SOURCETREE İLE GİT VE GİTHUB KULLANIMI

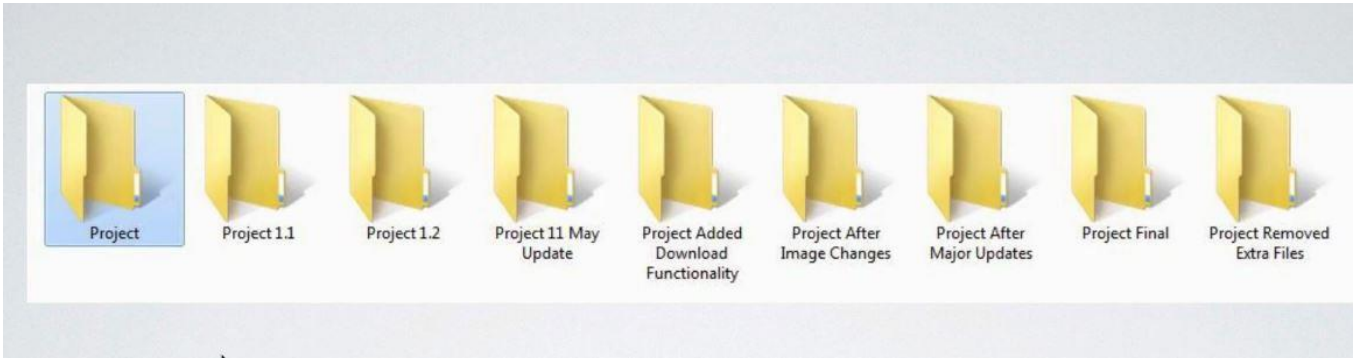
Muhammet sahin

@SiliconeLabs

VERSİYON KONTROLÜ NEDİR

Bir dosya veya bir küme dosyadaki değişiklikleri takip edebilmek için uyguladığımız bir yöntemdir.

Git ise bu değişikliklerin tarihçesini ve içeriğini bizim için takip eden ve kaydeden bir veritabanıdır.



VERSİYON KONTROLÜ NEDİR

VCS kullanarak herhangi bir anda üzerinde çalıştığınız dosyaların o anki hallerini kaydedip, daha sonra bu kaydedilmiş ve kontrol altına alınmış haline geri dönebilirsiniz.

Versiyon; dosyaların kayıt altına alınmış herhangi bir andaki hallerine denir.

Versiyon kontrolü kullanılan programlama dilinden, framework veya işletim sisteminden bağımsız bir yaklaşım olarak düşünülmeli. Çünkü vcs;

- HTML dosyalar için kullanılabileceği gibi, Android veya iPhone uygulaması kaynak kodu için de kullanılabilir.
- Dosyaların üzerinde çalışırken hangi işletim sistemini veya programları kullandığınızla ilgilenmez.

VERSİYON KONTROLÜ FAYDALARI NELERDİR

Uyumlu Ekip Çalışmasını sağlar.

- Eğer vcs kullanmazsak farklı kişilerle aynı dosyalar üzerinde çalışma durumunda sorunlar çıkabilir.
- Eğer vcs kullanırsak herkes özgürce değişikliği yapar ve güvenli bir şekilde merge işlemi yapabilir.

Versiyonları düzgün bir şekilde takip edilmesini sağlar

Önceki versiyonlara Geri Dönebilme

Dosyaların neden değiştiğini anlama

Yedekleme

KISA GİT TARİHÇESİ

2005 yılında başta Linus Torvalds olmak üzere Linux çekirdeğini de kodlayan ekip tarafından Linux kaynak kodunun versiyon kontrolünü yapmak için geliştirilmiştir.

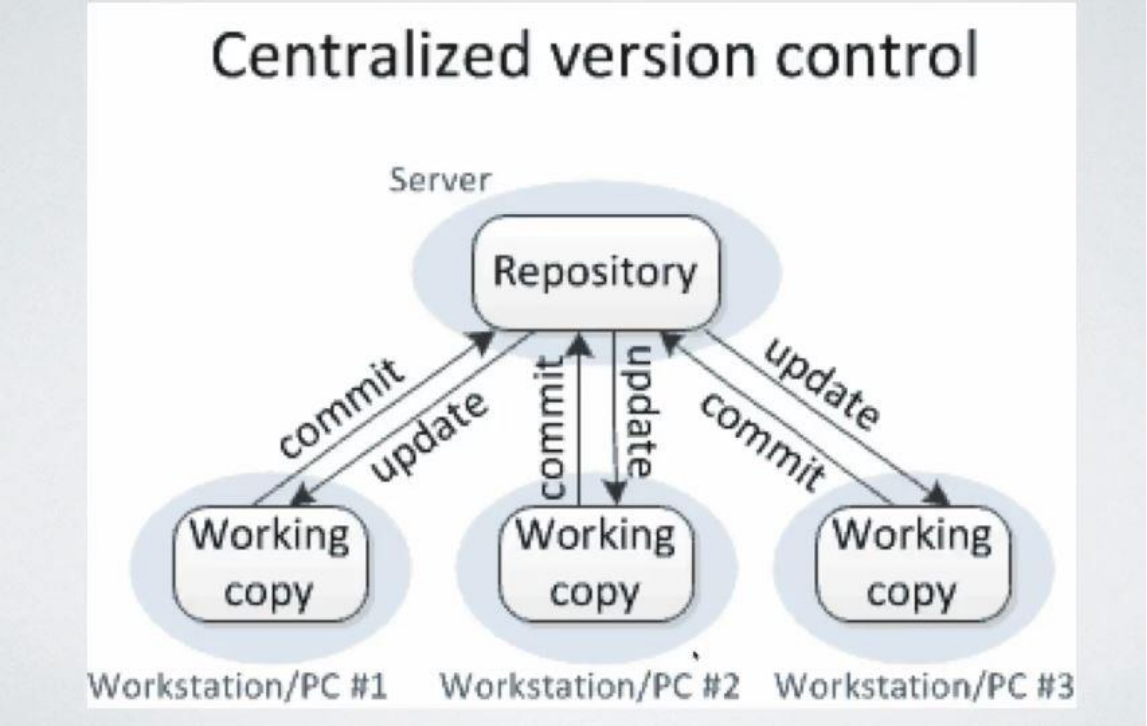
Aşağıdaki kriterleri sağlayan kendi yazılımlarını geliştirdiler

