Practical Git & GitHub Workflow

Kursun tamamında öğretilen her şeyi tek, gerçekçi bir uçtan uca Spring Boot senaryosunda birleştiren kapanış dersi: clone, feature branch, commit/amend, push, Pull Request, inceleme geri bildirimi, main üzerine rebase, conflict çözümü, force-with-lease, ve squash and merge.

Orta 30 dk

Son güncelleme: 27 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/practical-git-github-workflow

---

Bu kurstaki her komut şimdiye kadar izole olarak öğretildi. Bu kapanış dersi, hepsini tek, sürekli, gerçekçi bir senaryoda bir araya getiriyor — bir Java/Spring Boot geliştiricisinin gerçekten yaşadığı türden bir gün. Burada yeni bir sözdizimi yok; amaç, öğrendiğin her şeyin tek, doğal bir workflow'a nasıl bağlandığını görmek.

Senaryo

Küçük bir Spring Boot task-tracking API'sinde bir `Task` entity'sine bir due-date alanı ekliyorsun. İş tek bir feature branch için yeterince küçük, ama bir Pull Request'i, bir tur inceleme geri bildirimini, ve sen çalışırken başka birinin `main`'e merge ettiği değişikliklerle bir conflict'i içerecek kadar gerçek.

Clone'lamak ve Bir Feature Branch Başlatmak

Bu işe ilk kez katılıyorsun, bu yüzden repository'yi clone'luyorsun, sonra `main`'den bir feature branch başlatıyorsun:

```
git clone https://github.com/your-team/task-tracker.git
cd task-tracker
git switch -c add-task-due-date
```

Değişiklikleri Yapmak ve Gözden Geçirmek

Entity'yi düzenliyorsun, sonra herhangi bir şeyi stage'lemeden önce tam olarak neyi değiştirdiğini kontrol ediyorsun:

```
# ... Task.java'ya bir dueDate alanı eklemek için düzenle ...
git status
git diff
```

İşi Stage'lemek ve Commit Etmek

Diff'ten memnun kalıp stage'liyor ve commit ediyorsun — ve bir an sonra eşleşen veritabanı migration'ını unuttuğunu fark ediyorsun, bu yüzden ayrı bir "oops" commit'i oluşturmak yerine amend ediyorsun:

```
git add Task.java
git commit -m "Add dueDate field to Task entity"

# ... migration dosyasının eksik olduğunu fark et ...
git add V42__add_task_due_date.sql
git commit --amend --no-edit
```

Özellik ilerledikçe birkaç odaklı commit daha ekliyorsun:

```
git add TaskController.java TaskDto.java
git commit -m "Expose dueDate in the Task API response"

git add TaskControllerTest.java
git commit -m "Add tests for dueDate serialization"
```

Push Etmek ve Bir Pull Request Açmak

Özellik incelemeye hazır, bu yüzden onu push ediyorsun — bu branch'in ilk push'u olduğu için `-u` kullanarak — ve GitHub'da `main`'i hedefleyen bir Pull Request açıyorsun.

```
git push -u origin add-task-due-date
```

İnceleme Geri Bildirimine Yanıt Vermek

Bir takım arkadaşı PR'ı inceler ve bir değişiklik ister: tarih, varsayılana güvenmek yerine açıkça ISO-8601 formatında serileştirilmeli. Düzeltmeyi doğrudan aynı branch'te yapıyor ve yeniden push ediyorsun — yeni bir PR gerekmiyor, GitHub var olanı otomatik olarak güncelliyor:

```
# ... Task.dueDate üzerindeki @JsonFormat anotasyonunu ayarla ...
git add Task.java
git commit -m "Serialize dueDate as ISO-8601"
git push
```

main ile Güncel Kalmak

PR'ın incelenirken, bir takım arkadaşı `main`'e ilgisiz bir değişiklik merge etti. Merge etmeden önce, branch'inin en güncel `main`'e karşı test edilmesini istiyorsun — bu yüzden fetch edip rebase ediyorsun:

```
git fetch origin
git rebase origin/main
```

