



Git NOTES

▼ Introduction to Git

▼ Git Nedir ve Tarihi

What is Git?

Git; bir versiyon kontrol sistemidir. Projenin her zaman son haline ulaşmamızı ve güncel tutmamızı sağlayan bir araçtır.

Git; bir projenin bir çok kişi ile aynı anda yapılabilmesini sağlar.

Github ise projelerimizin saklandığı (depolandığı) uzak sunucudur.

Git, küçükten çok büyük projelere kadar her şeyi hızlı ve verimli bir şekilde ele almak için tasarlanmış ücretsiz ve açık kaynaklı bir sürüm kontrol sistemidir.

Git, yazılım geliştirme sırasında kaynak koddaki değişiklikleri izlemek için dağıtılmış bir **sürüm kontrol sistemi**dir. Programcılar arasındaki **çalışmaları koordine etmek için tasarlanmıştır**, ancak değişimi izlemek için kullanılabilir.

Git, çoğu ekibin ve bireysel geliştiricinin ihtiyaç duyduğu işlevsellik, performans, güvenlik ve esnekliğe sahiptir. Git, türünün en geniş çapta benimsenen aracıdır. Bu, Git'i aşağıdaki nedenlerle çekici kılar. Çok sayıda geliştirici zaten Git deneyimine sahiptir ve üniversite mezunlarının önemli bir kısmı yalnızca Git deneyimine sahip olabilir.

Brief History of Git

Linux çekirdek bakımının (1991–2002) ömrünün çoğu için, yazılımdaki değişiklikler yamalar ve arşivlenmiş dosyalar olarak dağıtıldı. 2002 yılında, Linux çekirdek projesi BitKeeper adlı tescilli bir Sürüm Kontrol Sistemini kullanmaya başladı.

2005 yılında, Linux çekirdeğini geliştiren topluluk ile BitKeeper'ı geliştiren ticari şirket arasındaki ilişki bozuldu ve aracın ücretsiz statüsü iptal edildi. Bu, Linux geliştirme topluluğunu (ve özellikle Linux'un yaratıcısı Linus Torvalds'ı) BitKeeper'ı kullanırken

öğrendikleri bazı derslere dayalı olarak kendi araçlarını geliştirmeye sevk etti. Git böyle doğdu.

More About Git

Git, sürüm kontrolüne bir örnektir. Sürüm kontrolü, belirli sürümleri daha sonra geri çağırabilmeniz için bir dosyada veya dosya kümesinde zaman içinde **yapılan değişiklikleri kaydeden bir sistemdir**.

Şunları yapmanızı sağlar:

- Dosyaları önceki duruma döndürme,
- Tüm projeyi önceki durumuna geri döndürme,
- Zaman içindeki değişiklikleri karşılaştırın,
- Bakın kim neyi değiştirmiş? Ve daha fazlası...

NOT : Bu, işleri batırırsanız veya dosyaları kaybederseniz, kolayca kurtarabileceğiniz anlamına gelir.

Why Git?

- Her şey yereldir (tam geçmiş ağacı çevrimdışı kullanılabilir),
- Her şey hızlı,
- Anlık görüntüler, farklılıklar değil,
- Merkezi değil, dağıtılır,

▼ **Git Konfigürasyonu ve İş Akışı**

What is a Repository?(Depo Nedir?)

Depo, projelerinizin yaşayabileceği bir dizin veya depolama alanıdır. Bazen “repo” olarak kısaltılır. Bilgisayarınızdaki bir klasörde yerel olabilir veya bulutta bir depolama alanı olabilir. Bir havuz içinde kod dosyaları, metin dosyaları, görüntü dosyaları vb. tutabilirsiniz.

Yerel bir repo oluşturalım ve neye benzediğini görelim.

Masaüstünüze gidin ve "git-projects" adlı bir klasör oluşturun.

```
$ mkdir git-projects
```

Get into the folder.

```
$ cd git-projects
```

Make another folder called "project1".

```
$ mkdir project1
```

Ve "git-projects" klasörünün içinde ne olduğunu kontrol edin.

```
$ ls -al
```

Bir repo(depo) oluşturmak için şunu yazın

```
$ git init
```

Güncellenen dosyaları görmek için şunu yazın

```
$ ls -al
```

Artık **.git** adında bir deponuz var. Birçok dosya ve klasörden oluşan **gizli** bir klasördür.

Deponun içinde ne olduğunu kontrol edelim.

```
$ cd .git  
$ ls -al
```

```
Ubuntu 18.04 LTS
guile@DESKTOP-ODR37SB:~/git-projects$ cd .git
guile@DESKTOP-ODR37SB:~/git-projects/.git$ ls -al
total 0
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 .
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 ..
-rw-rw-rw- 1 guile guile 23 Mar 25 22:25 HEAD
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 branches
-rw-rw-rw- 1 guile guile 92 Mar 25 22:25 config
-rw-rw-rw- 1 guile guile 73 Mar 25 22:25 description
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 hooks
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 info
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 objects
drwxrwxrwx 1 guile guile 512 Mar 25 22:25 refs
guile@DESKTOP-ODR37SB:~/git-projects/.git$
```

Ve bir Repo böyle görünüyor. Git, projenizdeki dosya ve klasörlerin tüm sürümlerini anlar ve bunları özel bir şekilde kaydeder. Tüm dosya ve klasörleriniz Git Veritabanında saklanacaktır.

Git'in Verileri Nasıl Yönettiğini Anlama

Git deposunda dosyanız üç ana durumda bulunabilir: **Modified(Değiştirilmiş)**, **Staged(Aşamalı)** ve **Committed(Bağlılık)**.

- **Modified** , dosyayı değiştirdiğiniz, ancak henüz veritabanınıza (repo) kaydetmediğiniz anlamına gelir.
- **Staged** , bir sonraki işleme anlık görüntünüze geçmek için değiştirilmiş bir dosyayı geçerli sürümünde işaretlediğiniz anlamına gelir.
- **Committed** , verilerin yerel veritabanınızda güvenli bir şekilde saklandığı anlamına gelir.

Bu bizi Git projesinin üç ana bölümüne götürür:

- The working tree (Çalışan ağaç),
- The staging area (Sahne alanı),
- The Git directory (Git dizini).

The working tree(Çalışan ağaç), projenin bir versiyonunun tek bir çıkışıdır. Bu dosyalar Git dizinindeki sıkıştırılmış veritabanından çıkarılır ve kullanmanız veya değiştirmeniz için diske yerleştirilir.

The staging area(Hazırlama alanı), genellikle Git dizininizde bulunan ve bir sonraki taahhüdünüze nelerin gireceği hakkında bilgi depolayan bir dosyadır. Git terminolojisiindeki teknik adı **“indeks”**tir, ancak “hazırlama alanı” ifadesi de aynı şekilde çalışır.