- Hızlı
- Kullanımı kolay
- Lineer olmayan geliştirme iş akışına uygun
- Tamamen DAĞITIK
- Büyük projeleri destekleyebilecek

CVS (CENTRALİZED VERSİON CONTROL)

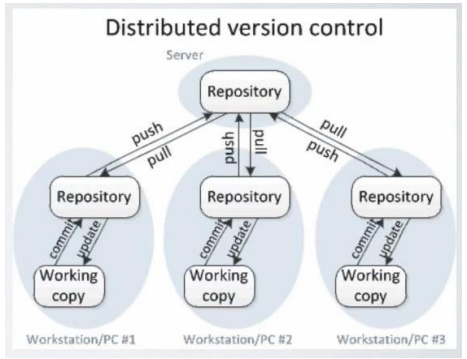
Tüm istemciler bir repoya yani depoya bağlı

Repo online olduğu sürece işlem yapılabilir.



DVCS (DISTRIBUTED VERSION CONTROL SYSTEM)

- Ana server'a sürekli bir bağlantı olmasına gerek yoktur
- Daha hızlı
- Network zorunlu değil
- Olası hatalara karşı daha dayanıklı
- Geliştiriciler birbirinden bağımsız olarak çalışabilir
- Yapılan değişiklikler kabul edilebilir veya reddedilebilir.



KİMLER GİT KULLANMALI

Değişiklikleri takip etmek isteyen her hangi biri

- Yapılan değişiklikler listesini görmek isteyen
- Versiyonlar arasındaki farklılıkları görmek isteyen
- Eski versiyonlara erişmek isteyen

Değiştirilen yapıları başkalarıyla paylaşmak isteyen her hangi biri

Programcı ve geliştiriciler

Resim, video, müzik dosyaları (text olmayan dosyalar) Git kullanımı için uygun değildir.

PDF, excel dosyaları uygun değildir.

GİT KURULUMU

Ücretsiz ve açık kaynaklıdır.

https://git-scm.com/downloads

Git'in kurulumu hem Windows hem de Mac OS X için oldukça kolay bir işlemdir. Her iki işletim sistemi için tek tıkla kurulum yapmanızı sağlayan kurulum sihirbazları vardır.

Git'in kurulumunun sorunsuz gerçekleştiğini teyid etmek için **Git Bash**'i açıp **git --version** komutunu yazın. Bu komut ekrana Git'in versiyon bilgisini basar.

BASİT OLARAK GİT İŞ AKIŞI

Versiyon kontrolünün en temel bileşeni repository denilen yapıdır.

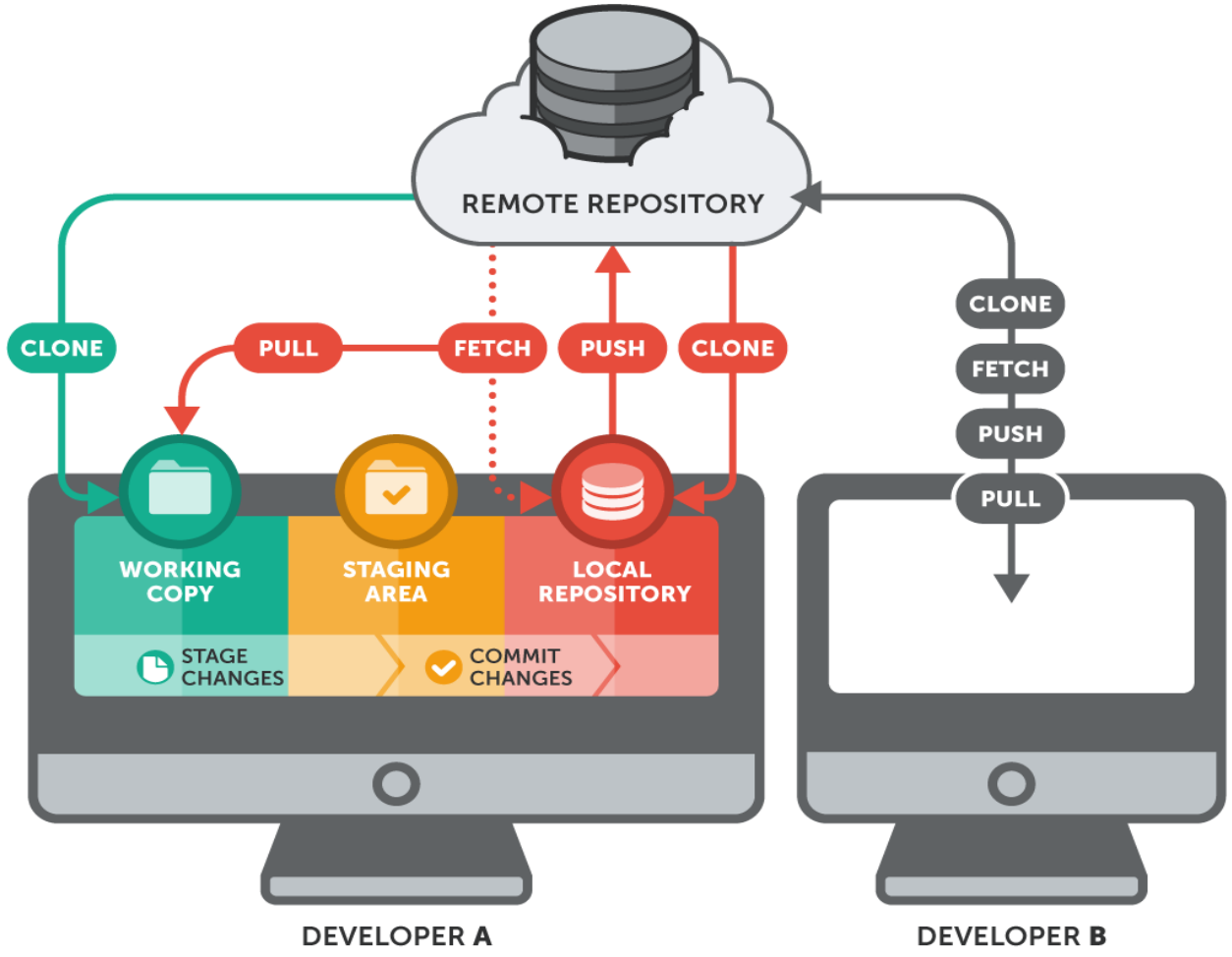
- Repository, dosyalarınızdaki tüm değişiklikleri ve bu değişiklikler ile ilgili ilave bilgileri (değişikliği kim, ne zaman yaptı ve değişiklik ile ilgili girilen açıklamalar) ayrı birer **versiyon** olarak kayıt altında tutan bir veri tabanıdır. Git tüm bu bilgileri genellikle dosya sisteminde gizli bir klasör olarak oluşturulan **.git** isimli klasör içinde bir dizi dosya olarak tutar.

