git status ile ilk önce herhangi bir git init oluşturulmuş diye bakalım.

git log ile bulunduğumuz branchin ilgili kayıtları gelecektir. Burada HEAD ile hangi branch’te olduğumuzu görürüz. HEAD bize hangi branch’te olduğumuzu ve o branch’in en son commit edilmiş versiyonunda olduğunu gösterir.

git branch, bize güncel branch’lerimiz gösterecektir. Hangi branch’te olduğumuzu görür ve başka hangi branch’lerin var olduğunu görürüz. git branch branch2 ile de yeni bir branch oluşturmuş oluyoruz. git switch branch2 ile de başka branch’lere geçiş yapıyoruz. Bu aşamada git log yaparsak HEAD ‘in branch2’yi gösterdiğini görürüz... daha sonra merge işlemi yapmak istediğimiz branch’e gidip, git merge branch2 yazıp, master branch’i ile branch2 branch’ini merge’lemiş oluyoruz.

Eğer ki yeni branchimizde herhangi bir değişiklik yaparsak ve bu esnada master branch’imiz için herhangi bir değişiklik yapmazsak, sanki master branch’inde devam ediyormuşuz gibi olacaktır. Bu durumda en son merge işlemi yaptığımızda ekranda “Fast Forwarding” yazacaktır. Bu işleme verilen addır. Mülakatlarda sorulabilecek bir sorudur.

Peki iki farklı branchte yapılan değişiklikler çakışırsa ne olacak? Conflict !!

Diğer branch’te varolan bir dosyayı sildik veya değiştirdik. Daha sonra master branch’inde de böyle bir işlem yaptık. “Tabii ki bu yapılan işlemleri hepsi commitlenmiş işlemlerdir”. İşte bu merge işleminde conflict çıkacaktır. Burada yapılacak birkaç şey vardır. Conflicte sebep olan commitleri düzeltmek. Yani, sildiğimiz veya değiştirdiğimiz şeylerin ne olacağına karar vermek. Ondan sonra bu dosyaları tekrar add-commit yapmamız lazım. En sonunda conflict i giderirsek sorun çözülecektir. Merge işlemini halledecektir.

Peki ya branchlerde ki değişklikleri commit etmeden branch değiştirir ve öyle merge yapmaya çalışırsak? !!

Yan branchte bir dosyanın içerisinde herhangi bir değişiklik yaptık ve commitlemeden master branchine geçtik. Daha sonra bu dosyanın içerisinde master branchine ait daha başka değişiklikler yaptık ve bu değişklikleri commit ettik. Sonra tekrar yan branche geçtik fakat en son yapıtğımız değişiklikler yok. Çünkü commit etmeden master branchine geçmiş olduk. Data kayboldu. 1- commit etmek, 2- bu son yaptığımız değişikliklerden vazgeçmek ve git restore file-name ile en son ki commit edilmiş haline geri almak, 3- stash yapmak.

Yan branchin içerisindeyiz ve birtakım değişklikler yaptık. Ama bunları kaydedip-commitlemek istemiyoruz. Ve bu dataların kaybolmasını da istemiyoruz. İşte bu durumda stash komutunu kullanıyoruz. git stash ile bu commitlemediğimiz değişiklikler özel bi yerde bizim için saklanıyor. Working Directory – Staging Area – Commited – Stash Area diyebiliriz.. Bundan sonra kalan işlerimize devam edebiliriz. En sonunda tekrar stash’e attığımız datayı geri getirmek istersek git stash pop dediğimiz de datamız geri gelecektir. Güncel stash listemizi öğrenmek istersek, git stash list diyoruz. Peki birden fazla stash datamız var ve biz hepsini geri çağırmak istemiyoruz. Bu durumda git stash list deyip hangi stash’i geri getireceğimizi listeden sırasını seçiyoruz. Stash numarasını alıp, git stash appyl **stash@{3}** deyip geri getiriyoruz. Burada kalın yazılmış stash yazısı stash listte yazan sırasını göstermektedir. Peki bütün stashleri silmek istersek, git stash clear diyoruz.

