Versiyon Kontrol Sistemi

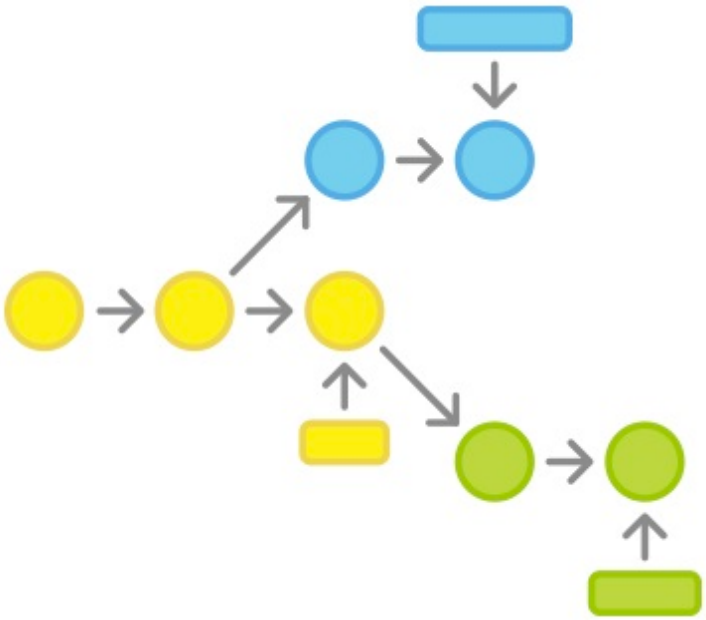
Versiyon Kontrol Sistemi

Nedir?

- Versiyon kontrolü nedir ve bizi neden ilgilendirmeli? Versiyon kontrolünü bir dosya veya bir küme dosyadaki değişiklikleri takip edebilmek için uyguladığımız bir yöntem olarak tanımlayabiliriz.
- Git gibi sistemler tüm bu değişikliklerin tarihçesini ve içeriğini elektronik olarak bizim için takip ederek kayıt altına almamızı sağlayan veri tabanları olarak düşünülebilir.
- Bu sistemleri kullanarak herhangi bir anda üzerinde çalıştığınız dosyaların o anki hallerini kaydedebilir, daha sonra da isterseniz bu dosyaların kaydedilmiş ve kontrol altına alınmış herhangi bir haline geri dönebilirsiniz.

Versiyon Kontrol Sistemi

Nedir?



- Dosyaların kayıt altına alınmış herhangi bir andaki hallerine **versiyon** diyoruz
- Versiyon kontrolünü, kullandığınız programlama dili, yardımcı programlama kütüphaneleri (framework), dosya tipi veya işletim sisteminden bağımsız bir yaklaşım olarak düşünmelisiniz. Çünkü versiyon kontrolü;
  - HTML dosyalar için kullanılabileceği gibi, mimari tasarım amaçlı proje dosyaları ve iPhone uygulaması kaynak kodunuz için de kullanılabilir
  - Dosyalarınız üzerinde çalışırken hangi işletim sistemini veya hangi programları kullandığınız ile ilgilenmez (Sublime Text, Notepad, Visual Studio, Word, AutoCAD)
- Versiyon kontrol sistemleri en basit anlamda dosyalarınızdaki değişikliklerin tarihçesini takip edip kayıt altında tutan sistemlerdir.

Versiyon Kontrol Sistemi

Neden İhtiyacımız Var?

- Uyumlu ekip çalışması
- Versiyonların düzgün bir şekilde takip edilebilmesi
- Önceki Versiyonlara Geri Dönebilme
- Dosyalarınızın neden değiştiğini anlama
- Yedekleme

Versiyon Kontrol Sistemi

Uyumlu ekip çalışması

Herhangi bir versiyon kontrol sistemi kullanmadığınızda beraber çalıştığınız diğer kişiler ile aynı dosyalar üzerinde çalışabilmek için muhtemelen herkesin erişimine açık paylaşımlı bir klasör kullanmak zorunda kalacaksınız.

Bu tür bir senaryoda kullanılan yazılımların çoğu değiştirilen dosyaya **kilit** koyar ve başka birisi aynı dosyayı düzenlemek istediğinde;

- Kullandığı programa bağlı olarak dosya yazma korumalı olarak salt okunur modda (readonly) açılır veya
- Değişiklikler kaydedilmek istendiğinde hata verir

Bu tür bir çalışma hem çok zahmetli hem de hatalara açıktır. Örneğin bir dosyanın en son geçerli versiyonunun nerede olduğunun takip edilmesi gibi çözüm bulunması gereken sorunlar ile uğraşmak zorunda kalırsınız.

Üzerinde çalıştığınız dosyada sizden önce başkasının değişiklik yapıp yapmadığından haberiniz yoksa hatalı içerik üretme ihtimaliniz vardır.

Versiyon Kontrol Sistemi

Uyumlu ekip çalışması

- Versiyon kontrol sistemi kullanıldığında ise ekibinizdeki herkes özgür bir şekilde istediği dosyalar üzerinde güvenli bir şekilde istediği değişikliği yapabilir.
- Herkes değişikliklerini tamamladıktan sonra da tüm değişiklikler versiyon kontrol sistemi kullanılarak sağlıklı bir şekilde **merge** (*birleştirme*) edilebilir.

Versiyon Kontrol Sistemi

Versiyonların düzgün bir şekilde takip edilebilmesi

Üzerinde çalıştığınız bir dosyanın veya bir dizi proje dosyasının zaman içinde farklı versiyonları oluşur ve bu versiyonların kayıt altına alınması gerekir. Bu sorumluluk genelde çok zahmetli ve sıkıcı bir iş ve süreçtir.

- Sadece değişen dosyalar mı yoksa bir projedeki tüm dosyaların versiyonları mı kaydedilmeli?
  - Bir sürü dosya içinden sadece değişen dosyaların belirlenmesi zordur
  - Her seferinde dosyaların hepsinin teker teker kaydedilmesi durumunda ise ihtiyaç duyulandan daha fazla disk alanı kullanılır

- Dosyalara verilecek isimler tam bir baş ağrısına dönüşebilir.
  - index.php
  - index1.php
  - index1_son.php
  - index1_gercekten_en_son.php şeklinde dosya isimleri üretmek zorunda kalabilirsiniz.

- Belki de canınızı en çok sıkacak şey projenizin iki versiyonu arasında tam olarak ne tür farkların olduğunu sağlıklı bir şekilde bilme şansınızın olmaması olacaktır

Versiyon Kontrol Sistemi

Versiyonların düzgün bir şekilde takip edilebilmesi

- Versiyon kontrol sistemi kullandığınızda sizin çalıştığınız disk alanında proje dosyalarının sadece bir versiyonu bulunur, bu dosyaların daha önceki halleri versiyon kontrol sisteminin denetimindedir.
- Bu sayede istediğiniz zaman önceki versiyonlara geri dönebilir, versiyonlar arasındaki farklılıkları rahatlıkla inceleyebilir ve versiyonları kaydederken eklediğiniz ilave bilgileri ve yorumlarınızı rahatlıkla görebilirsiniz.

Versiyon Kontrol Sistemi

Önceki Versiyonlara Geri Dönebilme

- Dosyalarınızın veya aslında tüm projenizin daha önceki versiyonuna geri dönebilme imkanın size ciddi anlamda özgürlük sağlar;

  dosyalarınızı ve projenizi istediğiniz gibi değiştirme özgürlüğü

- Yaptığınız değişiklikler projenizi çöpe döndürdüyse, geliştirdiğiniz bir işlev tam istediğiniz gibi olmadıysa veya müşteriniz veya patronunuz geliştirdiğiniz bir işlevi artık istemediğine karar verirse projenizin önceki temiz haline çok hızlı ve rahat bir şekilde dönebilirsiniz.

Versiyon Kontrol Sistemi

Dosyalarınızın neden değiştiğini anlama

- Versiyon kontrol sistemleri değişikliklerinizi tamamlayıp **commit** etmek istediğinizde **comment** adı verilen açıklamalar girmenizi isterler.
- Bu comment’ler sayesinde projenizin herhangi bir versiyonundaki değişikliklerin nedenlerini de kayıt altına alıp ihtiyaç halinde geri dönüp inceleyebilirsiniz.
- Git'de commit işlemi yapılırken comment (yorum metni) girilmesi zorunludur

Versiyon Kontrol Sistemi

Yedekleme

- Git gibi dağıtık versiyon kontrol (DVCS) sistemlerinin yan etki olarak sağladığı faydalardan birisi de yedeklemedir.
- Git sayesinde aynı projede çalışan herkesin kendi bilgisayarında projenin tam bir tarihçesi tutulur.
- Merkezi versiyon kontrol sistemi sunucusunda bir sorun oluştuğunda takımdaki herhangi birinin kendi diskindeki projeyi sunucuya geri yüklemesi yeterlidir.
- Diğerleri de kendi bilgisayarlarındaki proje dosyalarını geri yüklenen proje dosyaları ile senkronize edebilirler.

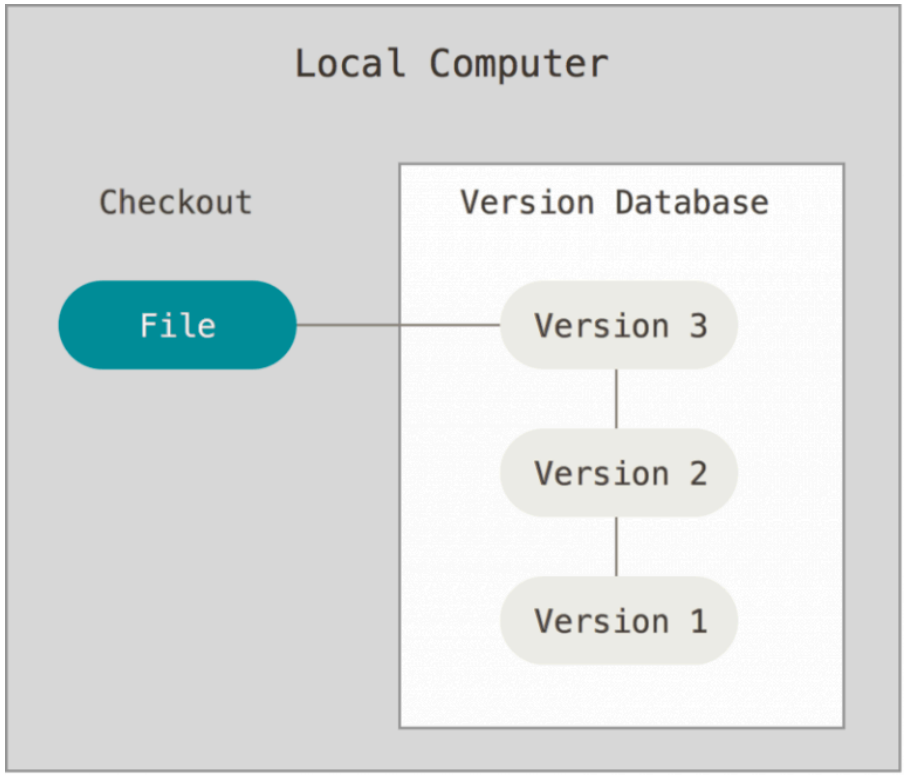
Versiyon Kontrol Sistemi

Repository(depo)

- VKS'ler ekip halinde çalışmayı sağlar. Bunu sağlayabilmesi için yapılan değişikliklerin ve kaynak kodun tarihçesinin ortak bir yerde yani bir sunucuda (remote server) saklanması şarttır. Birden fazla kaynak dosyanın ve tarihçesinin bulunduğu yapı **repository** (depo) olarak isimlendirilir.

