**Gitbash komutları**

ls -> İçerisinde bulunduğumuz klasörleri ve dosyaları listeliyor.

ls –la -> İçerisinde bulunduğumuz klasörleri, dosyaları ve GİZLİ olanlarını listeliyor.

pwd-> Güncel olarak bulunduğum klasörü göster.

cd-> Açmak istediğimiz klasörün adını yazıyoruz. Klasör değiştir demek.

cd ..-> Bir önceki klasöre geri gel demek.

clear -> Ekranı temizle demek. (Yazarken tab tuşuna bastığımızda otomatik dolduruyor.)

mkdir GitKursu -> GirKursu diye klasör oluşturuyor.

touch not.txt -> not.txt diye dosya oluşturuyor.

rm not.txt -> not.txt dosyasını siliyor. (Dosyaları siler)

rm –rf GitKursu -> GitKursu klasörünü siliyor.(Klasörleri siler)

git -> git’in yardım dökümantasyonunu getiriyor.

git –version -> Git’in yüklü olan versiyonunu gösteriyor.

***Kullanıcı adı ve Email Girme***

git config –global user.name “Ugur Arican” -> Projelerimizi kaydettiğimizde gözükecek isim.

git config –global user.email ugur-aricanotmail.com -> Projelerimizi kaydettiğimizde gözükecek email.

**Git Temelleri**

Commit kaydetmek demektir. Projeleri ara ara commitleriz. Oyunlardaki checkpoint gibi.

Aynı anda bir proje üzerinde birden fazla kişi çalışabiliyor. Burada bir proje iki farklı branch(dal)’a ayrılıyor diyelim. Ayrıldığı yerdeki ismini alır. Örneğin projede çıkış fonksiyonu yazıldı. Buradan sonra 2 branch’e ayrıldı. Bir tanesi resim paylaşma fonk. bir tanesi mesajlaşma fonk. Bunlara mesajlaşma brench’i, resim branch’i denir. Bu ikiye ayrılan proje ileride birleştirilebilir. Tek başına git branch komutu mevcut branchlerin hepsini gösterir.

Çalışma klasörü -> git add -> Index-Staging(sahne) -> git commit -> Local Repository

Çalışma klasöründeki verileri git add diyerek sahneye alırız. Buradan tekrar git commit komutunu kullanarak commitleriz.

git status -> Git’in güncel durumunu göster demek. Not a git repository diyorsa burada bu klasörün git ile bir alakasının olmadığını söylüyordur.

git init -> Git’i başlat demek. Burada bazı bilgiler verebilir. Örneğin master branch’ını kullanıyorsun değiştirmek istersen değiştirebilirsin gibi. Bu klasör içerisine konulan her şey git tarafından takip edilecek, loglanacak demek.

git init çalıştırmadan önce git status çalıştırıp ilgili klasör init edilmiş mi bakmak gerekiyor. 2 kere init yapmamız programda buglara sebebiyet verebiliyor.

.git dosyası bizim değişiklerimizin loglandığı tüm git ile ilgili işlemlerin döndüğü yerdir. Eğer bir yere yanlışlıkla git init etmişsek buradaki gizli .git dosyasını rm – rf . git komutuyla silip init edilmemiş haline döndürebiliriz.

Örnek olarak bir klasöre bir tane txt dosyası oluşturabiliriz. Mesh ekranından git status yaptığımızda karşımıza çıkan Untracked files (Takip edilmeyen dosyalar) çıkıyor. Git’in takip etmesini istemediğimiz video, fotoğraf dosyalarını eklemezsek burada takip edilmeyenlerde gözükür. Takip etmesi için git add \*dosyaIsmi yazabiliriz.

git add \*dosyaIsmi komutundan sonra git status yaparsak buradaki değişiklikler listelenecektir. Changes to be committed (Kaydedilmeyi bekleyenler) altındaki dosyalar commitlenmeyi bekliyor.

***İlk Commit***

git commit yalnız başına kullanılamıyor –m kullanmak zorunlu. Burada amaç her commitlemeye mesaj ile ne yaptığının açıklamasını yapmak. Bu mesaj kısmı da standartlara oturtturulmuştur, genelde İngilizce ve 2,3 kelimelik olur.Bu komuttan sonra çıkan ekranda x files changed( x dosya değiştirildi) x insertions(+) x deletions(-) (x ekleme yapıldı x çıkarma yapıldı. Buradaki ekleme ve çıkarmalar dosyaların içerisindeki satırlardır.) Buradan sonra tekrar git status dersek On branch master ifadesi gelir çünkü commit edeceğimiz hiçbir şey kalmadı.

***Git Log***

git log-> commit xx…xx (buna hash deniliyor.) (HEAD -> master) Her commitin kendine ait bir hashi olur. Bu commite geri dönmek için bu hash kullanılır. Master brach’inin içerisinde olduğumuzu söyler.

Author: Commitleyen kişinin bilgileri yer alır.

Date: Ne zaman commitlediği bilgileri yer alır.

Commitlediğimiz txt dosyasına gidip içerisine bir şeyler yazıp kaydediyoruz. git log dediğimizde Modified: (Değiştirilen) kısmında bizim dosyamız gözüküyor. Eğer burada başka bir dosya daha olsaydı mesela java dosyası. Onda da değişiklikler yaptık ve iki dosyayı da eklemek istiyoruz. git add x.txt git add x.java yazmak yerine git add . yapabiliriz, bu her şeyi ekler. Daha sonra git commit –m “yazmak istediğimiz mesaj” yaptığımızda 2 files changed, x insertion(+) yazacak. 2 dosayada değişiklik oldu x satır eklendi demektir. Tekrar git status çalıştırdığımızda nothing to commit, working tree clean yazısı çıkıyor. Yani commitlenecek bir şey yok, temiz bir çalışma yerin var diyor. git log dediğimizde her iki commit de gözükür. En üstteki commit en son attığımızdır. Burada eklediğimiz satıları da gösterir. Silmekte bir değişikliktir, örneğin java dosyasındaki yazılanları sildik ve git log dediğimizde midified: x.java gözükecektir. git add . diyip ekledik git commit –m “Java kodundan gereksiz kod silindi” dediğimizde bize 1 file changed, 1 deletion(-) yazacaktır. git status yaptığımızda bir değişiklik olmadığını söyleyecektir. git log yaptığımızda tüm logları gösterecek ve en üstte son yaptığımız commiti gösterecektir. Alt tarafında mesaj kısmında da Java kodundan gereksiz kod silindi yazacaktır.

git commit kodunu –m olmadan yanlışlıkla kullanırsak git bize default olarak hangi editör seçiliyse onu açıyor ve burada bize mesajımızı yazmamızı istiyor. Çıkmak istiyorsak :q kodunu kullanabiliriz.

***Git Ignore***

Commitlemek istemediğimiz (git’in görmezden gelmesini istediğimiz) belgeleri (fotoğraf, video, kütüphaneler, database ile ilgili bilgiler vs.) belirtiyoruz. Örneğin gizli.txt diye bir dosya oluşturalım.

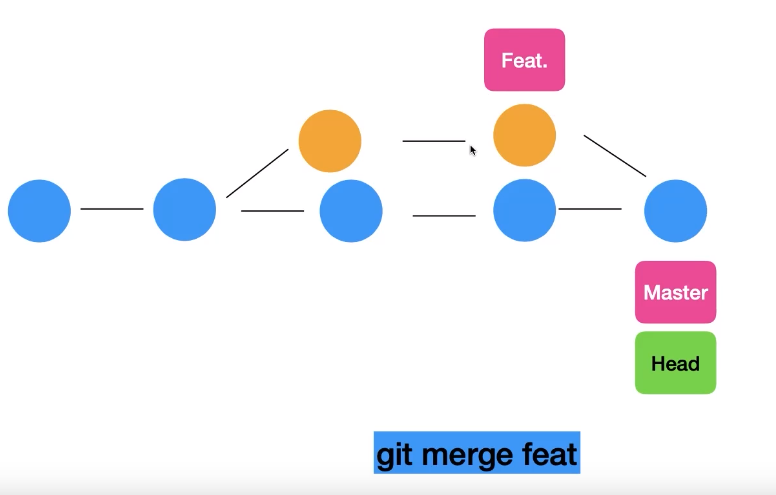
İlgili git sisteminin çalıştığı dosyada .gitignore olması gerekli bunun için Besh ekranından pwd ile ilgili dosyada olduğumuza bakıyoruz ve touch .gitignore diyere oluşturuyoruz. Bu dosyanın içerisine hangi dosyayı eklemek istemiyorsak onun adını tam olarak yazıyoruz örneğin gizli.txt gibi. git status yazıyoruz ve Untracked files’da .gitignore gözüküyor. Artık git add . yapsak bile bizim gizli.txt dosyamızı almayacak.

