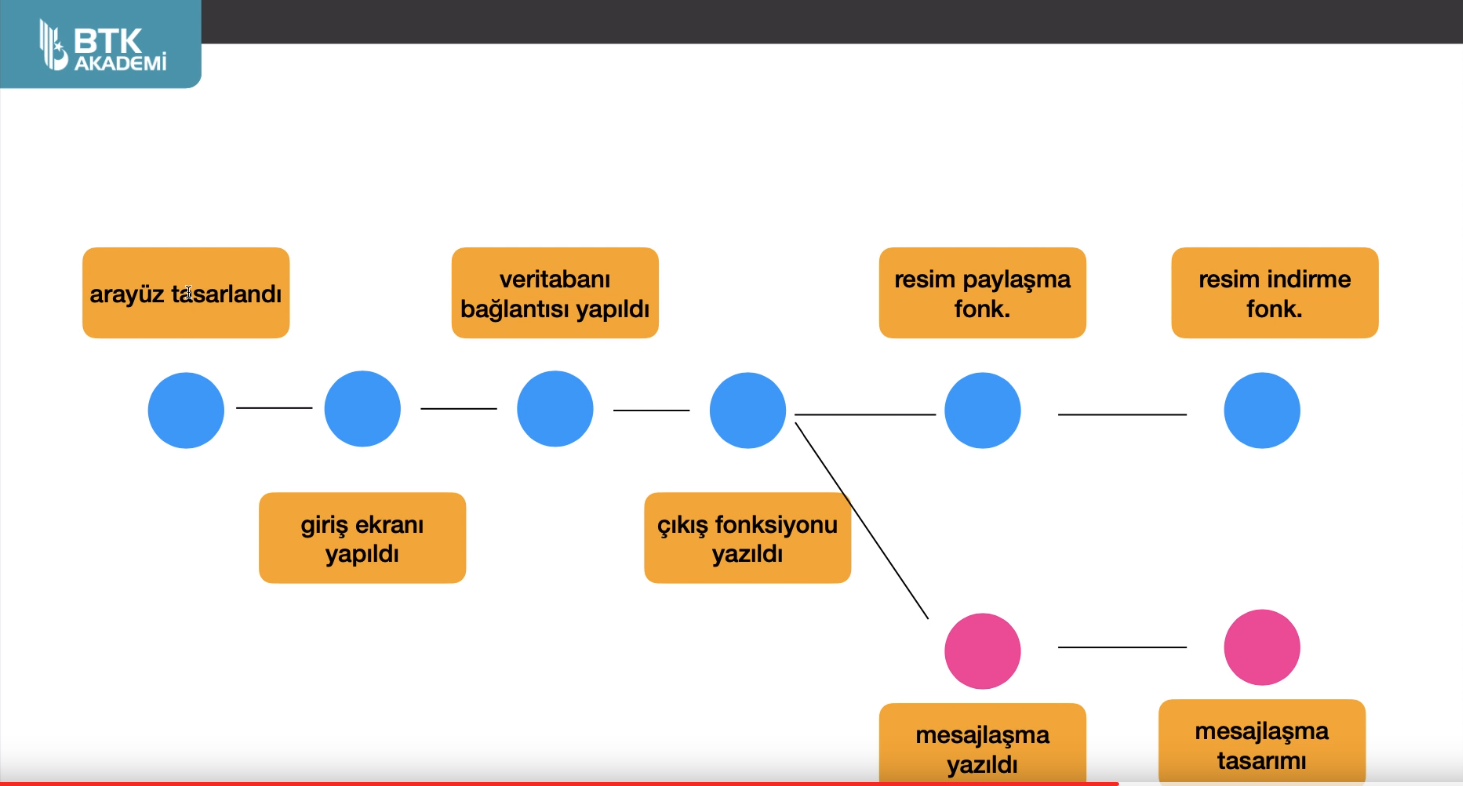
