**Git’e başlamadan önce şunları bilmek gerekir :**

Git, her bir commit işleminde kimin değişiklik yaptığını kaydetmek için kullanıcı adı ve eposta adresi bilgisine ihtiyaç duyar. Bu bilgiler bir kez yazıldıktan sonra Git, bu bilgileri global ayarlarınızdan otomatik olarak alır.

Commit : Git deposunda değişikliklerin kaydedildiği anlık görüntü.

HEAD : Git deposundaki mevcut konumunuzdur.

**Git kurulumu ve kontrolü**

* Komut: git --version (git kurulu mu değil mi, kurulu ise version kontrolu için)

1. Global kullanıcı adı ve e-posta nasıl ayarlanır?
   1. Global Ayar (tüm projeler için geçerli)

Komut: git config --global user.name “Kullanıcı Adı”

Komut: git config --global user.email email adresi

* 1. Bu bilgileri girdikten sonra bilgilerde bir değişiklik yapılmayacaksa Git için tekrar bu işlemleri yapmanıza gerek yoktur.
  2. Git ayarları belirli bir dosyada saklanır.

1. Yerel Ayar (Sadece bir proje için gecerlidir)
   1. Komut: git config user.name “projeye özel kullanıcı adı”
   2. Komut: git config user.email «email adresi»
2. Git, kullanıcı bilgilerini ayarlar dosyasında saklar, bu yüzden tekrar girmenize gerek yoktur. Eğer bu ayarları değiştirmek isterseniz, aynı komutları tekrar kullanarak düzenleyebilirsiniz.
   1. Komut: git config --global user.name “Yeni kullanıcı adı”
   2. Komut: git config --global user.email «Yeni email adresi »
3. Ayarların doğru yapılıp yapılmadığı kontrolü

Komut: git config user.name

Komut: git config user.email

**Terminal Kullanırken Gerekli Bazı Komutlar**

1. Komut1 : ls ile bir klasörün ya da dizinin içeriğini listeler.
2. Komut2 : start . ile o an terminalde hangi klasörde çalışıyorsanız, o klasörü Windows'un grafiksel dosya yönetim arayüzünde görmenizi sağlar.
3. Komut3 : clear ile terminal ekranı temizlenir.
4. Komut4 : pwd ile konumunuzu görürsünüz.
5. Komut5 : cd klasörAdi ile konum,dizin degistirebilirsiniz.
6. Komut6 : cd .. ile bir önceki konuma giderim.
7. Komut7 : touch dosyaAdi.uzantisi ile dosya oluşturulur. (klasör değil)
8. Komut8 : mkdir klasörAdi ile klasör oluşturulur.
9. Komut9 : rm dosyaAdi.uzantisi ile dosya silinir.(çöp kutusuna göndermez direk siler)
10. Komut10 : rm -rf klasörAdi ile klasörü içindekiler ile birlikte silersiniz.
11. Komut11 : ls -a ile gizli dosyaları listelerim.

**Bir Git Deposu Oluşturma :**

Git ile çalışmaya başlamadan önce bir depo (repository) oluşturmanız gerekmektedir.

Bir yerde GIT kullanmaya karar verirsem o proje klasörüne gidip git deposu oluşturmam gerekir. Her depo birbirinde ayrı olur. Istedigim kadar depo kurabilirim.

1. Komut : git init (bu komutu girerken çalışma yaptığınız dizinde olmalısınız.)

Bu komut çalışma dizinize .git isimli bir klasör ekleyerek orayı Git deposu haline getirir. Nerede olursanız olur orayi git deposu haline getirir.

Not: .git isimli klasör gizli klasör olarak oluştuğu için onu görmek için bulundugunuz dizinde terminalden ls -a komutu kullanılır.

iç içe git init yapılmaz. Git’in kafası karışır.

URL örneğin: <https://github.com/username/repo.git>

1. Git ile çalışacağınız yerel dosyaları izleme kapsamına alma

Komut: git add dosyaAdi.uzantisi (sadece belirtilen dosyayı ekler)

Komut: git add . (dizindeki her dosyayı ekler)

1. Durum kontrol etmek için:

Komut: git status (hangi dosya izleniyor veya izlenmiyor görmeye yarar)

1. Değişiklikleri kaydetme:

Komut: git commit -m «aciklama» (çift tırnak içinde bir açıklama)

Buradaki -m commit mesaji belirtir. Değişikliğimizi anlamlandırır.

1. Eğer GitHub gibi bir platformda uzak bir depo oluşturduysanız (remote repository), bu depoyu yerel (local) deponuza bağlamalısınız.

Komut : git remote add origin uzakBaglantıUrl

1. Uzak depoya gönderme (PUSH)

Komut : git push -u origin branchname (branch name: genellikle main ya da master olur ama eğer başka bir dal üstünde çalışıyorsanız onun adı yazılır.

-u ne işe yarar : Bu dalı (branch) varsayılan olarak ayarlar. Bir sonraki push işleminde tekrar dal ismi belirtmenize gerek kalmaz.

1. Uzak depodan güncellemeleri çekme

Komut : git pull origin branchname

Bu uzak depodaki değişiklikleri yerel depomuza çekmemizi sağlar.

1. Geçmişi görüntüleme

Komut: git log

Yapılan tüm commitleri listeler ve hangi commit kim tarafından yapılmış ve hangi mesaj eklenmiş görüntüleminizi sağlar. Git log ile commit geçmişi detaylarıyla birlikte görüntülenir. Git log sayfasını kapatmak için Q tuşuna bir kez basmanız yeterlidir. Ayrıca hash’lerin kısa halini görüntülemek için git log --abbrev-commit

Son olarak git log --oneline daha güzel ve sade bir görünüm sunar.