Versiyon Kontrol Sistemi

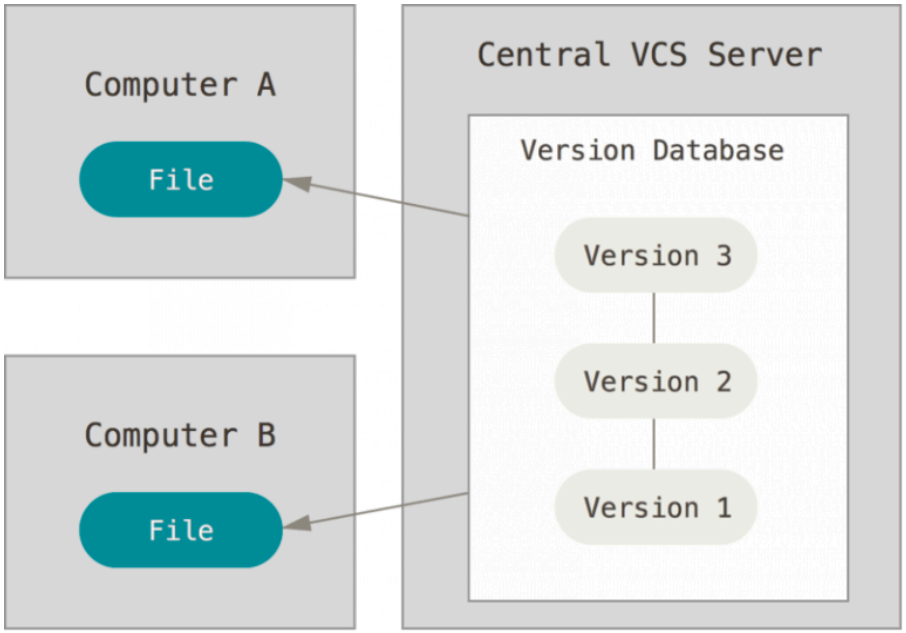
Yerel Sistemler



- Bu tür sistemler herhangi bir uzak sunucu (remote server) ile çalışmaz.
- Kaynak kodun tarihçesi de sadece yerel sistemlerde yapılır; yerelde versiyonlama yapar.
- Kişisel işler için kullanılabilir fakat ekip halinde çalışma imkanı sunmazlar.

Versiyon Kontrol Sistemi

Merkezi Sistemler(Central)



- Merkezi sistemler bu konuda en basit çözümlerden biridir. Kaynak kod ve tarihçesi yani **repository** merkezi bir sunucuda yer alır.
- Şekilden de anlayacağınız gibi istemcilerde kaynak kodun kopyası yer alır fakat kaynak kodun tarihçesi yer almaz.
- Kodların tarihçesi sadece merkezi bir sistemde bulunur.

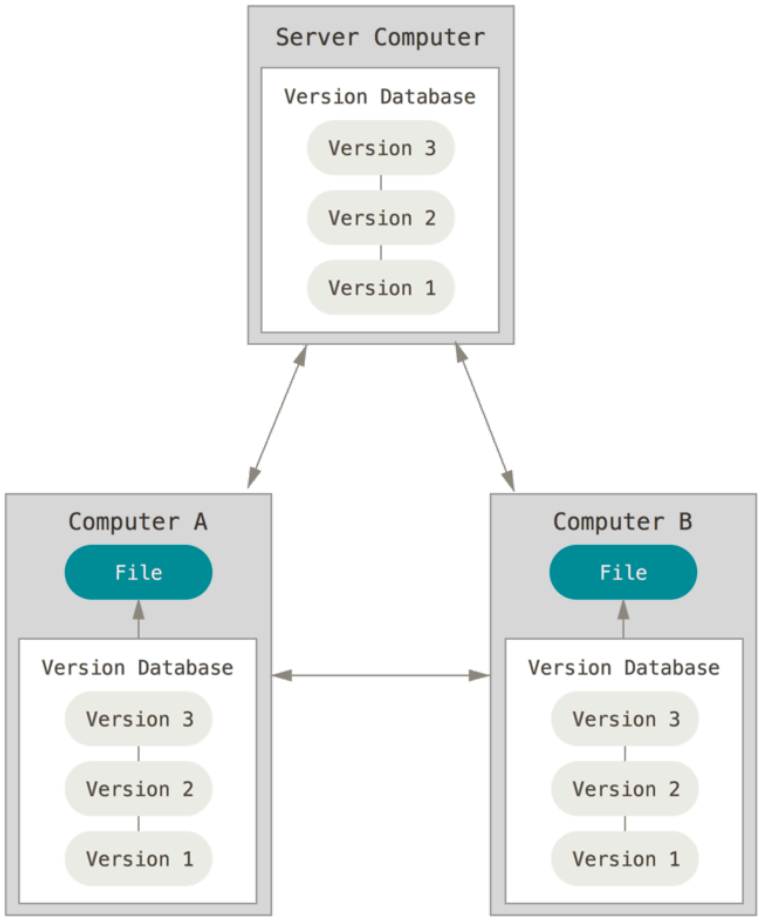
Versiyon Kontrol Sistemi

Merkezi Sistemler(Central)

- Kaynak kod üzerinde değişiklik yapmak isteyen bir kişi öncelikle değiştirmek istediği dosyayı alır. Daha sonra üzerinde geliştirme yaptıktan sonra dosyanın son halini merkezi sunucuya gönderir.
- Sadece bir sunucunun **repository** olduğu ve merkezi bir konumda yer aldığı için bu tür sistemlere merkezi (central) sistemler denir.
- Git'den önce en popüler VKS olan *SVN*, *CVS* ve diğer bir çok versiyon kontrol sistemi merkezi sistemlerdir.
- Bu sistemlerde eğer ki merkezi sunucunun başına birşey gelirse ve yedeğiniz yoksa kodun bütün tarihçesini kaybedersiniz. Yerel bilgisayarlarda kodun sadece son hali yer alır. Bu yapıdan dolayı **repository** kaybeden birçok ekip bulunmaktadır.

