GitHub & Remotes

Remote nedir, git clone/push/fetch/pull nasıl çalışır, fetch'in pull'dan farkı (fetch: entegre etmeden indir, pull: fetch + entegre), tracking branch'ler ve git push -u, ve yerel/remote branch'leri senkronize etmek.

Orta 35 dk

Son güncelleme: 27 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/github-and-remotes

---

Şimdiye kadarki her şey tamamen makinende gerçekleşti. Bu ders, yerel repository'ni GitHub'a bağlıyor — böylece işin yedeklenir, paylaşılabilir olur ve takımının geri kalanıyla senkronize olur. Baştan sona aklında tutman gereken temel fikir şu: yerel repository'n ve GitHub'daki repository, aynı projenin iki bağımsız, eksiksiz kopyasıdır, ve Git komutları özellikle commit'leri aralarında kontrollü bir şekilde taşımak için vardır.

Remote Nedir?

Bir **remote**, basitçe repository'nin başka bir kopyasına — genellikle GitHub'da barındırılan bir kopyaya — adlandırılmış bir referanstır. Canlı bir bağlantı ya da bir mount değildir; yalnızca Git'in hatırladığı bir URL'dir, artı o diğer kopyanın branch'lerinin en son kontrol ettiğinde neye benzediğine dair yerel bir kayıt.

Yerel Repository vs Remote Repository

Yerel repository'n ve bir remote repository, ikisi de *eksiksizdir* — remote, "kısmi" ya da "yalnızca-sunucu" bir sürüm değildir. Her tarafın kendi tam commit geçmişi vardır. GitHub ile etkileşime girdiğinde, her zaman commit'leri açıkça bir yönde ya da diğerinde aktarırsın (`push` seninkileri oraya gönderir, `pull` / `fetch` onlarınkini buraya getirir) — arka planda hiçbir şey otomatik olarak senkronize olmaz.

git remote

```
git remote           # remote isimlerini listele (genellikle yalnızca "origin")
git remote -v        # aynısı, ama gerçek URL'leri göster (fetch + push)
```

Konvansiyon olarak, push ettiğin ve pull ettiğin ana remote `origin` olarak adlandırılır — bu yalnızca bir isimdir, bir Git anahtar kelimesi değil, ama neredeyse her proje bunu kullanır.

origin Eklemek

Bir repository'yi önce yerel olarak oluşturduysan (`git init`) ve onu yeni, boş bir GitHub repository'sine bağlamak istiyorsan:

```
git remote add origin https://github.com/your-username/task-tracker.git
git push -u origin main
```

`git remote add`, URL'yi `origin` ismi altında kaydeder; sonrasındaki push'taki `-u` flag'i birkaç bölüm aşağıda ayrıntılı olarak ele alınıyor.

git clone

Repository zaten GitHub'da varsa ve yerel bir kopyasını istiyorsan, `git clone` bunu tek adımda yapar — tüm geçmişi indirir *ve* otomatik olarak ona geri işaret eden `origin`'i kurar:

```
git clone https://github.com/your-username/task-tracker.git
```

`clone`, neredeyse her var olan projede çalışmaya başlama şeklindir — yerleşik bir takımla çalışırken yepyeni, boş bir repo'yu nadiren `git init` edeceksin.

git push

`git push`, yerel commit'lerini remote'a gönderir:

```
git push origin main
```

Bu, `main`'in yerelde sahip olduğu ama `origin`'in `main` kopyasının henüz sahip olmadığı commit'leri yükler. Remote'ta yerel branch'inin sahip olmadığı commit'ler varsa, Git push'u reddeder (bu bir güvenlik özelliğidir — bilinçli override'ı, `--force`'u, bu kursta daha sonra ele alıyoruz).

git fetch

`git fetch`, remote'tan yeni commit'leri **working tree'ne ya da yerel branch'lerine dokunmadan** indirir:

```
git fetch origin
```

Bu, yerel repository'ni GitHub'da neler olduğu konusunda haberdar tutan iki adımlı sürecin ilk yarısıdır. Bir fetch'ten sonra, yerel `main`'in hiç hareket etmemiştir — Git yalnızca `origin/main`'in şu an nereye işaret ettiğine dair *kaydını* güncellemiştir.

git pull

`git pull`, fetch'i yapar, sonra alınan değişiklikleri hemen mevcut branch'ine merge eder:

```
git pull origin main
```

`git fetch`, remote değişikliklerini entegre etmeden indirir; `git pull`, fetch + entegre etmektir. Bu, bu dersteki en önemli tek ayrımdır. `pull` pratiktir, ama bir merge'in (ya da Git yapılandırmana bağlı olarak bir rebase'in) hemen ve otomatik olarak gerçekleşeceği anlamına gelir — ki bazı geliştiricilerin önce `fetch` etmeyi, neyin değiştiğine bakmayı ve sonrasında bilinçli olarak merge etmeyi tercih etmesinin tam olarak sebebi budur.

Fetch vs Pull

- **Remote commit'leri indirir** — hem `git fetch` hem `git pull` indirir.
- **Working tree'ni günceller** — `git fetch`: hayır. `git pull`: evet.
- **Mevcut branch'ine merge eder** — `git fetch`: hayır. `git pull`: evet, otomatik olarak.
- **Ne zaman kullanılır** — `git fetch`, entegre etmeden önce bakmak istediğinde; `git pull`, şimdi güncel olmak istediğinde.

