Terminal Crash Course: Navigation

ls (Masaüstü konumunu verir)

ls documentation/

ls documentation/Project

start . (Konumdaki tüm dosyaları açar)

clear (Tüm terminali temizler)

pwd (Şu an bulunduğun konumu gösterir)

cd Code (Dosyaya gider, sonra pwd yaz konumu takip et , daha sonra ls ile mevcut dosyaları gör sonra ) cd SecondCode (diğer dosyaya geç)

cd .. (işlemi geri alır)

touch purple.txt orange.py yellow.js strawberry.png dosya.pdf (Dosya oluşturur , yerini belirlemek için pwd kullan sonra o kısımda ls ve sonra start . ile konumunu aç sonra da dosyanı oluştur)

cd Pictures

cd Resimler

touch kangal.png balık.png

ls (Bu da özel dosya açma yöntemidir)

mkdir (dosya oluşturma komutudur.) mkdir Animals

#Aynı anda birkaç dosya oluştur

/Pictures/Resimler

mkdir Animals tigers lions Turtles sea\_turtles

rm (sayfa silme komutudur)

rm Kangal.png

rm -rf (Dosya silme komutudur)

rm -rf Animals

git status (Mevcut git reposunun durumunu gösterir)

git init (git reposu oluşturur. Fakat bu gizlidir.)

ls -a (git reposunu gösterir)

cd .git(içine girdik)

ls

git status

!!! Çok sık yapılan bir hata git reposunun bulunduğu bir yere bir git reposu daha açmaktır. Bu ileride çeşitli hatalara sebep olur bu yüzden ilk önce git status ile kontrol etmek gerekir.

Myfirstnovel

mkdir Intro

cd ıntro

mkdir Drıft

cd Drıft -🡪 git reposu içerir çünkü myfirstnovel git reposu içerir.

cd ..

cd ..

cd ..

mkdir book2 🡪 git reposu yok fakat ilk kontrol et(git status) sonra

git init

git add file1 file2

git commit -m “start work and define characters” (Git artık bu dosyayı izliyor)

Herhangi bir dosyada en ufak değişiklik dosyanın modifiye edildiğini gösterir (git status kullan)

Git log (Durumu gösterir – kimin ne zaman değişiklik yaptığını ve commit durumlarını gösterir.)

Git add . (tüm dosyaları ekler) ilk dosya eklenir sonra git commit -m “…..” ile dosyalar commitlenir.

Git status

Nothing , Work on tree clean

Belli bir noktaya geldikten sonra ( cd Myfirstnovel)

code . (git te kullandığın kodu açacaktır.)

Git docs --- > Git sayfasında documentation ve book kısmından eksik olduğun kısımları ve hatırlaman gereken yerleri okuyabilirsin.

Atomic commits

Bir bütünü en ince parçasına kadar ayırman gereken kısımdır. Özellikle git ile çalışırken buna dikkat etmek gerekir.

VScode 🡪 bütün parçalarda isim değiştirmek istiyorsan search kısmından yapabilirsin. Örneğin Myfirstnovel / chapter1 chapter2 outline characters hepsinde bulunan bi isim olan Gasby -- > Casby yapmak istersen böyle yap sonra da git add ve git commit mesajı yaz.

Git add Moodboard/cars1.jpg or git add Moodboard/

Commit mesajları present tense ile mi yoksa past tense ile mi yazılmalıdır?

Bunun net bir cevabı yok. Fakat daha yaygın kullanılan present tense. Bu yüzden açık kaynak kodu yazarken çalıştığın ekipte belirtilen duruma dikkat etmek gerekir.

git status

modified outline.txt

git add outline.txt

git commit

(Uzun söylenmek istenen ve eklenmek istenen veri mesajlarında kullanılır. Çünkü git commit -m”” genellikle satırlık mesajlar için kullanılır.)

I

Fix the typo in outline

:wq (bu kodlamalar bende gelmedi sanırım kullandığım sürümden kaynaklı direkt mesajı yazdım.)

git bash ‘ de

git config --global core.editor "code --wait" (documentation – books)

herhangi bi dosyayı değiştirdim

git add dosya

git commit

sonra direkt 🡪 düzenleyicine götürecektir.

(bu kısımları atladım çünkü beni direkt düzenleyiciye götürüyor \* vs code)

Eğer git log mesajlarını daha kısa görmek istiyorsak , çünkü epey bir git deposu oluşturulacak ve bunlar arasında commit mesajları çok olacaktır bu yüzden kullandığımız git logu kısaltmamız gerek.

