Rebase & Squash

Rebase nedir, merge'den farkı, git rebase -i ile commit'leri squash/reorder/reword etmek, rebase conflict'lerini çözmek, paylaşılan geçmişi yeniden yazmanın tehlikesi, ve git push --force ile --force-with-lease arasındaki fark.

Orta 35 dk

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

---

Bir branch'in değişikliklerini `main` ile bir araya getirmenin bir yolunu zaten biliyorsun: merge etmek. Bu ders ikinci bir yolu ele alıyor — rebase — ki bu daha temiz, doğrusal bir geçmiş üretir ama paylaşılan iş üzerinde dikkatsizce kullanılırsa gerçek bir risk taşır. Sonunda, rebase'in ne zaman doğru araç olduğunu, ne zaman olmadığını, ve commit'leri temiz, incelenebilir bir birime squash etmekle nasıl bağlantılı olduğunu tam olarak bileceksin.

Rebase Nedir?

Rebase etmek, bir branch'teki commit'leri alır ve onları tek tek, farklı bir başlangıç noktasının — tipik olarak en güncel `main`'in — üzerine yeniden oynatır. Commit'lerin kendileri yeni hash'ler alır (teknik olarak aynı değişikliklere ve mesajlara sahip yepyeni commit'lerdir), ama işinin *içeriği* birebir korunur.

Merge vs Rebase

İkisi de aynı temel sorunu çözer — "`main`'in yeni commit'lerini feature branch'ime getir" — ama farklı geçmiş şekilleri üretir:

```
Merge:                          Rebase:

main    A---D---------M         main    A---D
             \       /                       \
feature       B---C--                 feature  B'---C'   (D'nin üzerine yeniden oynatıldı)
```

- **Merge**, gerçekte ne olduğunu birebir korur, branch'lerin ne zaman bir araya geldiğini gösteren bir merge commit'i dahil — geçmiş, bazen dağınık olsa da gerçek bir kayıttır.
- **Rebase**, branch'inin commit'lerini sanki başından beri en güncel `main`'den başlayarak yazılmış gibi yeniden yazar — geçmiş, gerçekte ne olduğunu artık mükemmel yansıtmama pahasına, temiz, doğrusal bir hikayedir.

Hiçbiri evrensel olarak "doğru" değildir — birçok takım `main` gibi paylaşılan branch'lere merge eder, ama bir Pull Request açmadan önce kendi feature branch'lerini temiz tutmak için rebase eder.

git rebase

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

Bu, `add-password-reset`'e özgü her commit'i, `main`'in mevcut ucunun üzerine yeniden oynatır. Branch'lediğinden beri `main` değişmediyse, yapılacak bir şey yoktur. Değiştiyse, commit'lerinin her biri tek tek yeniden uygulanır.

Bir Feature Branch'i main Üzerine Rebase Etmek

```
#!/bin/sh
# Rebasing a feature branch onto the latest main, then resolving a straightforward
# conflict during the replay.

git switch add-password-reset
git rebase main
# Output:
# Auto-merging PasswordService.java
# CONFLICT (content): Merge conflict in PasswordService.java
# error: could not apply e4f5a6b... Add password validation
# hint: Resolve all conflicts manually, mark them as resolved with
# hint: "git add/rm <conflicted_files>", then run "git rebase --continue".

# ... open PasswordService.java, resolve the conflict markers ...
git add PasswordService.java
git rebase --continue
# Output:
# Successfully rebased and updated refs/heads/add-password-reset.

# The branch's commits now sit on top of main's latest commit, with new hashes.
git log --oneline -3
# Output:
# 9d8e7f6 Add password validation
# 8c7d6e5 Add password field to User entity
# a1b2c3d Add rate limiting to login endpoint   (main's latest, now the base)

# The branch was already pushed once before this rebase, so a normal push is
# rejected -- the rebased commits don't match what's on the remote.
git push
# Output:
# ! [rejected]        add-password-reset -> add-password-reset (non-fast-forward)
# error: failed to push some refs

# force-with-lease pushes the rebased history, but would refuse if origin had
# changed unexpectedly since our last fetch.
git push --force-with-lease
# Output:
# + e4f5a6b...9d8e7f6 add-password-reset -> add-password-reset (forced update)
```