**Git Branch**

***HEAD Nedir?***

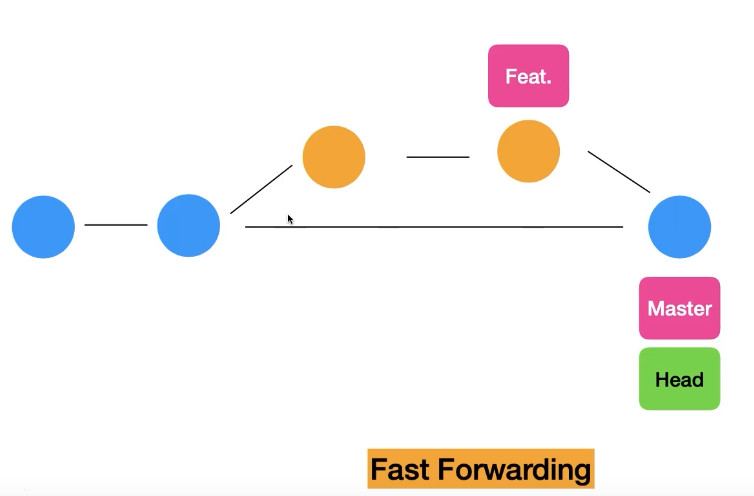
Commitlenmiş projelerden hangisinde olduğumuzu gösterir. Normalde en son commitlenen projede HEAD yazar fakat biz eğer daha önceki commite dönersen o döndüğümüz committe HEAD yazar. Bir de ilgili branch değişebilir. Normalde master branch’indeyiz ve burada en son commitlenme yapıldı Head burayı gösterir. Başka bir branch’imiz varsa ve orada en son commitlenme yapıldıysa bu sefer HEAD orayı gösterir fakat yanında hangi branch’de olduğumuzu da söyler.

***Merge***

******

Yeni branch oluşturmak için git branch x komutu kullanılır.Örnek olarak git branch feat ile feat adında yeni bracnh’ı oluturduk ve projemizde bununla ilerledik. Daha sonra git siwtch komutu ile main branch’ına döndük ye orada da değişiklikler yapıp commitleyip ilerledik. Artık bu iki branch’ı birleştirmek istiyoruz. Burada HEAD’in master branch’ında olduğunu varsayarak ve feat branch’ını master’a eklemek istiyorsak git merge feat komutunu kullanıyoruz. Bu komuttan sonra ekrana commit mesajı yeri geliyor. Burada yeni commit mesajı girmemiz. Mage branch “feature” şeklinde yazılabilir.

***Fast Forward***



Bu yöntem genelde ya kalabalık projelerde ya da kendi kodumuz güzel çalışıyor ve farklı özellik eklemek istiyorsak yeni ekleyeceğimiz özelliğin hata almasından çekiniyorsa yeni branch’te çalışıp sorun yoksa asıl branch’imize merge edebiliriz. Buna da fast forwarding deniliyor. Burada dikkat edilmesi gereken asıl branch’imizde hiçbir değişiklik yapmamış oluşumuz.

***Merge Conflict***

***Senaryo 1 (Yanlışlıkla Bir şey Ekleme)***

Bu bölümü örneklerle anlatacağım.

JavaKitabi diye bir klasör oluşturuyoruz (mkdir JavaKitabi) ve klasörün içine giriyoruz (cd JavaKitabi). git status ile bunun git reposu olup olmadığına bakıyoruz (fatal: not a git repository) gir reposu değil diye uyarı veriyor. git init yaparak bunu git reposu haline getiriyoruz. touch ilkbolum.txt diye bir dosya oluşturuyoruz.

Dosyanın içerisine “java kitabımız çok güzel olacak. Bence okuyun.” yazdık. Terminale gelip git status yaptık ve değişiklikler burada gözüktü. git add. Diyoruz ardından git commit –m “giriş cümlesi yazıldı.” diyoruz ve böylece ilk commitimizi yapmış olduk.

git branch arkakapak diyerek yeni branch oluşturduk. git switch arkakapak diyerek bu branch’e geçiyoruz. Bu branch’deyken yanlışlıkla ilkbolum.txt dosyasının içerisinde yazıalrın hepsini siliyoruz ve “arka kapak çok güzeldir.” diyoruz.

git add . ardından git commit –m “arka kapak yazıldı.” diyoruz. git switch master diyerek master’a dönüyoruz. ilkbolum.txt’yi kontrol ettiğimizde ilk yazdığımız yazılar gözüküyor. Burada bu ilkbolum.txt’yi silmek istiyoruz ve yerine bolum1.txt oluşturmak istedik. Terminalden rm ilkbolum.txt yazarak siliyoruz ve touch bolum1.txt diyerek yeni dosya oluşturuyoruz. bolum1.txt içerisine de “Java kitabımız çok güzel” diye ekleme yaptık. Ardından git add . ve git commit –m “ilk bolum tekrar yazıldı” diyerek yeniden commitledik.

Şimdi bu iki branch’i birleştirmek istedik ve git branch diyerejk hangi branch’de olduğumuza baktık. Master branch’inde olduğumuzu gördükten sonra git marge arkakaprak diyoruz ve şu çakışma hatası ile karşılaşıyoruz.

CONFLICY (modify/delete): (Burada çakışmanın ne olduğunu söylüyor.) ilkbolum.txt deleted in HEAD and modifed in arkakaparak. Version arkakaparak of ilkbollum.txt left in tree.

(ilkbolum.txt silinmiş ama arkakaparak’ta bunun içerisinde değişiklik yapmışsın sonra da bunu silmişsin.)

Automatic marge failed; fix conflicts and then commit the result.

(Atomatik birleştiremediğini çakışmaların giderilmesi gerektiğini söylüyor.)