The **Git directory**(**Git dizini**) (**.git**), Git'in projeniz için meta verileri ve nesne veritabanını depoladığı yerdir.

Dosyalarınızı yerel deponuza yükledikten sonra, istediğiniz zaman bunları kontrol edebilir, ardından istediğiniz gibi ekleyebilir ve yeniden oluşturabilirsiniz.

Yerel Git Örneğini Yapılandırma

Şimdi yerel git örneği yapılandırmamıza geçelim,

"touch" komutunu girerek bazı dosyalar oluşturalım.

```
$ touch index.html file.txt cprog.c javaprogram.java index.js style.css type.ts  
$ ls -al
```

Daha sonra boş bir yerel repo oluşturmak için "git init" komutunu yürütürsünüz.

```
$ git init
```

"home/guile/lab1.1/.git" içinde boş Git deposu başlatıldı.

Check Git status

```
$ git status
```

```
On branch master  
No commits yet  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    cprog.c  
    file.txt  
    index.html  
    index.js  
    javaprogram.java  
    style.css  
    type.ts  
nothing added to commit but untracked files present (use "git add" to track)
```

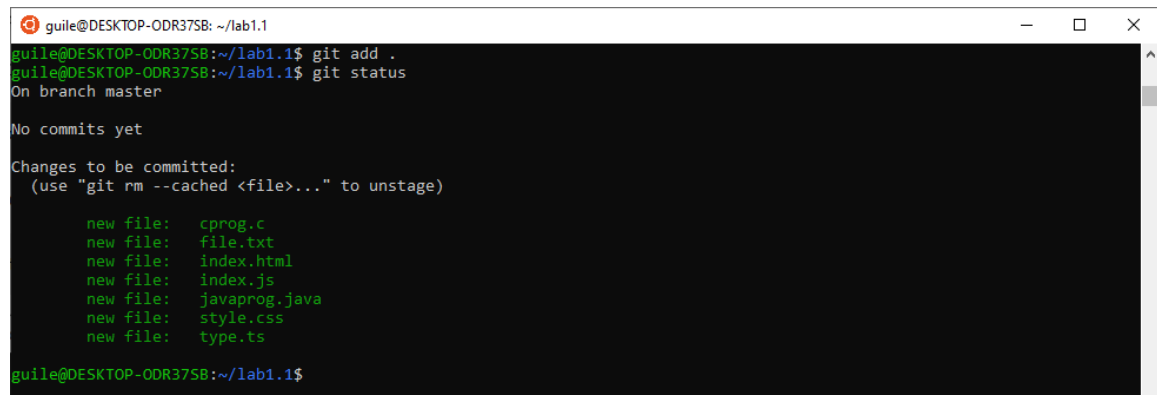
Kırmızı renkli bu dosyalar çalışma alanında duruyor ama Git tarafından izlenmiyor.

Dosyaları çalışma alanından hazırlama alanına eklemek için "Git add ."

```
$ git add .
```

Ve artık dosyaların yeni durumunu görebiliriz.(Yeşil renge dikkat edin)

```
$ git status
```



```
guile@DESKTOP-ODR37SB: ~/lab1.1
guile@DESKTOP-ODR37SB:~/lab1.1$ git add .
guile@DESKTOP-ODR37SB:~/lab1.1$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   cprog.c
        new file:   file.txt
        new file:   index.html
        new file:   index.js
        new file:   javaprogram.java
        new file:   style.css
        new file:   type.ts

guile@DESKTOP-ODR37SB:~/lab1.1$
```

Komutu ile dosyalarımızı local repoya atayalım.

```
$ git commit -m "This folder includes demo files"
```

Tebrikler! Yerel git deponuza(repo) ilk taahhüdünüzü tamamladınız.

▼ Version Control Systems

What is a Version Control System (VCS)?

Sürüm Kontrol Sistemi (Revizyon/Kaynak Kontrol Sistemi olarak da bilinir), **belirli sürümleri daha sonra geri çağırabilmeniz için zaman içinde dosyalarda yapılan değişiklikleri kaydetmek üzere tasarlanmış bir yazılımdır**. Kemerinizin altında sınırsız bir geri alma/yineleme özelliği olması gibi. Seçili dosyaları önceki bir duruma geri döndürmenize, tüm projeyi önceki bir duruma döndürmenize, zaman içindeki değişiklikleri karşılaştırmanıza, soruna neden olabilecek bir şeyi en son kimin değiştirdiğini, kimin bir sorunu ne zaman ortaya çıkardığını ve daha fazlasını görmenizi sağlar. Projeniz için çok güçlü bir araçtır.

VCS, kaynak kodundaki ve öğrenmedeki değişiklikleri izlemek için bir araçtır.

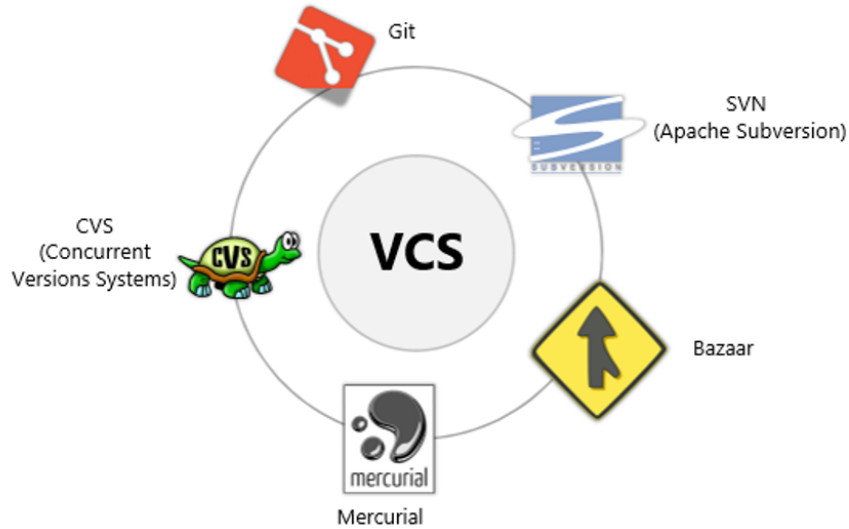
- Ne değişti
- Ne zaman değişti
- Neden değişti
- Kim değiştirdi

Versiyon Kontrol Sistemleri İhtiyacı

Bir sistemde Sürüm Kontrol Sistemi (VCS) kullanmanın birçok nedeni vardır. *İşte kullanma nedenleri.* Dosyaların bir yerde saklanması gerekir. Bunları istediğiniz yerde saklayabilirsiniz, **ancak bir VCS'de saklarsanız asla kaybetmezsiniz.** Bir dosya ile ilişkili bildirilen her değişiklik (check-in) de mevcuttur. Bir dosyanın önceki tüm sürümleri kolayca çıkarılabilir.