Commit Edilmemiş Değişikliklerin Varken git pull Çalıştırırsan Ne Olur

`pull` bir merge ile sona erdiği için, branch değiştirmekle aynı güvenlik kuralına tabidir: **Eğer merge, güvenle birleştiremeyeceği commit edilmemiş değişikliklerin üzerine yazacaksa, Git pull etmeyi reddeder.** Önce değişikliklerini commit etmeni ya da stash'lemeni söyleyen bir hata görürsün. Eğer commit edilmemiş değişikliklerin, gelen commit'lerin dokunduğu satırlara dokunmuyorsa, Git bunları otomatik olarak merge *edebilir* — ama buna güvenme; pull etmeden önce commit etmek (ya da stash'lemek, bu kursta daha sonra ele alınacak) güvenli alışkanlıktır.

Tracking Branch'ler

Bir repository'yi clone'ladığında, yerel `main`'in `origin/main`'i **tracking** etmesi otomatik olarak kurulur — Git bu ilişkiyi hatırlar, ki bu bağlantı var olduktan sonra her seferinde `origin main` belirtmeden düz `git push` / `git pull` çalıştırmana izin veren şeydir.

```
git branch -vv    # yerel branch'leri neyi tracking ettikleriyle birlikte göster
```

git push -u

Yepyeni bir yerel branch'i ilk kez push ederken, tracking ilişkisini kurmak için `-u`'ya (`--set-upstream`'in kısası) ihtiyacın vardır:

```
git push -u origin add-password-reset
```

Bu tek seferlik kurulumdan sonra, o branch'te düz `git push` ve `git pull`, her seferinde `origin add-password-reset` belirtmene gerek kalmadan commit'leri tam olarak nereye gönderip nereden alacağını bilir.

Remote Branch'leri Silmek

Yerel bir branch'i silmenin (`git branch -d`) remote kopya üzerinde hiçbir etkisi yoktur — bağımsızdırlar. GitHub'ın kendisinde bir branch'i silmek için:

```
git push origin --delete add-password-reset
```

Yerel ve Remote Branch'leri Senkronize Etmek

Tam resmi bir araya getirirsek, bir remote ile tipik bir gün şöyle görünür:

```
#!/bin/sh
# A typical day: check for remote changes, pull them in, push a new branch,
# and keep it up to date with main.

git fetch origin
# Output:
# From github.com:your-username/task-tracker
#    a1b2c3d..e5f6a7b  main       -> origin/main

# Two new commits exist on origin/main that we don't have locally yet.
git pull origin main
# Output:
# Updating a1b2c3d..e5f6a7b
# Fast-forward
#  TaskController.java | 9 +++++++++
#  1 file changed, 9 insertions(+)

# Start a new feature branch and push it for the first time.
git switch -c add-due-dates
# ... edit Task.java, TaskController.java ...
git add .
git commit -m "Add due date field to Task"
git push -u origin add-due-dates
# Output:
# * [new branch]      add-due-dates -> add-due-dates
# branch 'add-due-dates' set up to track 'origin/add-due-dates'.

# From now on, plain push/pull on this branch work without extra arguments.
git push
```

Yaygın Hatalar

- **`git fetch`'in dosyalarını güncellediğini varsaymak** — yalnızca Git'in remote hakkındaki *bilgisini* günceller; merge edene (ya da pull edene) kadar working tree'n dokunulmamış kalır.
- **Commit edilmemiş değişikliklerle `git pull` çalıştırıp bir conflict'e ya da reddedilmeye şaşırmak** — önce commit et ya da stash'le.
- **Yeni bir branch'in ilk push'unda `-u`'yu unutmak**, sonra düz `git push`'un neden upstream branch olmadığından şikayet ettiğine şaşırmak.

En İyi Pratikler

- Her gün işe başlamadan önce, nasıl entegre edeceğine karar vermeden önce neyin değiştiğini görmek için `git fetch` yap.
- Yeni bir branch'i push ederken ilk seferde her zaman `-u` kullan — bundan sonraki her push/pull'da bunun karşılığını alırsın.
- Bir Pull Request'i tamamlamanın parçası olarak GitHub'da bir branch'i sil (sonraki derste ele alınacak) — eski remote branch'lerin birikmesine izin verme.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Bir remote, repository'nin başka bir kopyasına (genellikle GitHub'da, `origin`) adlandırılmış bir referanstır.
- `git clone` bir repository'yi indirir ve `origin`'i otomatik olarak kurar; `git remote add`, var olan bir yerel repo'yu yeni bir remote'a bağlar.
- `git fetch`, değişiklikleri entegre etmeden indirir; `git pull` = fetch + merge, ve branch değiştirmek gibi commit edilmemiş yerel değişikliklerle engellenebilir.
- `git push -u`, yeni bir branch'i ilk kez push ederken tracking ilişkisini kurar.

Cheat Sheet

```
git clone <url>                   # bir repo indir, origin'i otomatik kur
git remote -v                     # remote URL'lerini göster
git fetch origin                  # değişiklikleri indir, entegre etme
git pull origin main              # değişiklikleri indir VE entegre et
git push origin main              # commit'lerini yükle
git push -u origin <branch>       # yeni bir branch'i push et, tracking'i kur
git push origin --delete <branch> # remote'ta bir branch'i sil
```

Terimler Sözlüğü

- **Remote** — repository'nin başka bir kopyasına (genellikle GitHub'da `origin`) adlandırılmış bir referans.
- **Fetch** — remote commit'lerini, branch'lerine merge etmeden indirmek.
- **Pull** — fetch etmek, sonra hemen mevcut branch'ine merge etmek.
- **Tracking branch** — belirli bir remote branch'e bağlı yerel bir branch, düz `push`/`pull`'a olanak tanır.