Burada ilkbolum.txt veya bolum1.txt dosyasını silebiliriz yada içerisindeki yazıları diğerine taşıyabiliriz ve silebiliriz. Hatta arkakapak.txt açıp ilkbolum.txt’de yazdıklarımızı buraya koyup silebiliriz.

Biz burada bu iki txt dosyasında değişiklik yapmadan kaydediyoruz ve yeniden git marge arkakapak diyoruz ve error:Merhing is not possible baceuse you have unmerged files. Hatası ile karşılaşıyoruz.

Hint: (burada ipucu veriyor.)

Burada dosyalarını kaydettin ama git add olarak ekle ve ardından commit yap diyor.

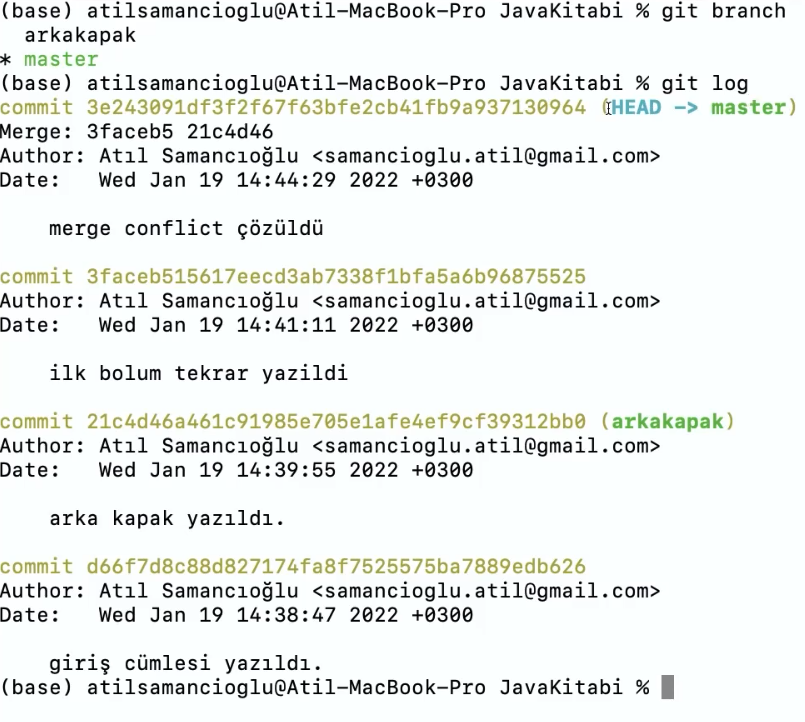
git status çalıştırıyoruz ve Unmerged pats: kısmında deleted by us: ilkbolum.txt gözüküyor.

Yani ilkbolum.txt geri geldi diyor. git add . ardından git commit –m “merge conflict çözüldü”

Git merge arkakapak dediğimizde Already up to date. (Zaten birleştirme yapıldı) diyor.

git status dediğimizde nothing çıkıyor. git branch çalıştırıyoruz ve hala masterdayız.

git log çalıştırıyoruz ve aşağıdaki satırlarla karşılaşıyoruz.



Burada çözüm olarak ilk bolum tekrar yazıldı commitini alıyor ve master branch’inde ilerletiyor. Çakışma yapan bu dosyadan bir tane daha oluşturuyor onu da arkakaparak branch’inde içerisindeki önceki yazılanlarla birlikte tutuyor.

***Senaryo 2 (Aynı dosya içerisinde çakışma)***

Master branch’ındayken yeni bir dosya oluşturuyoruz touch ornek.java bu ornek.java dosyasının içerisine String email=“ugur-arican.hotmail.com”; git add . dedik ardından git commit –m “email eklendi” dedik.

git branch feat ile feat branch’ini oluşturduk ve git switch feat ile bu branch’e geçtik.ornek.java dosyasının içerisini düzenledik ve String email=“ugur\_842.hotmail.com.tr”; yaptık ve git add . dedik ardından git commit –m “email değiştirildi” diyerek yeniden commitledik ve git log ile bu değişimi kontrol ettik.

git switch master diyip master’a geçiyoruz git status ile değişim var mı kontrol ediyoruz. Herhangi bir değişiklik yok fakat ornek.java dosyasının içeriğinde ugur-arican.hotmail.com mailini gösteriyor ve biz bu satırı siliyoruz. git add . ve git commit –m “email silindi” diyoruz.

git merge feat diyoruz CONFLICT(content): Merge conflict in ornek.java

Automatic merge failed; fix conflicts and then commit the result.

Burada ornek.java içerisinde bir çakışma var ve otomatik merge yapılamadı diyor.

Bu sefer senaryo 1’den farklı olarak ornek.java dosyasının içerisinde bize ayrıntılı olarak gösteriyor.

