Branches & Merging

Branch oluşturmak/silmek/yeniden adlandırmak, git switch ile branch değiştirmek, git merge ile fast-forward ve merge commit'in farkı, merge conflict'lere giriş, ve gerçekçi bir Spring Boot feature branch workflow'u. git switch'in git checkout'a neden tercih edildiği.

Orta 35 dk

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

---

Önemsiz olmayan her proje, birbirini etkilemeden aynı anda birden fazla şey üzerinde çalışmanın bir yoluna ihtiyaç duyar — yeni bir özellik, bir hata düzeltmesi, bir deney — ve bunu yaparken `main`'i her zaman çalışır bir durumda tutmak. Branch'lerin var olma sebebi budur. Bu ders, küçük bir Spring Boot projesinde gerçekçi bir feature branch workflow'u kullanarak branch oluşturmayı, aralarında geçiş yapmayı ve merge etmeyi ele alıyor.

Branch Nedir?

Bir **branch**, basitçe bir commit'e işaret eden hareketli bir pointer'dır. Commit attığında, o an üzerinde olduğun branch ileri hareket eder ve yeni commit'e işaret eder. `main` (bazı eski projelerde hâlâ `master` olarak adlandırılır) diğerleri gibi yalnızca bir branch'tir — teknik olarak özel bir statüsü yoktur, yalnızca "birincil geliştirme hattı" olarak konvansiyonel bir statüsü vardır.

Bir branch yalnızca hafif bir pointer olduğu için (tüm projenin bir kopyası değil), bir tane oluşturmak anlık ve ucuzdur — küçük bir değişiklik için bile branch açmadan önce tereddüt etmenin bir sebebi yoktur.

Branch Oluşturmak — git branch

```
git branch add-password-reset       # yeni bir branch oluştur, ama mevcut olanda kal
git branch                          # tüm yerel branch'leri listele
```

Tek başına `git branch <name>` yalnızca branch'i *oluşturur* — seni üzerine taşımaz. Neredeyse her zaman oluşturmayı ve geçişi tek adımda yapmak isteyeceksin, sıradaki bölümde ele alınıyor.

Branch Değiştirmek — git switch

`git switch` seni farklı bir branch'e taşır, working tree'ni buna uyacak şekilde günceller:

```
git switch main
git switch add-password-reset
```

Branch değiştirmek, working tree'nde gördüğün dosyaları değiştirir — bu, Git'in proje klasörünü o branch'in son commit'ine uyacak şekilde gerçekten yeniden yazmasıdır, bir tür görünüm ya da filtre değil.

> Eğer güvenle birleştiremeyeceği commit edilmemiş değişikliklerin üzerine yazılmasına neden olacaksa, Git branch değiştirmeyi reddeder. İşin bitmediyse, geçiş yapmadan önce işini commit et ya da stash'le (bu kursta daha sonra ele alınacak).

Oluşturmak ve Geçmek — git switch -c

Birleşik, gündelik versiyon — yeni bir branch oluştur ve tek komutla üzerine geç:

```
git switch -c add-password-reset
```

Gerçekte onlarca kez yazacağın şey budur; tek başına `git branch`'i daha nadir, "yalnızca branch'leri listele ya da yönet" komutu olarak düşün.

Branch'leri Listelemek

```
git branch          # yerel branch'ler, mevcut olan * ile işaretli
git branch -v       # yerel branch'ler, en son commit'leriyle birlikte
git branch -a       # yerel VE remote-tracking branch'ler
```

Branch'leri Yeniden Adlandırmak

```
git branch -m old-name new-name     # bir branch'i yeniden adlandır (başka bir branch'ten)
git branch -m new-name              # o an üzerinde olduğun branch'i yeniden adlandır
```

Yerel Branch'leri Silmek

Bir branch'in işi merge edildikten sonra, branch listeni yönetilebilir tutmak için sil:

```
git branch -d add-password-reset    # güvenli silme: tamamen merge edilmediyse reddeder
git branch -D add-password-reset    # zorla silme: merge edilmemiş olsa bile siler
```

Küçük harfli `-d`'yi tercih et — bu, yalnızca o branch'te var olan commit'leri yanlışlıkla kaybetmeni önleyen yerleşik bir güvenlik kontrolüdür.

Branch'leri Merge Etmek — git merge

`git merge`, bir branch'teki değişiklikleri başka birine getirir. Bunu değişiklikleri *almak* istediğin branch'ten çalıştırırsın:

```
git switch main
git merge add-password-reset
```

Sıradaki ne olacağı, tamamen branch oluşturulduğundan beri `main`'in hareket edip etmediğine bağlıdır — ki bu bizi iki olası sonuca getiriyor.

Fast-Forward Merge

Eğer `add-password-reset` branch'ten ayrıldığından beri `main` hiç değişmediyse, Git yeni bir şey oluşturmak zorunda kalmaz — yalnızca `main`'in pointer'ını `add-password-reset`'in son commit'ine uyacak şekilde ileri taşır:

```
Önce:  main → A
                \
                 add-password-reset → B → C

Sonra: main → A → B → C
```

Buna **fast-forward** merge denir çünkü Git yalnızca bir pointer'ı "fast-forward" ediyor — yeni bir commit oluşturulmaz, ve geçmiş tamamen doğrusal kalır.

Merge Commit

Eğer bu arada `main` ilerlediyse (başka biri önce başka bir şeyi merge ettiyse), bir fast-forward mümkün değildir — bunun yerine Git, her branch'ten biri olmak üzere iki ebeveynli yeni bir **merge commit** oluşturur:

```
Önce:  main → A → D
                \
                 add-password-reset → B → C

Sonra: main → A → D ------→ M   (M = merge commit, ebeveynler: D ve C)
                \           /
                 B → C ----
```

Bu, birden fazla kişi aynı branch'e merge etmeye başladığında normal, beklenen sonuçtur — bir sorun değil, yalnızca gerçek takımlarda fast-forward'dan daha yaygın olan farklı bir geçmiş şeklidir.

Merge Conflict'leri

Bazen her iki branch de aynı dosyanın *aynı satırlarını* farklı şekillerde değiştirir — Git hangi sürümün doğru olduğuna otomatik olarak karar veremez, bu yüzden durur ve senden bunu elle çözmeni ister. Buna **merge conflict** denir. Conflict'lere bu kursta daha sonra tam bir özel ders ayırıyoruz ("Merge Conflicts") — şimdilik, bunun bir şeylerin yanlış gittiğinin işareti değil, merge etmenin normal, beklenen bir parçası olduğunu bil.

Merge Conflict'lerini Çözmek

Yüksek seviyede: Git, çakışan bölümü doğrudan dosyanın içinde işaretler, sen dosyayı doğru içeriği koruyacak şekilde düzenlersin, sonra merge'i tamamlamak için stage'ler ve commit edersin:

```
# bir conflict'ten sonra, dosyayı düzenleyip çözdükten sonra:
git add ConflictedFile.java
git commit
```

Tam mekanik — conflict işaretlerinin gerçekte neye benzediği, ve tam bir çözüm rehberi — "Merge Conflicts" dersinde derinlemesine ele alınıyor.

Merge Edilmiş Branch'leri Silmek

`add-password-reset`, `main`'e merge edildikten sonra, amacına hizmet etmiştir:

```
git switch main
git branch -d add-password-reset
```

Merge edilmiş branch'leri süresiz olarak tutmak yalnızca `git branch` çıktısını kirletir — silmenin hiçbir maliyeti yoktur, çünkü commit'leri zaten kalıcı olarak `main`'in geçmişinde yaşamaya devam eder.

Gerçekçi Bir Feature Branch Workflow'u

Hepsini bir araya getirirsek, küçük bir Spring Boot projesi için:

```
#!/bin/sh
# A complete feature branch workflow on a small Spring Boot project: branch,
# work, merge back, clean up.

git switch -c add-password-reset
# Output:
# Switched to a new branch 'add-password-reset'

# ... edit PasswordController.java and PasswordService.java ...
git add .
git commit -m "Add password reset endpoint"
# Output:
# [add-password-reset f7e1a92] Add password reset endpoint
#  2 files changed, 31 insertions(+)

git switch main
# Output:
# Switched to branch 'main'

git merge add-password-reset
# Output (fast-forward, since main hasn't moved since branching):
# Updating a1b2c3d..f7e1a92
# Fast-forward
#  PasswordController.java | 18 ++++++++++++++++++
#  PasswordService.java    | 13 +++++++++++++
#  2 files changed, 31 insertions(+)

git branch -d add-password-reset
# Output:
# Deleted branch add-password-reset (was f7e1a92).
```