Git log –abbrev-commit (commit ‘den sonraki harf ve sayı karışımını kısaltır , geri kalan her şey aynı)

Git log –oneline (gönderenin ismi ve maili dahil her şeyi gizler sadece gönderilen mesajı tek satır halinde gösterir.(ilk satırı)

Git add chapter1.txt chapter2.txt characters.txt

Git commit -m “add files to heading”

(sonra bakıyoruz ki eklememiz gereken bir dosya eksik, unutulmuş)

Onu eklemek için başka bi işlem yapmadan

Git add outline.txt

Git commit –amend ( commit mesajını düzeltmek , hataları gidermek ya da sadece eksik dosyayı eklemek için kullanılır )

Gitignore

Bir uygulama üzerinde çalışırken , gizlemen gereken dosyalar varsa yani githuba yükleyeceğin zaman insanların görmesini istemediğin örneğin kimlik bilgileri , kredi kartları , veya API key formlar gibi dosyalarda bunları ignore etmek ( görmezden gelmek isteriz)

Günlük yaptığımız basit işlerde , yüz kişinini bir projede çalışması veya bazı programlamalarda Python,java,javascript,angular vs. gibi günlük yaptığımız ve çok da önemli olmasa da proje kalitesini artıracak fakat başkalarının görmesinin lüzumu olmayıp dosya sayısını artırmaya gerek duyulmayan durumlarda da git ignore kullanılır ve çok işlevlidir. Yine detay bilgi git docs dan ulaşılabilir.

Mkdir Gitignoredemo

Cd gitignordemo

Git status

Git init

Touch index.js

Touch secrets.txt

Touch art.py

Code .

Dosyaları değiştir sonra git commit yap fakat secret ve art a dokunma

Sonra touch .gitignore

.gitignore (dosyası içinde)

.DS\_Store ( bu mağaza genellike göz ardı edilmeli) olsa da olmasa da yaz garantiye al

Secrets.txt

Art.py

Git add .gitignore

Git commit -m “ add git ignore “

Sonra bu secrets.txt ve art.py de istediğin kadar değişiklik yapsanda gözükmeyecektir.

Nedeni üst tarafta açıklandı. Şu sayfa hangi kodları yazmaman gerektiği hakkında bilgi verir :

<https://www.toptal.com/developers/gitignore>

Git branch kavramı

Bir proje üzerinde çalıştığımızda her commit yaptığımızda aynı dal üzerinden farklı bir bölüme geçeriz. Örneğin başta 1293xjfkd kısmındayken bir dallanma olmadığında commit yaptığımızda 1239043dfdfeg kısmına geçmiş olalım. Bu master dediğimiz kısım Git in ilk yaptığımız depo için oluşturduğu kısımdır. Bir projede herkes farklı bir alanda çalışıyorsa tek bir dal olmaz. Dallanmalar olur çünkü örneğin oluşturulan web sitesi için çok farklı alanlar vardır. Bunun için bir dal üzerinden değil onlarca dal üzerinde çalışma gerçekleştirilir. İşte buna dallanma deriz. HEAD Kavramı ise en son nerede calısma yapıyorsak onu işaret eden yapıdır. Fakat master değişmez ana bölümdür ve en son birleşilen kısımdır.

Git branch (branchleri gösterir , kaç tane branch var )

Pictures –

Mkdir roadtripplaylist

git status

git init

touch playlist.txt

playlistte(birkaç değişiklik yap)

git add playlist.txt

git commit -m”değişiklik”

git branch : \*master HEAD 🡪 Master

git branch oldies

git branch : \*master oldies

git switch oldies

playlistte(birkaç değişiklik yap)

git add playlist.txt

git commit -m”değişiklik2” HEAD 🡪 oldies

git switch master (Artık değişiklik2 görünmez)

DALLANMALARA DİKKAT ET

Git branch saian

Git switch saian

Git commit -a -m “add two Saian songs” (-a (add .) kısaltmasıdır. )

git switch master

git branch emptyplaylist

git commit -a -m “empty list”

burada master ve empty list dallanmış diğer tarafta master , oldies , saian branchları dallanmıştır.

Git checkout == git switch

Git switch -c oldies (direkt geçiş yapar)

Git commit yapmayı geçiş yapmadan önce yapmayı unutma !!

Git branch -d name (silmek için uyarır) 🡪Head farklı bir konumda olmalı

Git branch -D name (siler)

Git branch -m rename (isim değiştirir) 🡪 Head o konumda olmalı

Diyelim ki masterdasın

Git ls -a

Cd .git

Cat HEAD

Böylece (HEADIN bulunduğu dosyayı görürsün) git log ,, hashi kontrol edersin

Cat .git/HEAD (alternatif)

Git branch -v (tüm yapılan dalları ve son commitleri gösterir)

Git switch master

Git merge oldies

\*\*\*(İlk önce master klasörüne geliyoruz daha sonra master klasörünü en son yaptığımız branch ile değiştiriyoruz. (Merge komutu) ,, bunu yapmamızın sebebi sonradan eklenen dallanmaların (ekip calısmasıysa beğenilirse ana klasör yapılmak istenmesi)\*\*\*

\*\*\*Aşamalı merge\*\*\*

İlk önce masterda işlem yapılır

Sonra yeni bir branch (ABBA) işlem yapılır

Sonra masterda tekrar işlem yapılır

Ardından ABBA branch ile merge edilir. Çift daldan türeme olur. Örneğin siz bugfix yaptınız fakat ekip arkadasınız masterda işlem yaptı o zaman birleşme de çift işlem olacaktır.

