**GIT**

**Git Bash Temel Komutları**

**git config –global user.name “adiniz\_soyadiniz” :** sisteme isim ve soyismimizi tanıtırız.

**git config –global user.email “email\_adresiniz” :** sisteme mail adresimizi tanıtırız.

**git config –global user.name :** sistemden isim ve soyisim sorgulaması yapabiliriz.

**git config –global user.email** **:** sistemden email sorgulaması yapabiliriz.

**pwd :** mevcut dizini gösterir.

**ls (list) :** bulunduğumuz dizindeki dosyaları listeler.

**cd (change directory) :** dizin değiştirmek için kullanılır.

**cd .. :** ile bir onceki dizine geçilebilir.

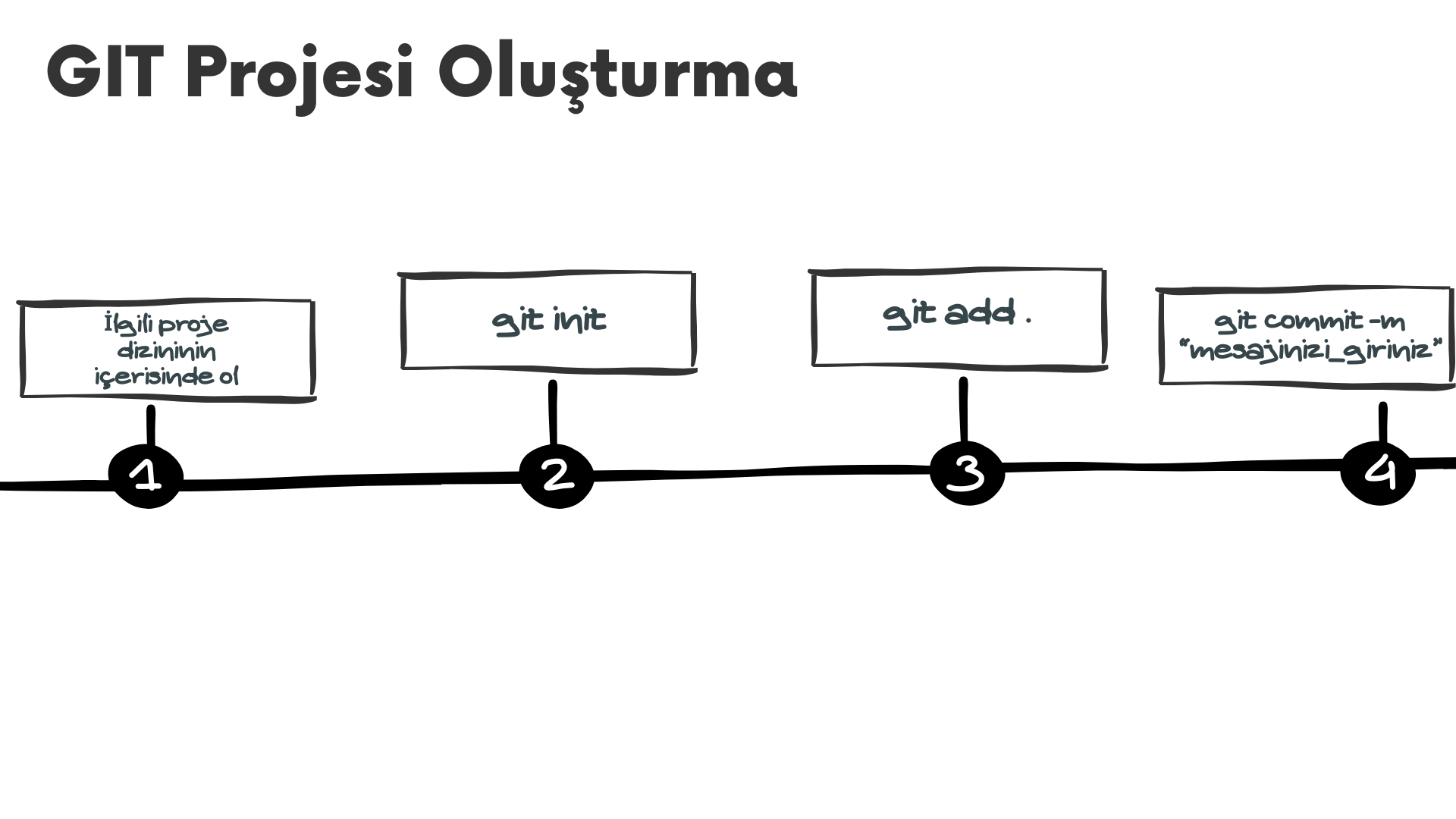
**clear :** terminali temizlemek için kullanılır. Ya da ctrl + L ile de kısaca temizlenebilir.