Popular Version Control Systems

Some of the most preferred and popular open-source version control systems and tools are listed:



Versiyon Kontrol Sistemlerinin Önemi

VCS neden önemlidir? Çünkü bize yardım ediyor.

Sürüm kontrolü, değişiklikleri takip etmek ve her ekip üyesinin doğru sürüm üzerinde çalışmasını sağlamak için önemlidir. Birden fazla ekip üyesinin üzerinde ortak çalışacağı tüm kodlar, dosyalar ve öğeler için sürüm kontrol yazılımı kullanmalısınız.

Sürüm Kontrol Sistemleri(VCS) Türleri

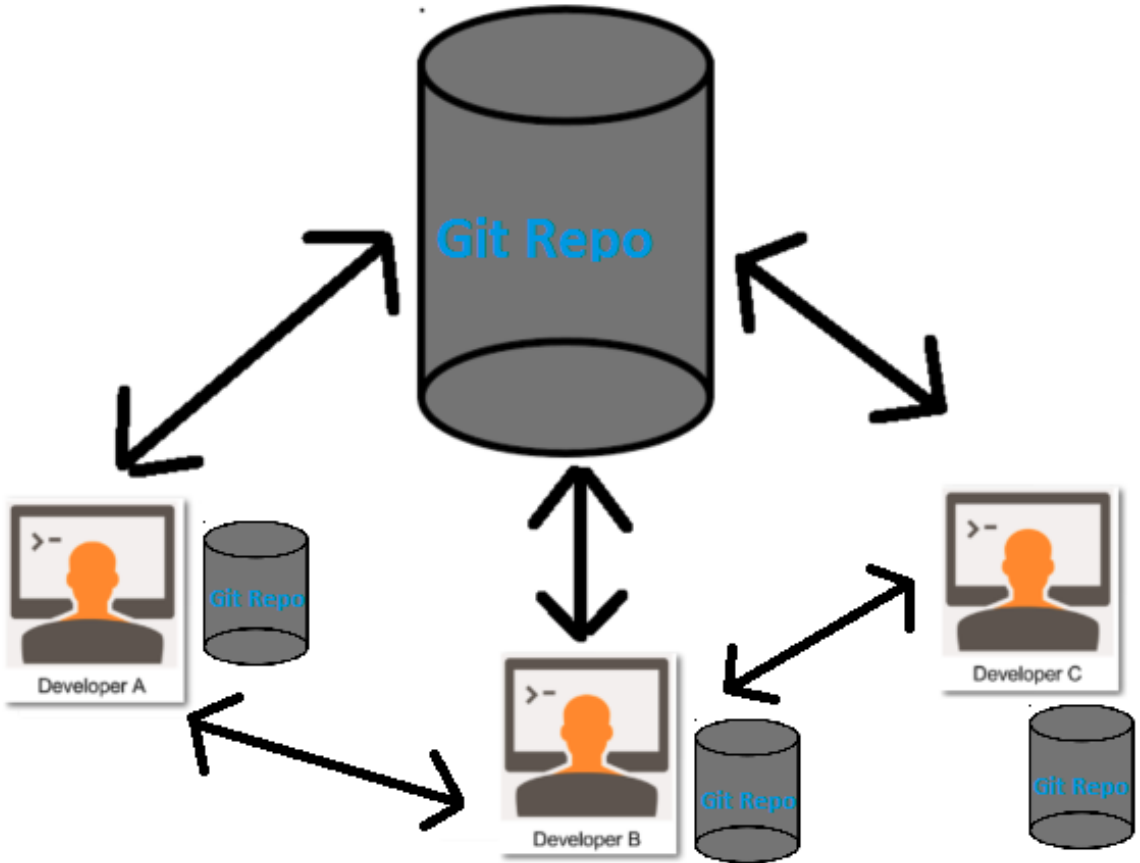
Üç tür sürüm kontrol sistemi vardır:

- File-based(Dosya tabanlı),
- Client-server type(İstemci-sunucu türü),
- Distributed(Dağıtılmış)

Dosya tabanlı sürüm kontrol sistemleri eskidir ve pek kullanılmamaktadır. Dağıtılmış VCS'ler piyasada daha yaygındır. *Git, Distributed(dağıtılmış) bir VCS türüdür.*

Distributed(Dağıtılmış) Version Control Systems

Dağıtılmış sürüm kontrol sistemleri, *her bilgisayarda deponun kopyalarını oluşturur*. Her kullanıcının bir replika üzerinde çalışması gerekir ve bunu ağ bağlantısı kesilse bile yapabilir. Bağımsız olarak çalışabilen ve değişiklikleri birleştirme için uygulayabilen büyük projeler ve bağımsız geliştiriciler için uygundurlar.



Dağıtılmış VCS'nin Özellikleri

- Güvenilir yedek kopyalar
- Hızlı birleştirme ve esnek dallanma
- Hızlı geri bildirim ve daha az birleştirme çakışması
- Çevrimdışı çalışma esnekliği