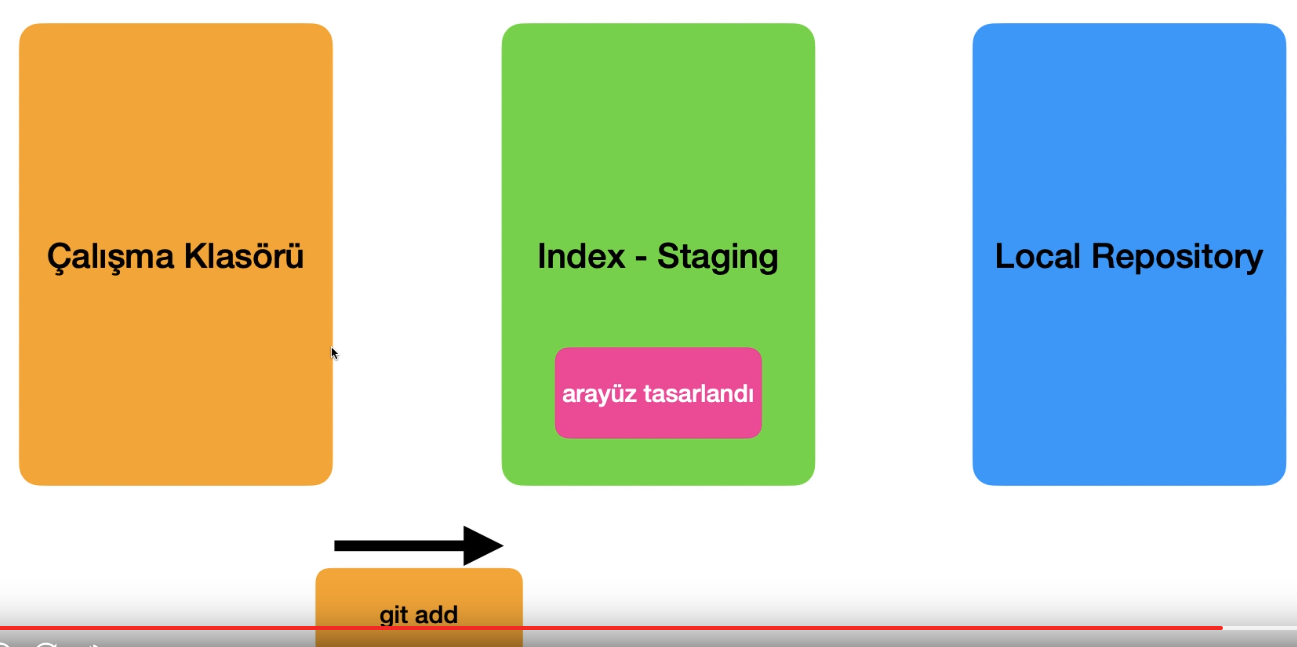
**GİT**