**Git Projesi Oluşturma**

Git projesi yapılmak istenen dosya dizininine gidilir ve **git init** komutu ile bu dosya git dosyası haline getirilir.

**ls -a :** oluşturulan git projesinin içerisindeki gizli dosyaların görüntülenmesini sağlar.

Git Repositroy (Depo) : Proje dosyalarının git tarafından saklandığı yerdir.



**git init :** projelerimizi içinde olduğumuz git dizinine eklemek için kullanılır.

**git add . :** dizindeki bütun nesneleri git deposuna ekler. (Aslına nesneler Staging Area denilen geçiş bögesindedir, tam anlamıyla repo ya yüklenmiş değildir.)

**git commit -m “mesajinizi\_giriniz” :** nesnelerin geçiş bölgesinden repo’ya tam anlamıyla geçiş yapmasını sağlar.

**git log :** aldığımız versiyonları listelemek için kullanılır.

**git status :** projemizde yapılan değişiklikleri gözlemlemek için kullanılır. Eğer zaten çalışma dizini ile git deposu arasında bir fark yoksa yani, çalışma dizinindeki bütün değişiklikleri commit edilip git deposuna gönderilmişse bu kodun çalışması sonucu herhangi bir çıktı alınmaz.

**git diff :** dosyalarda yapilan degisiklikleri satir satir listeler.

Dosya içeriğinde bir değişiklik yapıldıysa ve bu değişiklilkleri tekrardan git deposuna gönderilmek istendiğinde de aynı işlemler uygulanır.

**git diff --staged :** Staging Area’daki degisiklikleri gösterir.

**Git Üzerinden Dosyaları Silme**

2 yöntem vardır:

1. Çalışma dizininden elle silinebilir.

Normalde bir dosya Git’e eklenmeden önce geçiş bölgesine “ git add . “ komutu ile göderilir. Aynı işlem “ git rm “ (rm: remove) ile de yapılabilir. Yine bu kod silinen dosyayı geçiş bölgesine taşıyacaktır.

1. git komutu ile silme

git rm

**git rm -r klasor\_adi/ :** hem klasorun icindekileri hem de klasorun kendisini siler.

**Dosya İsimlendirme**

Dosya ismi ve konumu değiştirme

Dosya adını değiştirmek için 2 seçenek vardır:

* **Elle manuel olarak değiştirilebilir:** Elle dosya adı değiştirildikten sonra; git status, git add ., git commit -m “mesaj” komutlarının sırasıyla çalıştılması ile işlem tamamlanmış olur.
* **Git komutu ile değiştirilebilir.** (daha sağlıklı ve daha kolay): **git mv [dosyanin\_mevcut\_ismi] [dosyanin\_yeni\_ismi]** örnek -> git mv den2.txt den2.1.txt Yani; “den2.txt” dosyası, “den2.1.txt” olarak değiştirildi.

**Dosya Dizini Değiştirme**

mv komutu : hem dosya adını değiştirmek için hem de dosyayı başka bir dizine taşımak için kullanılabilir.

**git mv [tasinacak\_dosya\_adi] [tasinacak\_dizin\_adi]** örnek; **git mv den2.1.txt yeniKlasor/** Yani; den2.1.txt dosyası, yeniKlasor klasörüne taşındı.

