ATIL SAMANCIOĞLU VERSİYON KONTROLLERİ: GİT VE GİTHUB EĞİTİMİ NOTLARI

1.2 Git Nedir?

Versiyon control sistemidir.

Git –version komutu ile yüklü olan git versiyonunu öğrenebiliriz.

Git’de kullanıcı adı ve mail tanımlama:

git config --global user.name “user name” değişikliği teyit etmek için git config user.name komutunu çalıştırabiliriz.

git config --gloabal user.mail “user mail” değişikliği teyit etme için git config user.mail komutunu calıştırabiliriz.

A diagram of a company

Description automatically generated

A diagram of a network

Description automatically generated

git add: bu komutla kodumuzu index-staging dediğimiz bir yere alıyoruz. Bu commitlenmeden önce bir yerde tutmak için yapıtığımız komuttur.

A screenshot of a computer screen

Description automatically generated sonrasında git commit komutu ile commitimizi oluşturabiliriz.

A screenshot of a video

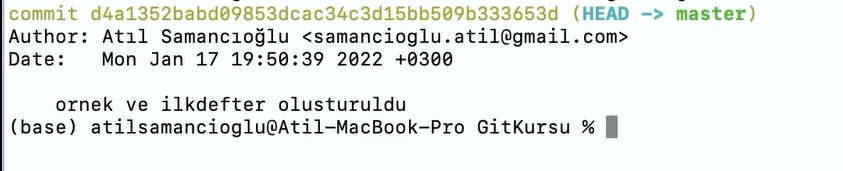
Description automatically generated

Bir klasörde güncel git durumunu öğrenmek için git status komutunu çalıştırmalıyız. Eğerki git reposu değil diye çıktı alırsak repomuzu git reposu haline getirmek için yani git ile entegre edebilmek için git init komutunu çalıştırmalıyız. Yanlışlıkla bir yere git eklersek kaldırmak için: rm -rf .git komutu ile giti kaldırabiliriz, aslında git status komutununda control ettiği şey .git isimli dosyanın var olup olmadığıdır.

git add komutu ile eklediğimiz dosyalar staging area’ya alınır yani git tarafından takip edilir (tracked file). git add komutunun ardından commit yapmak istersek komut şu şekilde:

git commit -m “write a message for the commit.”

Her commitin kendine ait bir hash’I yani numarası/id’si vardır. git log komutu ile commitlerimiz hakkında bilgi alabiliriz:



Hem add hem de commit’I tek satırda yapmak istersek şu komutu kullanabiliriz:

git commit -a

değişiklik yaptığımız tüm dosyaları tek seferde indexlemek için (staging) şu komutu kullanabiliriz:

git add .

Repomuzda bazı dosyaların git tarafından takip edilmesini istemiyorsak şu komutla bunu sağlayabiliri<, tabii öncesinde .gitignore isimli bir dosya oluşturmamız gereklidir.

Bunun içinde hangi dosya/dosyaların izlenmesini istemezsem bu dosyanın içine onun ismini yazmamızla beraber artık git tarafından takip edilmez, bu dosya başka dosyalar tarafından tabii ki görünür tek fark artık git tarafından takip edilmez. Uğraştığımız dil için gitignore dosylarının örneğini internetten aratabiliriz: gitignore template …

Head güncel olarak nerede olduğumuzu gösterir:



git branch komutu ile yeni bir branch açıyoruz:

git branch “branch adı”

git switch “branch adı” 🡪bu komutla geçmek istediğimiz branche geçeriz.

git branch 🡪 komutu ile var olan bütün branchlerimi görebilirim

2 branchi birleştirmek için hangi branchi birleştiriceksek yani diğerinin üzerine alacaksak onu komuta yazmalıyız, örneğin feature branchini master branch’inin içine almak istiyorsam:

git switch master

git merge feature

fast forwarding

A diagram of a diagram

Description automatically generated

Yeni bir branch açtıkdan sonra orada hep commit atıp masterda hiçbir şey yapılmamışsa o sırada, bunu zaten merge edebilirim buna fast forwarding yani “ileri sarma” diyoruz.

A screenshot of a computer program

Description automatically generated

!!!!!!Bir branchi başka bir branchin üzerine alırken çakışma olursa buna “conflict” diyoruz. Ve bunu çözmek için çakışmanın olduğu dosyalara gidip hangisini seçeceğimize karar verip ardından bunu commitledikten sonra merge komutunu çalıştırabiliriz.