**Git Terminal Command**

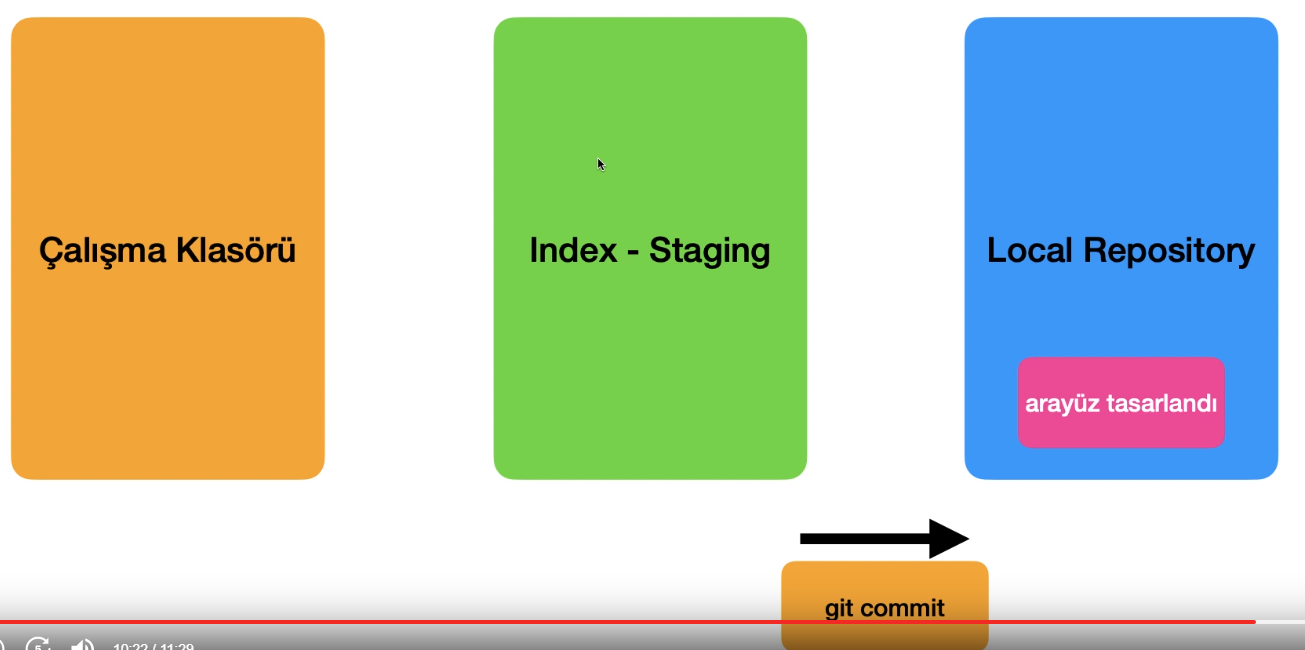
* ls => list, klasör içinde bulunan dosyaları göster.
* ls –la => klasör içinde bulunan gizli dosyalar ve klasörleri gösterir.
* pwd => print working directory, güncel olarak bulunulan klasörü göster.
* cd “DosyaAdı” => change directory, bulunulan klasörün içindeki klasöre gider.
* cd .. => Bir önceki klasöre tekrar gel.
* clear => Ekran temizlenir.
* cd Des + tab tuşu => cd Desktop/ şeklinde tamamlar.
* mkdir => make directory, klasör oluşturur.
* touch “dosyaAdı”.txt => .txt uzatılı dosya oluşturur.
* rm “dosyaAdı”.”dosyaUzantısı” => remove, ilgili dosyayı kaldırır.
* rm –rf “klasörAdı” => ilgili klasörü kaldırır.
* git => Git’in help dokümantasyonunu gösterir.
* git –version => Git’in yüklenmiş versiyonunu gösterir.
* git config --global user.name "Kemal Usanmaz" Git konfigürasyonuna Kullanıcı adı kaydetme
* git config user.name => Git konfigrasyonunda kayıtlı kullanıcı adı görüntülenebilir.
* git config --global user.email “kemalusanmaz1996@gmail.com” Git konfigürasyonuna email kaydetme
* git config user.email => Git konfigrasyonunda kayıtlı email görüntülenebilir.
* git status => klasörün git güncel durumunu gösterir.
  + Not a git repository => bu klasörün git ile herhangi bir bağlantısı yok.
  + On branch master => klasör default oluşturulan master branch’ine bağlanmıştır.
  + No commits yet => henüze commit yapılmadı.
  + Nothing to commit => klasör, commit yapılacak bir dosyaya sahip değil.
  + Untracked files => index-stagging’e atılmamış dosyalar git add “dosyaAdi” ile indexe atılır.
  + Changes to be comitted => git add ile index’e atılmış dosya.
* git init => klasörü git’e bağlamak için komut girilir. \*git init çalıştırılmadan önce git status ile klasörün git’e bağlı olup olmadığı kontrol edilir. Çünkü git init bir klasörde iki kez çalıştırılırsa sistem hatası olabilir.
  + Initialized empty Git repository in => git’e bağlanmaya başladığını gösterir.
* git add “dosyaAdi” => ilgili dosyayı index kütüphanesine ekler.
* git add . => çalışma klasörü içindeki tüm dosyaları index’e atar.
* git commit –m “message”=> index’e eklenen dosya repo’ya commit edilir. –m, zorunlu bir description alanıdır.
  + İnsertion => Eklenen bilgi sayısıdır.
* git log => git reposu içinde yapılan değişiklikleri tarih bilgisiyle verir. Bu bir commit’i temsil eder ve her commitin bir hash’i (id numarası) vardır.
* git commit –a => dosyayı tek komutta indexe alıp commit eder ve repoya alır.
* touch .gitignore => istenilen dosyaların repo içerisine alınması istenmiyorsa .gitbranch –d dosyası git’in başlatıldığı klasöre eklenir.dosyastudio code’da açılır ve içine repo içerisine alınması istenmeyen dosya uzantısıyla birlikte yazılır. Gitignore dosyası içine alınan dosya değiştirilirse git bunu görmez. Bireysel olarak yönetilmeli.(gitignore template ile ilgili dil ile ilgili gitignore şablonu bulunabilir.)