Versiyon Kontrol Sistemi

Dağıtık Sistemler(Distributed)



- Dağıtık sistemler, merkezi sistemlerin aksine kaynak kodu ve tarihçesi merkezi bir yerde toplamak yerine kod üzerinde değişiklik yapan her istemcinin sistemlerine de dağıtır.
- Kaynak kodu ve tarihçesi sadece bir sunucuda değil aynı zamanda geliştiricilerde de yer alır.
- Yani her geliştiricinin kendi bilgisayarında yerel bir sunucudakinin birebir kopyası yani **repository** bulunur.

Versiyon Kontrol Sistemi

Dağıtık Sistemler(Distributed)

- Bu yaklaşımda kendi bilgisayarınızda bir **repository** olduğu için işlemler uzak sunucuda değil yerelinizde yapılır.
- Daha sonra yerelinizdeki işlemler uzak sunucuya gönderilir. Aynı projede çalıştığınız diğer kişiler de uzak sunucudan sizin yaptığınız işlemleri alıp yerelinde projenin son haline ulaşır.
- Dağıtık sistemlerin en büyük avantajı sürekli merkezi bir sunucuya ihtiyacı olmadığı için bazı işlemleri hızlı bir şekilde gerçekleştirebilir. Kendi sisteminize özgü uzak sunucudan farklı işlemler yapılmasına da imkan sağlar.
- Ayrıca dağıtık bir yapıda birden fazla uzak sunucu ile de çalışabilirsiniz. Kaynak kodunuzun bir versiyonunu bir **repository**'de saklarken farklı bir versiyonunu farklı bir **repository**'de saklayabilirsiniz.
- Uzak sunucudaki **repository**'yi kaybetmeniz durumunda her geliştiricinin bilgisayarında aynı *repository* olacağı için tarihçeyi kurtarmanız oldukça kolaydır.
- Git, Mercurial ve Bitkeeper bu sistemlere örnek verilebilir.

Versiyon Kontrol Sistemi
Git

Git 2005 yılında, başta Linus Torvalds olmak üzere Linux çekirdeğini de kodlayan ekip tarafından Linux kaynak kodunu versiyon kontrolü altında tutmak ve kendi iş akışlarını düzenlemek için geliştirilmiştir

Linux'un kaynak kodu 1991-2002 yılları arasındaki dönemde manuel olarak dosyaların paylaşılması şeklinde yönetiliyordu. 2002 yılında Linux geliştiricileri normalde ücretli olan ancak açık kaynak projeler için ücretsiz lisanslama modeli sunan BitKeeper isimli dağıtık versiyon kontrol sistemini kullanmaya başladılar. 2005 yılında BitKeeper'ın ücretsiz sağladığı lisansı geri çekmesi üzerine Linus Torvalds ve Linux ekibi kendi dağıtık versiyon kontrol sistemini geliştirmeye karar verdiler.

Linux ekibi BitKeeper ile olan deneyimlerini de dikkate alarak öncelikli olarak aşağıdaki kriterleri sağlayan kendi yazılımlarını geliştirmeye başladılar

- Hızlı
- Kullanımı kolay
- Lineer olmayan geliştirme iş akışına uygun (branching)
- Tamamen dağıtık
- Büyük projeleri destekleyebilecek

2005 yılından bugüne Git gelişmeye devam ediyor. Git'e yeni eklenen özelliklere rağmen Git bugün bile yukarıdaki kriterlerden taviz vermeden milyonlarca yazılım geliştiricinin hayatını kolaylaştırmaya devam ediyor.

Git

Komut satırı mı yoksa görsel arayüz mü?

- Git ile çalışmak için git'in kendi **komut satırı arayüzünü** (Git Command Line Interface) veya görsel kullanıcı arayüzü olan masaüstü uygulamalar (SourceTree, Tortoise Git, Tower, Fork veya GitHub) kullanabilirsiniz.
- Git ile çalışırken görsel arayüzü olan bir uygulama kullanmanız üretkenliğinizi arttırıp Git'in çok sayıdaki karmaşık komutuna daha hızlı ve kolay erişmenizi sağlar.
- Diğer yandan Git'in komut satırı arayüzünü kullanmanız Git ile ilgli daha ayrıntılı bilgilenmenizi ve 3. parti uygulamalara bağımlı kalmadan Git ile çalışabilmenizi sağlar.

Git

Kurulum

- İşletim sisteminiz Windows ise git ile çalışmak için "https://gitforwindows.org" adresini kullanabilirsiniz.
- Bu adreste yer alan kurulum uygulamasını indirip çalıştırmalısınız. Kurulum adımları sırasında karşınıza çıkacak olan ekranlarda varsayılan ayarları seçili olarak bırakarak kurulumunuzu tamamlayabilirsiniz.
- Kurulum tamamlandıktan sonra Windows Başlangıç menüsünden *Git* klasörü altındaki **Git Bash** uygulamasını çalıştırıp Git'in komut satırı arayüzünü kullanmaya başlayabilirsiniz.
- Git'in kurulumunun sorunsuz gerçekleştiğini teyid etmek için **Git Bash**'i açıp **git --version** komutunu yazın. Bu komut ekrana Git'in versiyon bilgisini basar.