1. Yeni bir dal (branch) oluşturma

Komut: git branch branchname

Yeni bir dal ismi yazıyoruz. Örneğin git branch feature-x

1. Dalları değiştirme (dallar arası geçiş yapma)

Komut: git checkout branchname

Bu komut ile belirtilen dala geçiş yaparız.

1. Oluşturduğumuz yeni dalı uzak depoya gönderme

Komut: git push -u origin branchname

1. Yeni bir dal (branch) oluştururken direk oluşturduğunuz dal’a geçmek isterseniz:

Komut: git checkout -b branchname

Bu komut ile yeni bir dal oluşturup o dala geçmiş olursunuz.

1. Birleştirme (merge) işlemi:

Komut: git merge branchname

Merge esnasında komuta branchname yazılırken bulunduğumuz dal değilde birleştirmek istediğimiz dal ismi yazılır ve bulundugumuz dal ile yazdığımız dal birleşmiş olur.

**Ileri seviye Rebase ve Hash yapabilme**

1. Iki branch'imiz var olduğunu varsayalım; bulundugumuz branch’ın seçili olduğunu varsayarsak (ki bu main değil yani ana dal değil), çalışmamızı doğrudan main branch'indeki çalışmanın üzerine taşımak istiyoruz. Bu durumda bu iki daldaki özelliğin sırayla geliştirildiği görünecek, ama gerçekte eşzamanlı olarak geliştirildiği anlaşılacaktır. Yani burada kopya oluşturarak bir yaptığımız bu işlemi komutu

Komut: git rebase main (main yazdık çünkü biz diğer dal üzerindeyiz)

Komut: git rebase branchname (main üzerinde iken bu komut girilir)

1. Hash'leri görmek için git log kullanmanız gerekecek. Komut: git log

Örnek bir hash: fed2da64c0efc5293610bdd892f82a58e8cbc5d8

Git'in hash'lerle mantıklı çalışır. Sadece commit'i benzersiz şekilde tanımlayabilen hash'in yeterince uzun kısmını belirtmeniz gerekir. Yani yukarıdaki uzun string yerine fed2 yazabilirim.

Komut: git checkout C4; git checkout HEAD^ burada C4 commitinde bulunan … isimli branch ile HEAD birbirinden ayrıldı ve sonrasında HEAD ile 1 adım geri gidildi.

1. Göreceli commit’ler güçlüdür, ancak 2 basit örneği aşağıdaki gibidir.

* ^ (caret) ile bir commit yukarıya hareket etmek ve ~sayı (bir sayi girilir) ile birden fazla commit yukarıya hareket etmek mümkündür. (~ tilde operatörü)

Komut: git checkout main^

Bu sayede HEAD^ ile zaman içinde geri gidebiliriz.

Komut: git checkout HEAD~4 (4 adım geri gitme komutu)

Bu komut git checkout C4; git checkout HEAD^ komutunu 4 kez uygulamaya gerek kalmadan tek seferde HEAD ile 4 adım geri gitmeyi sağladı.

* Şimdiki komut ile hangi adıma hangi branch^hi taşıyacağımizi söylüyoruz.

Komut: git branch -f main HEAD~3 yani burada main branchini 3 adim geri taşıdık.

Ya da git branch -f main C6 ile C6 commitine maini taşımış oluruz.

Not : git branch -f ile dalı, git checkout HEAD~num ile HEAD'i taşıyabilirsiniz.

Yani git checkout HEAD~num komutu HEAD’ i hareket ettirir.

Ama git branch -f … komutlari ile main ya da yazdığımız branch hangisi ise onu istediğimiz noktaya hareket ettirir.

Geri Alma İşlemleri

1. Komut git reset HEAD~num, bir dalı geçmiş bir commit'e taşır ve o committen sonraki değişiklikleri iptal eder. Bu anlamda, onu "tarihi yeniden yazmak" olarak düşünebilirsiniz; git reset, bir dalı, sanki commit hiç yapılmamış gibi geriye doğru hareket ettirir.
2. Değişiklikleri geri almak ve geri alınan değişiklikleri başkalarıyla paylaşmak için git revert kullanmamız gerekir.

Komut git revert HEAD

1. git cherry-pick <Commit1> <Commit2> <...>
2. örneğin: git cherry-pick C2 C4

**Amending commits (Taahhütlerin değiştirilmesi) :**

Komut1 : git commit -m ‘some commit’

Komut2 : git add forgotten\_file

Komut3 : git commit --amend (bu komut ile ne yapabiliriz: örneğin 4 tane dosya da değişiklik yaptım ama 3 dosyayı add yaptım 1 tane dosyayı unuttum ve git commit -m “biraciklama” ile kaydettim sonra 1 dosyanın commitlenmedigini fark ettim. İşte bu gibi durumda son kalan dosyayı git add yaparak ekledikten sonra git commit --amend yaparak son commit noktasına dönerim ve oraya ekleme yapmış olurum. Bu işlem ile son commit noktasına dönüş yapılır.

git remote -v : bu komut ile git bash üzerindeyken gönderdiğimiz dosyalarin github’ta nereye gittiğini görebiliriz.

git branch : bu komut ile hangi dalda işlem yaptigimizi görürüz.

git branch -vv : Hangi dalın hangi uzaktaki dala bağlandığını kontrol etmek için kullanılır.

git remote show origin : Varsayılan olarak, Git hangi remote ve dalı kullanacağını bilir. Bunu kontrol etmek için kullanılan komuttur.

Eğer farklı bir GitHub deposuna push yapmak istiyorsanız, remote URL'sini değiştirebilirsiniz: git remote set-url origin https://github.com/yeni-kullaniciadi/yeni-depo.git