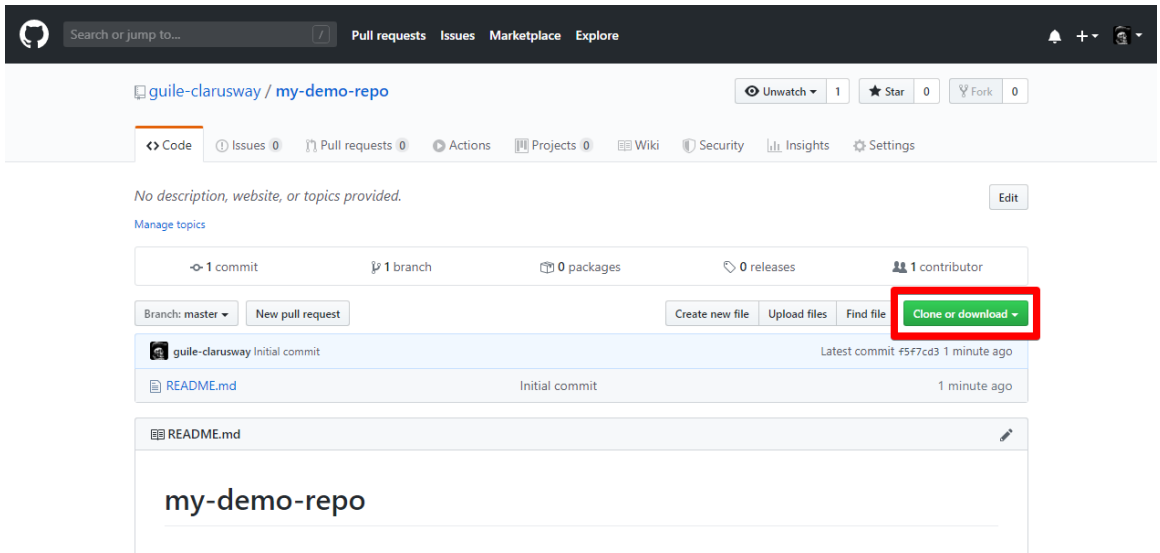
▼ Git Operations

▼ Creating a Remote Repo on GitHub

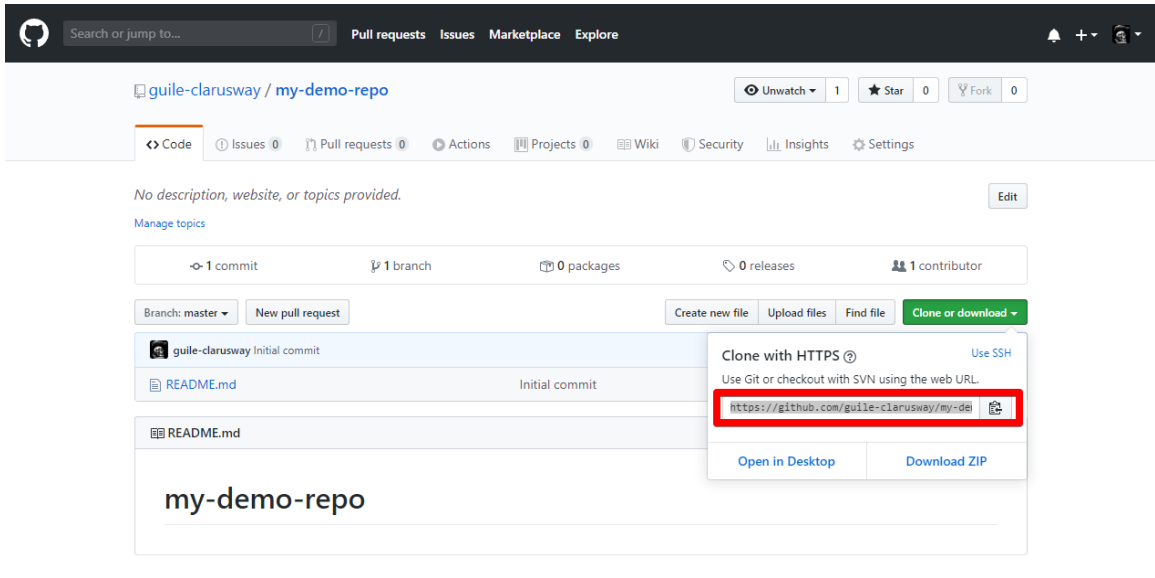
Bu dersimizde hem yerel Git deposu hem de uzak GitHub deposu üzerinde çalışacağız. Aralarındaki etkileşimi göreceğiz.

Gidip GitHub'da oturum açalım, ardından yeni bir repository(depo) oluşturalım, ancak **Read Me** dosyası başlatmayı seçmeyi unutmayın. Bu otomatik olarak ilk taahhüdümüzü oluşturacak, böylece daha sonra onu yerel depomuza kopyalayıp üzerinde çalışabileceğiz. Bu demo için repo'muza my-demo-repo'muz adını vereceğiz. İşiniz bittiğinde **Create repository** ye tıklayın.

Yeni repo oluşturuldu. Şimdi **Clone or download** düğmesine tıklayın.



Ardından deponun URL'sini kopyalayın. Bu URL'yi daha sonra klonlamada kullanacağız.



▼ Cloning Remote Repo to Local

Artık repomuz oluşturuldu ve hazır, üzerinde çalışmaya başlayabiliriz. Ancak, Git'in Dağıtılmış bir VCS olduğunu ve aynı projedeki diğer bazı meslektaşlarınızın aynı depo ile çalışması gerekebileceğini unutmayın. Böylece onu yerel depomuza klonlayacağız ve yerel olarak üzerinde çalışacağız.

GitBash/Terminal'i açın ve aşağıdaki komutu yazın. Oluşturduktan sonra kopyaladığınız kendi repo URL'nizi kullanın.

```
$ git clone YOUR_REPO_URL
```

Bu durum için örnek budur. Kendi repo URL'nizi kullanacaksınız.

```
$ git clone https://github.com/guile-clarusway/my-demo-repo.git
```

```
guile@DESKTOP-ODR375B: ~  
guile@DESKTOP-ODR375B:~$ git clone https://github.com/guile-clarusway/my-demo-repo.git  
Cloning into 'my-demo-repo'...  
remote: Enumerating objects: 3, done.  
remote: Counting objects: 100% (3/3), done.  
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), done.  
guile@DESKTOP-ODR375B:~$
```

Böyle bir şey görürseniz, onu yerel deponuza başarıyla klonladınız. Hadi kontrol edelim,

```
$ ls
```

ve klasörü açın.

```
$ cd my-demo-repo
```

Şimdi geelim repoda neler var.

```
$ ls -al
```

```
guile@DESKTOP-ODR375B: ~/my-demo-repo  
guile@DESKTOP-ODR375B:~$ ls  
my-demo-repo  
guile@DESKTOP-ODR375B:~$ cd my-demo-repo/  
guile@DESKTOP-ODR375B:~/my-demo-repo$ ls -al  
total 0  
drwxrwxr-x 1 guile guile 512 Apr  7 23:16 .  
drwxr-xr-x 1 guile guile 512 Apr  7 23:16 ..  
drwxrwxr-x 1 guile guile 512 Apr  7 23:16 .git  
-rw-rw-r-- 1 guile guile 14 Apr  7 23:16 README.md  
guile@DESKTOP-ODR375B:~/my-demo-repo$
```

Depodaki tek dosya olan **README.md**'ye ek olarak **.git**'in zaten mevcut olduğunu fark etmelisiniz. Uzaktan kumandadan klonladığınızda bir repo başlatmak için `git init` kullanmanıza gerek yoktur.

Artık repomuzda local olarak değişiklik yapabiliriz.

▼ Creating a New Branch(Yeni Dal Oluşturma)

Diyelim ki yeni bir fikrimiz var ama bundan tam olarak emin değiliz. Bir deney yapmak istiyoruz. **Branches(dallar)** bu tür bir durumun anahtarıdır. Yeni bir **branch(dal)** oluşturabilir, üzerinde biraz ilerleme kaydedebilir ve yeni daldan memnun değilseniz ana dal'a geri dönebilirsiniz. Dallar belirli bir taahhüt için referans veya yer imi olarak düşünebilirsiniz. Bu demo için yeni bir dal oluşturacağız ve üzerinde çalışacağız.