Git
Konfigürasyon

- Ayarlar için Git bize **git config** isimli bir araç/komut sunar. Git ayarlarını bir defa yapmanız yeterli olacaktır.
- Bu ayarları istediğiniz zaman değiştirebilirsiniz.
- Git ayarlarınız aşağıda belirtilen üç konumda kaydedilir ve hiyerarşik olarak bu konumlardan yüklenir
    - 1.Seviye (/etc/gitconfig dosyası) : Tüm kullanıcı ve projeler için geçerli olan ayarlar bu dosyada kaydedilir. **git config** komutunu **--system** seçeneği ile çalıştırırsanız ayarlar bu dosyada kaydedilecek ve bu dosyadan okunacaktır
    - 2.Seviye (/.gitconfig dosyası) : Sadece sizin kullanıcınız için tanımlanan ayarların kaydedildiği dosyadır. **git config** komutunu **--global** seçeneği ile çalıştırısanız ayarlar bu dosyaya kaydedilecek ve bu dosyadan okunacaktır
    - 3.Seviye : Proje klasörünüzün (projenizin Git ile versiyon kontrolüne alınmış olması gerekiyor) altında yer alan **.git/config** dosyasında ise proje bazındaki git ayarlarınız yer alır.
    - Git, ayarlarınızın değerini belirlemek için bu üç konumdaki dosyaları 3. seviye, 2. seviye ve 1. seviye sıralaması ile hiyerarşik olarak okur. Belirli bir ayar'a ilişkin değere ilk hangi seviyede rastlandıysa o seviyedeki değer dikkate alınır diğer seviyelerdeki değerler dikkate alınmaz.
    - Windows'da global (**git config --global** komutu) git ayarlarınız Windows'un $HOME klasörü altında yer alan (genellikle C:\Documents and Settings$USER) **.config** dosyasında yer alır. Proje seviyesindeki ayarlarınız ise OS X'de olduğu gibi **[Projenizin Ana Klasörü].git\config** dosyasında kayıt altına alınır.

Git

Konfigürasyon

- Git ayarlarından en önemli olanları kullanıcı adınız ve email adresinizdir.
- Git, ayar olarak tanımladığınız değerleri **commit** vb işlemlerde otomatik olarak kullanır.
- Bu ayarların değerini belirlemek için komut satırında aşağıdaki komutları çalıştırıyoruz
- git config --global user.name “Adınız Soyadınız”
- git config --global user.email “email@adresiniz.com”

Git

Versiyon Kontrolü İş Akışı

- 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.
- Yukarıda bahsettiğimiz **repository**'yi kendi bilgisayarınızda oluşturmak için iki yöntem kullanabilirsiniz.
  - Henüz versiyon kontrolünde olmayan bir projeniz varsa **git init** komutu ile projenizi tüm klasör ve dosyaları ile birlikte versiyon kontrolüne alabilirsiniz
  - Projeniz uzaktaki veya şirket ağınızdaki bir Git sunucusunda versiyon kontrolü altında tutuluyorsa projeyi kendi bilgisayarınıza **git clone** komutu ile indirebilirsiniz.

Git

Versiyon Kontrolü İş Akışı

Projenizin repository'sini oluşturduktan sonra dosyalarınız üzerinde istediğiniz değişiklikleri istediğiniz uygulamayı kullanarak yapabilirsiniz. Bu aşamada yaptığınız değişiklikleri versiyon kontrolü için birebir ve doğrudan takip etmenize gerek yoktur.

Yaptığınız değişiklikler istediğiniz bir noktaya ulaştığında veya bir özellik veya sorun giderme düzenlemesi ile ilgili çalışmanız tamamlandığında versiyon kontrolü bakış açısı ile değişikliklerinizi değerlendirmeniz gerekir. Bu aşamada değişikliklerinizi **commit** adı verilen komut ile git e kayıt altına almaya başlamış oluruz. Böylece projenizin yeni bir versiyonunu oluşturma işleminin ilk adımını tamamlamış olacaksınız.

Git

Versiyon Kontrolü İş Akışı

commit işlemi öncesinde dosyalarınızda yaptığınız değişikliklerin bir özetini görmek isteyebilirsiniz. ***git status*** komutu ile hangi dosyaları değiştirdiğinizi, sildiğinizi veya hangi dosyaları eklediğinizi kolayca görebilirsiniz.

Git

Versiyon Kontrolü İş Akışı

Bir sonraki aşamada değişen dosyalarınızdan hangilerinin commit'e dahil olduğunu belirlemeniz gerekiyor. Bu adımda commit'e dahil etmek istediğiniz dosyaları **staging area** denilen ara bir alana alırız.

Dosyaların içeriğinin değiştirilmiş olması, silinmesi veya yeni dosya eklenmesi bu dosyaların otomatik olarak **staging area**'ya eklenmesini sağlamaz. Bu işlemi ilgili dosyaları seçerek sizin yapmanız gerekir.

Dosyalarınızı **staging area**'ya ekledikten sonra şimdi *commit* işlemine hazırsınız. Commit işlemi ile dosyalarınızdaki değişiklikler yeni bir versiyon olarak Git'de kayıt altına alınır.

Git

Versiyon Kontrolü İş Akışı

Zaman zaman, özellikle de bir takım çalışması söz konusu ise, projenizdeki değişikliklere göz atmak isteyebilirsiniz. Projeniz için oluşturduğunuz commit'lerin tarihçesini incelemek için ***git log*** komutunu kullanabilirsiniz.

Git

Versiyon Kontrolü İş Akışı

Yaptığınız değişikliklerin takımın geri kalanı tarafından da görülmesini ve kullanılmaya başlanmasını sağlamak için değişikliklerinizi zaman zaman uzaktaki repository'de yayınlamanız gerekir. Bunun için ***git push*** komutunu kullanırız.