\*Merge – Resolving ( Karışık durumda )

Eğer iki taraf ana dalda ortak bir işlem yapmışsa , örneğin biri beğenmediği bir durumu siler , diğeri de beğendiği bu durumu cogaltırsa , ortak işlemden dolayı merge fail olur. Bunu düzeltmek için şu adımlar gerekli:

Git switch master

Git switch -c Enes (Enes klasörünü oluşturur ve orada işlem yapar)

Git switch -c Semih

\*\*çeşitli değişiklikler yapılır , ekleme veya çıkarma

Git switch Enes

\*\*çeşitli değişiklikler yapılır , ekleme veya çıkarma

Git switch -c combo (birleştirme amaçlı)

Git merge Semih

\*\*merge fail olur. Çünkü farklı 2 işlem vardır. (VSC Bunu otomatik düzeltir)

Daha sonra ;

Git add songs.txt

Git commit -m “resolving conflication” ➡➡➡➡ Merge işlemi tamamlanır.

Git checkout -b == git switch -c

Git diff komutu

Bu komutta hangi değişiklikleri yaptığımızı görürüz. Örneğin bir dosya da değişiklik yaptıysak ve bunu commit etmeden önce bu değişikligi görmek istiyorsak git diff kullanırız. Bu komut bize hangi işlemi çıkartıp hangi işlemi eklediğimizi gösterecek.

Aşalandırılmamış olanları bu komut ile gösteririz.

KULLANIMI ŞÖYLE:

OMEN@DESKTOP-0LI898T MINGW64 ~/colours (master)

$ git add colors.txt

OMEN@DESKTOP-0LI898T MINGW64 ~/colours (master)

$ git commit -m "add blue and purple"

[master ca80a46] add blue and purple

1 file changed, 3 insertions(+), 1 deletion(-)

OMEN@DESKTOP-0LI898T MINGW64 ~/colours (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: colors.txt

no changes added to commit (use "git add" and/or "git commit -a")

OMEN@DESKTOP-0LI898T MINGW64 ~/colours (master)

$ git diff

diff --git a/colors.txt b/colors.txt

index 853f2fd..5a75b9c 100644

--- a/colors.txt

+++ b/colors.txt

@@ -3,4 +3,4 @@ orange

yellow

green

blue

-purple

\ No newline at end of file

+indigo

\ No newline at end of file

Git diff HEAD komutu

Bu komut tüm yapılan değişiklikleri (tüm dosyalarda) gösterir, aşalandırılmış ya da aşalandırılmamış tüm değişiklikleri de gösterir.

Git add colors.txt

Git diff – boştur

Git diff HEAD:

-purple

+violet

Numbers.txt:

+ONE

Yani commit edilmeden önceki add kısmında farklı dosyalar olsa da gösterir

Git diff –staged (git add den sonra aşalandırılmış dosyaları gösterir)

Git diff –cached

Git diff HEAD style/main.css (İstenilen dosyaya gider ve oradaki değişiklikleri gösterir)

Git diff branch1..branch2

Branchler arasında karşılaştırma yapar.(stagelenmiş durum) Örneğin birinci branchden ikinci branche geçerken çıkarılanları ve eklenenleri gösterir.

Git diff master..odd-numbers

ONE -TWO THREE -FOUR +FİVE

Git master..odd-numbers (iki nokta yerine boşlukda aynı işlevi görür)

OMEN@DESKTOP-0LI898T MSYS ~/colours (odd-numbers)

$ git log --oneline

21c4e3d (HEAD -> odd-numbers) Odd numbers colors

5a44254 add Five

7508a80 remove two and four

e9999a4 (master) new things

454dcfd add two

38e1a3d Add One

2057a5d add violet

1c476ce Add numbers file

63555a1 Add indigo and remove purple

ca80a46 add blue and purple

f0b907a add orange, yellow and green

c9496d6 add red

\*\*\*$ git diff 63555a1..f0b907a\*\*\* ( iki commit arası karşılaştırma )

GİT STASH

Bu yapı özellikle branchlar arasında geçiş yaparken , yapılan değişikliklerin aynı branch da kalmasını sağlar. Birden fazla branchda çalışırken , bir branch da işimiz yarım kaldığında git stash kullanarak onu saklarız ve diğer brancha geçiş yaparız. [Commit yapmak istemediğimiz durumlarda]

Git switch ekip

(1-2 dosya da değişiklik yaptım fakat commit etmeyeceğim henüz tamamlanmadığı için)

Git stash (dosyalardaki değişikliği saklar)

Git switch master

(burada işimi hallettim)

Git switch ekip

Git stash pop [değişiklikleri geri getirir.]