Interactive Rebase — git rebase -i

`git rebase -i` (interactive rebase), branch'inin commit'lerini listeleyen bir editör açar ve *her birine* ne olacağına karar vermene izin verir — yalnızca onları yeniden oynatmakla kalmaz, yeniden sıralayabilir, birleştirebilir, düzenleyebilir ya da düşürebilirsin:

```
git rebase -i HEAD~4    # son 4 commit'i interaktif olarak rebase et
```

Git şuna benzer bir şey açar:

```
pick a1b2c3d Add password field to User entity
pick e4f5a6b Add password validation
pick 7c8d9e0 Fix typo in validation message
pick 1f2a3b4 Add tests for password validation

# Commands:
# p, pick <commit> = use commit
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like squash, but discard this commit's message
# r, reword <commit> = use commit, but edit the commit message
# d, drop <commit> = remove commit
```

Bu gerçekten düzenlediğin bir metin dosyası — her satırdaki kelimeleri değiştir, kaydet, ve Git talimatlarını yerine getirir.

Commit'leri Squash Etmek

Herhangi bir commit'te `pick`'i `squash`'e (ya da kısaltması `s`'ye) değiştirmek, onu *üstündeki* commit'e birleştirir:

```
pick a1b2c3d Add password field to User entity
squash e4f5a6b Add password validation
squash 7c8d9e0 Fix typo in validation message
squash 1f2a3b4 Add tests for password validation
```

Kaydettikten sonra, Git dört commit'in tamamını bire birleştirir ve sonuç için tek, temiz bir commit mesajı yazmana izin verir — dağınık bir work-in-progress geçmişini tek, odaklı bir commit'e dönüştürür. `fixup` aynı birleştirmeyi yapar ama fixup edilen commit'in mesajını sessizce atar, ki bu tam olarak son mesaja hiçbir şey katmayan "fix typo" gibi commit'ler için istediğin şeydir.

Commit'leri Yeniden Sıralamak

Rebase listesi yalnızca bir dosyadaki satırlar olduğu için, commit'leri yeniden sıralamak satırları yeniden sıralamak kadar basittir — Git onları yukarıdan aşağıya, bıraktığın sırayla uygular.

Commit Mesajlarını Düzenlemek

Bir commit'te `pick`'i `reword`'e (ya da `r`'ye) değiştirmek, onu olduğu gibi korur ama mesajını düzenleyebilmen için rebase'i duraklatır — bir yazım hatasını düzeltmek ya da eski bir commit'in mesajını netleştirmek için, gerçek değişikliklerine dokunmadan işine yarar.

Rebase Conflict'lerini Çözmek

Rebase, commit'leri tek tek yeniden oynattığı için, bir conflict *herhangi bir* bireysel commit'te olabilir, yalnızca sonunda bir kez değil. Olduğunda, Git rebase ortasında duraklar:

```
# çakışan dosya(lar)ı çözdükten sonra:
git add ConflictedFile.java
git rebase --continue
```

Belirli bir commit'in conflict'i değmeyeceği kadar zor çıkarsa, ya da rebase etmenin sonuçta doğru hamle olmadığına karar verirsen:

```
git rebase --abort    # tüm rebase'i iptal et, öncesindeki duruma dön
```

Conflict işaretlerini çözmenin tam mekaniği, özel "Merge Conflicts" dersinde ele alınıyor — süreç, conflict bir merge'den mi yoksa bir rebase'den mi geldiğine bakılmaksızın aynıdır.

Ne Zaman Rebase ETMEMELİ

Rebase hakkındaki en önemli tek kural: başkalarının zaten pull ettiği commit'leri asla rebase etme. Rebase, yeniden oynatılan her commit'e yeni bir hash verdiği için, o commit'lerin *eski* sürümlerine zaten sahip olan herkesin geçmişi artık seninkinden sapmıştır — Git bunları "aynı iş, taşınmış" olarak değil, ilgisiz commit'ler olarak görür. Bir sonraki pull'ları en iyi ihtimalle kafa karıştırıcı, en kötü ihtimalle kopya, karışık bir geçmiş üretir.

Güvenli kural: henüz push etmediğin sürece (ya da kimse push ettiğin sürümü pull etmediyse), kendi feature branch'inde özgürce rebase et. `main`'i ya da başkalarının aktif olarak üzerinde çalıştığı herhangi bir branch'i asla rebase etme.