* git branch => güncel branchler gösterilir.
* git branch feat/branchAdi => Master branchten ayrılacak yeni bir branch açılır. Genelde isim, feat/yapılacak işlem olarak verilir.
* git branch -d <branch> => branch silinir.
* git branch –a => hem remote hem local branchleri gösterir
* git switch branchAdi => istenilen branche geçiş yapılır. Head buraya taşınır.
* git merge branchAdi => istenilen branch master branch ile birleştirilir, merge yapılır. Merge yapılmadan önce switch ile birleştirilmesi istenilen branche gidilmelidir. Daha sonra birleşme commiti oluşur. Buraya bir commit mesajı yazılmalıdır.
* git restore dosyaAdi => en son commit ne durumdaysa ona döndürür.
* git stash => git dosyayı çalışma klasörü ve index dışında ayrı bir deoplama alanına saklar.
* git stash pop => çağrılması istenilen bilgiyi çağırır. Studio Code’da ilgili dosyaya gidilmelidir.
* git stash list => güncel olarak saklanılan bilgileri çağırır.
* git stash apply stash@{sayı} => stash listesinde birden çok bilgi varsa ve istenilen bilgi çağrılması isteniyorsa sayı yerine istenilen bilgiye ait id no yazılır ve çağrılır.
* git stash clear => tüm stash listesi silinir.
* git checkout hash => hash numarası yazılan commite geri döndürür.
* git reset hash=> hash’i verilen commite kadar commitler gitten silinir fakat script’te bu değişiklik yapılmaz. Scripte değişiklik yapıldığı için indexe ve repoya tekrar alınmalıdır.
* git reset --hard hash => hash’i verilen commite kadar commitler gitten silinir. Scripte buna göre değişir.
* git revert hash => hash’i verilen committeki bilgiyi silip commiti logda tutar ve aynı branchten devam edilir. Logun tutulması istenmiyorsa direkt git reset –hard komuut kullanılır.
* git diff => commitler, branchler ve dosyalar arasındaki güncel değişiklikleri gösterir.
* git diff HEAD => HEAD yani son commite göre yapılan değişiklikleri gösterir.
* git diff hash1 hash2 => iki hash arasında güncel farklılıkları gösterir.
* git diff master branchName => iki branch arasındaki güncel farklılıkları gösterir.
* git config --global http.sslVerify false => ["unable to get local issuer certificate](https://stackoverflow.com/questions/23885449/unable-to-resolve-unable-to-get-local-issuer-certificate-using-git-on-windows)” hatası alındığında SSL doğrulamasını kapatmak için
* git config --global http.sslVerify true => SSL doğrulamasını açmak için
* Commit => Kaydedilen yazılım scriptleri
* Branch => Commitlerin dallanması
  + Master => Ana Branch, Git varsayılan olarak oluşturur.
  + HEAD => Son committe ve branchte olma durumunu gösterir.
* Merge => Branch ile ayrılan commitlerin birleştirilmesi
* Stash => Git bilgileri başka bir deoplama alanında saklar. Bu ilgiler istenildiği zaman çekilebilir.