<<<<<<HEAD

========

String email = “ugur\_842.hotmail.com.tr”;

>>>>>>> feat

Burada güncel olan yerinde silmişsin fakat feat içerisinde String email = “ugur\_842.hotmail.com.tr”;

Yazmışsın diyor. Bunu komple silebiliriz bunu yaptığımızda master’daki halini tut demektir.

Sadece String email = “ugur\_842.hotmail.com.tr”; yazabiliriz bu da feat’i tut demek. Ya da yeni bir yazı yazabiliriz.

String mesaj= “oyle bir şey yaptım ki, her şeyi ben belirledim.”;

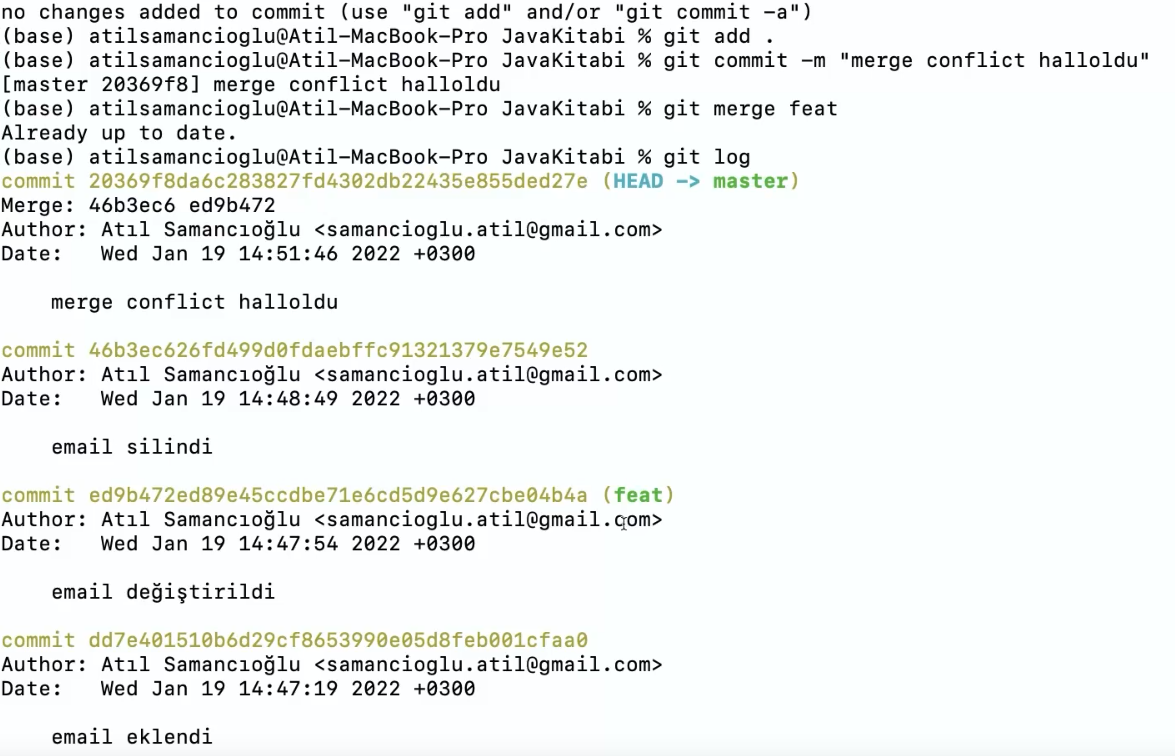
(Not: Bazı ide’lerde seçenekler de çıkabilir. Bunu kabul et, şunu kabul et gibi)

Biz buradaornek.java içerisinde sadece String mesaj= “oyle bir şey yaptım ki, her şeyi ben belirledim.”; ifadesini bırakıp save ediyoruz. git status yapıyoruz ve both modified: ornek.java gözükmekte.

git add . ve git commit –m “merge conflict halloldu.” diyoruz ve git merge feat yapıyoruz. Already up do date (zani zaten yapıldı diyor.) git log yapıyoruz ve aşağıdaki çıktıyı veriyor.

Burada master merge conflict hallolduyu point ediyor ve feat’de yapmış olduğumuz email değiştirildi de bu en sonki commitin içerisinde.

Sonuç olarak iki branch’ın merge edilebilmesi için conflict olmaması gerekli.



***Stash (Depola) – Pop (Ap)***

Endüstride commitlemenin kullanım amacı bitmiş projeler üzerinden yapılır, yani yarım kalmış bir projede commitleme yapılmaması gerekir.