Biz tam olarak içimize sinmeyen şeyi commit etmeyiz genelde, yani genelde bitirdiğimiz istenen amaca ulaşmış şeyleri commit ederiz. Örneğin mobile branchi isimli bir branchimiz olsun ve bende o branchin üzerinde çalışıyor olayım. Çalışmalarım çok ilerlemiş ama bitmemiş olsun, o sırada takım lideri benden başka bir branche geçip acil olan başka bir şeyi düzeltmemi istemiş olsun. Ben de yapılan çalışmalarımı kaybetmek istemiyorum ama aynı zamanda commit etmekte istemiyorum çünkü yapacaklarım tam bitmemişti. Böyle bir durumda kendi branchimi değiştirmeden bulunduğum branchte sadece git add “dosya adı” komutunu çalıştırarak çalıştığım dosyayı da kendimle beraber yeni branche götürüp, geri dönerken de geri getirebilirim. Bu sadece ilk branche özgü olan bir dosya için geçerlidir. Eğer geçiş yapacağım branchte de o dosya mevcut ise yani her iki branchte bulunuyorsa bu yöntem işe yaramaz, bu halde ya commit yapıp öyle dönmem lazım ya da stash denilen yere o değişikelerimi saklaması için git’e komut verebilirim. Bu ne çalışma dizininde ne repoda ne de indexte tutulur. Bu gitin kendine has bir yapısıyla sağlanır. Bunu yapmak için branch değiştirmeden önce şu komut çalıştırılır:

git stash

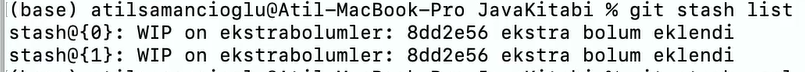
A screenshot of a computer program

Description automatically generated

Stashlediğim bilgiyi geri getirmek için bunu uygulamak istediğim yere geçip şu komutu çalıştırıyorum:

git stash pop

git stash list komutu ile güncel sakladıklarımı görebilirim



Eğerki listeden sadece birini eklemek istersem:

git stash apply stash@{0} şeklinde komutu çalıştırırım.

Istersem tüm stash’leri şu komutla silebilirim:

git stash clear

eğer bir stash’I hem uygulayıp hem listede kalmasını istiyorsak o zaman pop yerine apply kullanmalıyız:

git stash apply

o zaman bu stash’I başka branchlerde kullanmak istersek listede tutulu kaldığı için kullanabiliriz.

git restore “dosya adı”🡪commit atmadan önce yapılam değişiklikleri sıfırlamak için bu komutu kullanırız.

Bir veya birçok commit yaptıkdan sonra geriye dönmek için checkout komutunu kullanırız:

git checkout “commit\_hashi” aynı zamanda checkout ile bracnh’de değiştirebiliriz.

Eğer birkaç commit ilerledikten sonra o commitleri beğenmeyip önceki commite dönmek istersek eğer, burada amaç bu commitleri tamamen silmek bunu reset ve reset hard komutları ile yapabiliriz. Arada ki fark ise şudur: reset comitleri silip dönmek istediğiniz commite döner ama dönerken sildiğiniz commit içindeki değişiklikleride döndüğünüz commite taşır. reset hard ise bunu yaparken değişiklikleri getirmez yani commitlerle beraber orada yaptığınız değişiklikler de silinir ve döndüğünüz commit ondan sonraki commitlerde eklediğiniz değişiklikleri içermez.

git reset “commit\_hashi”

git reset --hard “commit\_hashi”

tabii bu durumlarda başka arkadaşlarımız ile merge yaparsak conflict olması muhtemeldir, çünkü bu commitlerin loglarını biz kendimizden silmiş olacağız. Peki hem bu commitlerden kurtulmak isteyip hem de aynı branch üzerinden devam etmek istiyorsak ne yapmalıyız? İşte burada kullanılması gereken komut revert komutudur. Bu komutun yanına hangi commiti çıkarmak istiyorsak onun hash’ini yazmamız lazım:

git revert <cıkarılacak\_commit\_hashi>

bu sayede yeni bir commitle aynı branch üzerinden diğer commit içeriklerini elimine ederek bulunduğum commitin içeriğiyle devam edeceğim, bu sayede diğer kişilerde bulunan loglar bende de duracak.

git diff ile yapılan değişiklikleri görebiliriz.

2 branch arasındaki farkı görmek için:

git diff branch1 branch2

2 commit arasındaki farkı görmek için:

git diff <commit\_hash1> < commit\_hash2> aradaki boşluk yerine : da koyulabilir.

A screenshot of a computer

Description automatically generated

git rebase komutunu kullanma amacım sürekli merge işlemini yaptığım commitleri yapmamak ve bu komutu kullanarak commitleri sıralayıp, merge için commit yapmamak yani bu komutu kullandığımda bu sekilde commitler yeniden düzenlenir ve log değişmiş olur,

git rebase’I eğer ki şu durumda kullanırsam çok sıkıntı yaşarım: projede çalışan başka insanlarda benim commitlerimi kullanmaya başladılarsa sonrasında rebase edip tekrar paylaşırsam herkesin projesini darmadağın edebilirim. Eğer rebase yapmadan önce hiç paylaşmadıysam kimsenin bunlardan haberi yoksa o zaman rahatlıkla rebase yapabilirim.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated