Git ile Versiyon Kontrol Sistemi

Ali Özgür

İçindekiler

1

Versiyon Kontrolüne Giriş

Bu Bölümde

Bu bölümde genel anlamda versiyon kontrolü kavramına değinip bize sağlayacağı faydaları ele alarak, Git kurulumunu ve kişisel ayarlarımızı nasıl yapacağımızı göreceğiz. Ayrıca, çalışma ortamımızın kurulumunu yaptıktan sonra versiyon kontrolü ile genel bir akışı size sunup ardından projelerimiz için depolar (repository) oluşturarak temel Git komutları ile ilk işlemlerimizi nasıl yapabileceğimizi öğreneceğiz.

Bu şekilde bir çalışma yöntemi çok zahmetlidir. Aynı zamanda da hatalara açıktır. Örneğin, bir dosyanın en son geçerli halinin nerede olduğunun takip edilmesi gibi çözüm bulunması gereken sorunlar ile uğraşmak zorunda kalırsınız.

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

Versiyon kontrol sistemi kullanıldığında ise ekibinizdeki herkes özgür bir şekilde istediği dosyalar üzerinde güvenli olarak istediği değişikliği yapabilir. Herkes değişikliklerini tamamladıktan sonra da tüm değişiklikler versiyon kontrol sisteminin sağladığı araçlar ve komutlar kullanılarak sağlıklı bir şekilde birleştirilebilir.

VERSİYONLARIN DÜZGÜN TAKİP EDİLEBİLMESİ

Üzerinde çalıştığınız bir dosyanın veya bir dizi projenin zaman içerisinde farklı versiyonları oluşur. Bu versiyonların kayıt altına alınması gereklidir. Bu sorumluluk genelde çok zahmetli ve sıkıcı bir süreçtir. Aşağıdakine benzer sorular canınızı gereğinden fazla sıkabilir.

1. Sadece değişen dosyalar mı yoksa bir projedeki tüm dosyaların versiyonları mı kaydedilmeli?

» Bir sürü dosya içinden sadece değişen dosyaların belirlenmesi zordur.

» Her seferinde dosyaların tamamının teker teker kaydedilmesi durumunda ise ihtiyaç duyulandan daha fazla disk alanı kullanılır.

2. Dosyalara verilecek isimler tam bir baş ağrısına dönüşebilir.

» Personel_Maas.xlsx

» Personel_Maas1.xlsx

» Personel_Maas_Ozet.xlsx

» Personel_Maas_BrutHaricDetay.xlsx

şeklinde dosya isimleri üretmek zorunda kalabilirsiniz.

3. Projelerinizin veya dosyalarınızın iki farklı versiyonu arasında tam olarak ne tür değişikliklerin yapıldığını sağlıklı bir şekilde bilme şansınız olamaz.

» Versiyon kontrol sistemi kullandığınızda sizin çalıştığınız disk alanında herhangi bir anda proje dosyalarının sadece tek 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.

Önceki Versiyonlara Geri Dönebilme

Versiyon kontrol sistemleri, dosyalarınızın veya projenizin daha önceki versiyonuna geri dönebilme ve değiştirme özgürlüğü sağlamaktadır. Yapılan değişikliklerin projenizi çöpe döndürdüğünü düşündüğünüzde, geliştirdiğiniz bir işlev tam istediğiniz gibi olmadıysa veya müşteriniz veya patronunuz geliştirdiğiniz bir işlevi artık istemediğine karar verirse projenizin önceki herhangi bir haline çok hızlı ve rahat bir şekilde geri dönebilirsiniz.

Değişikliklerin Nedenini Anlama

Versiyon kontrol sistemleri değişikliklerinizi tamamlayıp yeni bir versiyon oluşturmak istediğinizde yorum olarak açıklama girmenizi isterler. Bu yorumlar sayesinde projenizin herhangi bir versiyonundaki değişikliklerin nedenlerini de kayıt altına alarak ihtiyaç halinde inceleyebilirsiniz. Özellikle karmaşık, satır sayısı fazla olan ve kalabalık ekiplerin çalıştığı yazılım geliştirme projelerinde değişikliklerin neden yapıldığına ilave olarak değişikliğin kendisi ile ilgili altın değerindeki bilgilere bu yorumları inceleyerek vakıf olabilirsiniz.

DİKKAT Git'de versiyon oluşturma işlemi sırasında yorum metni girilmesi zorunludur. Boş da olsa bir metin girmeniz gerekir.

Yedekleme

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

Kısa Git Tarihçesi

Git 2005 yılında, başta **Linus Torvalds** olmak üzere Linux çekirdeğini de kodlayan ekip tarafından Linux işletim sistemi çekirdeğinin 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.

2

Dallar ve Değişiklikleri Birleştirmek

Bu Bölümde

Bu bölümde herhangi bir proje üzerinde bireysel olarak veya takım olarak çalışırken değişikliklerinizi daha özgür şekilde yapabilmeniz için vazgeçilmez Git yapılarından birisi olan dalları ele alacağız.

DALLARIN ÇALIŞMA YÖNTEMİMİZE ETKİSİ?

Bazı araçların sağladığı imkânlar günlük iş yapma şeklimizi çok derinden etkileyip, yaptığımız işe daha farklı bakabilmemizi sağlar. Git'in branching yaklaşımı (Türkçe'ye **dallanma** olarak tercüme edilebilir) bahsettiğim bu dönüştürücü etkiye sahip araçlardan birisidir. Dallanma konusundaki hâkimiyetinizin artması ve sağlamlaşması ile birlikte daha farklı iş yapmaya başlayıp daha iyi birer yazılım geliştiricisi olabilirsiniz.

Dallanma denilen yöntem aslında Git dışındaki diğer versiyon kontrol sistemlerinde de öteden beri kullanılmakta ve yazılım geliştiricilerin hayatını önemli derecede kolaylaştırmaktadır. Ancak, Git'deki dallanma yaklaşımının kullanım kolaylığı ve yüksek performansı nedeniyle kendine has bir yapısının olduğunu da söylemeliyiz.

Öyleyse gelin şimdi dallanmanın ne olduğunu anlayalım.

DAL (BRANCH) NEDİR?

Proje dosyalarınızda bir değişiklik yaptığınız zaman Git dosyalarınızın önceki hali ile son hali arasındaki farkları kayıt altına almak yerine commit işlemi gerçekleştiği andaki dosyalarınızın son halinin tam bir kopyasını (*snapshot*) kayıt altına alır ve değişikliğe ilişkin bilgileri içeren (değişiklik kim yaptı, ne zaman yapıldı vs.) commit nesnesini gösteren bir işaretçi oluşturur. Dal, commit nesnesini gösteren birer işaretçidir. `git branch` komutunu çalıştırdığınızda Git kaynak dalınızdaki son commit'i gösteren işaretçinin bir kopyasını yaratır sizin için yeni bir dal oluşturur. Git'de dal oluşturulduğunda proje dosyalarının bilindik anlamda birer kopyası oluşturulmayıp sadece işaretçinin bir kopyası oluşturulduğu için `git branch` işlemi disk kapasitesi kullanımı ve dal oluşturma süresi açısından çok çok düşük maliyetli bir işlemdir.

BİRDEN FAZLA BAĞLAMDA ÇALIŞMA

Zaman zaman aynı projenin birbirinden farklı dosya veya dosya gruplarında değişiklik gerektiren farklı özellikleri ile ilgili eş zamanlı veya paylaşımlı çalışma yürütmeniz gerekebilir. Büyük projelerde bu durum kişisel bir tercih olmaktan çıkıp iş bölümü/uzmanlık gibi kriterlere bağlı olarak proje/ürün yönetiminin önemli bir parçası halinde ele alınır. Örneğin, 5 kişilik bir ekibin her üyesi aynı yazılımın farklı özellikleri ile ilgili çalışabilir veya iki farklı kişi aynı özelliğin farklı şekillerde nasıl geliştirilebileceği ile ilgili deneysel çalışma yapıyor olabilirler. Bahsettiğim tüm bu alternatif senaryolar aslında kendi yaşam döngüleri olabilen, çoğu zaman kısa veya uzun süreli eş zamanlı ilerleyen farklı birer bağlama denk gelir.