Master branchinde olduğumuzu varsayalım. Ve birkaç önceki commite dönmek istedik. git log diyerek git kayıtlarına bakıyoruz. Buradan dönmek istediğimiz commit seviyesinin SHA numarasını alıyoruz. Daha sonra git checkout SHA diyerek, SHA numarasını aldığımız commite gitmiş oluyoruz. artık HEAD bu commiti göstermiş oluyor. Yani güncel olarak bulunduğumuz yeri işaret ediyor. Buradan yeni bir branch oluşturabiliriz. Veya farklı bir değişikliler yapıp tekrar master branch’imizin eski son uzantısına gidebiliriz.

git diff dersek wd ile repo arasındaki farklılıkları görürüz. git diff file\_name dersek dosya özelinde ki değişiklikleri görürüz. git diff - -staged dersek stage ile repo arasında ki farkı görürüz. pek iki commit arasındaki farkları görmek istersek, git diff SHA1 SHA2, yazarsak iki commit arasında ki farkları görürüz. SHA1-2 ise commitin kendine ait numarasını gösteriyor. Aralarında da boşluk vardır. Boşluğu kabul etmezse .. koyabiliriz.(iki nokta). git diff master branch2 dersek de iki branch arasında ki farkları göstermiş oluyor.

Peki branch’in daha önceki commitlerine geri dönmek istersek ve son yaptığımız commitleri artık istemezsek; git reset SHA dersek, dönmek istediğimiz commitin SHA numarasını yazıyoruz ve artık o commitimize dönmüş oluyoruz. git log dersek istemetiğimiz commitlerin kaybolduğunu görürüz. Ama dosyalarda ki değişiklikler hala gözükür-sorun değil, silinebilirler.

git reset –hard SHA bu ise, commitleri hem siliyor hem de commitlerin değişikliklerini de siliyor. Böylelikle son yapılan hiçbir değişiklik artık gözükmüyor.

git revert SHA dersek, örneğin master branchindeyiz ve 6.committeyiz. ama 4.commite geri dönmek istiyoruz, bunu yaparken de arada ki commitlerde yaptığımız değişiklikleri istemiyoruz. Ve bu commitlerin tamamen silinmesini de istemiyoruz. İşte bu durumda git revert SHA diyoruz. Değişikliklerini geri almak istediğimiz commitin SHA numarasını yazıyoruz. Mesela son yaptığımız commitin değişikliklerini istemiyoruz ve ondan bir önceki committen devam etmek istiyoruz. İşte burada ki SHA’a bu geri almak istediğmiz commitin SHA numarasını yazıyoruz. Böylece devam etmek istediğimiz committen HEAD olarak devam etmiş oluyoruz hem de git log da bu geri aldığımız-istemediğimiz- commitlerin de kayıtlarını tutmuş oluyoruz.

Branch2’de çalışıyoruz ve kendi commitlerimizi yazıyoruz. Daha sonra master branch’inde de yapılmış commitler olduğunu görüyoruz. Bu commitler bizimle çakışmadığı için onları git merge master diyerek, master’da yapılan yeni commitleri almış oluyoruz ve kendi işimize devam ediyoruz. Yani kendi branchimizde olup merger-master diyerek master branch’inde olan değişiklikleri kendi branchimize dahil etmiş oluyoruz. Eğer ki bu şekilde devam ediyorsak, kendi branchimizde çalışıp master branch’inde olan değişiklikleri alıp bu şekilde ilerliyorsak ve bizim branchimizden herhangi bir data alınmıyorsa, git rebase master yapabiliriz. Eğer ki bu şartlar var ve rebase yaparsak, git log umuz şu şekilde olacaktır. En başta masterdan merge yapıtğımız commitler sırasıyla yazacaktır. Daha sonra da bizim kendi branch’imizde yaptığımız commitler yazacaktır. Kronolojik sırayı piç ediyoruz ama bu durumda bir sorun olmayacaktır. Çünkü başka kimse bizim branchimize dahil olup merge işlemi yapmıyor. Bu yüzden sorun olmayabilir. Ancak başkaları da bizim branchimize gelip merge alıyorsa işte o zaman fatal-error olacaktır. Bu durumda git rebase master önerilmiyor.