Yerel veya uzak repositoryler olabilir.

- yerel için git init komutu
- uzak için git clone komutu

Dosyalarımızda yaptığımız değişiklikler belli bir noktaya ulaştığında, yapılan değişiklikler bütününü **commit** etmemiz gerekir.

BASIT OLARAK GIT İŞ AKIŞI



GİT ÇALIŞMA AŞAMALARI

Working copy: Projenizin ana klasörüne *Working Copy* veya *Working Directory* ismi verilir. Bu klasörde projenizde yer alan dosyaların ve klasörlerin bir kopyası bulunur. Versiyon kontrol sistemine projenizin herhangi bir versiyonunu Working Copy'nize kopyalamasını söyleyebilirsiniz, ancak bir anda Working Copy'nizde projenizin sadece bir versiyonu yer alır.

Değişikliklerinizin kayıt altına alındığı bir alan daha vardır ki buna **Staging Area** denir ve git'in en temel kavramlarından birisidir. Staging Area'yı, proje dosyalarımızdaki bir dizi değişikliği repository'ye göndermeden önce kayıt altında tuttuğunuz veri tabanı/alan olarak tanımlayabiliriz.

Staging area'da bulunan dosya ve klasörler commit komutu ile git veritabanına eklenir.

BASİT OLARAK GİT İŞ AKIŞI

Commit işlemi öncesinde yapılan değişikliklerin özetini **git status** ile görebiliriz.

Değiştirilen dosyalardan hangilerinin commit işlemine dahil olacağını belirleriz. Bu dosyaları **git add** komutu ile **staging area** kısmına taşırız.

Artık **git commit** methodu ile yapılan değişiklikler yeni bir versiyon olarak Git'de kayıt altına alınır.

Yapılan tüm commit özetini **git log** ile görürürz.

GİT ÇALIŞMA AŞAMALARI KODLARI

Henüz version kontrolü altında olmayan bir projenizi versiyon kontrolü altına almak için **git init** komutunu kullanırız.

git add dosya_adi veya git add . diyerek klasörde bulunan tüm dosyaları staging area'ya ekleriz

git commit –m «Initial Commit» komutuyla mesaj ekleyerek commit işlemini gerçekleştiririz.

git commit –a –m «ınitial Commit» komutu da hem ekleme hem commit işlemini aynı anda yapar.

GİT PROJESİNDE ÇALIŞMAYA BAŞLAYALIM

Dosya Durumları

- **Untracked (Takip Edilmeyen):** Bu dosyalar versiyon kontrolü altında olmayan veya sizin henüz versiyon kontrolü yapmak için git'e eklemediğiniz dosyalardır. Bu dosyalardaki değişiklikler siz dosyaları git'e eklemediğiniz sürece versiyon kontrolüne tabi değildir
- **Tracked (Takip Edilen):** Bu dosyalar ise git'in versiyon kontrolü takibi altında olan dosyalardır. Bu dosyalar üzerinde yapacağınız tüm değişiklikler git tarafından takip edilmektedir.

git status komutu ile değişiklikleri izleyebilirsiniz.

git log komutu ile yapılan commitleri listeleyebilirsiniz.

VERSİYON KONTROLÜNÜN ALTIN KURALLARI

Değişikliklerinizi commit etmeye karar verdiğinizde birbiri ile alakalı değişiklikleri tek bir commit olarak ele almaya özen gösterin. Birbiri ile alakalı olmayan değişiklikleri aynı commit ile versiyon kontrol sisteminde kayıt altına aldığınızda aşağıdakilere benzer sorunlar yaşama ihtimaliniz artacaktır.

Commit işlemi sırasında yazacağınız bilgilendirici bir mesaj hem ekibinizdeki diğer kişilerin hem de daha sonra kendinizin yapılan değişikliği daha rahat ve hızlı anlamanızı sağlayacaktır. Mesajınıza kısa bir özet satırı yazdıktan sonra bir sonraki satırda da değişikliğin nedeni ve içeriği hakkında bilgi verebilirsiniz.

İYİ BİR COMMİT ÖZELLİKLERİ

Commit'inizde sadece kavramsal olarak ilişkili değişiklikleri içermeye özen göstermelisiniz. Zaman zaman iki farklı konu veya sorun ile ilgili aynı anda veya çok kısa aralıklarla değişimli olarak çalışmak zorunda kalabilirsiniz. Bu şekilde yapılan bir çalışma sonrasında commit zamanı geldiğinde mümkün ise iki konu ile ilgili değişikliklerinizi bir defada commit etmek yerine iki defada ayrı ayrı commit edin. Bu çok zor oluyorsa kısa yoldan bir anda tek bir değişikliğe odaklanmayı da düşünebilirsiniz.