İlk önce mevcut branch(dal ınızı) kontrol edin

```
$ git branch
```

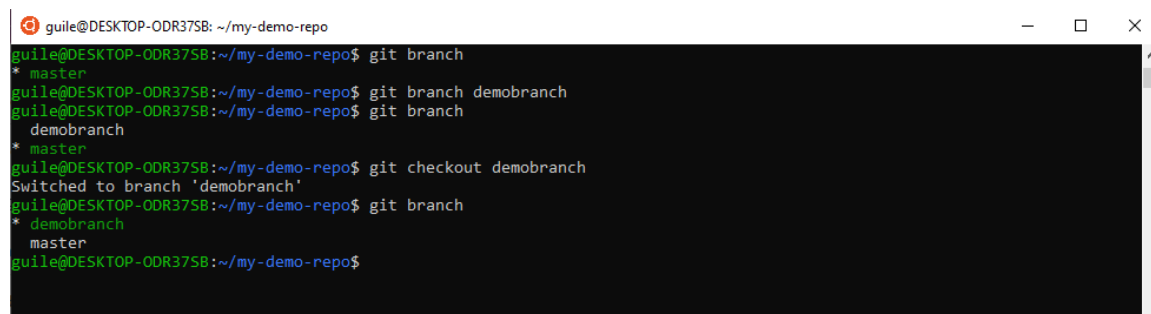
Ardından yeni bir dal oluşturun. Bu ders için demobranh kullanacağız.

```
$ git branch demobranh
```

Tekrar kontrol edin ve gördüğünüz gibi yeni dal oluşturuldu ancak yeşil renkli olduğu için hala ana daldayız. Şimdi demobranh'a geçin

```
$ git checkout demobranh  
  
$ git branch
```

Artık yeni daldayız ve çalışmaya başlayabiliriz.

A terminal window titled 'guile@DESKTOP-ODR37SB: ~/my-demo-repo' showing the following commands and output:

```
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch  
* master  
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch demobranh  
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch  
demobranh  
* master  
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git checkout demobranh  
Switched to branch 'demobranh'  
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git branch  
* demobranh  
master  
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

Projemize basit bir Python dosyası ekleyelim. Favori düzenleyicinizi kullanın, dosyaya `print("Hello World")` ekleyin.

```
$ vim hello.py
```

Dilerseniz kontrol edebilirsiniz.

```
$ cat hello.py
```

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ vim hello.py
guile@DESKTOP-ODR37SB:~/my-demo-repo$ cat hello.py
print("Hello World")
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

We can stage(sahneleyebiliriz),

```
$ git add .
```

And commit it now.

```
$ git commit -m "New python file added"
```

Let's check our log

```
$ git log
```

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git add .
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git commit -m "New python file added"
[demobranch aff9ce6] New python file added
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git log
commit aff9ce61534ccdb1196de4d3c3dd9f997eee0f47 (HEAD -> demobranch)
Author: guile <guile@clarusway.com>
Date: Tue Apr 7 23:54:50 2020 +0300

    New python file added

commit f5f7cd3ef4ee19268b67a0a71730d0006fcfaa17 (origin/master, origin/HEAD, master)
Author: guile-clarusway <59561051+guile-clarusway@users.noreply.github.com>
Date: Tue Apr 7 22:36:37 2020 +0300

    Initial commit
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

▼ Pushing Branch to Remote(Dal'ı Remote'a aktarma)

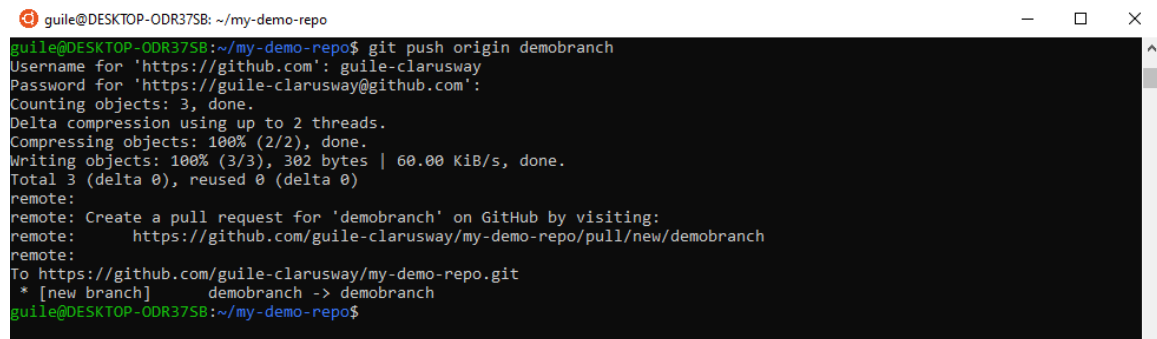
Artık local repo ile işimiz bittiğine göre, onu remote'a itelim. Aşağıdaki komut, yerel repo içeriğimizi uzak repoya yükleyecektir.

```
$ git push origin demobbranch
```

Enter your GitHub username and password if asked.

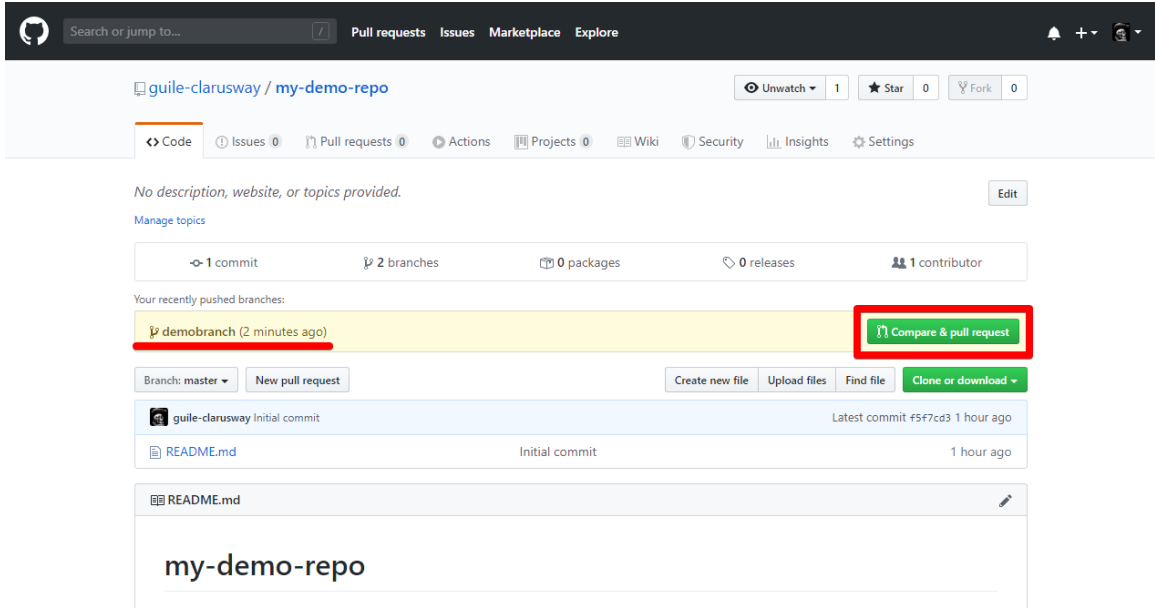
İlk GitHub Dersini hatırlıyorsanız, git push komutundan önce git remote add Origin komutunu kullandık. Burada bu komuta ihtiyacımız yok çünkü repomuzu uzaktan klonladık ve hem uzak hem de yerel repo zaten birbiriyle ilişkili.

Bastıktan sonra görmeniz gereken şey budur.