git branch -r(remote) ile remote repoda olan branchleri görebiliriz. Ancak bunlar arasında geçiş yapmak istersek git switch branch2 işe yaramayacaktır. Çünkü bu yöntem localdeki branchler arasında geçiş yapmamızı sağlar. Remote branchler arasında geçiş yapmak istersek git checkout origin/master yapmamız gerekiyor.

git fetch origin master dediğimizde, local repomuza değişiklikleri getiriyor fakat kendi localimizde ki dosyaları değiştirmiyor. Böylelikle fetch’den gelen değişiklikleri görüp-inceleyip daha sonrasında hangi işlemi yapacağımıza karar veriyoruz. Yani dosyalarımızı merge-commit etmeden önce incelemize izin veriyor.

git pull origin master ise, fetch benzeri fakat olan değişiklikleri direkt kendi branchimize yansıtıyor. İnceleme yapmadan, çakışma olup olmadığına bakmadan yapılan değişiklikleri kendi branchimize yansıtmamızı sağlıyor.

Peki ya projesini beğendiğimiz bir repoyu nasıl kendi localimize dahil edip üzerinde çalışabiliriz. git clone URL dersek, istediğimiz repoyu kendi bilgisayarımıza kurmuş oluyoruz. Ama bu şekilde yaparsak başka bir hesapta bulunan repodaki projeye müdahil olmak için izin almak gerekiyor. Sonuçta başka bir hesapta bulunan bir repodaki proje üzerinde çalışabilmek için izin vermek gerekir. Buna gerek kalmadan ayrı olarak bu projeyi kendi repomuzda çalıştırmak ve geliştirmek için github’da fork kısmında, o projeyi fork lamamız lazım. Böylece bu proje bizim kendi repomuza dahil olmuş oluyor. Buradan bu projeyi istediğimiz gibi kontrol edebiliriz. İşte bu durumda ayrı bir fork olarak projeyi geliştirebiliriz. Eğer ki bu geliştirdiğimiz yeni özellikleri forklerdiğimiz kullanıcıya da “ben bu proje şöyle bir özellik ekledim, isterseniz siz de ana projeye dahil edebilirsiniz.” Şeklinde bir mesajla söyleyebiliriz. Bunun için git sayfamızda bu projeye gelip, pull request oluşturup, mesajla bereaber istek atabilliriz. Daha sonra bu pull isteği ve mesaj o kullanıcıya ulaşacaktır. Bu aşama da 3 seçenek vardır. İlki reddetmek, ikincisi bir comment ile “la gardaş şurası olmamış, değiştir ve bir daha istek yolla”, üçüncüsü ise yine bir mesajla(default tabii ki) pull isteğini kabul etmek.

Private bir repoda işlem yapdığımız zaman, işte onaylar cart curttan sonra, projeye dahil olup, ayrı bir branch açıp orada bir şeyler yaptıktan sonra bunları add-commit-push yaparız. Projenin admini, bu repoya bakıp, pull requestlere bakıp bizim yaptığımız değişiklikleri pull isteği olarak görür. Ve onları oluşturduğumuz yeni branchten, git pull origin newbranch2 ile alıp yaptığımız değişikliklere bakıp ona göre değerlendirir. Onaylarsa projeye commit eder, master branch i ile merge etmiş oluruz.

**0 –**

**Git terminaline girdiğimizde ilk olarak (git kurduktan sonraki ilk işlemde) kendimizi tanıtmak için, user-name ve user-email i yazarız.**

git config –global user.name “hasan bala”

git config –global user.email [balabala@gmail.com](mailto:balabala@gmail.com)

git config –global user.name

**Bash kısayolları**

+ pwd komutu bize tam olarak hangi konumda olduğumuzu gösteriyor. Hangi directory’yiz.

pwd=> home/balah

**ls komutu dizinde varolan tüm dosyaları bize gösterectir.**

ls -a(gizli dosyalar dahil tüm içeriği gösterir)

ls -l (dizinin tüm içeriğini uzun formatta ve dosya izinlerini gösterir)

ls -t(dosya ve dizinleri, son değiştikleri tarihe göre sıralar)