Git stash dan sonra diğer bir brancha geçip

Git stash apply (bu kod ile sakladığım kodları istediğimiz branchda uygularız ayrıca apply da tek kullanımlık değildir (pop gibi değil) bunun yerine tüm branchlarda kullanılabilir)

Çoklu değişiklikler yapabilirz çeşitli.

Bu durumda git stash list kullanırız ve

MEN@DESKTOP-0LI898T MINGW64 ~/pictures/git-stash-exercise (master)

$ git stash list

stash@{0}: WIP on master: 718d6d5 new codes

stash@{1}: WIP on master: 718d6d5 new codes

stash@{2}: WIP on master: f5b4ab4 new codes added

OMEN@DESKTOP-0LI898T MINGW64 ~/pictures/git-stash-exercise (master)

$ git stash apply stash@{0}

Yaparız.

Git stash drop [listedeki stashleri kaldırır]

Uygulanan stash@{0} kaldıralım artık gerek yok.

Git stash drop stash@{0}

Ve tüm listeyi temizleyen;

Git stash clear

**DETEACHED HEAD**

İstediğimiz zamanda geriye ve ileriye gitmeye yarar. Aynı zamanda ulaşılan bi noktadan sonra geriye dönüp yeni bir repo oluşturulabilir ve birtakım değişiklikler yapılabilir.

Git status

$ git log --oneline

cc60893 (HEAD -> master) add headings to all files

b7c29b7 add new file to two images

22de7a4 add Hasan characters list

ccbd098 I change "?" to " ' " fix: bazi küçük yazim hatalari düzeltildi

b6db59f fix the typo in outline

3ec6935 create mood board

e21adf4 rename three to four

bf62611 finished chapter1

29cecbf begin to work on chapter1

113eb06 Start work and meeting characters

Örneğin chapter1 bitirdiğim kısma dönüp yeni bir repo oluşturacağım ve chapter1i düzenleyeceğim.

Git checkout bf62611

Git switch -c chapter2-repo

(dosyaları tekrar yazdım)

Git add .

Git commit -m”—“

(artık yeni bir repoya sahibim) geçmişe gittim.

Git switch master (eski repoma geri döndüm)

Git checkout HEAD~1 (1 commit geriye gider ve başlıklar yok olur)

Git switch – ( en son brancha gider örn: master , git switch master ile aynı sey)

Git checkout HEAD~8 (biraz geriye gittik 8 commit geri)

Git switch -(mastera döndük değişiklik yok)

Git checkout HEAD dog.txt

Git checkout HEAD cat.txt

Son yapılan değişikleri commit edilmemişse geri alır , en son haline getirir. [çoklu dosyalarda tek tek kaldırmak yerine tüm dosyadaki değişikliği kaldırır]

Git restore dog.txt (tümüyle en başa döndürür son commite)

Git restore –source HEAD~2 dog.txt cat.txt (2 commit aşağı döndürür fakat commit etmen gerek değişiklik için)

Git restore staged—secrets.txt [örneğin 3 dosyamız var yanlışlıkla add . eklemesi yaptık ve birini commit etmeden çıkartmak istiyorsak bunu kullanacağız.]

Yanlışlıkla birkaç commit üst üste yaptın bunu nasıl düzeltebiliriz? (reset işlemi ile)

$ git log --oneline

7147656 (HEAD -> master) mistake commit

19735fc Fourth commit

9573879 new file

dc2dd04 fixing

6a67c6e third commit

f521d50 second commit

31f8814 First commit

93e7e3a new files

Git reset 19735fc [en son kısmı alıyorum]

\*\*commit durumundan çıkarır fakat değişiklikler hala duruyor

Bunun için yeni bir branch oluşturalım

Git switch -c badstuff

Git add .

Git commit -m”bad things”

Git switch master

(böylece fourth commitde duracaktır)

Git reset –hard 6a67c6e [direkt üçüncü commite döner.]

Tüm değişiklikler gitti.

Git Revert komutu

Yapılan değişiklikleri geri alır ve yeni bir commit oluşturur aynı commit isminde revert edilmiş (düzenlenmiş) değişikliklerdir.

Örneğin bad commit oluşturalım

a982db6 (HEAD -> master) make a bad commit

git revert a982db6

4085f1c (HEAD -> master) Revert "make a bad commit"

a982db6 make a bad commit

**GİTHUB BASİCS**

Git Clone

Bir repoda olmadığından emin ol.

