**Basit Anlamda Versiyon Kontrolü İş Akışı**

Git'in derinliklerine dalmadan önce gelin basit bir versiyon kontrol iş akışına adım adım göz atalım.

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.

Projeniz için yukarıdaki yöntemlerden biri ile *repository* oluşturduktan sonra aşağıdaki basit akışı kullanarak değişikliklerinizi yapmaya başlayabilirsiniz

1. 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.
2. 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 bir bütünü olarak tarif etmelisiniz. Böylece projenizin yeni bir versiyonunu oluşturma işleminin ilk adımını tamamlamış olacaksınız.
3. Fakat, commit işlemi öncesinde dosyalarınızda yaptığınız değişikliklerin bir özetini görmek isteyebilirsiniz. *git status* komutu ile hangi dosyaları değiştirdiğinizi, sildiğinizi veya hangi dosyaları eklediğinizi kolayca görebilirsiniz.
4. 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.

1. Dosyalarınızı **staging area**'ya ekledikten sonra şimdi *commit* işlemine hazırsınız. Commit işlemi ile dosyalarınızdaki değişiklikler yeni bir versiyon olarak Git'de kayıt altına alınır.
2. 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.
3. 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.

**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.

# 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.

**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.

## Versiyon kontrolü altına almak istemediğimiz dosyalar

Tüm geliştirme ortamları ve işletim sistemlerinde kullandığımız araçlar tarafından ara bir ürün olarak üretilen ve aslında doğrudan versiyon kontrolü altına almak istemediğimiz dosya veya klasörler olacaktır. Örneğin Mac OS X'in otomatik olarak ürettiği gizli DS\_Store isimli klasör veya C++ derleyicileri tarafından üretilen .o uzantılı obj dosyaları gibi. Hangi dosyaların versiyon kontrolü altında tutulacağına ve hangilerinin göz ardı edileceğine Git otomatik olarak karar vermez, bu kararı sizin vermeniz gerekir.

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

Versiyon kontrolü altına almak istemediğiniz dosya ve klasörleri tanımlamak için proje klasörüne eklenen .gitignore dosyası kullanılır. Bu dosyaya göz ardı etmek istediğiniz dosya ve klasörlerin tespit edilebilmesi için doğrudan isimler veya basit kurallar ekleriz. Projelerinizi versiyon kontrolü altına aldıktan sonra ilk iş olarak GitHub'ın yayınladığı derlemeyi veya kendi deneyiminiz ve bilginiz ile karar vereceğiniz dosya ve klasörleri **.gitignore** dosyasına ekleyiniz. Projenizin ilerleyen aşamalarında bu işlemi yapmanız biraz daha zahmetli olacaktır.

Şimdi gelin .gitignore dosyasında kuralları nasıl tanımlayabileceğimize bir göz atalım

\*.[oa]

.~

İlk satırda **o** veya **a** uzantısı ile biten dosyaların versiyon kontrolü dışında tutulması için bir kural tanımlıyoruz. İkinci satırda ise **~** karakteri ile biten (çoğu metin düzenleme uygulaması geçici dosyaları ~ ile biten dosyalar olarak otomatik oluşturur) dosyaların versiyon kontrolü haricinde tutması için kural tanımlıyoruz.

.gitignore dosyasında tanımlama yaparken aşağıdaki kurallar geçerlidir

* Boş satırlar veya # ile başlayan satırlarda yaptığınız tanımlamalar Git tarafından dikkate alınmaz.
* \*, ?, [ ], { }, [!] ve \ gibi karakterler kullanılarak oluşturulan ve *[globbing patterns](http://www.tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm" \t "_blank)* adı verilen tanımlayıcılar kullanabilirsiniz
* Klasörleri belirtmek için **/** karakteri kullanılır. Örneğin **/projemde/versiyon/kontrolü/istemedigim/bir/klasor/** şeklinde bir tanım yaptığımızda ilgili klasör ve altındaki tüm dosyalar Git tarafından göz ardı edilir.
* Tanımladığınız bir kuralın tersini **!** simgesi ile tanımlarız. Örneğin !/projemin/kaynak/kodu/ şeklinde bir tanım yaptığımızda bu klasör dışındaki tüm klasör ve dosyalar Git tarafından göz ardı edilecektir.

## İlk commit’imiz

Projemizi versiyon kontrolüne alıp göz ardı edilmesini istediğimiz klasör ve dosyaları da belirledikten sonra aşağıdaki komutlar ile ilk commit işlemimizi yapabiliriz

$ git add -A

$ git commit -m "İlk commit işlemimizi yaptık"

Bu komutların ne işe yaradığına sonraki bölümlerde değineceğiz, şimdilik

* İlk komutun tüm proje dosyalarının Staging Area'ya eklenmesi için,
* İkinci komutun ise dosyalarımızın bir açıklama ile commit edilmesi için kullanıldığını söylemek ile yetinelim.