Git

Local (Yerel) & Remote (Uzak) Repository’ler

Local repository, kendi bilgisayarınızda proje klasörünüzün altında bulunan **.git** klasörüdür. Bu repository üzerinde sadece siz çalışabilirsiniz ve değişiklikler yerel diskinize kaydedilir.

Remote repository'ler ise genellikle uzaktaki bir sunucuda yer alırlar ve bu sunucudaki **.git** klasöründen ibarettirler. Takım çalışması söz konusu ise takımdaki kişiler değişikliklerini bu uzaktaki repository üzerinden paylaşırlar.

Git

Local bir proje oluşturmak

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

Bu işlemi gerçekleştirmek için Mac OS X'de Terminal uygulamasını Windows'da ise Git Bash'i açarak aşağıdaki komutları çalıştırmanız gerekir

```
$ cd proje/klasörünüzün/yolu/
$ git init
```

Bu işlemden sonra

```
ls -la
```

komutu ile proje klasörünüz altındaki dosyaları listelediğinizde klasörün içinde *.git* isimli gizli bir klasörün olduğunu göreceksiniz. *git init* komutu ile projemiz için **boş** bir repository oluşturduk.

Ancak proje klasörümüzde dosyalar ve başka klasörler bulunmasına rağmen bu dosya ve klasörlerin hiç biri henüz Git tarafından versiyon kontrolü altına alınmadı.

Git

Git'in Dosyalara Bakışı

Git dosyaları kendi içerisinde ikiye ayırıyor. Bunlar tracked files (takip edilen dosyalar), untracked files (takip edilmeyen dosyalar).

Takip Edilmeyen Dosyalar

Git init komutu verdiğiniz bir dizinde oluşturduğunuz herhangi bir yeni dosya varsayılan olarak takip edilmeyen dosya olarak kategorilendirilir.

Takip Edilen Dosyalar

Dosyaları takip etmek ve kalıcı olarak git veri tabanına yazmak için öncelikle dosyaları Git'e eklemeniz gerekiyor. Git'e eklediğiniz bu dosyalar "**Tracked Files**" (takip edilen dosyalar) olarak adlandırılıyor. Bu dosyalarda yaptığınız değişiklikler komut verdiğiniz takdirde git veri tabanına kalıcı olarak işleniyor. Herhangi bir dosyayı takip edilen dosyalar arasına almak için add komutunu vermeniz gerekiyor.

```
$ git add /dosya/yolu/dosya_ismi
```

Git

Git'in Dosyalara Bakışı

Dosyamızı Git'e ekledikten sonra bu dosyalarımızı Commit edilecek değişiklikler (changes to be committed) hanesine yazdı.

Şimdi bunun ne anlama geldiğine bir bakalım. Git, takip edilen dosyaları **staged**, **modified** ve **unmodified** olarak üçe bölüyor.

- **Staged**

Dosyamızı alıp gite ekledikten sonra üzerinde hiçbir değişiklik yapmazsak bu dosyamız staged olarak adlandırılıyor. Stage, "sahne" anlamına geliyor ve staged da sahnelenen gibi bir anlama geliyor. Bu dosyalarımız commit edilmeye tamamen hazır olmuş oluyor.

- **Modified**

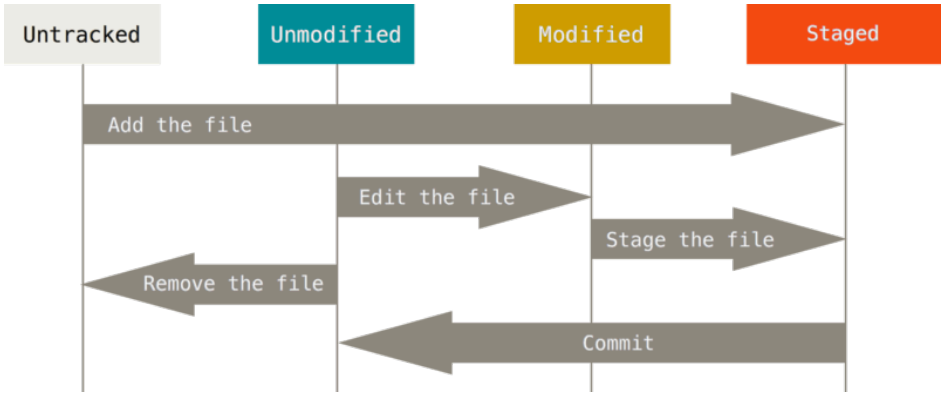
Modified olarak adlandırdığımız dosyalarımız git'e eklendikten sonra üzerinde değişiklikler yaptığımız ve daha sonra tekrar git'e eklemediğimiz dosyalarımız oluyor.

- **Unmodified**

Bu dosyalarımız da en az bir kez commit edildikten sonra bir daha değişikliğe uğramamış dosyalar.

Git

Git'in Dosyalara Bakışı



Untracked dosyalarımız git add komutunu verdiğimiz an Git'e ekleniyor ve staged oluyor.

Git commit komutunu verdiğimiz zaman staged olan dosyalarımızın hepsi unmodified haline geliyor.

Unmodified dosyalarımızda herhangi bir değişiklik yaparsak bu dosyamız modified haline geliyor ve son olarak da bu modified dosyamızı git add komutu ile Git'e eklersek bu dosyamız da tekrar staged olmuş oluyor.