URL Kopyala (Github) ve git clone <URL> ile yapıştır. Paylaşılan tüm proje bilgisayarında olacak.

Git Remote

Githubdan kopyaladığın projeleri bu remote işlevi ile görebilirsin.

Git remote

\*Origin

Git remote -v

\*Origin <URL>

Aynı zamanda önceden gitte yazdığın bir projeyi , githuba yüklemek istersen githubda ayrı bir depo açtıktan HTTPS kısmından kopyaladığın url ile onu da git ile birleştirebilir ve yükleyebilirsin.

NOT: Origin ismi genel bir isimdir. Genellikle söylenir fakat değiştirebilirsin örneğin myfirstDepo olabilir

git remote add origin <https://github.com/AlaattinUysal/github-demo-novel.git>

git remote remove <name>

Git Push

Git push <name> <branch>

\*Hali hazırda git de bir repom var ve bunu github’a aktarmak istiyorum. İşte burada git push devreye giriyor. Bu eklentiyi kullanmadan git de yaptığım komutlar , githuba aktarılmayacak.

OMEN@DESKTOP-0LI898T MINGW64 ~/Pictures/resimler/myfirstnovel (master) [sayfasında olduğumuz düşünelim]

Git push origin master

Bu bizi githubdaki sayfamıza yönlendirecek ve başka bir branch daha ekleyebiliriz.

Git push origin empty

İkinci branchı yükledik. Böylece branchları çeşitlendirdik. Şimdi Githubdan istediğimiz duruma bakabiliriz.

Git remote -v

origin https://github.com/AlaattinUysal/github-demo-novel.git (fetch)

Sayfaya gidip kontrol sağlayabiliriz.

\*\*\*NOT!: Tekrar değişiklik yaparsak bunu da push etmemiz gerekir.

--NEW FEATURE

git push origin cats:master

git push <remote>

<local-branch> : <remote-branch>

\*Cats branchındekileri mastera iter. Cats branchı degismez fakat master branchı cats dekileri de kapsar.

**Git push -u**

Git push -u origin master ,, yazdıktan sonra olusturacagımız tüm dosyalarda sadece

Git push yazmamız yeterlidir.

Start from scratch

Henüz bir repon yoksa ve yeni oluşturacaksan ve bu projeyi githuba atacağını biliyorsan , githubda yeni bir repo oluşturursun ve bunu klonlayarak gite aktarırsın daha sonrasında burada yapacağın değişiklikleri git push origin <name> özelliği ile githuba aktarabilirsin.

Dikkat bunun için önemli olan iki tarafta da henüz bir oluşum olmaması her şeyin baştan başlamışıdır.

Git Merge

Git switch main

Git merge foods [Foods dosyasının içindekiler main dosyasının içine aktarıldı burada aynı dosya kullanıldı] 🡪 ileride daha detaylı işlenebilir.

Git Fetch

Git Fetch ; Githubda bi ekiple çalıştığında ekip herhangi bir değişiklik yaptığında sen o değişikliği git kısmına yüklediğinde sadece local depon etkileniyor. Böylece Genel depodaki master hala aynı kalıyor. Örneğin food ve game sayfası olsun genel, burayı etkilemez ama senin local değişkenindeki origin/food ve origin/game kısmında değişiklik meydana gelecektir.

metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu görsel açıklamak için yeterli. Değişikliği github algılıyor ve sen git fetch yaptığında local deponda değişiklikler meydana geliyor fakat uzak depo etkilenmiyor.

Git fetch origin

Or git fetch [Githubdaki değişikliği gite taşımak için]

Git Pull

Uzak depoyuda günceller.

Git pull origin (food)

Git branch / Git switch food [güncellendi]

metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bazı durumlarda çakışma olabilir. Örneğin Semih isminde arkadaşın ile aynı dosyayı yüklediniz fakat senin bundan haberin yoktu bu durumda merge edilecektir. Git pull merge conflicts olur.

Unutma github🡪git [Local depo – git fetch / Genel depo – git pull] Git -> github [git push origin] metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