Yukarıdaki iki komutu arka arkaya kullanmak yerine aynı işlemi git commit -a komutu ile de yapabiliriz.

**Remote bir proje oluşturmak**

Versiyon kontrolü Git ile yapılan bir projede yer alıyorsanız *remote repository*'lerinizi nasıl yöneteceğinizi de öğrenmeniz gerekir. Remote repository'leri projelerinizi internette veya sınırlı erişime izin verilen şirket ağında yer alan versiyonları olarak düşünebilirsiniz.

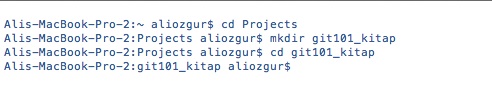
Diğer ekip üyeleri ile birlikte verimli çalışabilmek, onların yaptığı değişiklikleri kendi yerel çalışma alanınıza almak, kendi yaptığınız değişiklikleri onlar ile paylaşabilmek için remote repository'lerinizi doğru ve etkin bir şekilde yönetmelisiniz.

Git ile versiyon kontrolü yapılan bir projeye dahil olduğunuzda size verilecek ilk bilgiler projenin Git adresi (URL) ve projeye erişim için kullanacağınız kullanıcı adı ve şifrenizdir. Uzaktaki bir repository'nin (URL) adresi aşağıdaki formatlardan birinde olacaktır

* ssh://user@server/git-repo.git
* kullanıcıadı@sunucuadı:git-repo.git
* <http://example.com/git-repo.git>
* <https://example.com/git-repo.git>
* git://example.com/git-repo.git

Bu adres formatlarından ilk iki tanesi [SSH](http://en.wikipedia.org/wiki/Secure_Shell) (Secure Shell) protokolüne karşılık gelir. http:// ve https:// protokolleri ise normal internet erişimi için de kullanılan protokollerdir. Son format ise git'in kendi protokolüne karşılık gelir.

Remote repository'nizin adresini ve erişim için gerekli kullanıcı adınızı ve şifrenizi öğrendikten sonra yapmanız gereken tek şey bu adresten projenizin dosyalarını yerel diskinize klonlamak. Bunun için öncelikle yerel diskinizde projenizi indireceğiniz bir klasör oluşturmanız ve Terminal'den bu klasöre gitmeniz gerekiyor. Sırasıyla aşağıdaki komutları Terminal'de yazınız



Yukarıdaki ekran görüntüsünde yer alan ilk **cd** komutu ile proje klasörümün içinde yer alacağı ana klasör olan **Projects** klasörüne konumlanıyoruz. İkinci komut olan **mkdir** ile proje klasörümüz olan **git101\_kitap** klasörünü oluşturuyoruz. Üçüncü komutumuz ile de yeni oluşturduğumuz **git101\_kitap** klasörüne konumlanıyoruz.

Yerel diskimizde boş proje klasörümüzü oluşturduğumuza göre şimdi remote repository'mizi yerel klasörümüze **git clone** komutu ile indirebiliriz.



Kullanıcı adınızı ve şifrenizi vererek remote repository'yi klonlamak için aşağıdaki **git clone** komutuna bu bilgileri aşağıdaki formatta vermeniz gerekiyor

git clone [https://kullanıcıadı:şifre@github.com/username/repository.git](https://kullan%C4%B1c%C4%B1ad%C4%B1:%C5%9Fifre@github.com/username/repository.git)

# Projemiz Üzerinde Çalışmaya Başlayalım

Üzerinde çalışacağımız projenin dosyaları artık yerel diskimizde yer aldığına göre projemiz ile ilgili normal çalışmamıza başlayabiliriz.

Projenizi ister local bir proje olarak oluşturmuş olun isterseniz remote bir repository'yi klonlamış olun tüm değişiklikleriniz yerel diskinizde gerçekleşecek ve **commit’leriniz** ile oluşturacağınız tüm versiyonlar git tarafından yerel diskinizdeki .git klasöründe takip edilecektir. İlerleyen bölümlerde ayrıntılı olarak ele alacağımız **git push** komutunu çalıştırmadığınız sürece yaptığınız değişiklikler sadece yerel diskinizde kayıt altına alınır.

## Dosya Durumları

Git'de dosyalarınız genel olarak iki durumda olabilir

* 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.