```
guile@DESKTOP-ODR375B: ~/my-demo-repo
guile@DESKTOP-ODR375B:~/my-demo-repo$ git push origin demobbranch
Username for 'https://github.com': guile-clarusway
Password for 'https://guile-clarusway@github.com':
Counting objects: 3, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 302 bytes | 60.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'demobbranch' on GitHub by visiting:
remote:   https://github.com/guile-clarusway/my-demo-repo/pull/new/demobbranch
remote:
To https://github.com/guile-clarusway/my-demo-repo.git
 * [new branch]      demobbranch -> demobbranch
guile@DESKTOP-ODR375B:~/my-demo-repo$
```

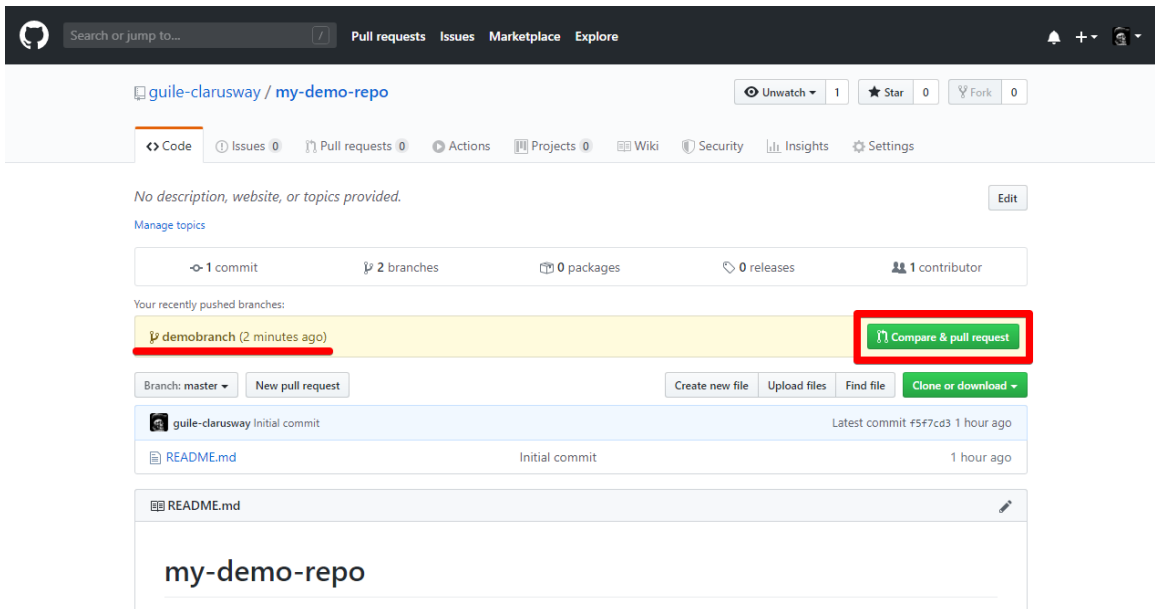
Şimdi gidip uzak depomuzu kontrol edelim. Dalımızın başarıyla ittiyseniz, demobbranch'ı görmelisiniz. Uzak depomuzu ilk oluşturduğumuzda yalnızca bir dalımız olduğunu unutmayın. Şimdi iki tane var.



▼ GitHub'da Çekme İsteği (PR) Oluşturma(Pull Request)

Çekme İsteği (PR), diğer katkıda bulunanlara veya ekip üyelerine depoda bazı değişiklikler (hataları düzeltme, yeni özellikler ekleme vb.) yaptığınızı ve bunları depoya katkıda bulunmak istediğinizi söylediğiniz bir yöntemdir. Bu değişiklikler yetkili kişiler tarafından kontrol edilip onaylandıktan sonra repo ile birleştirilir.

Burada kendi repomuz için bir çekme talebi oluşturacağız. Bir çekme isteği açmak için **Compare & pull request**'e tıklayın.



Able to merge açıklamasına dikkat edin. Şubemiz birleştirilebilir. İsterseniz bir açıklama yazabilirsiniz, son taahhüt mesajınız otomatik olarak eklenecektir ("New python file added").

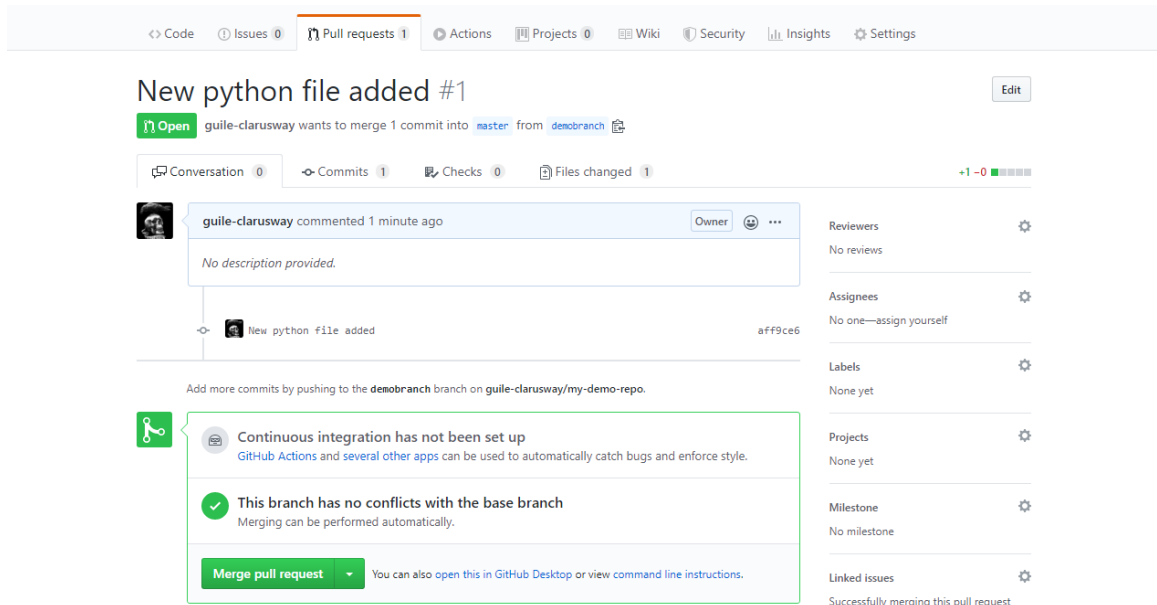
Click **Create pull request**.