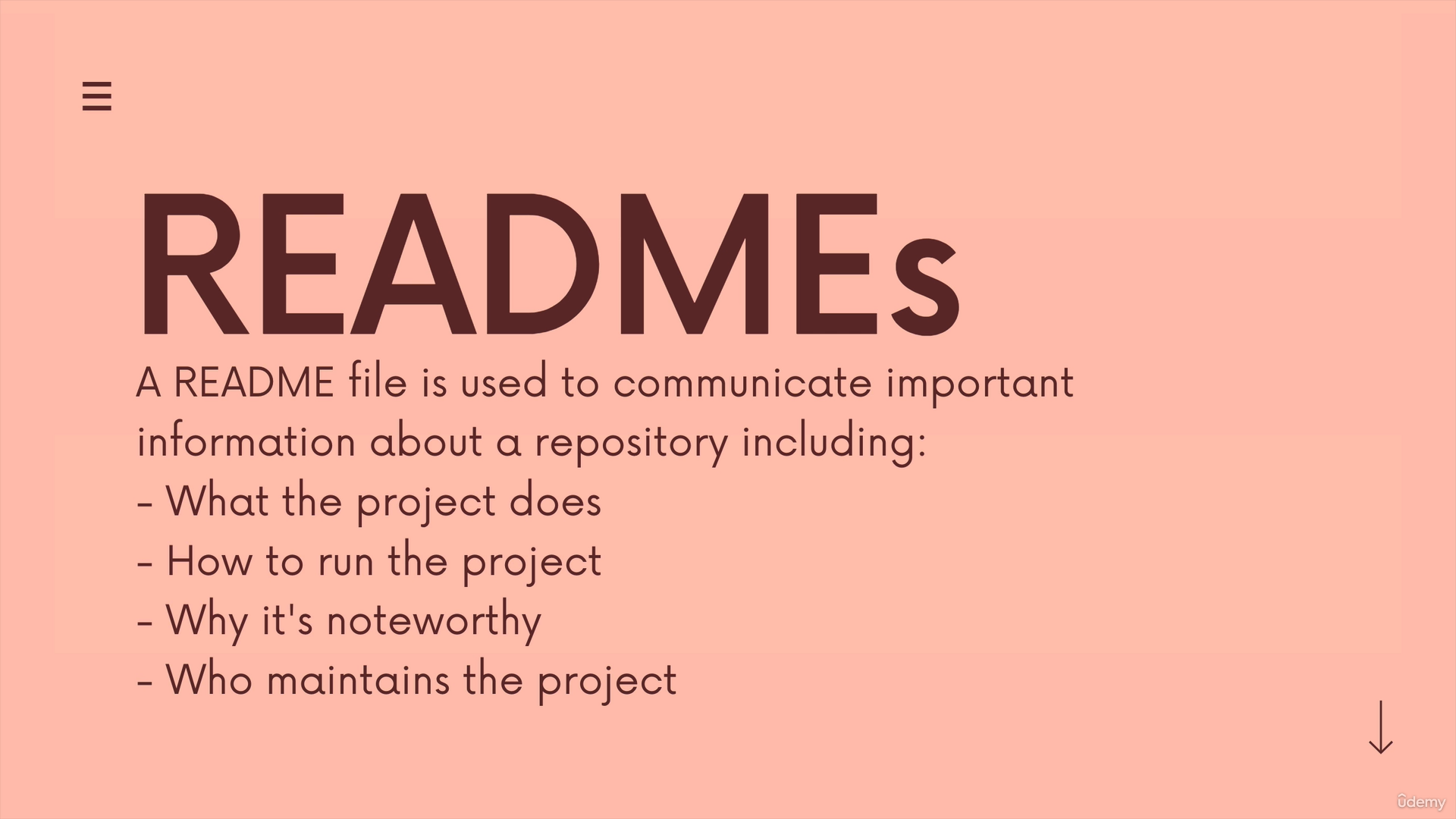
Açıklama otomatik olarak oluşturuldu

Eğer o dalda isen ve sadece originden oluşmuş dosyaların varsa yani food veya game dalındasın veya sadece maindesin. O zaman git pull origin food yazmak yerine git pull yazsanda aynı işlemi yapmış olacaksın.

metin, yazı tipi, ekran görüntüsü, doküman, belge içeren bir resim

Açıklama otomatik olarak oluşturuldu

Readme about that :



<https://markdown-it.github.io/>

<https://daringfireball.net/projects/markdown/>

Centralized Workflow Demonstration

Git collaboration yaparken bazı durumlarda git pull izin verilmez (bunun için değişiklikleri saklamak gerekir yani git stash yapmamız lazım)

İlk önce karşı taraf git de değişiklik yapmış mı bunu bilmemiz gerekir bu yüzden git push yapmadan önce daima git pull yapılmalı eğer o kadar master üzerinde calıstık ve karşı taraf da git üzerinde calıstıysa

Bu durumda merge edilir ya da failure olur. Çünkü tek bir ana dal üzerinde çalışıyoruz.

Git rebase vs Git merge

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Vs

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Git rebase 🡪 \*Tek bir satır halinde düzenliyor ve commit mesajları daha rahat görmemizi sağlıyor.

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tekrar commit eklendiğinde ve değişiklik yapıldığında durum değişir.

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Rebase tarihi yeniden yazmak ve güncellemektir. Her zaman son tarihten başlar ve geçmişi tamamen o tarihten itibaren alır. Bu yüzden master/main dalında rebase gerçekten tehlikelidir. Eğer kendi dalında işlem yaparken kullanıyorsan işlevsel olabilir ama master (özellikle collobaration kısmında) kullanmamak gerekir. WARNING!

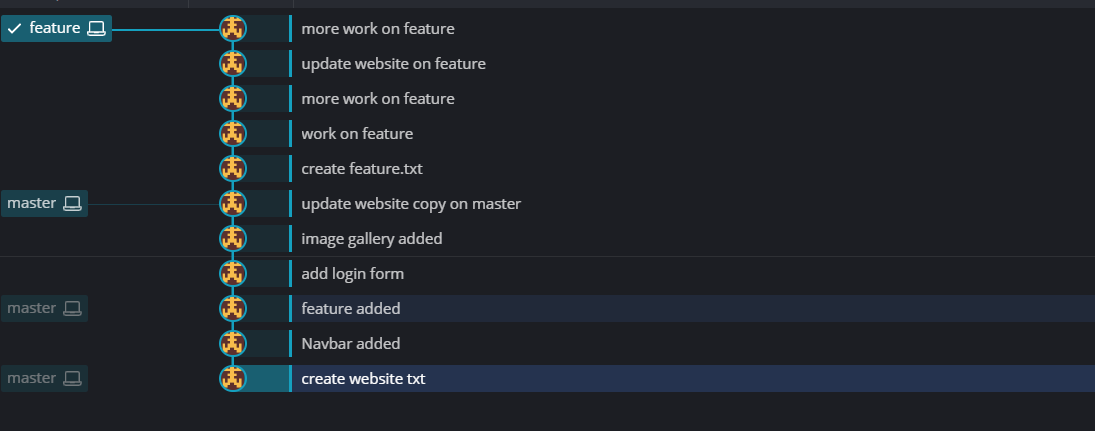
Git rebase –abort (Değişiklikleri iptal et eğer merge etmeyeceksen)

// eğer değişiklikleri değiştireceksen yani iki değişikliği kabul edeceksen:

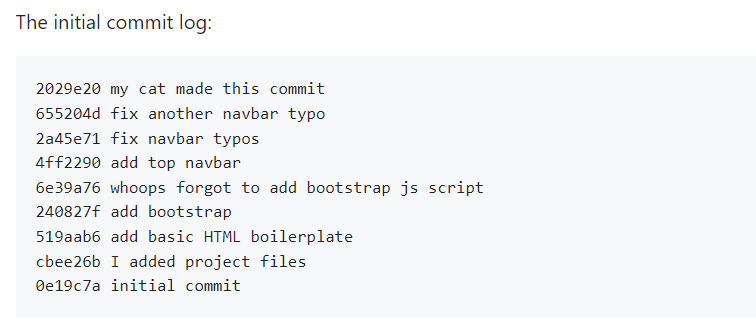
Git status

Git add website.txt

Git rebase –continue



Git rebase -i HEAD~n (kaç komut geriye gitmek istiyorsan)



Reword -> commit yeniden yazar.

Fixup -> commit önceki commite eklenir (içerik değişmez)

Drop –> commit silinir , içerik silinir.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Git Tag

Git tag (Tüm yapılan tagleri gösterir) 1.0.0 -> 1.0.1 ( yama değişikliği – Çok küçük , userlara hissettirilmez.)

* 1. -> Küçük değişiklikler (Fakat kullanıcı deneyimi iyileştirilir)

2.0.0 -> Büyük Çapta değişiklikler.

Git tag -l “\*v17\*” 🡪 Direkt v17 değişikliklerini gösterir.

Git checkout 15.3.1 🡪 Bu tag’e gidip değişikler yapabiliriz.

Git diff v17.0.0 v17.0.1 🡪 İki tag arasındaki farklılıkları ölçebiliriz.

Yeni değişikliklerden sonra tag eklemek istediğimizde misal:

Git tag v18.3.3 şeklinde ekleriz.

Açıklamalı tag nasıl yapılır

Git tag -a “v18.4.0” [metin düzenleyicide mesajı yazarsın]

Git show v18.4.0 🡪 Detaylı bir şekilde tag gösterilir , mesajlar , commitler , düzenlemeler vs.

Git tag mytage(isimlendirme) aa8469fefb 🡪 Commit seçtiğinde ve commiti yapıştırdığında ona bir isimlendirme yapabilirsin.

Git tag v18.3.3 8a70d31ba9 -f 🡪 Tag yerini değiştirir.

Git tag -d 18.3.1 🡪 Tag silinir geriye sadece değişiklik kalır.

git push alaaddin mytage 🡪 1 tag githuba itmek

git push alaaddin –tags 🡪 Tüm tagleri itmek

Git Config

Local depoda çeşitli değişiklikler yapabilirsin

ls -a

cd .git

code .

config dosyası ulaşım

<https://git-scm.com/docs/git-config>

Object file |

Tüm geçmişin barındığı fakat bizim dahi açamadığımız git file.