Pratikte, üzerinde çalıştığınız projenin her zaman son kararlı durumu yansıtan ana bir bağlamı ve X numaralı hata bildiriminin düzeltilmesi, yeni bir Y özelliği üzerinde yapılan çalışma veya deneysel bir özellik ile ilgili yapılan çalışma gibi birden fazla yan bağlamları olacaktır. Git'in sağladığı esnek ve güçlü dallanma yapısı ile tek bir izlek üzerinde lineer çalışma yaklaşımından uzaklaşıp birden fazla ve eş zamanlı var olabilen izlekler üzerinde paralel olarak çalışabilirsiniz.

Net olarak birbirinden ayrılmış farklı bağlamlar oluşturmak için branching benzeri yapılar olmasaydı aşağıdakilere benzer senaryolarda nasıl davranılması gerektiği konusunda sıkıntılar yaşanacaktı.

» Örneğin, müşteriniz veya yöneticiniz iki alternatif sayfa tasarımından birincisini değil de ikincisini beğendi. Buna paralel olarak siz de sayfa tasarımı dışında birkaç tane **hata giderme** (*bug fix*) ve birkaç tane de dokümantasyon değişikliğini bu karar henüz verilmeden tamamladınız. Bu senaryoya göre müşterinizin beğendiği ikinci tasarımı diğer tüm düzenlemeleri kaybetmeden nasıl devreye alacaksınız?

» Üzerinde çalıştığınız alış veriş sitesi için özel olarak geliştirdiğiniz **Sepet** modülü yerine 3. parti hazır bir sepet modülü kullanmaya karar verdiniz. Bu durumda kendi geliştirdiğini modül kodunu tespit edip diğer modülleri etkilemeden bu işlemi tamamlamanız gerekir.

» Yeni geliştirdiğiniz **Beni Haberdar Et** işlevi yazılımınızın geri kalan işlevlerinin birçoğunda değişiklik yapılmasına neden olmuşken bu işlevinin gereksiz olduğuna karar verilmiş olsun. Bu işlevi aradan geçen sürede yazılımın farklı yerlerinde yapılan ve korunması gereken diğer değişikliklerden yalıtarak nasıl sökeceksiniz?

Birden fazla konu ile ilgili değişikliklerin tamamını tek bir bağlam ile yönetmeye çalışırsanız işler hızla karmaşık hale gelecek ve bu değişiklikleri hatırlamak, yönetmek ve diğer değişiklikler ile uyumlu hale getirmek için daha fazla zaman harcamaya başlayacaksınız. Bu karmaşayı yönetebilmek için her bir değişiklikte 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.

Dallanma (branching) yukarıda değindiğimiz tüm sorunların önüne geçmek için kullanabileceğimiz çok güçlü ve esnek bir araçtır. Branching ile farklı bağlamları birbirinde kolayca yalıtarak her birini kolayca ve ayrı ayrı yönetebilirsiniz.

4

Uzak Depolar ile Çalışmak

Bu Bölümde

Git oldukça esnek yapısı ve yüksek performanslı veri yapıları sayesinde kişisel projelerinizi rahat şekilde kendi bilgisayarınızda versiyon kontrolü altına alarak çalışabilirsiniz.

Ancak takım halinde yapılan projelerde değişikliklerin merkezi bir yerde paylaşılması hem kaynak kodunuzun kendi bilgisayarınız dışında bir konumda yedeğinin olmasını hem de kod okuma ve gözden geçirme faaliyetlerini daha rahat yapmanızı sağlar.

Bu bölümde uzak depolar ile çalışabilmek için Git'in hangi imkanları sağladığını ele alıyoruz.

UZAK DEPOLAR

Günlük çalışmanız sırasında versiyon kontrolü ile ilgili işlemlerin büyük bölümünü diskimizde yer alan yerel proje deposu üzerinde yaparız. Proje üzerinde tek başınıza çalışıyorsanız büyük olasılıkla bulut Git servisleri üzerinde (GitHub, Bitbucket, GitLab benzeri) veya şirket ağınızda uzak proje deposu oluşturmanıza gerek kalmayacaktır. Kendi bilgisayarınızdaki Git deposunu herhangi bir sıkıştırma uygulaması ile zaman zaman sıkıştırıp **Google Drive** veya **Dropbox** benzeri servislerde yedekleyebilirsiniz.