The screenshot shows the GitHub interface for creating a pull request. At the top, the repository name 'guile-clarusway / my-demo-repo' is displayed along with 'Unwatch', 'Star' (1), and 'Fork' (0) buttons. Below this is a navigation bar with links to 'Code', 'Issues' (0), 'Pull requests' (0), 'Actions', 'Projects' (0), 'Wiki', 'Security', 'Insights', and 'Settings'. The main heading is 'Open a pull request', followed by a subtext: 'Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).' Below this is a comparison bar showing 'base: master' and 'compare: demobranh' with a green checkmark and the text 'Able to merge. These branches can be automatically merged.' The main content area has a title 'New python file added' and a 'Write' tab. Below the title is a 'Leave a comment' text area and an 'Attach files by dragging & dropping, selecting or pasting them.' section. A green 'Create pull request' button is at the bottom right. On the right side, there are settings for 'Reviewers' (No reviews), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), 'Milestone' (No milestone), and 'Linked issues'.

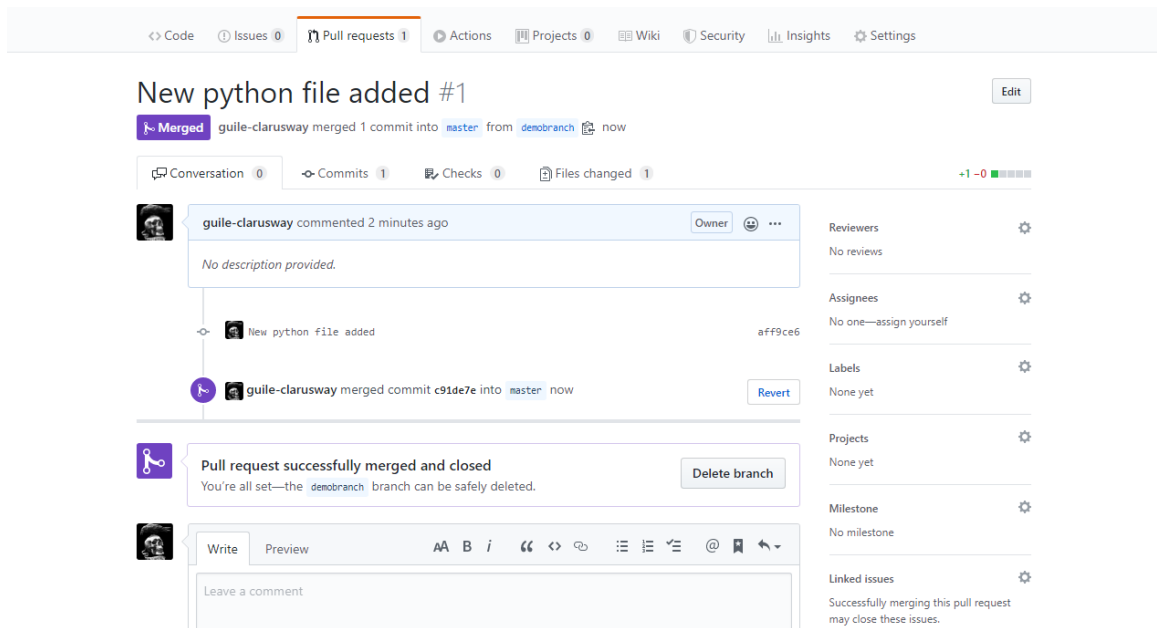
▼ Merging Pull Request(Çekme İsteğini Birleştirme)

Şimdi çekme isteğini birleştirme zamanı. Bu deponun tek sahibi ve katkıcısı olduğumuz için kendi çekme talebimizi birleştireceğiz.

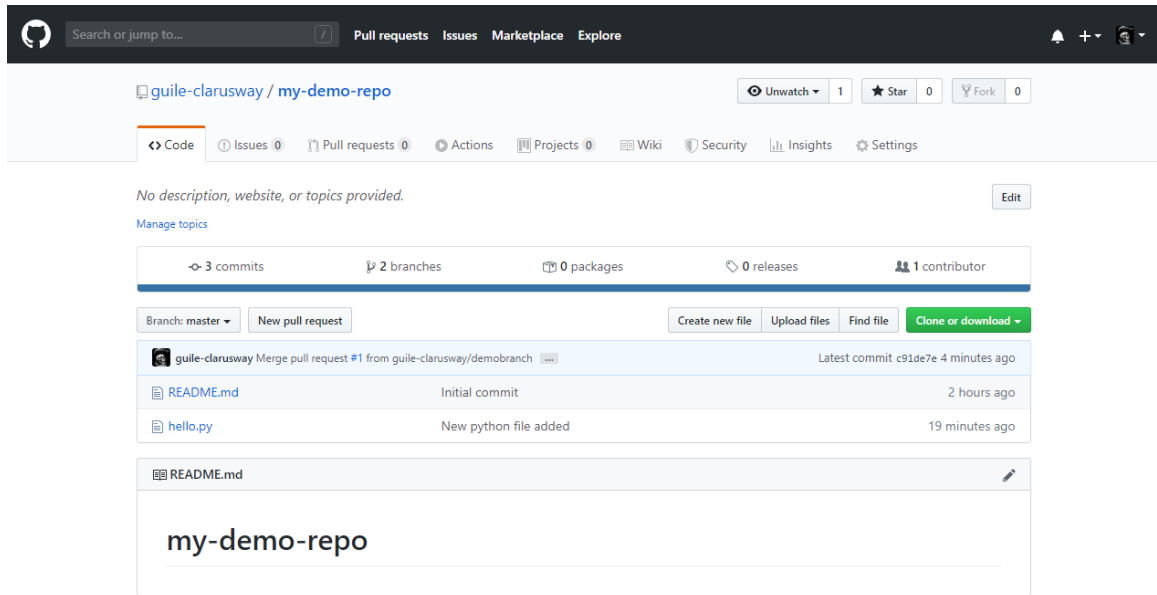
Gördüğünüz gibi birleştirilecek bir çakışma yok. **Merge pull request**'i tıklayın ve onaylayın.



Pull request successfully merged and closed.



Birleştirmeden sonra deponun nasıl görüldüğünü görmek için **Code** sekmesine tıklayın. Gördüğünüz gibi demobranh **master** **branch** ile birleştirildi.



▼ Updating Local Repo

Bildiğiniz gibi yerel repomuzda yeni bir şube oluşturduk ve onu uzak repoya ittik. Artık uzak repomuz (GitHub repo) ana dalda ve güncel, ancak yerel repo değil. Öyleyse GitBash'e geri dönelim

İlk önce, master branch(ana şubey)e geçin.