Ref / HEAD

Echo “hi” | git hash-object —stdin

40 basamaklı SHA HASH verir ve sabittir. Eğer böyle bir şeyi saklamak isteseydim hangi anahtarı verirdi? Fakat asla bunu depolamaz

Echo “hi” | git hash-object —stdin -w

Dosyayı oluşturur ve object de saklar.

<https://git-scm.com/docs/git-hash-object>

Genel bir özet :

**commit** Çalışma ağacınızın belirli bir zamanda anlık görüntüsünü   
 temsil eden nesne (ayrıca o anlık görüntüyü oluşturan   
 işlem)  
**blob** Bir dosyayı temsil eden nesne (özellikle içeriği)   
**tree** Bir dizini temsil eden nesne (özellikle içeriği)  
**tag** Belirli bir nesneyi işaretler  
**lightweight tag** Değişmez bir dal gibi, sadece bir dosya (yani   
 ilişkili meta veri yok)  
**annotated tag** Etiket nesnesi  
**branch** Geliştirme hattı   
**master** Konvansiyonel olarak, ana geliştirme hattının (branch)   
 adı  
**HEAD** An itibarıyla çalışılan branşı veya commit'i gösteren işaretçi   
 $ cat .git/HEAD  
 ref: refs/heads/main  
**fetch** Uzak sunucunun kod havuzunda 'origin' dalından   
 (belirtilmişse) tüm değişiklikleri alır ve bunları yerel   
 olarak kullanılabilir hale getirir, ancak değişiklikleri   
 uygulamadan önce değerlendirmenize izin vermek için   
 bunları çalışma ağacına yerleştirmez.  
 *$ git fetch origin master***merge** İki branşı veya uzak sunucudaki kodları kendi   
 branşınızla birleştirmek için kullanırsınız. Uzak   
 sunucudan ***fetch*** ile getirdiğiniz değişikliklerden   
 memnunsanız, bunları ***merge*** ile birleştirebilirsiniz.  
 *$ git merge origin/master***pull** Uzak sunucudaki değişikliklerin çalışma dizininize hemen   
 uygulanması konusunda rahatsanız (fetch ve ardından   
 merge işlemlerini yürütmek yerine), çekme işlemini   
 kullanabilirsiniz.  
 *$ git pull origin master***push** Yerelde yaptığınız değişiklikleri uzak sunucuya   
 göndermek için push komutunu kullanıyoruz. Çalıştığınız   
 branştan commit'lerinizi uzak sunucuya göndermek   
 istediğinizde uzak sunucuyu etiketlediğimiz ayarın url   
 bilgisine göndeririz. Aşağıdaki ekran görüntüsünde   
 "main" branşında çalışırken *$ git push* komutunu   
 çalıştırdığımızda "remote = origin" bilgisine bakara   
 "origin" etiketli uzak sunucu bilgilerine bakar. "  
 [remote origin]" içindeki url/pushurl alanındaki adrese   
 gönderir. Ayrıca uzak sunucu etiketi ve branş adı  
 *$ git push origin main*

**remote** Bir depodan klonlamak istiyor, ancak başka bir depoya   
 göndermek istiyorsanız, yapılandırmanıza kodu   
 göndereceğiniz diğer uzak sunucuyu ***remote add*** ile   
 ekleyebilirsiniz. Bir uzak deponuz varsa, "origin"   
 varsayılan etiketidir. Başka bir uzak kod havuzuna kodu   
 göndermek istediğiniz durumda, uzak kod havuzun adını   
 push komutuna parametre olarak açıkça belirtmeniz   
 gerekir.  
 *$ git remote add diger\_repo\_etiketi ssh://*[*hede@hodo.net*](mailto:alice@gate.collabnet.medienstadt.net)*/proje*  
 *$ git push diger\_repo\_etiketi diger\_reponun\_brans\_adi*checkout Bulunduğunuz branştan başka bir branşa geçmek istediğimizde kullanırız. Ayrıca uzak sunucudaki bir branşı yerelde farklı bir branş adıyla oluşturmak istediğimizde kullanırız.  
 *$ git checkout -b yeni origin/cem*

git cat-file -t 79dd….(hash)

be

commit

git cat-file -p 79dd.. (hash)

be

tree 12323543245eeddf4343fdd….

Author

Committer

Bir commit 🡪 tree ve bloblardan oluşur.

Bir tree 🡪 bloblardan oluşur.

Blob –> içerik , tree -> dizin belirtir. Örneğin o dizinin içinde iki tane dosya olabilir. Cats.txt , dogs.txt olduğunu düşünürsek bunların herbiri birer blobdur. Yani içerikleri vardır. İçerik değişmediği sürece hashler değişmez,sabittir. Fakat içerik değişirse blob hash de değişir.