## Staging Area

Çoğu versiyon kontrol sisteminde değişiklikleriniz iki yerde kaydedilir

* Yerel diskinizdeki çalışma klasörünüz (working folder) veya
* Versiyon kontrol sisteminin veri tabanı

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

Versiyon Kontrolünün Altın Kuralları

**#1 Sadece Birbiri İle Alakalı Değişiklikleri Commit Edin**

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

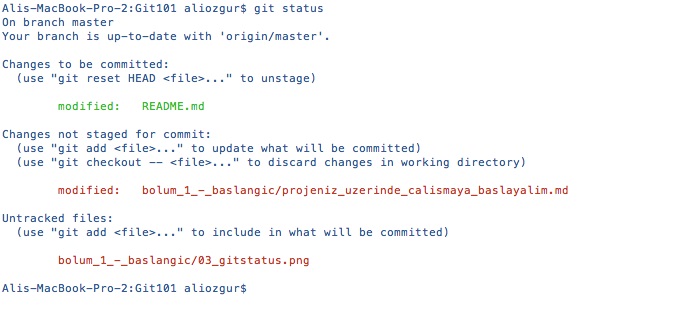
* Commit’inizdeki değişiklikleri inceleyen ekip arkadaşlarınız yaptığınız değişikliklerden hangisinin hangi konu ile ilgili olduğunu anlamakta güçlük çekeceklerdir.
* Alakalı alakasız değişiklikler tek bir commit içinde yer aldığı için herhangi bir nedenle belirli ve tek bir değişikliği geri almakta güçlük çekeceksiniz.

Alakalı alakasız değişiklikleri tek bir commit ile ele almak yerine örneğin iki ayrı sorunu gidermek için yaptığınız değişiklikler iki ayrı commit ile kayıt altına alınmalı veya daha büyük bir özellik üzerinde çalışırken bu özelliği oluşturan ve anlamsal bir bütün olarak ifade edilen daha küçük özellikleri de ayrı commit’ler ile kayıt altına almalısınız.

Projeniz üzerinde çalışırken belirli bir zaman aralığında yaptığınız değişikliklerin tamamının aynı konu veya özellikle ilgili olması mümkün olmayacaktır. Tam da bu noktada **Staging Area** mekanizmasının güzelliği ortaya çıkar, çünkü git hangi değişikliğinizin Staging Area'ya gideceğine karar vermeniz için sizin devreye girmenizi ister. Daha önce de belirttiğimiz gibi yaptığınız değişiklikler git tarafından otomatik takip edilmez, bunun yerine git tüm değişiklikleri sizin gözden geçirerek kontrollü bir şekilde Staging Area'ya almanızı ister.

## Yaptığınız Değişiklikleri Listelemek

Son commit işleminizden sonra proje dosyalarınızda yaptığınız değişiklikleri listelemek için **git status** komutunu kullanabilirsiniz.

 Yukarıdaki terminal ekran görüntüsünde de görebileceğiniz gibi git oldukça ayrıntılı durum bilgisi sunmaktadır. **git status** komutu ile git aşağıdaki 3 ana grupta yer alan dosyaları size listeler

* Changes to be committed (Commit edilmeye hazır dosyalar): Bu gruptaki dosyalar **git add** veya **git rm** komutu ile Staging Area'ya eklediğimiz dosyalardır. Bu dosyalar bir sonraki commit'imizin içinde yer alacaktır
* Changes not staged for commit (Commit için henüz hazır olmayan dosyalar): Bu gruptaki dosyalar değişiklik yaptığımız fakat henüz Staging Area'ya eklemediğimiz dosyalardır. Bu dosyalar bir önceki grubun içine eklemediğimiz sürece bir sonraki commit'e dahil olmayacaklarıdır
* Untracked files (Versiyon takibinde olmayan dosyalar): Bu gruptaki dosyalar ise henüz versiyon kontrolü altına almadığımız dosyalardır.

## "git add" ve "git rm" komutları

Bir önceki başlıkta değindiğimiz ve **git status** komutu sonrasında git'in bize özetlediği 3 gruptan son ikisinde yer alan dosyaların ilk gruba dahil edilmesi için **git add** ve **git rm** komutlarını kullanabiliriz.

Aşağıda oluşturduğumuz **git add** komutu ile **baslik\_2.md** ve **baslik\_2\_1.md** dosyaları ile **resimler** klasörü altındaki tüm dosyaların Staging Area'ya eklenmesini sağlayabiliriz.

$ **git add** baslik\_2.md baslik\_2\_1.md resimler/\*

Benzer şekilde aşağıdaki **git rm** komutu ile **ornek2.md** dosyasının bir sonraki commit’imizde yer almayacağını belirtebiliriz.