Bir projede yeni bir branch üzerinde bir dosyada çalışıyoruz örneğin x.txt ve acil olarak master branch’ına dönmemiz gerekti. git add . yaptık ve commitlemeden master branch’ına geçtik diyelim. Bu x.txt dosyası da bizimle master branch’ına geçmiş oluyor. Ama eğer biz master branch’ine geçmeden commitleseydik bu sefer x.txt dosyası master’a geçmeyecekti ama tam bitirilmemiş projeyi de commitlemememiz gerekiyor. Bir sorun da iki branch de de aynı dosya varsave git add yapmadan master branch’ine geçtiğimizde bir önceki branch’teki aynı isimdeki dosyaların içeriğini master’a aktarır ve git status yaptığımızda modified olarak gözükür. Fakat biz burada master branch’ında değişiklik yapmamıştık eğer git add yaparsak diğer branch’takileri master’a atmış olacak ve master’ın içindekiler kaybolacak. Bu status ekranındayken git restore x.txt dersek x.txt dosyasının master branch’ında almış olduğu son commiti getirir. Burada ger switch diyip diğer branch’a geçersek bu sefer de o branch’taki bilgiler kaybolur.

Bu senaryoyu baştan alırsak bir branch’da çalışıyoruz ve acil master’a geçmemiz gerekti. Commitlemek istemiyoruz ama verilerimizin kaybolmasını da istemiyoruz bu durumda git stash komutu çalıştırılır. Bunu yaptığımızda çalıştığımız branch çalışmaya başlaamdan önceki haline döner. Git status yaptığımızda da zaten hiçbir değişiklik olmadığını söyler ve artık master branch’ına geçerbiliriz. Master’da değişiklik yapıp git add . ve git commit’le commitledik. Şimdi git switch ile diğer branch’a geçiyoruz git status ile baktığımızda yine bir değişiklik olmadı olarak gözüküyor. git stush pop diyoruz zulaladığımız veri geri geliyor ve stash’lerden siliniyor. Bu komutu bu branch’da değil başka bir yerde kullansaydık oraya getirirdi.

Birden fazla stash komutu da kullanabiliriz. Örneğin bu branch’ta bir değişiklik yaptık ve git stash ile zulaya gönderdik. Bu branch eski haline dönmüş oldu bu eski haline da farklı bir değişiklik daha yaptık ve bunu da git stash ile zulaya gönderdik. git stash list ile bunların listesini görebiliriz ve başlarında stash @ {0}, stash @ {1} gibi gözükecektir. Hangi git stash apply stash @ {0} komutunu kullanırsak istediğimiz stash’i istediğimiz yere koyabiliriz.

git stash clear ile tüm stash’leri silebiliriz. git stash pop komutunu kullandığımızda en sonki stash’ı getiriyor ve listeden siliyordu. git stash apply dediğimizde en sonki stash’i getiriyor ve listeden silmiyor.

**Geçmişe Dönme**

***Checkout***

Normalde eklediğimiz bir şey olursa git status çalıştırdığımızda bize 2 seçenek veriyordu.

* Use “git add <file>…” (Yani git add yaparak bunu commitlemeye hazır hale getirebilirsin.)
* Use “git restore <file>…” (Git restore komutuyla bunu önceki haline alabilirsin.)

Eğer commitlenmişse ve daha önceki commite dönmek istiyorsak git checkout (hesh) komutu kullanılır. Buradaki hesh ifadesi git log dediğimizde listelenen commitlerden hangisine geri dönmek istiyorsak ona ait numaradır. Bunu yaptığımızda ;

You are in ‘detached HEAD’ stade. Şeklinde hata veriyor bu hatada HEAD(Olduğumuz yeri gösteren) kopuk bir durumda diyor ve iki seçenek sunuyor. Ya geri dönmelisin ya da yeni branch açmalısın diyor. Hatta buradayken git branch çalıştırdığımızda (HEAD detached at xxx) diyor.

* Eğer buradayken git switch master dersek ve master branch’ine dönersek git chechout ile silinen bilgiler geri gelir ve hiç işlem yapmamış gibi oluruz.
* Git branch feat diyerek feat adında yeni branch açıp git switch ile bu branche geçebilir ve ardından git add . daha sonra git commit –m (mesaj) ifasini girebiliriz.

***Reset ve Revert***

Aynı branch’taki ve önceki commite dönmek istiyorsak git reset (gitmek istediğimiz commitin hesh numarası) yazıyoruz. Ama burada dikkat edilmesi gereken yer HEAD ve master önceki commite dönüyor fakat içerik olarak en son commit’te yazılanlar hala duruyor. Dönülen commit’ten sonraki commitleri de siliyor.

Eğer önceki commit’e dönmek, döndüğümüz committen sonrakileri silmesini ve içerik olarak sadece döndüğümüz commit’in kalmasını istiyorsak git reset –hard (gitmek istediğimiz commitin hesh numarası) komutunu kullanabiliriz.