Daha önceki bölümlerde bahsetmiştik, Git ile versiyon kontrolü yapabilmek için ihtiyaç duyulan tek şey `.git` isimli bir klasör ve bu klasörün içinde Git'in oluşturduğu veritabanıdır. Bu nedenle kendi bilgisayarınızda herhangi bir veri kaybı yaşamanız durumunda Dropbox'da yedeklediğiniz klasörü bilgisayarınıza indirmeniz tekrar çalışmanıza dönebilmeniz için yeterli olacaktır.

Ancak, takım çalışması söz konusu olduğunda takımdaki geliştiricilerin birlikte çalışabilmesi için herkesin değişikliklerini ortak bir alanda yayınlaması ve diğerlerinin de bu ortak alan üzerinden değişiklikleri kendi kişisel dallarına entegre etmesi gerekir. Bu durumda başvuracağınız en etkin araç Git'deki **Uzak Depo** (*Remote Repository*) araçlarıdır. Uzak depolar en basit anlamda ekibinizin erişimi olan basit bir dosya sunucusu olarak düşünebilirsiniz.

> **NOT** GitHub, Bitbucket ve GitLab üzerinde gördüğünüz işlevlerin çoğu Git'in sağladığı işlevler değildir. Bu tür bulut Git servisleri Git'in ana işlevleri üzerine inşa edilir ve kullanıcılarına Git'in sunmadığı pull/merge request, sorun bildirimi, depo tarihçesinin daha anlaşılır bir şekilde gösterimi, kimlik yönetimi ve doğrulama gibi araçları sunarlar. Bu servislerin sağladığı ilave hizmetlerin Git'in uzak depo araçları ile sağlanığı yanılgısına kapılmayın.

Şimdi gelin yerel ve uzak depoları birbirinden ayırmamıza yardımcı olan temel özelliklere bir göz atalım.

KONUM

Yerel depolar, yukarıda da bahsettiğimiz gibi geliştiricilerin kendi bilgisayarlarında yer alırken, uzak depolar çoğunlukla internet olmak üzere ekipteki herkesin erişebileceği uzak bir sunucuda yer alırlar.

Özellikler

Teknik açıdan uzak depolar ile yerel depolar arasında Git'in çalışma mekaniği açısından bir fark yoktur. Yerel depolarda kullandığınız `git add`, `git commit` ve `git branch` gibi komutların tamamı uzak depolar için de kullanılır. Ancak, tüm bu benzerliklere rağmen uzak depolar için **Çalışma Kopyası** (*Working Copy*) yani aktif daldaki dosyaların diskimizde normal bir dosya olarak varolabilmesini tanımlayan yapı geçerli değildir. Uzak depolarda sadece Git'in veri tabanın yer aldığı `.git` klasörü yer alır. Projenizin dosyaları bu veri tabanında bildiğimiz anlamda birer dosya olarak değil Git'in özel veri tabanı formatına uygun birer içerik olarak kayıt altında tutulurlar.

İPUCU

Daha önceki bölümlerde **yalın depo** (*bare repository*) kavramından kısaca bahsetmiştik. Uzak depolar ağ içinde veya ağlar arasında uzaktan erişimin mümkün olduğu yalın depolardır.

Repository (Depo) Oluşturma

Yerel bir depo ancak iki yöntem ile oluşturulabilir;

» Boş bir depo olarak sıfırdan `git init` komutu ile,

ya da

» Uzak bir depoyu `git clone` komutunu kullanarak diskinize kopyalayarak

Uzak depoları da iki yöntem ile oluşturabiliriz;

» Yerel depomuzu `git clone` komutu ve `--bare` seçeneği ile kullanarak uzak bir depoya kopyalayabiliriz.

» Boş bir uzak depo olarak sunucu üzerinde, yerel depolarda yaptığımıza benzer bir şekilde, `git init` komutunu `--bare` seçeneği ile çalıştırarak.