$ **git rm** ornek2.md

## Değişikliklerimizi Commit Edelim

Değişikliklerinizi **git add** ve **git rm** ile Staging Area'ya aldıktan sonra **git commit** komutu ile yeni bir versiyon olarak kayıt altına alabilirsiniz.

$ **git commit** -m "1.7 numaralı alt başlık içeriği tamamlandı"

Yukarıdaki komutta yer alan **-m** parametresi ile yaptığınız değişiklikleri özetleyen bir mesajı da commit'inize ekleyebilirsiniz. Eğer birden fazla satırı olan bir commit mesajı gireceksiniz **-m** parametresini kaldırmanız yeterli olacaktır. Bu durumda 1.3 numaralı bölümde ayarladığınız editör açılır ve bu editöre mesajınızı istediğimiz uzunlukta girebilirsiniz.

Versiyon Kontrolünün Altın Kuralları

**#2 Anlamlı Commit Mesajları**

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.

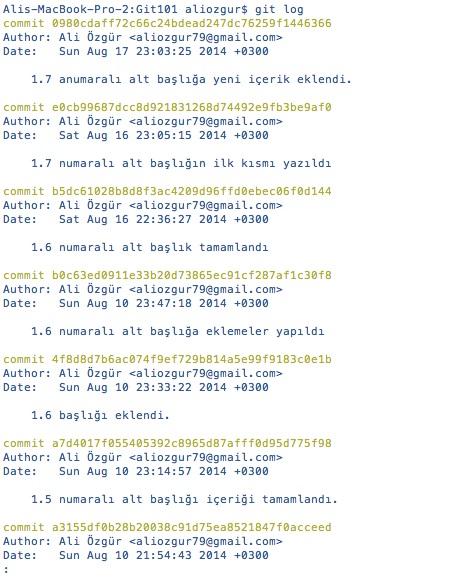
### İyi Bir Commit Nasıl Olmalı?

1. 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.
2. Tamamlanmamış değişikliklerinizi kesinlikle commit etmemeye özen gösterin. Eğer zaman zaman değişikliklerinizi kayıt altına almak istiyorsanız commit işlemi yerine Git'in **Stash** özelliğini/komutunu kullanabilirsiniz.
3. Test edilmemiş değişiklikleri commit etmemeye özen gösterin. Bu öneri aslında bir önceki önerimiz ile pratikte aynı anlama geliyor
4. Commit'leriniz kısa ve açıklayıcı mesajlar içermeli.
5. 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.

## Commit Tarihçesi

Git projeniz üzerinde çalıştığınız her anda yaptığınız commit işlemlerini kayıt altına almaktadır. Özellikle ekip çalışması söz konusu ise commit işlemleri ile ilgili git tarafından kayıt altına alınan bu bilgiler daha da önem kazanmaktadır.

Git'in commit’leriniz ile ilgili kayıt altına aldığı tarihsel bilgileri görmek için **git log** komutunu kullanıyoruz. Bu komut tüm commit’ler ile ilgili bilgileri, en son commit en üstte olacak şekilde, tarihsel olarak sıralar. Eğer Terminal pencerenize sığmayacak kadar çok tarihsel kayıt var ise son satırda **:** simgesi yer alacaktır, klavyenizden **SPACE/BOŞLUK** tuşuna basarak bir sonraki sayfanın listelenmesini **q** tuşuna basarak da listelemenin sonlandırılmasını sağlayabilirsiniz.



Terminal'de listelenen her commit tarihçesi kaydı, diğer bilgilerin yanı sıra, aşağıdaki temel bilgileri içerir

* Commit'in Hash değeri
* Commit'i gerçekleştiren kişinin adı ve email'i
* Commit tarihi
* Commit mesajı

**Commit Hash** : Her bir commit'in benzersiz ve tek bir tanımlayıcı değeri vardır. Bu değer git tarafından commit'e dahil olan tüm değişiklikleriniz ve commit'in kendisi ile ilgili bilgiler de kullanılarak otomatik hesaplanır. Genel olarak git'in listelemelerinde ve bazı komutların parametresi olarak bu değerin ilk 7 karakterinin kullanılması yeterlidir. Çünkü bu ilk 7 karakterin de nerdeyse benzersiz ve tekil olduğunu söyleyebiliriz.

git log komutu ile birlikte commit işlemi ile ilgili bilgilendirici çoğu bilgiyi görmekle birlikte parametre olarak **-p** değerini kullanırsanız dosyalarda yapılan değişiklikler de ayrıntılı olarak listelenecektir.



Kitabımızın ilerleyen bölümlerinde **git log -p** komutu ile gördüğümüz bilgileri nasıl yorumlayacağımızı ayrıntılı olarak ele alacağız.