**GIT Üzerinden Geri Alma İşlemleri**

**Çalışma Dizini Üzerinden Geri Alma**

Örneğin bir dosya oluşturduk ve bunu git e ekledik. Sonra dosya içeriğinde yanlışlıkla bir hata yaptık ve kaydedip çıktık.

Bunun için önce git bash e gelip “git status” ile kontrol yapıldığında değişiklik gözlenlenebilir. **git checkout --[dosya\_adi] :** örnek; git checkout –index.txt ile işlem geri alınabilir.

**Checkout:** commitler arasındaki geri alma işlemidir.

**Restore:** commit edilmese bile yapılan değişiklik üzerinden geri alma işlemini sağlar. (Mevcut commit üzerinden değişiklik yapılabilmesine olanak tanır.) get restore [dosya\_adi] şeklinde kullanilabilir.

**Geçiş Bölgesi Üzerinden Geri Alma**

İçeriğinde değişiklik yapılmış ve geçiş bölgesine alınmış bir dosyanın geri alınması için: git add komutunun tam tersi işlem yapan, **git reset HEAD index** komutu kullanılmalıdır. Bu işlem ile geiş bölgesinden geri alma işlemi tamamlanır ancak dosya dizininden icelelndiğinde dosyadaki değişiklik gözlemlenemez. Bunun için bir de dosya dizinininden geri alma işlemi yapılmalıdır. Yani **git checkout --[dosya\_adi]** komutu yazılarak içerisinde yapılan değişikler geri alınmış/kurtarılmış olur.

**Versiyon Değiştirme**

Örneğin 3 versiyon var elimizde:

versiyon1, versiyon2, versiyon3

versiyon 3 ten –> versiyon 2 ye geçmek için;

**git checkout [hashcode] -- . :** kodu ile bu işlem yapılabilir.

**GITHUB**

GIT ile GITHUB sıkça karışıtırılır. Burada git; versiyon kontrol sistemiyken github; git yazılımı ile entegre olmuş bir depolama alanıdır. GITHUB, git projelerini uzak bilgisayarlarda saklayıp yönetebileceğimiz bir yerdir.

Aynı GITHUB gibi depolama hizmeti veren birçok web sitesi vardır. GITLAB, BITBUCKET gibi ...

**GITHUB’a Proje Gönderme**

Önce GITHUB üzerinde yeni bir repository oluşturulur ve,

Mevcut proje dizinindeyken,

**git remote add githubRepo [url]** : GITHUB’da oluşturduğumuz gönderinin bağlantısını buraya gönderilir. (Uzak bağlantı, githubRepo yerine başka bir şey de yazabilirdik. ) githubRepo; GITHUB bağlantısının takma adını belirler.

**git remote :** ile bağlantının eklenip eklenmediği kontrol edilebilir.

**git push -u githubRepo master :** kodu ile projemiz github a yüklenir.

-u : bütün dosyaların alınmasını sağlar.

githubRepo : bağlantısını kurduğumuz github deposu

master : bu git projesinin ana dalı

Şimdi eklenen proje üzerinde local de bir değişklik yapalım.

Örneğin bir dosyayı sildik ve bu işlemi github da da güncellememiz gerekecek. Değişkliği tamamladıktan sonra,

**git push -githubRepo master** kodunu tekrar çalıştırarak güncelleme işlemini tamalarız. Bu işlemler, GITHUB arayüzünden de yapılabilir.

**.gitignore Kullanımı**

GIT prejesinde, git tarafından takip edilmesini istemediğiniz dosyalar varsa bu dosyaların ismini .gitignore adında bir dosyaya yazılmalıdır. Bu dosyalar git tarafından görmezden gelinir.

Örneğin, açık kaynak kodlu bir proje paylaşıyoruz ama veritabanı dosyasının GITHUB a gonderilmesini istenmeyen bir durumdur. Bunun için bu istenmeyen dosyalar .gitignore doyasına yazılarak GIT tarafından takip edilmesini engelleyebiliriz.

cat >> .gitignore : komutu ile gitignore adında bir gizli dosya oluşturulur. Gizli dosya . karakteri ile oluşturulur. Bu işlemden sonra enter a basılır, comut sistemi bizden dosyanın içine koymak istediklerimizi ister buraya eklemek istediklerimizi ekledikten sonra “ctrl + c” yaparak dosyayı kapatırız.