Tamamlanmamış değişikliklerinizi kesinlikle commit etmemeye özen gösterin. Eğer zaman zaman değişikliklerinizi kayıt altına almak istiyorsanız commit işlemi yerine Git'in **Stash** özelliğini/komutunu kullanabilirsiniz.

İYİ BİR COMMİT ÖZELLİKLERİ

Test edilmemiş değişiklikleri commit etmemeye özen gösterin. Bu öneri aslında bir önceki önerimiz ile pratikte aynı anlama geliyor.

Commit'leriniz kısa ve açıklayıcı mesajlar içermeli.

Son olarak da sık sık commit işlemi yapmayı alışkanlık haline getirmenizi önerebiliriz. Bu alışkanlık ile birlikte yukarıdaki maddeleri de yerine getirebilirseniz iş yapma şekliniz ve konsantrasyonunuz da olumlu yönde etkilenecektir.

Pomodoro Teknigi : https://onedio.com/haber/11-madde-ile-verimli-calisamama-derdini-sonsuza-kadar-bitiren-pomodoro-teknigi-628834

BRANCHİNG VE MERGİNG (DALLANMA VE BİRLEŞTİRME)

Günlük iş yapma şeklimizi etkileyip, yaptığımız işe daha farklı bakabilmemizi sağlar.

Branchlerin aktif kullanılmasıyla daha hızlı bir şekilde uygulama geliştirilebilir.

Branch; kendi yaşam döngüleri olan, çoğu zaman kısa veya uzun süreli eş zamanlı ilerleyen bağlamalardır.

BRANCHİNG OLMASAYDI

Birden fazla konu ile ilgili değişikliklerin tamamını tek bir bağlam ile yönetmeye çalışırsanız işler hızla sarpa saracaktır. Bu karmaşanın önüne geçmek için her bir değişiklik için projenizin tamamının farklı klasörlere kopyalamayı deneyebilirsiniz. Ancak bu durumda;

- Bu klasörler versiyon kontrolünde olmadığı için ekibin geri kalanı ile iş birliği yapmanız çok zorlaşacak
- Farklı değişiklikleri entegre etmek çok zor ve hataya açık bir işlem olacak

VERSİYON KONTROLÜNÜN ALTIN KURALLARI

Değişikliklerinizi commit etmeye karar verdiğinizde birbiri ile alakalı değişiklikleri tek bir commit olarak ele almaya özen gösterin. Birbiri ile alakalı olmayan değişiklikleri aynı commit ile versiyon kontrol sisteminde kayıt altına aldığınızda aşağıdakilere benzer sorunlar yaşama ihtimaliniz artacaktır.

Commit işlemi sırasında yazacağınız bilgilendirici bir mesaj hem ekibinizdeki diğer kişilerin hem de daha sonra kendinizin yapılan değişikliği daha rahat ve hızlı anlamanızı sağlayacaktır. Mesajınıza kısa bir özet satırı yazdıktan sonra bir sonraki satırda da değişikliğin nedeni ve içeriği hakkında bilgi verebilirsiniz.

Branch’ler git'in en güçlü özelliklerinden birisidir. Hızlı ve kullanımı kolay branching mekanizması git'in tasarımında ilk gününden itibaren ciddi bir gereksinim olarak ele alınmıştır. Branch'ler farklı bağlamlarda çalışmaktan kaynaklanabilecek karmaşanın önüne geçmek için biçilmiş kaftandır. Branch'leri bug fix'ler, yeni özellikler üzerinde çalışmak veya deneysel özellikleri geliştirmek için bol bol kullanın.

BRANCH'LER İLE ÇALIŞMAK

Git projeyi oluşturunca varsayılan olarak sizin için master isimli branch oluşturur.

git branch test komutu test isimli bir branch oluşturulur. Ama aktifleştirmez.

git branch komutu ile var olan branchleri görebilirsiniz.

git status ile projenin durumuna baktık, commit edilmemiş bir değişiklik var ve bunu yeni branch'e eklemekte şüpheliyiz. Ozaman bunu commit mi edelim yoksa göz ardı mı?

VERSİYON KONTROLÜNÜN ALTIN KURALLARI

Değişikliklerinizi commit etmeye karar verdiğinizde birbiri ile alakalı değişiklikleri tek bir commit olarak ele almaya özen gösterin. Birbiri ile alakalı olmayan değişiklikleri aynı commit ile versiyon kontrol sisteminde kayıt altına aldığınızda aşağıdakilere benzer sorunlar yaşama ihtimaliniz artacaktır.

Commit işlemi sırasında yazacağınız bilgilendirici bir mesaj hem ekibinizdeki diğer kişilerin hem de daha sonra kendinizin yapılan değişikliği daha rahat ve hızlı anlamanızı sağlayacaktır. Mesajınıza kısa bir özet satırı yazdıktan sonra bir sonraki satırda da değişikliğin nedeni ve içeriği hakkında bilgi verebilirsiniz.

Branch'ler git'in en güçlü özelliklerinden birisidir. Hızlı ve kullanımı kolay branching mekanizması git'in tasarımında ilk gününden itibaren ciddi bir gereksinim olarak ele alınmıştır. Branch'ler farklı bağlamlarda çalışmaktan kaynaklanabilecek karmaşanın önüne geçmek için biçilmiş kaftandır. Branch'leri bug fix'ler, yeni özellikler üzerinde çalışmak veya deneysel özellikleri geliştirmek için bol bol kullanın.