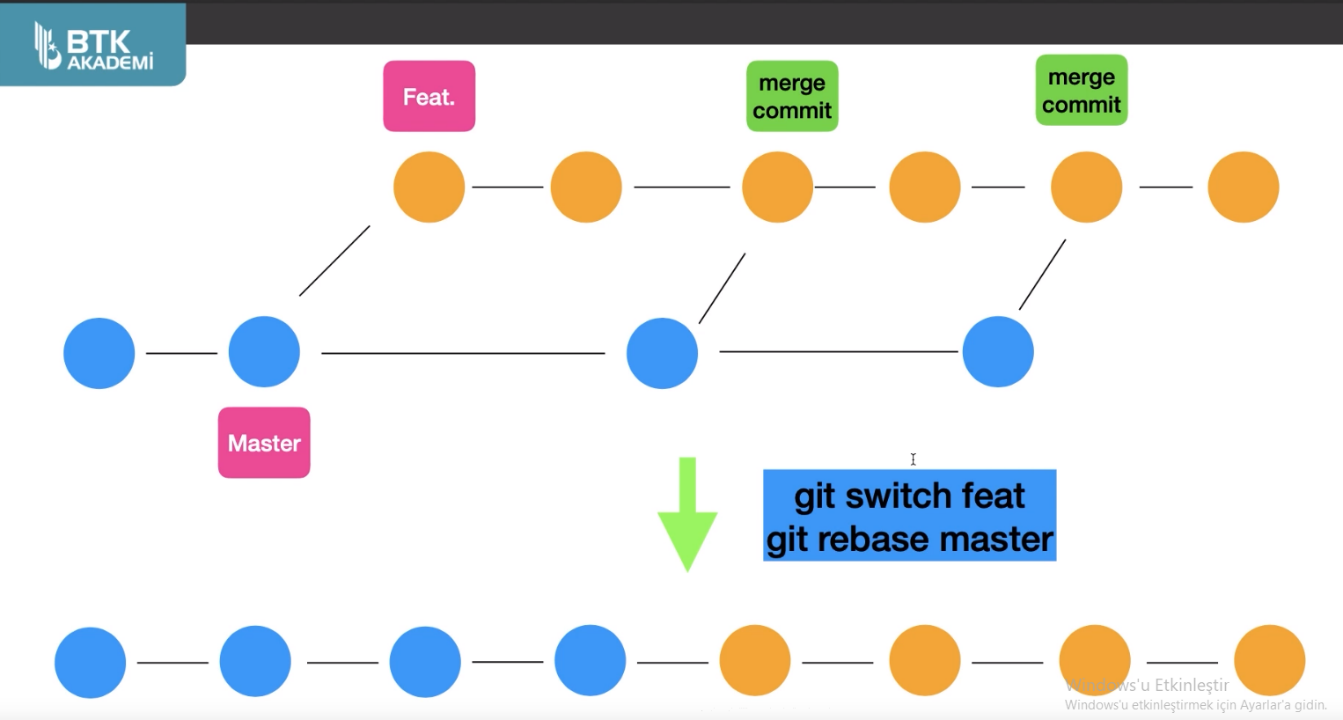
En sondaki commit’i geri olmak istiyorsak da git revert (geri alınmasını istediğimiz hesh numarası) komutuyla kullanabiliriz. Bu komutu kullandığımızda mesaj girme ekranı çıkacaktır. Bu komutu kullandıktan sonra hangi commit numarasını girdiysek o geri alınacaktır. Burada dikkat edilmesi gereken en sondaki commiti revert etmemiz olacaktır aksi taktirde conflict’ler meydana gelebilir. Git log komutuyla da kontrol ettiğimizde Revert olarak gözükecektir. Revert’un reset’den farkı git log ekranında hala diğer commit’lerin gözüküyor olması yani ileride tekrar o commit’lere gitdebiliriz.

***Git Diff***

Commit’ler, branch’ler, dosyalar arasındaki farkları gösterir. Git diff komutunu yalnız başına kullandığımızda ilgili dosya içerisindeki değişiklikleri gösterir, git add yapılırsa ve tekrar git diff yapılırsa bir değişiklik göstermez. Eğer bu durumdaki değişiklikleri görmek istiyorsak git diff HEAD yapmamız gerekir, burada add yapılsa dahi commit’lenmediyse bu değişiklikler gözükecektir.

Eğer iki commit arasındaki değişiklikleri görmek istiyorsak git log ile hangi iki commit arasındaki değişiklikleri görmek istediğimizi belirleyebilir ve git diff (karşılaştırmak istediğimiz 1. Commit’in hesh numarası) (karşılaştırmak istediğimiz 2. Commit’in hesh numarası) ile bu iki commit arasındaki farkı görebiliriz. (Eğer burada komut çalışmazsa iki commit numarası arasına boşluk yerine “:” koyulup çalıştırılabilir.)

Eğer iki branch arasındaki farkı görmek istersek bu sefer git diff (1. Branch adı) (2. Branch adı) olarak komut çalıştırılır. (Boşluk sıkıntı çıkarırsa yine “:” ile çalıştırılabilir.)