Rebase Neden Force Push Gerektirebilir

Rebase etmek commit'lerinin hash'lerini değiştirdiği için, yerel branch'in ve zaten push edilmiş remote karşılığı artık tamamen farklı (ama benzer görünen) geçmişlere sahiptir. Normal bir `git push` reddedilir — Git bunu doğru bir şekilde "remote'ta benim sahip olmadığım commit'ler var" olarak görür ve bir güvenlik kontrolü olarak reddeder. Rebase edilmiş branch'ini push etmek için, bu kontrolü açıkça override etmen gerekir.

git push --force vs git push --force-with-lease

İkisi de rebase edilmiş geçmişini remote'takinin üzerine push eder — ama önemli bir güvenlik farkıyla:

```
git push --force origin add-password-reset
git push --force-with-lease origin add-password-reset
```

- `--force`, remote branch'in üzerine koşulsuz yazar — son fetch'inden beri başka birinin oraya push etmiş olabileceği commit'ler dahil, onları sessizce atarak.
- `--force-with-lease` önce kontrol eder: eğer remote branch, son fetch'inden beri değiştiyse (başka birinin görmediğin bir şey push ettiği anlamına gelir), reddeder ve işlerini üzerine yazmak yerine güvenle başarısız olur.

Gerçekten bir force push gerektiğinde `--force-with-lease`'i tercih et — sana aynı gücün tamamını gerçek bir güvenlik ağıyla birlikte verir. Bu ayrımı, force-push'un ne zaman uygun olduğu/olmadığıyla birlikte, bu kursta daha sonra özel "Force Push" dersinde daha derinlemesine ele alıyoruz.

Yaygın Hatalar

- **Başkalarının zaten pull ettiği bir branch'i rebase etmek** — bu, rebase'in bir takımda gerçek sorunlara yol açmasının en yaygın tek yoludur.
- **`--force-with-lease`'in beklenmedik bir remote değişikliğini yakalayacağı yerde alışkanlıktan düz `--force`'a başvurmak.**
- **Ne durumda olduğunu anlamadan conflict ortasında paniğe kapılıp iptal etmek** — `git rebase --abort` her zaman güvenlidir ve seni tam olarak başladığın yere geri götürür; bir rebase conflict'i kafa karıştırıcı hale gelirse özgürce kullan.

En İyi Pratikler

- Bir Pull Request açmadan önce geçmişi temiz tutmak için kendi, henüz paylaşılmamış feature branch'lerini rebase et.
- İnceleme istemeden önce "fix typo" tarzı commit'leri ebeveynlerine squash etmek için `git rebase -i` kullan.
- Rebase edilmiş bir branch'i push etmen gerektiğinde her zaman düz `--force` yerine `--force-with-lease`'i varsayılan yap.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Rebase, branch'inin commit'lerini yeni bir taban (genellikle `main`) üzerinde yeniden oynatır, yeni commit hash'leri pahasına doğrusal bir geçmiş üretir.
- `git rebase -i`, paylaşılmadan önce commit'leri squash etmene, yeniden sıralamana, yeniden adlandırmana ya da düşürmene izin verir.
- Başkalarının zaten pull ettiği commit'leri asla rebase etme — bu, onların geçmişini seninkinden ayırır.

- Rebase edilmiş, zaten push edilmiş bir branch bir force push gerektirir; düz `--force` yerine `--force-with-lease`'i tercih et çünkü beklenmedik remote değişikliklerinin üzerine yazmayı reddeder.

Cheat Sheet

```
git rebase main                  # branch'inin commit'lerini main üzerine yeniden oynat
git rebase -i HEAD~4             # son 4 commit'i interaktif olarak düzenle
git rebase --continue            # bir conflict'i çözdükten sonra devam et
git rebase --abort               # iptal et, rebase-öncesi duruma dön
git push --force-with-lease      # rebase edilmiş bir branch'i güvenle push et
```

Terimler Sözlüğü

- **Rebase** — bir branch'in commit'lerini farklı bir taban commit'in üzerinde yeniden oynatmak, yeni hash'ler üretmek.
- **Interactive rebase** — her commit için pick/squash/fixup/reword/drop seçtiğin bir rebase.
- **Force-with-lease** — son fetch'inden beri remote beklenmedik şekilde değiştiyse reddeden bir force push.