Tam anlamıyla bitirmediğiniz ve test etmediğiniz bir değişikliği asla commit etmeyin. Üzerinde çalışacağınız değişiklikleri planlarken bu değişiklikleri mümkün olduğunca küçük parçalar halinde ele almaya özen gösterirseniz yaptığınız değişiklikleri kayıt altına almak için henüz tamamlanmamış değişiklikleri commit etmek zorunda kalmazsınız. Buna rağmen ara safhada kayıt altına almak istediğiniz değişiklikler olursa Git'in **Stash** özelliğini kullanabilirsiniz.

DEĞİŞİKLİKLERİ GEÇİCİ OLARAK KAYDETMEK

Tam bitmemiş, commit etmek istemediğiniz ama kaybetmek de istemediğiniz değişikleri korumak için git stash komutu kullanılabilir.

git stash sonrasında aktif branch üzerinde bekleyen bir değişiklik kalmaz.

git stash list komutu ile tüm stashleri listeleyebilirsiniz.

git stash pop : Listede en üstte bulunan değişikliği geri yükler ve listeden siler.

git stash apply stash@{0} : Değişikliği yükler ama listeden silmez.

git stash drop stash@{1} : Değişikliği listeden siler.

STASH KULLANMA DURUMLARI

Aktif bir branch'i temiz duruma getirmek için kullanılır. Ayrıca;

- Farklı bir branch'i aktif hale getirmeden önce
- Remote Repository değişikliklerinizi yerel diskinize indirmeden önce
- Branch'inizi merge etmeden önce

.GİTİGNORE DOSYASI

- Bazen çalışma dizinimizdeki bazı dosyaların git ile takip edilmemesini veya değişikliklerin kaydedilmemesini isteyebiliriz.
- Bunun için .gitignore dosyası kullanılır.
- Bu dosyaya doğrudan isimler veya basit kurallar ekleyebiliriz.

- Kullandığınız geliştirme araçlarına bağlı olarak hangi dosyaların göz ardı edilebileceği ile ilgili GitHub'ın yayınladığı derlemeye göz atabilirsiniz.
- https://github.com/github/gitignore

BASİT BİR BRANCHİNG AKIŞI

Bir web sitesi üzerinde çalışıyorsunuz,

Yeni bir özellik eklemek için branch oluşturdunuz (git branch yeniozellik, git checkout yeniozellik)

Bu yeni branch üzerinde değişiklik yapmaya başladınız.

Bu sırada web sitesinde çok büyük bir sorun tespit edildi ve acilen düzeltilmesi gerekiyor. Ne yapacağız !!!

Son stabil branche geri dönülür. (git checkout master)

Sorunu çözmek için yeni bir branch oluşturulup, değişiklikler yapılır.(git branch loginsorunu, git checkout loginsorunu)

Sorun çözüldüğünde commit ile işlemler kaydedilir.

Son stabil branche geri dönülür. (git checkout master)

Sorun çözülen branch master branche merge edilir. (git merge loginsorunu)

Kaldığınız yerden devam edebilirsiniz (git checkout yeniozellik)

CHECKOUT, HEAD KAVRAMLARI

Git'de bir branch otomatik olarak o branch için yaptığınız son commit işlemine bir işaretçi tutar ve hangi dosyaların o branch'e ait olduğunu bilir.

Herhangi bir anda bir proje için tek bir branch **aktif** olabilir. Bu branch'e **HEAD** denir ve Working Copy içindeki (Working Copy'yi projenizin yerel diskinizdeki dosyalarının tamamı olarak düşünebilirsiniz) dosyalar aktif olan branch'e yani **HEAD**'e aittir.

Diğer branch'lerinizdeki dosyalar diskiniz üzerinde değil Git'in veri tabanında (.git klasörü içinde özel bir formatta) bulunur.

Farklı bir branch'i aktif hale getirmek için **git checkout** komutu kullanılır. Bu durumda Git otomatik olarak sizin için iki şey yapar

- Aktif hale getirdiğiniz branch'i **HEAD** yapar ve
- Aktif hale getirdiğiniz branch'e ait dosyaları Git veri tabanınızdan yerel diskinize kopyalar ve önceki branch'e ait dosyaları diskinizden kaldırır. Yani Working Copy'nize yeni branch'e ait olan dosyaları koyar.

GİTHUB'DAN PROJE CLONE'LAMAK

Github üzerindeki bir projeyi local repomuza alabilir, güncelleyebilir ve eğer yetkimiz var ise tekrar github üzerinde merge işlemi yapabiliriz.

git clone github_proje_linki → projeyi localimize indirir.

Projedeki tüm commitleri branchler, görebiliriz.

Localde değişiklikler yapıp commit edebiliriz.

git remote –v ile projede tanımlı uzak repoları görebiliriz.

git push remote_repo_kisa_adi/remote_repo_linki master(hangi branche gönderilecek)

Localde yapılan değişiklikler github üzerinde de gerçekleştirilir.

GİT ALİAS KAVRAMI

Uzun komutlara kısa takma isimler takabiliriz.

git log –all –graph –decorate –oneline

Bu komutu belirten bir komut oluşturubiliriz.

git config –global alias.hist «log –all – graph – decorate –oneline

git hist

GİT MERGEDİFF ARAÇLARI

Görsel olarak farkları görmek ve değişiklikleri birleştirmek için ekstra bir yazılım kullanalım

P4Merge bu araçlardan sadece bir tanesi, ücretsiz ve güzel

Git'e bunu tanımlamamız için;

git config –global merge.tool p4merge

git config –global mergetool.p4merge.path «Bilgisayardaki yeri»

git config –global diff.tool p4merge

git config –global difftool.p4merge.path «Bilgisayardaki yeri»

git config –global –e → Default editörle tüm ayarları açar

DOSYA KARŞILAŞTIRMA

Dosyalarımızın belli durumlardaki hallerini karşılaştırmak isteyebiliriz.