**cat .gitignore :** işlemin gerkçekleşip gerçekleşmediği kontrulü yapılabilir.

NOT: takip edilmesi istenmeyen dosyaların adları .gitignore dosyasına kaydedilmeden git add veya git commit işlemlerinin yapılmaması gerekir. Eğer bu işlemleri yapıldıktan sonra .gitignore dosyasına kaydedilirse hiçbir işe yaramayacaktır.

.gitignore dosyasını git deposuna gonderelim.

Şimdi de bir klasör içerisinde birden fazla dosya bulunan bir klasörü .gitignore dosyasına atmak için,

**gedit .gitignore :** ile .gitignore dosyası açılır.

Dosya içerisine -> **[klasör\_adi]/\***

Örneğin; htmlFiles/\* ile tüm dosyaları .gitignore a atabiliriz. Burada **\*** = tüm dosyalar anlamına gelir. Eğer tüm dosyalar arasından hariç tutmak istediğimiz bir kaç dosya var ise, bir alt satıra geçerek:

**![klasör\_adi]/[dosya\_adi] :** şeklindeki bir kullanım ile hariç tutulabilir.

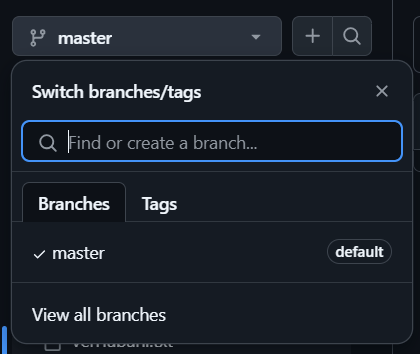
Örneğin; **!htmlFiles/dosya1.html**

Böylece GITHUB da yalnızca dosya1.html dosyası görüntülenmiş olur.

**Branches (Dallar)**

Branche : git projemizin bir kopyasıdır. Master dalı, git ile bize varsayılan olarak gelir ve git projemizinin tamamını kapsayan ana daldır. Master dalından bir kopya alındığı zaman, yeni bir dal oluşturulmuş olur.

Bu dallar şu işe yarar: Örneğin projede birden fazla kişi ile çalışıyorsak ve her çalışanın da görevi belliyse takım arkadaşlarınıza projenizinden bir dal oluşturursunuz. İşler bittikten sonra da bu dalları master dalı ile birleştiririz. Böylece herkesin ayrı ayrı geliştirdiği modüller birleşerek tek bir proje halini alır.

Önce GitHub üzerinden dal oluşturalım:

Buradaki kutucuktan oluşturacağımız dalın adını yazarak dal oluşturma işlemi yapılabilir.

GitHub ‘dan yapılan değişikliklerin local’e çekilmesi için : **git pull** komutu kullanılabilir.

Terminal üzerinden dal oluşturma;

**git branch :** projedeki mevcut dalları listeler.

**git branch --all :** uzak depodaki dalları da listeler.

git branch komutununun yanına oluşturmak istediğimizi dalın ismini girerek yeni dal oluşturma işlemi gerçekleştirebiliriz. Örneğin;

**git branch yanDal2**

**git checkout [dal\_Adi] :** oluşturduğumuz yeni dala geçişi sağlar. Örneğin: git checkout yanDal2

master ile yanDal2’yi birlestirelim bunun için önce master dalına geçiş yapmamız gerekir.

git diff master yanDal2 : yazarsak master ile yanDal2 dalları arasındaki farkları görebiliriz.

git merge yanDal2 : komutu ile master ve yanDal2 dalları birleştirilebilir. Zaten master dalında olduğumuz için master dalının adını yazmaya gerek yoktur.

Yapılan bu değişiklikleri GitHub’a yollayalım.

**git push –u githubRepo master**