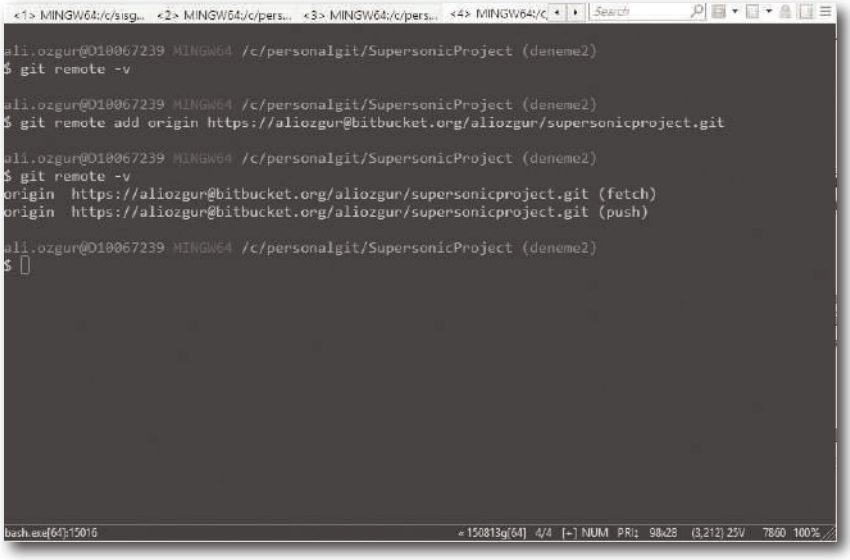
İş Akışı

Git'de sadece uzak depo işlemleri için kullanılabilecek çok az sayıda komut vardır. Günlük çalışmamız sırasında, bölümün başında da belirttiğimiz gibi Git işlemlerimizin çoğu yerel diskiniz üzerinde gerçekleşir. Bu nedenle herhangi bir ağ bağlantısına ihtiyaç duyulmaz. Ancak uzak depo komutlarını kullanabilmek için ağ bağlantısının olması zorunludur.

Git, sizin komutunuz veya haberiniz olmadan arka planda uzak depolar ile herhangi bir iletişim veya veri alış verişi yapmaz.

UZAK DEPO İLE BAĞLANTI KURMAK

Uzak bir depoyu yerel diskinize `git clone` komutu ile indirdiğinizde Git otomatik olarak bu işlemi yaparken kullandığınız uzak depo erişim bilgilerini hatırlar. Git uzak deponuz için `origin` adı verilen varsayılan bir kısaltma oluşturup uzak depo ile ilgili komutlarda kullanılmak üzere veri tabanında kayıt altına alır. Yerele bir depo için ise böyle bir bilgi tutulmaz.



Ekran görüntüsünde yer alan örneğimizde yerel bir depo olarak oluşturup üzerinde işlem yaptığımız proje deposunu herhangi bir Git sunucusu üzerinde oluşturduğumuz boş bir uzak depo ile nasıl ilişkilendirebileceğimizi görüyorsunuz. Yerel depo ile uzak depo arasındaki ilişkiyi kurmak için `git remote add` komutunun kullanıyoruz. Bu komutun en basit anlamdaki kullanım şablonu aşağıdaki gibidir.

```
git remote add <kısaltma_adı> <uzak_depo_adresi>
```

Yerel deponuzdaki her bir uzak depo eşleştirmesi için Git tarafından biri `fetch` diğeri de `push` işlemlerinde kullanılmak üzere aynı kısaltma ile iki kayıt tutulur. `fetch` kaydındaki uzak depo adresi yerel deponuza uzak depodan veri okuma işlemlerinde, `push` kaydındaki uzak depo adresi ise yerel deponuzdaki değişiklikleri uzak depoya yayınlarken yapılan yazma işlemleri için kullanılır. Genel pra-

6

Git Araç ve Servisleri

Bu Bölümde

Kitabımızın son bölümünde Git serüveninizde size yardımcı olacağını düşündüğüm araçların adlarını sizinle paylaşmak istiyorum.

Git Servisleri

Takım çalışması söz konusu olduğunda en önemli konulardan birisi de kaynak kodunun veya daha genel anlamda dosyaların nasıl paylaşılacağına karar vermektir. Bu noktada iki seçeneğiniz var;

» Dosyalarınızı kendi sunucularınız üzerinden paylaşmak veya

» İşi paylaşım ve barındırma hizmeti vermek olan online servisler kullanmak

Dosyalarınızı kendi sunucularınız üzerinden paylaşmanın aşağıdaki gibi avantajları vardır;

» Düşük maliyet,

» Dosyalarınız kendi sunucularınızdadır,

» Git'in veya hangi versiyon kontrol sistemini kullanıyorsanız bu sistemin tüm özelliklerini istediğiniz gibi kullanabilirsiniz.