Git - Uygulama - 1

Lokal Repository ile ilgili uygulama

- Lokal repo oluşturmak - (git init)
- Lokalde çalışma dosyalarını oluşturmak
- Çalışma dosyalarını repoya stage'e eklemek (git add)
- İlk commiti yapmak (git commit)
- Değişiklik yapıp stage'e atmak (git add)
- Stage'de yer alanları commit yapmak (git commit)
- Repo geçmişini incelemek (git log)

Git - Uygulama - 2

Remote Repository ile ilgili uygulama

- Github’ta hesap oluştur
- Github’ta repository oluştur
- Repository readme.md oluşturma ve stiller
- Dosyaları yükleme
- Remote repoyu lokalde kullanmak (git clone)
- Lokalde değişiklik yapıp stage’e atmak (git add)
- Stage’de yer alanları commit yapmak (git commit)
- Lokaldeki commit’i remote repoya atmak (git push)
- Repo geçmişini incelemek (git log)

Git

Commit noktalarına gitmek

- Projemizde belirli zamanlarda yapmış olduğumuz commit işlemleri neticesinde git yapmış olduğumuz değişiklikleri kayıt altına alır ve biz bu kayıtlı alanlara tekrar geri dönüş yapabiliriz

  git checkout “commit id”

  git checkout ea8d5d231169d513ac4f5cd483e15641870f0f05

- Daha sonra projemizin ana(master veya origin) dalının(branch) son güncel haline geri dönmek istersek aşağıdaki şekilde komut yazarız

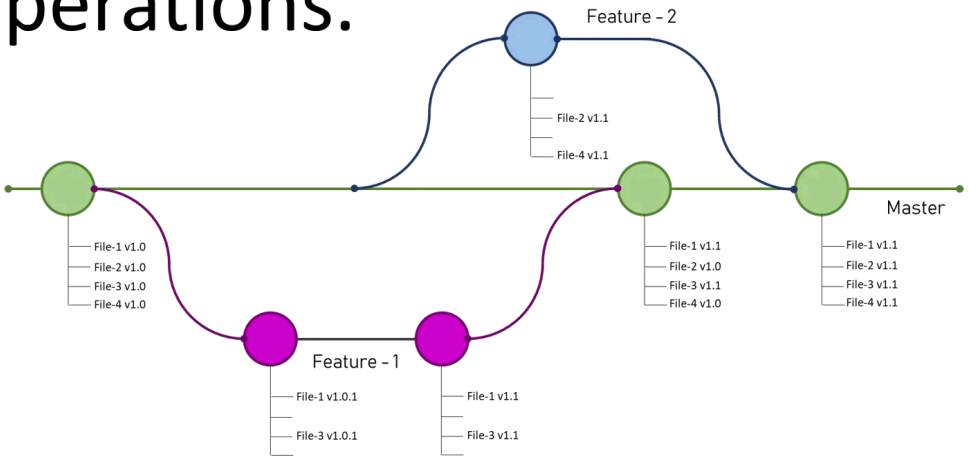
  git checkout “dalın adı”

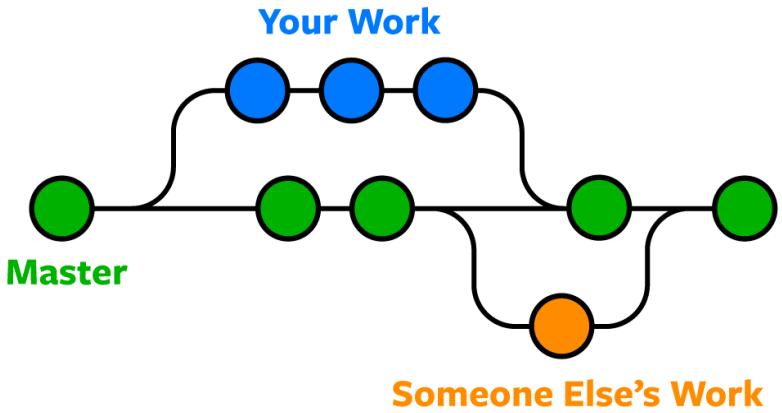
  git checkout master

Git

Branch(Dal)

GIT Branch and its Operations.





Git

Branch(Dal)

- Projemizde git init yaptığımızda zaten git bir tane ana(master) dal bizim için oluşturur ve değişiklikleri bu dal üzerinde takip eder
- Özellikle de ekip çalışmalarında ekipteki herkesin aynı dal üzerinde çalışması pek verimli olmaz
- Herkesin üstlendiği görev ile ilgili ana dalın bir kopyası üzerinde gerekli geliştirmeleri yapması ideal bir çözümdür
- Böylece ana dalda oluşabilecek problemlerin önüne geçilmiş olur
- Ayrıca geliştirci de oluşturduğu yeni dal üzerinde özgürce çalışır
- Yeni bir dal oluşturmak için;

  git branch dalın adı

  git branch login-modulu

Git

Branch(Dal) Birleştirme

- Oluşturmuş olduğumuz dal üzerinde gerekli işlemlerimizi tamamladıktan sonra projemizin ana dalına birleştirme(merge) işlemini gerçekleştirebiliriz
- Projemizdeki mevcut dalları görüntülemek için *git branch* komutunu çalıştırmalıyız
- Ana dalda oluğumuzdan emin olmalıyız
- Başka dala geçiş için “git checkout dalın adı”
- Daha sonra mesela “login-modulu” dalını ana dala birleştirmek için;