Bu tam döngü — branch aç, çalış, geri merge et, sil — profesyonel takımların günlük işlerini nasıl organize ettiğinin belkemiğidir, ve bu kurstaki kalan her ders bunun üzerine inşa edilir (aynı temel akışın etrafına GitHub, Pull Request'ler ve kod incelemesi ekleyerek).

git checkout Hakkında Bir Not

`git switch` var olmadan önce (Git 2.23'te tanıtıldı), branch değiştirme `git checkout <branch>` ile yapılırdı. `checkout`'u eski öğreticilerde, Stack Overflow yanıtlarında ve var olan projelerde hâlâ sürekli göreceksin, bu yüzden onu tanıman gerekiyor — ama yeni kod için `switch`'i (branch değiştirmek için) ve `restore`'u (dosya değişikliklerini geri almak için, önceki derste ele alındı) tercih et. Sebebi: `checkout`, bu ilgisiz işlerin İKİSİNİ birden, artı birkaç tane daha yapacak şekilde aşırı yüklenmişti, ki bu da onu yaygın bir hata kaynağı yapıyordu. `switch` ve `restore`, bu sorumlulukları odaklı, yanlış kullanımı daha zor komutlara ayırıyor.

Yaygın Hatalar

- **Önce bir feature branch oluşturmak yerine doğrudan `main` üzerinde commit atmak** — `main`'i her zaman çalışır, deploy edilebilir bir durumda tutmayı çok daha zorlaştırır.
- **Değişiklik yapmadan önce hangi branch'te olduğunu unutmak** — `git status` her zaman ilk satırında mevcut branch'i gösterir; kontrol etme alışkanlığı edin.
- **Merge edilmiş branch'leri süresiz olarak silmeden bırakmak**, `git branch` çıktısını gürültülü ve taranması zor hale getirir.

En İyi Pratikler

- Özellik ya da düzeltme başına bir branch — branch'leri odaklı ve kısa ömürlü tut.
- Bir branch'in `main`'den çok fazla uzaklaşmasını önlemek için sık sık merge et (ya da rebase et, daha sonra ele alınacak).
- Merge ettikten hemen sonra branch'leri sil — hiçbir şey kaybedilmez, çünkü geçmişleri `main`'de yaşamaya devam eder.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Bir branch, bir commit'e işaret eden hafif, hareketli bir pointer'dır — bir tane oluşturmak anlıktır.
- `git switch -c <name>`, yeni bir branch'i tek adımda oluşturur ve üzerine geçer.
- Merge etmek, mümkünse bir fast-forward'dır (pointer hareket eder, yeni commit yok), değilse iki ebeveynli bir merge commit oluşturulur.
- Merge conflict'leri, her iki branch de aynı satırları değiştirdiğinde olur — elle çözülür, sonra `git add` + `git commit` ile tamamlanır.
- `git checkout`, `git switch` / `git restore`'un daha eski, çok amaçlı öncülüdür — tanı, ama yeni komutları tercih et.

Cheat Sheet

```
git switch -c <name>       # yeni bir branch oluştur ve üzerine geç
git switch <name>          # var olan bir branch'e geç
git branch                 # yerel branch'leri listele
git branch -d <name>       # merge edilmiş bir branch'i sil (güvenli)
git merge <branch>         # <branch>'i mevcut branch'e merge et
git branch -m <new-name>   # mevcut branch'i yeniden adlandır
```

Terimler Sözlüğü

- **Branch** — bir commit'e işaret eden hareketli bir pointer; paralel çalışmanın temeli.
- **Fast-forward merge** — yalnızca bir branch pointer'ını ileri taşıyarak merge etmek, yeni commit yok.
- **Merge commit** — bir fast-forward mümkün olmadığında oluşturulan, iki ebeveynli bir commit.
- **Merge conflict** — her iki branch de aynı satırları düzenlediği için Git'in değişiklikleri otomatik birleştiremediği durum.