Working Directory ile Staging Area (git diff)

Working Directory ile Repository(son commit) (git diff HEAD)

Staging area ile Repository (son commit) (git diff –staged HEAD)

Tek bir dosya karşılaştırması yapmak için (git diff --DosyaAdi)

İki commit arasındaki farklar için (git diff id1 id2)

MERGE ÇEŞİTLERİ

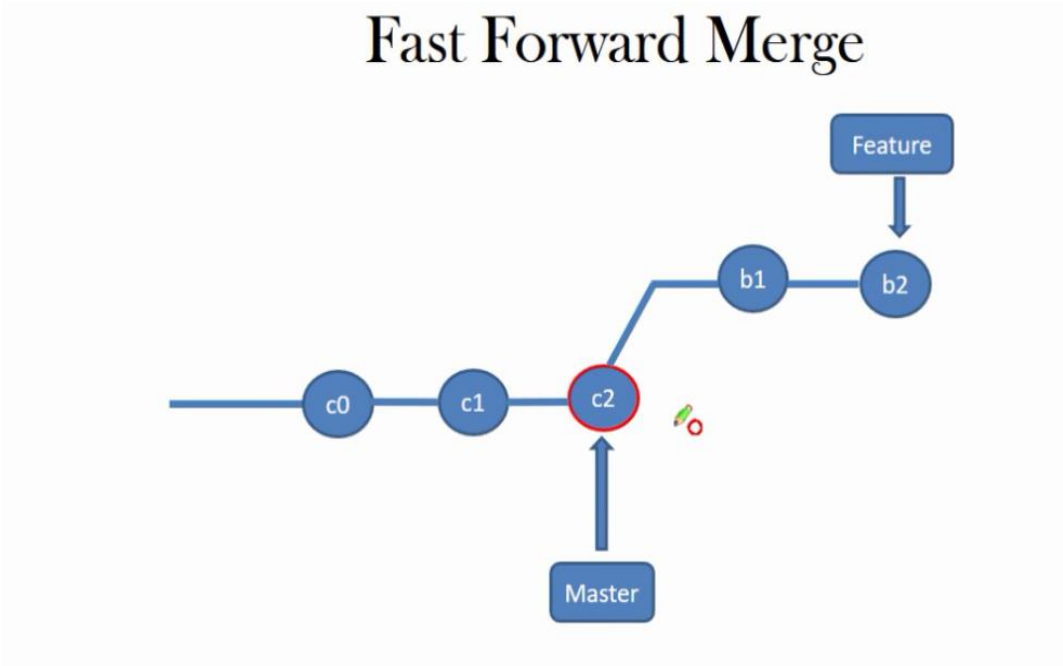
- Branch oluştururken git branch yeniBranchAdi diyebiliriz.
- Branch değiştirmek için git checkout branchAdi
- Tüm branchler için git branch –a
- Bir branchi silmek için git branch –d branchAdi
- Yeni bir branch oluşturup ona geçiş yapmak için git checkout –b yeniBranch

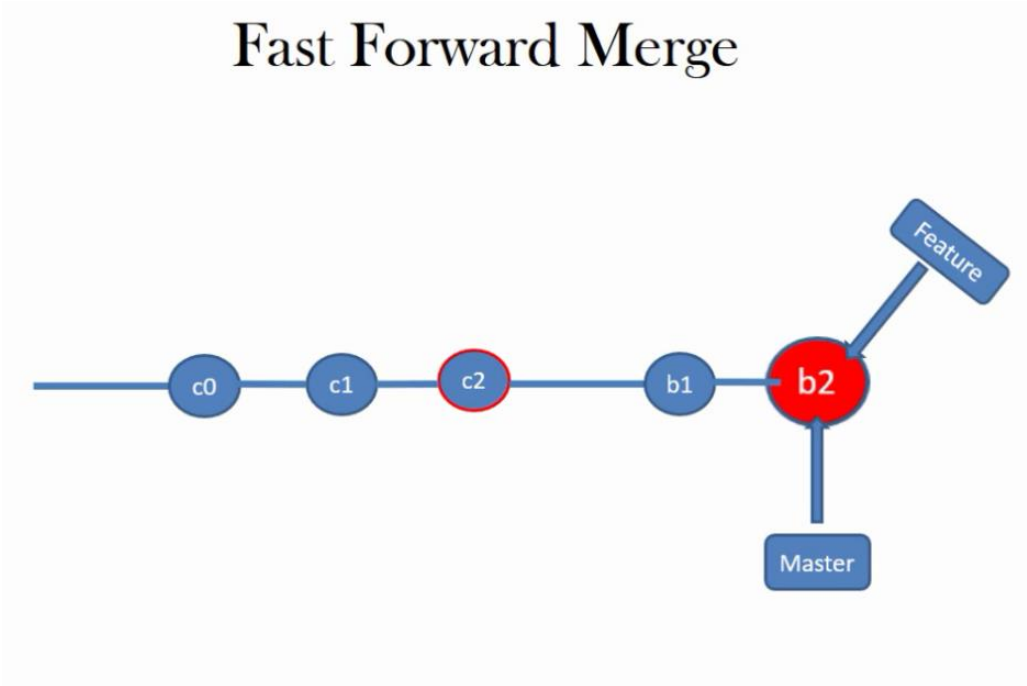
- Yeni branchteki değişiklikleri master branch ile birleştirmek(merge) için master'a geçtikten sonra
- Git merge branchAdi demek yeterlidir.

- Merge çeşitlerine sırasıyla bakalım..