Rebase Sırasında Bir Conflict'i Çözmek

Rebase bir conflict'e çarpıyor — diğer değişiklik de yakın bir satırda `Task.java`'ya dokunmuş. Git rebase ortasında duraklıyor:

```
#!/bin/sh
# Rebasing a feature branch onto an updated main, hitting a conflict in
# Task.java, and resolving it -- the middle of the full end-to-end scenario.

git fetch origin
git rebase origin/main
# Output:
# Auto-merging Task.java
# CONFLICT (content): Merge conflict in Task.java
# error: could not apply 4f5a6b7... Add dueDate field to Task entity
# hint: Resolve all conflicts manually, mark them as resolved with
# hint: "git add/rm <conflicted_files>", then run "git rebase --continue".

# Task.java now contains conflict markers:
#
# public class Task {
#     private String title;
# <<<<<<< HEAD
#     private Priority priority;
# =======
#     private LocalDate dueDate;
# >>>>>>> 4f5a6b7 (Add dueDate field to Task entity)
#     // ...
# }
#
# Both fields are needed -- someone else added `priority` on main while this
# branch added `dueDate`. The correct resolution keeps both.

# ... edit the file to keep both fields, remove all three marker lines ...

git add Task.java
git rebase --continue
# Output:
# Successfully rebased and updated refs/heads/add-task-due-date.

git log --oneline -1
# Output:
# 9e8f7a6 Add dueDate field to Task entity
```

Pull Request'i Güncellemek

Rebase branch'inin commit hash'lerini yeniden yazdığı için, ve bu branch zaten bir kez push edildiği için, düz bir `git push` reddediliyor. Bu feature branch'i kimse başka pull etmediği için, bir force push güvenli — ve `--force-with-lease` bunu yapmanın güvenli yolu:

```
git push --force-with-lease
```

GitHub'ın PR sayfası, rebase edilmiş commit'leri göstermek için otomatik olarak güncellenir.

Pull Request'i Tamamlamak

Reviewer onaylıyor. GitHub'da **Squash and merge**'ü seçiyorsun — branch'in beş work-in-progress commit'i (amend ve inceleme-geri-bildirimi düzeltmesi dahil) `main`'de tek, temiz bir commit haline geliyor: "Add dueDate field to Task entity." GitHub, merge edilmiş PR sayfasında tek tıkla "Delete branch" butonu sunuyor, ki bunu hemen kullanıyorsun.

Terminaline geri dönüp, artık merge edilmiş yerel branch'ini temizliyor ve yeni `main` ile senkronize oluyorsun:

```
git switch main
git pull
git branch -d add-task-due-date
```

Özellik `main`'de canlı, branch gitti (yerel ve remote olarak), ve yerel repository'n tamamen güncel — tam olarak bir sonraki özelliğe başlayacağın yer.

Anahtar Çıkarımlar

- Bu workflow'un tamamı, zaten tek tek bildiğin komutlardan inşa edilmiştir — burada yeni bir sözdizimi yok, yalnızca gerçekçi bir işlem sırası.
- Amend etmek, bilinçli olarak stage'lemek ve odaklı commit'ler yazmak, hepsi bir şey paylaşılmadan *önce* olur — push edildikten ve incelemeye alındıktan sonra, yeni iş yeni commit'ler haline gelir, reviewer'ın zaten gördüğü şeylerin yeniden yazılması değil.
- Kendi henüz merge edilmemiş feature branch'ini güncellenmiş bir `main` üzerine rebase etmek — ve sonrasında `--force-with-lease` ile force-push yapmak — bir takımla çalışmanın tamamen normal, güvenli bir parçasıdır, tam olarak kimsenin o branch'i pull etmemiş olması nedeniyle.
- Squash and merge, sonra branch'i silmek, döngüyü temiz bir şekilde kapatır: `main`, tek, odaklı bir commit alır, ve branch'ten sonrasında hiçbir şey kalmaz.