**touch komutu ise yeni bir dosya oluşturmak için kullanılır**

touch keyboard.txt

touch file{1..4}.html (4 html dosyası)

**cat ile tek bir dosyanın içeriğini görüntüleyebiliyoruz**

cat Desktop/comedy/club.txt

**cp ise dosyaları kopyalamamızı sağlar**

cp source.txt destination.txt

**tüm dosyaları kopyalamamızı sağlar**

cp \* my\_directory/

**w ile başlayan tüm dosyaları kopyalar**

cp w\*.txt my\_directory/

**taşıma işlemi yapar ve yeniden adlandırır**

mv my\_file.txt my\_directory/

**dosyayı kaldırma işlemi yapar**

rm file.txt

**bu dizini ve onun da tüm alt dizinlerini siler**

rm -r file.txt

**1 -**

(git init) **=> working directory (git add) => staging area (git commit) => repository**

git init komutu ile aslında wd.’ye girmiş oluyoruz. Sonrasında yaptığımız tüm işlemler orada yapılmaktadır. git add komutu ile de wd. ‘de ki dosyalarımızı sa. ‘ya taşımız oluyoruz. git commit ile de sa. ‘da ki dosyalarımızı repoya taşımış oluyoruz.

**commit edilen dosyaların versiyonlarını gösterir**

git log

**Git projemizde ki değişklikleri görmek için;**

git status (yapılan dizin değişkiklerinin durumunu kontrol ederiz.

**-git status, commit edildikten sonra herhangi bir değişiklik olunca;**

git diff (wd ile repo arasındaki farklılıkları görürüz)

git diff file\_name (şeklinde dosya özelinde de değişiklikleri görebiliriz)

git diff - -staged(stage ile repo arasında ki farka bakıyoruz)

peki git sistemine göre dosyayı kaldırmak istersek;

git rm file\_name(şeklinde daha sağlıklı silmiş oluruz. Gerisi aynı)

peki dosya(klasör) silmek istersek; (recursive)

git rm -r file\_name(şeklinde siliyoruz. Gerisi yine aynı)

**git sistemi ile dosyada isim değişikliği ve taşıma yapmak için;**

git mv file\_name1.txt file\_name2.txt

(file\_name1.txt => file\_name2.txt şeklinde dönüştürmüş olduk. Sonrasında yine stage ve repo işlemlerini yapmamız lazım)

git mv file\_name1.txt files/

(burada da dosya taşıma işlemi yapıyoruz ve stage-repo işlemlerini yine aynen yapıyoruz)

**Git değişikliklerini geri alma işlevi; (git status yazınca, yardımcı yönerge çıkıyor)**

(dosyamızda değişiklik yaptık fakat (add .) lemedik. geri almak istersek, yani working directory’de bulunuyoruz.)

git checkout -- file\_name(böylelikle repoda ki son versiyonu tekrar dosyamıza geri alıyoruz. Geri yükleme işlemi tamamlanıyor)

(peki dosyamızı stage gönderdikten sonra geri almak için, yani add ledik.)

git reset HEAD file\_name(working directory ‘e dosyamızı tekrar geri gönderiyoruz. Buradan da eski haline döndürmek için yukarıda yaptığımız gibi. Staging area da ki dosyayı HEAD commit ile aynı olacak şekilde sıfırlar. Yalnızca staging area dan kaldırır yani. Working directory de tekrar düzenleyip tekrar commit edebiliriz) staging areadan wordking directory’ye geri aldık. Bundan sonra yukarıda yaptığımız gibi working directory den değişiklikleri geri almak istersek, hemen aşağıdaki satır gibi yapıyoruz.

git checkout - - file\_name( gerisini yine aynı şekilde yapıyoruz)

**working directory de yaptığımız en son değişikliği iptal eder**

git checkout HEAD file\_name

**en son yaptığımız değişikliği gösterir**

git show HEAD

**Dosyalarımızda ki versiyonlar arasında değişiklik yapmak istersek; git log bize versiyon durumlarını gösterecektir. Versiyon5 ‘te ise versiyon2 ‘ye geçmek istersek;**

git checkout SHA **- - .**

SHA, commit numarası oluyor, ya tamamını ya da ilk 7 hanesini SHA kısmına yazıyoruz. Böylelikle versiyon2 artık hangi versiyonu istersek, o versiyonda ki tüm hallerini getirmiş oluyoruz. .(nokta) ise tüm dosyaların geri getirileceği anlamına gelir. Ama bu değişikliklerin de commit edilmesi gerekir, yine -commit statüsünde commit ederiz.