FAST FORWARD MERGE

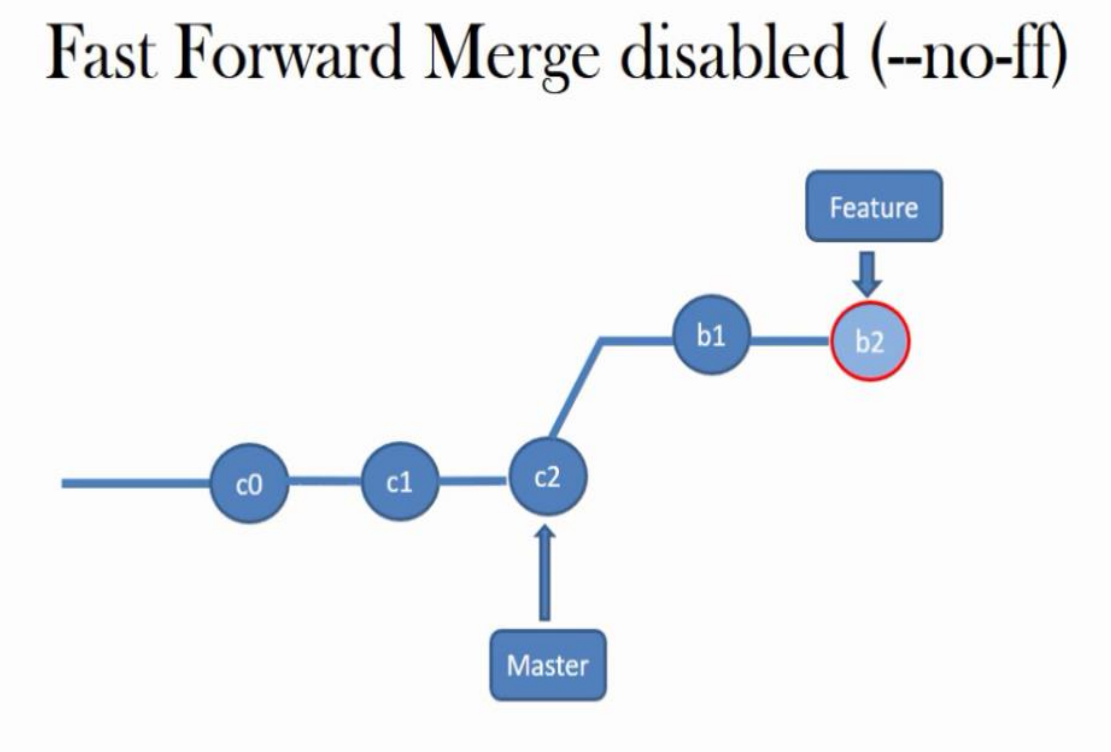
Eğer master branchta yeni branch oluşturulduktan sonra her hangi bir değişiklik olmamışsa olur.

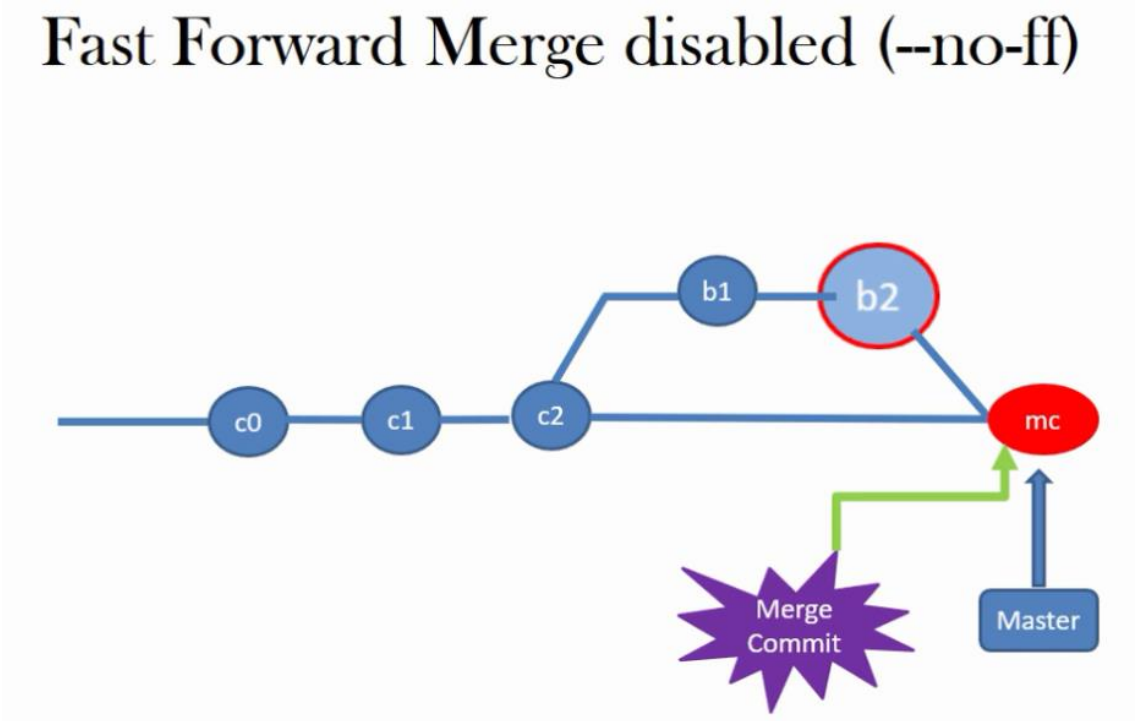




NO FAST FORWARD MERGE

- Eğer master branchta yeni branch oluşturulduktan sonra her hangi bir değişiklik olmamışsa olur.
- Yeni branchta yapılan değişiklikler farklı bir düzlem üzerinde ilerler.
- Git merge branchAdi –-no-ff