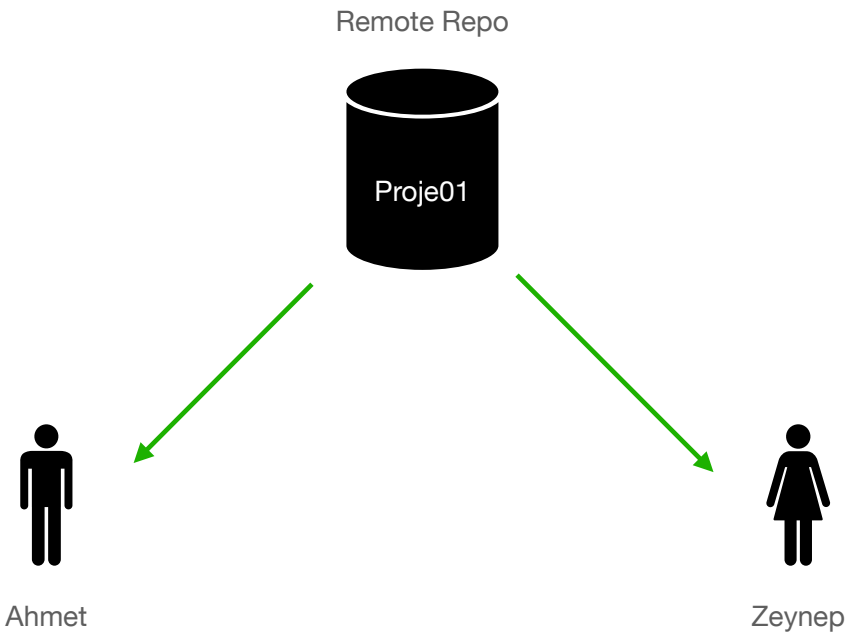
  git merge “dal adı”

  git merge “login-modulu”

  komutunu yazmalıyız

Git

Remote Repo ile Çalışmak

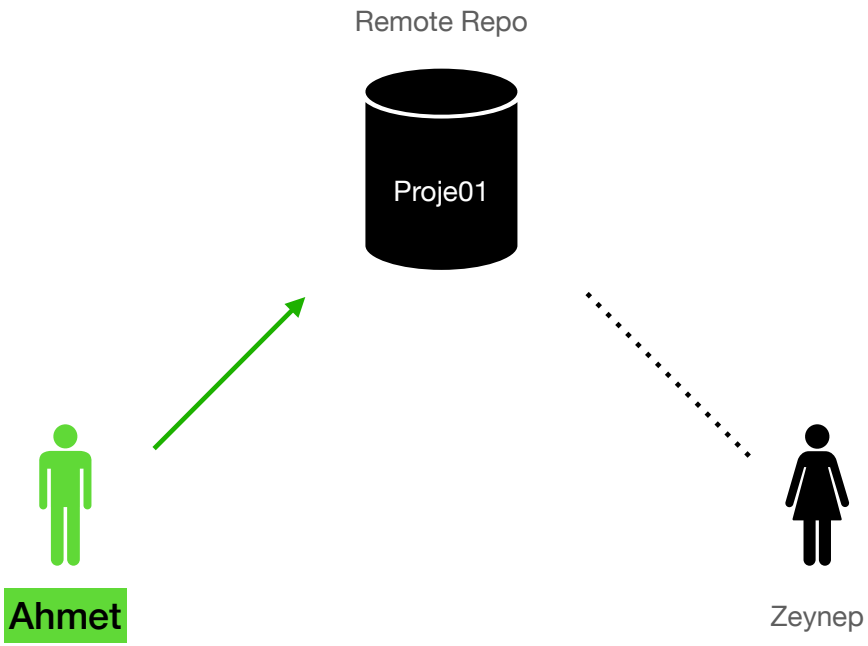


- Öncelikle ekipte bulunan kişilerin “Proje01” adlı repoyu kendi lokal alanlarına (bilgisayar) kopyalamaları gerekir

  git clone “proje git adresi”

Git

Remote Repo ile Çalışmak - Senaryo 1

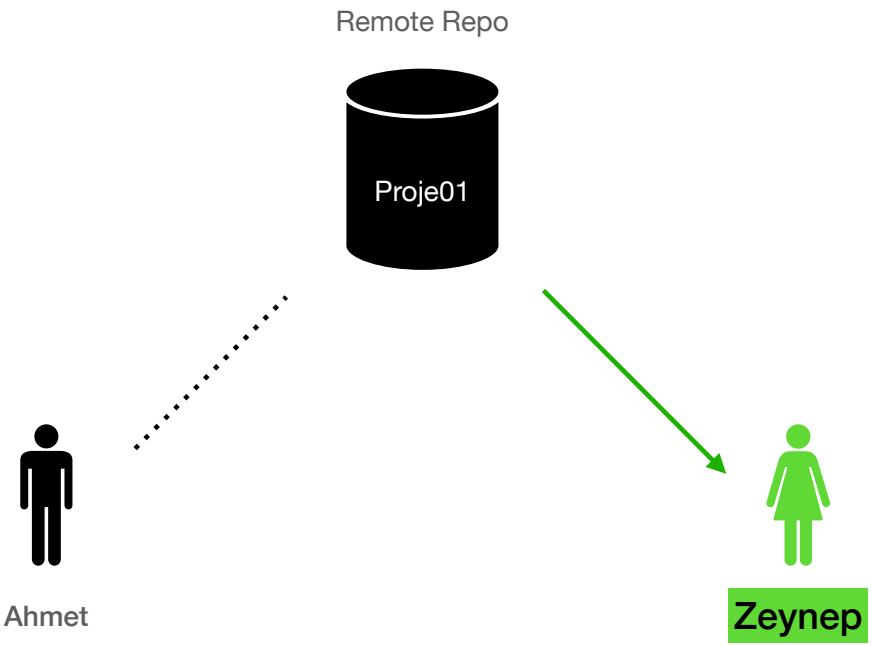


- Ahmet “index.html” adlı dosyada değişiklik yapıp bunu Remote Repo’ya gönderir. Değişiklik yaptıktan sonraki git işlemleri;

```
git add index.html
git commit -m “bir güncelleme..”
git push
```

Git

Remote Repo ile Çalışmak - Senaryo 2



- Zeynep Remote Repo'dan Ahmet'in yaptığı güncellemeleri kendi lokaline alır. İlgili git işlemleri;

  git pull