**Github ile dosyamızı bağlamak istersek;**

Github ‘dan yeni bir repo oluşturup, https kısmını yani url sini alacağız, lazım olacak çünkü. Sonra localde ki dosyaya gelip (cmd den tabi ki) uzak bağlantı kuruyoruz. Git\_name i bizim kendi githupRepo ismimiz oluyor. Herhangi bir isim koyabiliyoruz. url de reponun url si.

git remote add git-name url **(-git remote ile kontrol de edebiliriz)**

git push -u git-name master

git push origin master

git pull origin master

username: ..

password:..

remote ile uzak bağlantıyı kuruyoruz. İsmini verip, url sini girip. Push ile localde ki

dosyalarımızı kurduğumuz bağlantı ile gönderiyoruz. -u demek, bütün dosyalarımız anlamına

geliyor. Master ise ana yönetim dosyamızı olduğu anlamına geliyor. Yan dalları da olabilir..

username ve password de giriyoruz. İşlem tamamdır. Peki ya değişiklik yaparsak;

**file5.txt dosyasını sildik, commit ettik ve bu değişikliği githuba master olarak push ettik.**

git rm file5.txt

git commit -m “file5.txt deleted”

git push -u git\_name master

**Takip edilmesini istemediğimiz dosyayı .gitignore kullanarak github da gizlemiş oluyoruz.**

cat >> .gitignore

databese.txt

^C

cat .gitignore

database.txt dosyasını paylaşmak istemiyoruz. Bunun için **cat >> .gitignore** komutunu yazıp hangi dosyayı saklamak istiyorsak onun ismini yazıyoruz. Daha sonra ^C ile de satırdan çıkıyoruz. -cat ignore ile de kontrol ediyoruz işlemi.

gedit .gitignore(bu şekilde ignoreladığımız dosyanın içerisine girmiş oluyoruz. Ve burada şöyle bir işlem yapabiliriz.

belgeler/\*

!belgeler/file1.html

(belgeler klasörünün de takip edilmesini istemedik ama içerisinde ki file1.html dosyasının takip edilmesini istedik. Burada bu işleri yapmış olduk) daha sonra yine add, commit ve push işlemlerini yapınca repoya kaydetmiş oluyoruz.

**Github da ki diğer dallar ile master dallar arasında ki bağlantıyı kurmak için;**

Github üzerinden farklı bir dal açıp o dal içerisinde işlemler yapıp daha sonra master dalı ile merge edebiliriz. Peki bunu locale nasıl çekeceğiz.

git pull(komutu ile github da olan son değişiklikleri localde ki dosyamıza çekebiliyoruz)

git branch, projedeki mevcut dalları gösterir

git branch - -all, uzak git deposundaki bağlantıları, dalları gösterir

git branch yandal2, yandal2 adında başka bir dal oluşturmamızı sağlar

git switch yandal2, komutu yandal2 dalına geçmemizi sağlar

yandal2 de değişiklikler yaptıysak ve bunu master ile senkronize etmek istersek;

git add . & git commit -m “....” (yandal2 de ki değişikleri repoda güncellemiş olduk)

-git switch master, master branch ine geçmiş olduk

**-git diff master yandal2**, master ve yandal2 arasındaki farklılıkları gösterir

**-git merge yandal2**, yandal2 ile master ı merge eder

git push -u git\_name master, github a push edecektir

**branch silme işlemi**

git branch -d branch\_name

git clone’ları ve remote durumları

**comp.name: ~$ git clone remote\_location clone\_name** (remote\_location: filepath ya da web adresste olabilir. Yeni bir isim vererek clone oluşturmuş oluyoruz)