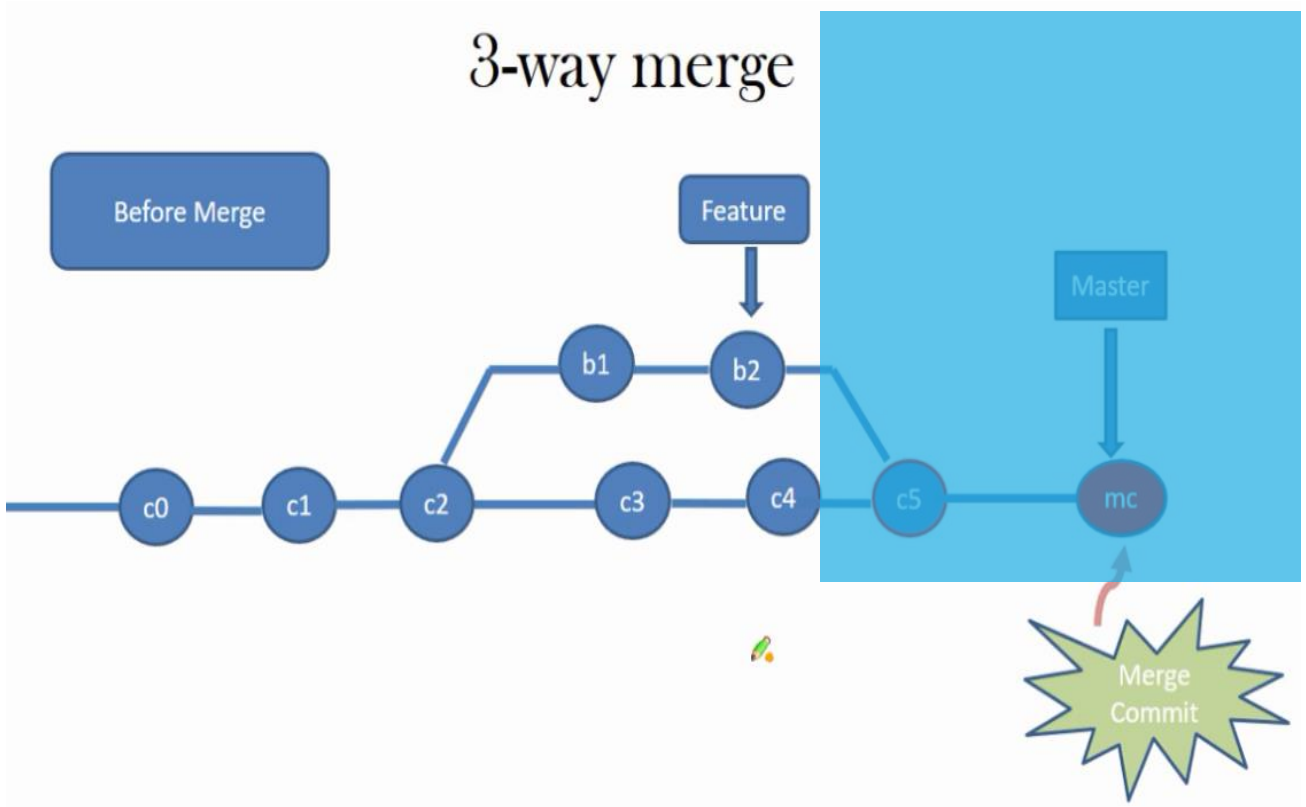


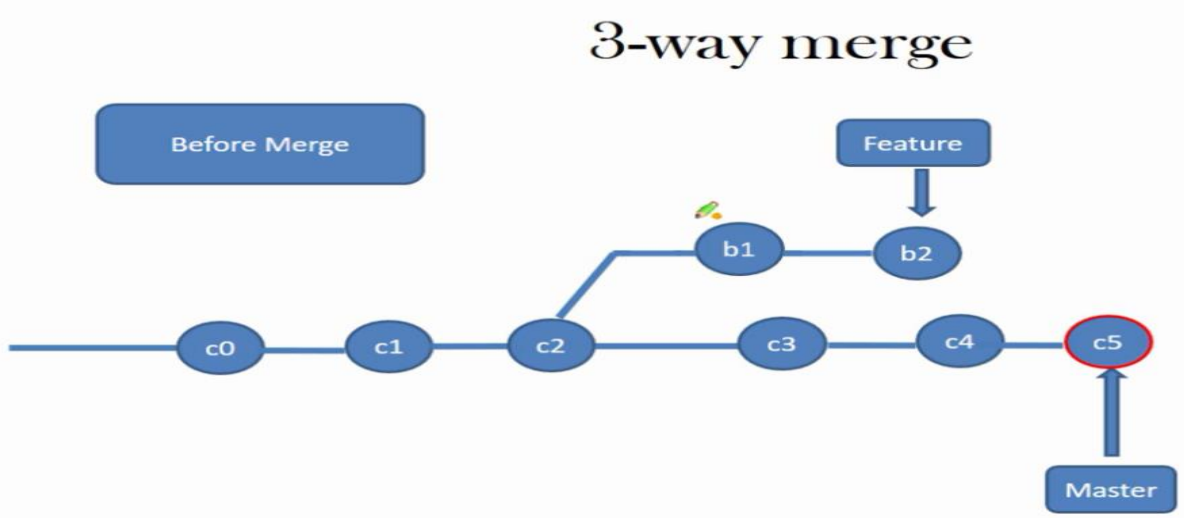


3 WAY MERGE (AUTOMATİC MERGE)

Eğer master branchta yeni branch oluşturulduktan sonra her hangi bir değişiklik olmuşsa olur.

Git otomatik olarak birleştirme yapar.





MERGE CONFLİCTS ÇÖZÜMÜ

- İki farklı branchte aynı dosyanın aynı satırında değişiklik yapılmış olabilir.
- Bu iki branchi birleştirmeye çalıştığımızda çatışma (conflict) oluşur ve bu çözülene kadar merge işlemi askıda kalır.
- Conflict oluşan dosyada düzeltme işlemini yapmak bizim sorumluluğumuzdadır.
- Git mergetool ile bu düzeltme işlemi yapılıp commit edilir.
- Bu sayede merge işlemi de başarıyla tamamlanmış olur.

- Bu işlemler sonrasında git bizim için .orig dosyaları oluşturur.
- Bunları önlemek için ya .gitignore dosyasına bu uzantıdaki dosyaları takip etmemesini söyleriz.
- Ya da aşağıdaki komut ile bunların oluşmasını engelleriz.
  - Git config –global mergetool.keepBackup false

GİT REBASE KAVRAMI

- Master branchteki değişikliklerin yeni branche aktarılmasıdır.
- Böylece fast forward merge yapılabilir.