```
$ git checkout master
```

Then type

```
$ git pull origin master
```

to update our local repo.

```
guile@DESKTOP-ODR37SB: ~/my-demo-repo
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
guile@DESKTOP-ODR37SB:~/my-demo-repo$ git pull origin master
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.com/guile-clarusway/my-demo-repo
* branch      master      -> FETCH_HEAD
f5f7cd3..c91de7e master    -> origin/master
Updating f5f7cd3..c91de7e
Fast-forward
 hello.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 hello.py
guile@DESKTOP-ODR37SB:~/my-demo-repo$
```

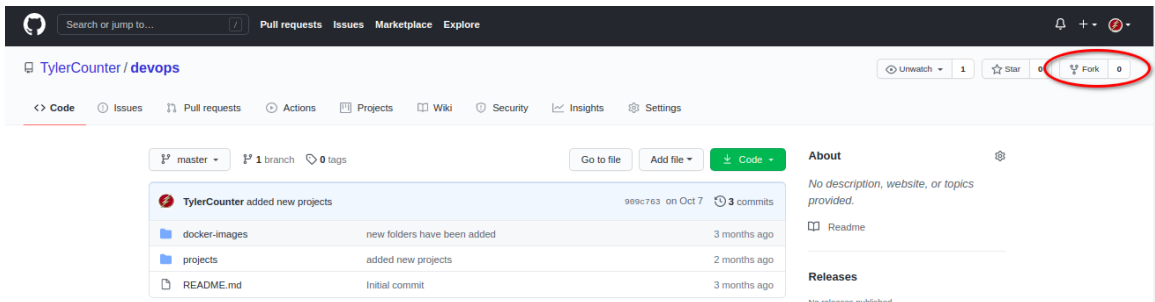
▼ Forking Projects(Çatallı Projeler)

GitHub'ı bir süre kendi başınıza kullandıktan sonra, kendinizi başka birinin projesine katkıda bulunmak isterken bulabilirsiniz. Ya da belki birinin projesini kendi projeniz için başlangıç noktası olarak kullanmak istersiniz. Bu işlem çatallama olarak bilinir.

Bir "fork" oluşturmak, başka birinin projesinin kişisel bir kopyasını üretmektir. Çatallar, orijinal depo ile kişisel kopyanız arasında bir tür köprü görevi görür. Değişikliklerinizi orijinal projeye kadar sunarak diğer kişilerin projelerini daha iyi hale getirmeye yardımcı olmak için Çekme İstekleri gönderebilirsiniz. Forking, GitHub'da sosyal kodlamanın merkezinde yer alır.

Fork the repository

Devops deposunu çatallamak için deponun başlığındaki Çatal düğmesini tıklayın.



Clone your fork

Devops deposunu başarıyla çatalladınız, ancak şu ana kadar yalnızca GitHub'da var. Proje üzerinde çalışabilmek için, onu bilgisayarınıza kopyalamanız gerekecek.

Nasıl klonlayacağınız size kalmış. Bazı seçenekler komut satırıyla veya GitHub Desktop kullanılarak klonlanıyor.

Değişiklik yapmak ve zorlamak

Devam edin ve favori metin düzenleyicinizi kullanarak projede birkaç değişiklik yapın. Örneğin, GitHub kullanıcı adınızı eklemek için [README.MD](#)'deki metni değiştirebilirsiniz.

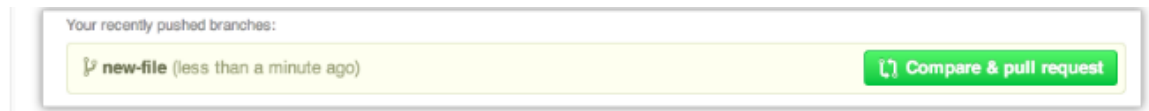
Değişikliklerinizi göndermeye hazır olduğunuzda, değişikliklerinizi gerçekleştirin ve `commit` edin.

Şu anda Git'e aslında "Tamam, değişikliklerimin anlık görüntüsünü aldım!" dediniz. Daha fazla değişiklik yapmaya devam edebilir ve daha fazla kesinleştirme anlık görüntüsü alabilirsiniz. Değişikliklerinizi [GitHub.com](#)'a aktarmaya hazır olduğunuzda, değişikliklerinizi uzaktan kumandaya aktarın.

Çekme Talebi Yapmak(Making a Pull Request)

Sonunda, ana projede değişiklik önermeye hazırsınız! Bu, bir başkasının projesinin çatalını oluşturmanın son adımı ve tartışmasız en önemlisidir. Topluluğa bir bütün olarak fayda sağlayacağını düşündüğünüz bir değişiklik yaptıysanız, kesinlikle geri katkıda bulunmayı düşünmelisiniz.

Bunu yapmak için projenizin yaşadığı [GitHub.com](#)'daki depoya gidin. Bu örnek için <https://www.github.com//devops> adresinde olacaktır. Yakın zamanda yeni bir şubeye ittiğinizi ve bu şubeyi "upstream" olarak orijinal depoya gönderebileceğinizi belirten bir başlık göreceksiniz:



[Compare and Pull Request](#) e tıklamak sizi bir başlık ve isteğe bağlı açıklama girebileceğiniz bir tartışma sayfasına gönderir. Bu Çekme Talebini ilk etapta neden yaptığınıza dair çok sayıda yararlı bilgi ve bir gerekçe sağlamak önemlidir. Proje sahibinin, değişikliğinizin düşündüğünüz kadar herkes için yararlı olup olmadığını belirleyebilmesi gerekir.

İçten argümanınızı yazmaya hazır olduğunuzda, *Send a pull request*'e tıklayın.

Create CONTRIBUTING.md

Write Preview

⌨️ Parsed as Markdown ⌕ Edit in fullscreen

Let's add a contributing file so we can work better, together|

Attach images by dragging & dropping, selecting them, or pasting from the clipboard.

✓ Able to merge.
These branches can be automatically merged.

Create pull request

Çekme İstekleri, tartışma için bir alandır. Bu durumda, Tyler çok meşguldür ve muhtemelen değişikliklerinizi birleştirmeyecektir. Diğer projeler için, proje sahibi Çekme Talebinizi reddederse veya bunun neden yapıldığına dair daha fazla bilgi isterse alınmayın. Hatta proje sahibi sizin çekme talebinizi birleştirmemeyi seçmiş olabilir ve bu tamamen sorun değil. Kopyanız internette rezillik içinde var olacak. Ve kim bilir belki de hiç tanışmadığınız biri değişikliklerinizi orijinal projeden çok daha değerli bulacaktır. Paylaşın ve aynı şekilde paylaşın!