Ancak bu seçeneğin aşağıdaki dezavantajlarını da göz ardı edemeyiz;

» Sunucuların çalışır halde ve erişilebilir olmasını sağlamak sizin sorumluluğunuzdadır.

» Yedekleme sorumluluğu sizde olacak.

» Güvenlik ve yazılım güncellemelerini de sizin takip etmeniz gerekir.

Eğer sunucu kaynakları yeterli olan, yedekleme, güncelleme gibi sunucu yönetimi konularında ayrı ve uzman ekibi olan bir kurumda çalışıyorsanız dosyalarınızı kendi sunucularınızda barındırmak ilk tercihiniz olacaktır.

Ancak küçük bir girişimseniz veya açık kaynak bir projeniz varsa sunucu yönetimi ile ilgili yeterince uzmanlığınız ve kaynağınız olmayabilir. Bu durumda dosyalarınızı online bir servis üzerinde barındırmak ve buradan paylaşıma açmak sizin için daha mantıklı olacaktır.

GitHub

Özellikle açık kaynak projeler için oldukça popüler bir servis olan GitHub'ı kullanabilirsiniz. GitHub açık kaynak projeler için ücretsiz, kurumlar ve özel projeler için ise oldukça makul fiyatlara Git sunucu hizmet'i sunmaktadır. GitHub'ı ayrıca kendi sunucularınıza da lisans satın alıp çalıştırabilirsiniz.

GitLab

GitHub'ın sunduğu işlevlerin tamamını sunan başka bir Git servisidir. Açık kaynak projelerinizi bu servis üzerinde ücretsiz olarak oluşturabilirsiniz. İsterseniz GitLab'ın ücretsiz sürümünü kendi sunucularınıza da kurarak sadece kurum içi kullanıcıların erişebileceği GitLab servisi oluşturabilirsiniz.

BitBucket

Daha önce küçük bir girişim olarak **Mercurial** (bu da dağıtık bir versiyon kontrol sistemi) hizmeti sunmak için kurulan **BitBucket Atlassian** tarafından satın alındıktan sonra Git sunucu hizmeti de sunmaya başladı. BitBucket açık kaynak veya özel 5 kullanıcıya kadar olan sınırsız sayıda projeniz için ücretsiz hizmet sunar aynı zamanda oldukça makul fiyatlara da daha fazla kullanıcı için ücretlis hizmet seçeneği de var.

Visual Studio Team Services

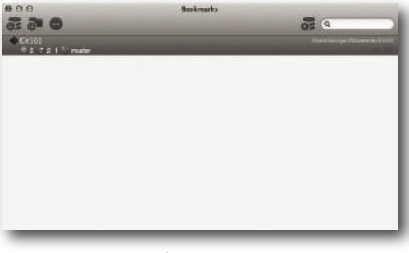
Microsoft tarafından sunulan bu servis ile 5 kullanıcıya kadar sınırsız sayıda Git deposu oluşturabilirsiniz. Daha büyük takımlarınız için yapacağınız aylık ödemelerle de bu servisten yararlanabilirsiniz. Ayırca Visual Studio Team Services'de, Git'in yanı sıra Microsoft'un kendi versiyon kontrol sistemi olan Team Foundation Server (TFS) ile de projelerinizin kaynak kodunu versiyon kontrolü altına alabilirsiniz.

Görsel Git İstemcileri

Kitabımızda komut satırı ara yüzü (**Git Bash** veya **Terminal**) kullanarak birçok Git komutunun nasıl kullanıldığını size gösterdik. Ancak günlük çalışmanızda her bir komutun ayrıntılı olarak ne işe yaradığını, hangi parametreler ve seçenekleri kabul ettiğini aklınızda tutmanız zor olacaktır. Bu nedenle Git'i günlük iş akışınıza katıp Git kavramlarını öğrendikten sonra görsel bir Git istemcisi kullanmak işinizi ciddi oranda kolaylaştıracaktır.

Atlassian SourceTree

SourceTree ücretsiz bir uygulama. SourceTree'yi Mac OS X ve Windows işletim sistemlerinde kullanabilirsiniz.



Yerel Depo Listesi

Yerel Depo