***Rebase***

Feat branch’inde projemiz üzerinde çalışıyoruz ve master branch’ine şirket içerisinden başkaları commit yaptı. Bu yeni master bracnh’ini bizim çalıştığımız feat branch’ine merge ettik yani birleştirdik. Daha sonra biz feat üzerinde çalışırken yeniden master’a bir commit geldi biz bu yeni gelen master’la da merge ettik yani birleştirdik. Artık üzerinde çalışmamız bitti ve master branch’iyle birleştirmek istiyoruz. Bu durumda rebase komutunu kullanıyoruz. Git siwtch feat ardından git rebase master ile master’da yeniden oluşturuyoruz. Buradaki görselde de gözüktüğü gibi master commitlerini tutuyor ve ardından feat bracnh’ini ekliyor. Feat bracnh’ini eklerken merge ile birleştirdiğimiz yerleri almıyor. Burada commit tarihleri yeniden yazılıyor yani kronolojik olarak sıralama değişiyor.

Burada çok dikkatli olmamız gereken husus şu;

Eğer biz bu üzerinde çalıştığımız feat branch’ini başkalarıyla paylaştıysak yani paylaşılan kişiler bizim bu commitleri kullanmaya başladılarsa kesinlikle bu komutu kullanmamalıyız.

Eğer biz üzerinde çalıştığımız feat branch’ini başkalarıyla paylaşmadıysak bu komutu kullanmamızda bir sakında yok.

Şimdi bunu uygulamalı bir örnek üzerinden daha iyi anlayabiliriz;

Mkdir RebaseTestProjesi ile yeni klasör oluşturduk. cd RebaseTestProjesi ile bu klasörün içerisine giriyoruz. Git status ile git reposu olmadığını görüyoruz ve git init ile git reposu haline getiriyoruz.

İki dosya oluşturuyoruz. touch birinci.txt, touch ikinci.txt ve git add . ile ekledik git commit –m “ilk commit” diyerek commitledik.

Birinci.txt içerisine “master ikinci commit için bu satırı yazıyorum.” kaydet diyip çıkıyoruz. Git add . ve git commit –m “master 2. Commit” diyip commitledik.

Git switch feat ile bu bracnh’e geçtik. İkinci.txt içerisine “feat’e ilk commit atabilmek için bu satırları yazıyorum.” Kaydet diyip çıkıyoruz ve git add . ardından git commit –m “feat’in ilk commiti’ dedik.

Git switch master diyip bu branch’e geçiyoruz. Bunu yaptığımızda ikinci.txt içerisindekiler silinmiş oluyor çünkü feat bracnh’inde eklemiştik. Birinci.txt içerisine girip “master’da bir commit daha atacağız.” Diye yeni yazı ekledik ve kaydedip çıktık. Git add . ardından git commit –m “master’da üçüncü commiti de attık.” Diyoruz. git switch feat yapıyoruz, burada biz normalde feat branch’inde çalışıyorsak master’da bir commit daha gelmiş, burada bu master’da yapılan değişikliklere göre bizim yapacağımız proje değişebilir diye bu master’daki değişiklikleri bizim bu feat branch’ine almak istedik. Git merge master yapıyoruz ve yeni bir commit oluşturulacağı için mize commit mesajı soruyor, bu çıkan mesaja tamam diyip olduğu gibi kabul ediyoruz, git log ile commitlerimizi kontrol ettik.

En sondan öncekilere doğru “feat ikinci commit” – “Merge branch ‘master’ info feat” – “masterda üçüncü commiti de attık” – “feat’in ilk commiti” – “master ikinci commit” – “ master ilk commit”

İkince.txt dosyasını açıp “feat’in de bir ikinci commiti olmasın mı” diyip kaydettik. Git add . ve git commit “feat ikinci commiti” diyerek commitledik, git log ile tekrar commitlerimizi kontrol ettik. Git bracnh ie feat branch’inde olduğumuzu doğruladık, git rebase master dedik. Konsolda “Successesfully rebased and update refs/heads/feat.” Çıktı. Git log ile commitleri kontrol etmek istedik ve en sondan öncekilere doğru “feat ikinci commit” – “feat’in ilk commiti” – “master’da üçüncü commiti de attık” – “master ikinci commit” – “master ilk commit”

Yani git rebase’i hem log temizlemek için hem tarihleri tekrar yazmak için kullanabiliriz.

**GitHub**

***GitHub Temelleri***

Star: Sosyal medyadaki like ve kaydetme gibi düşünülebilir. Profilden stars kısmından star’ladığımız projelere erişebiliriz.

Explore: Sosyal medyadaki keşfet kısmı gibi çalışıyor.

Bir projedeyken Issues(sıkıntı) kısmından iş:open açık olan sıkıntı gösteriyor filtreyi kaldırırsak kapanmış olanları da gösterir. Burada kendimiz de açabiliriz, burada bug var gibi yazabiliriz.

Git remote add origin https://.. Bu komut ile uzak git adresimizi veriyoruz. Origin uzak repomuzun adresinin ismi, değişken olarak düşünülebilir.

Git push –u origin main Bu komutda commitleri uzak repomuza yani origine it manasındadır.