**comp.name: ~$ git remote -v** (uzak bağlantı ile tüm git projelerini görebiliriz)

**comp.name: ~$ git fetch** (remote bağlantıda değişiklik yapılıp yapılmadığını görmenin ve değişiklikleri yerel kopyamıza indirmenin kolay yoludur)

**reset =** soft-mix-hard;

git reset –soft SHA(first 7 int) => yumuşak bir resetleme olur. Yine sonraki commitlerin hepsi gider. Resetlediğimiz committeki yaptığımız işlemleri staging area ya taşır.

git reset –mix SHA(first 7 int) => mix bir resetleme olur. Yine sonraki commitlerin hepsi gider. Resetlediğimiz committeki yaptığımız işlemleri working directory ya taşır.

git reset –hard SHA(first 7 int) => sonraki tüm commitler silinir. Kesin bir dönüş yapmaktadır.ne commit ne de add lenecek bir dosya kalır. Resetlediğimiz commitin ilk haline gelir.

revert

git revert SHA(first 7 int) => o committeki yaptığımız değişklikleri geri almak, default haline döndürmek (ayrı bir sayfada ne olduğunu detaylı bir şekilde açıklar. Orada commit(-m “blablabal”) gibi yorum da yapabilmemize imkan sağlar. (esc ye basıp, :q) şeklinde çıkış yaparız.böylelikle seçtiğimiz bir committeki değişkileri geri almış olduk.

git commit –amend ( yaptığımız yorumları değiştirmemizi sağlar )

checkout

branch ooluşturmanın ; git branch name ; şeklinde bir yolu var fakat bu branch in içerisine girmek için tekrardan ; git checkout name; dememiz gerekir. Bunun biraz daha kısa yolu vardır, şöyle ki ; git checkout -b namexx ; şeklindedir. Aynı zamandan bu bize o branch in içerisine de girmemizi sağlayacaktır.

Eğer ki merge edilmemiş branch i silmeye çalışırsak ; git branch -d name ; hata verecektir. Çünkü merge edilmemiştir. Bunun yerine silmeye zorlamak için ; git branch -D name ; yazarsak silecektir.(master a gelip, git merge name, şeklinde merge yapabiliyoruz)

fetch-pull diffence

git fetch origin; diyerek uzak repo da ki değişklikleri kendi branch imize dahil etmiş oluyoruz fakat, directory miz de herhangi bir değişiklik olmayacaktır. Git branch all; dersek tüm uzak bağlantıları da bize gösterecektir. Eğer ki bu uzak bağlantılar ile merge işlemi yapmazsak pull isteği yapmayacaktır, hata verir. Git branch -a ; ile var olan remote bağlantılara bakıyoruz hangisine merge yapacaksak, git merge origin/master; böylece merge işlemini tamamlanır

pull ise, git pull origin master; ile bu yukarıda ki ile aynı mantıkta oluyor. Ama daha kısa yolu bu oluyor.

Git stash komutu

Aynı projede birden fazla kişi ile çalıştığımız zaman git pull ile güncel versiyonu almış oluruz. Fakat kendi yazdığımız kod uçar. Bunu korumak için stash ile yazdığımız kodu muhafaza ediyoruz. Yani daha push lanmak için hazır olmayan dosyalar için kullanırız. Nasıl ? ; git stash save “stash edited” şeklinde stash ile korumuş olduk. Peki hangi dosyalar stash lenmiş ? git stash show -p ; git stash show -p 0 ; sırayla kaydedildikleri için en son gelen 0 oluyor. ; git stash list ; ile de tüm listeyi görebiliriz. ; git stash drop; ile de en son stash lenen dosyayı stash den kaldırmış oluyoruz. ; git stash drop 1; id si 1 olanı temizler. Nasıl geri getiririz ? git stash pop ; diyerek id si 0 olan stash i geri getirmiş olur.

Git alias

Git congif –global alias.st status => status ü artık st olarak değşitirdik, yani status gibi ya da başka belirlediğimiz bir kelimeyi uzun uzun yazmak yerine bu kısaltmaları yazabiliriz.

Git congif –global alias.co ‘commit -m’

Git config –list ; ile de bu tür kısaltmaların hepsine erişebiliyoruz, neleri atadık diye.

git push –set-upstream origin master

git internals