* Çalışma Klasörü => İçinde kodların bulunduğu Proje klasörü
* Index-Stagging => git add komutu ile yazılan scriptler sahneye alınır.

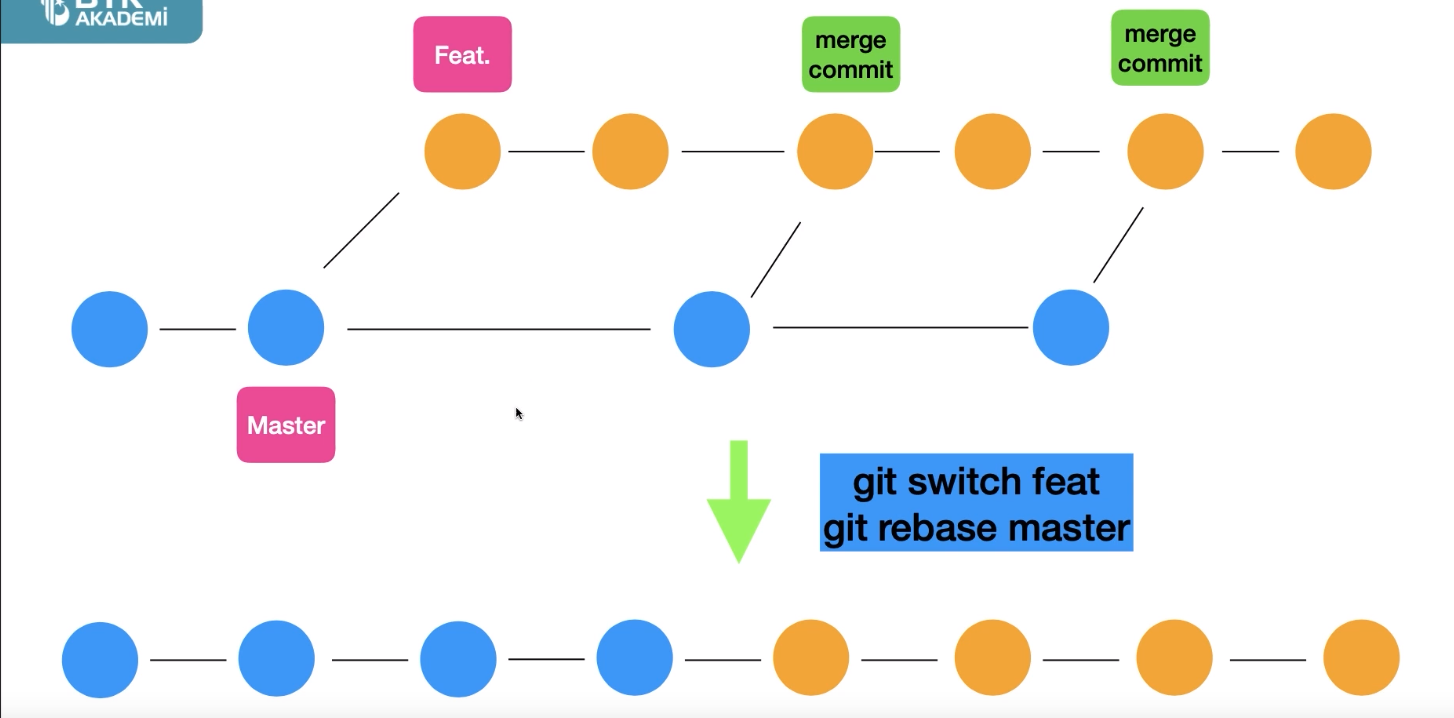


* Local Repository => git comit ile sahneye alınan scriptler git projesinde commitlenir.



Fast forwarding => master branchte bir commit yapmadan, diğer branchte commitler yapılır, daha sonra master branch ile diğer branch merge edilir. Diğer branchte yapılan commitler artık master branch içinde görülür. Bu yapı branchler arası conflicti engeller.

Rebase => feat branchinde çalışırken master branchinde de değişiklik olabilir.bu değişiklikler merge ile feat branchine de alınabilir fakat logu merge commitler ile şişirilmiş olur. Rebase yaparak masterda yapılan değişiklikler ve feat hizalanabilir. Git rebase log temizlemek ve tarihi değiştirmek için kullanılır.



**GİTHUB**

* git remote add origin https://URL.git => git klasörünü github’da oluşturulan repo’ya ekler.
* git branch –M main => git’te ana branch main yapılır.
* git push –u origin “ana branch adı” => master branchini origin (git URL)’e push eder. –u’yu bir kere kullanak yeterlidir. Aynı url’e atılacaksa git push kullanılabilir.
* gir remote => github bağlantısı yani origini verir.
* git branch –r => remottaki yani githubdaki brancleri gösterir.
* git fetch origin master=> github yani originden değişikleri alır local branche yani master’a getirir.
* git pull origin master => git fetch + git merge değişiklikleri al ve branche göm.
* git chekout origin/master => remote branchlere geçmek için checkout kullanılır.
* git clone Github\_URL => remote repo yu al ve bulunduğun yola indir.
* git push origin –d “remote branch adı” => github remote branchi git üzerinden silmek için kullanılır. Silinien branch ana branch ise github üzerinden default branch ayarı başka bir branch yapılıp silinmelidir.

\* Yapılan değişiklik githuba eklenmek isteniyorsa push yapılır.

\* Githubda yapılan değişiklik lokale eklenmek isteniyorsa pull yapılır.