Advanced Git & Best Practices

git cherry-pick, git reflog (özellikle kayıp commit'leri kurtarma mekanizması olarak), git blame, alias'lar, iyi branch/commit konvansiyonları, ve merge/rebase/squash/revert'i özetleyen bir karşılaştırma. Bir geliştirme takımı için Git en iyi pratikleri.

Orta 30 dk

Son güncelleme: 27 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/advanced-git-and-best-practices

---

Bu ders, kursu `add` / `commit` / `push`'tan daha az sıklıkla başvuracağın ama gerçekten ihtiyaç duyduğun gün çok önem taşıyan bir avuç gerçekten faydalı araçla tamamlıyor — özellikle `git reflog`, ki bu, bu kurstaki her şeyin işin gittiğini söyleyeceği anda seni kurtarabilir. Ders, önceki her dersi birbirine bağlayan bir dizi takım seviyesi pratikle kapanıyor.

git cherry-pick

`git cherry-pick`, repository'nin herhangi bir yerinden **belirli tek bir commit'i** mevcut branch'ine kopyalar — başka bir branch'ten tam olarak tek bir düzeltmenin seninkiyle ilgili olduğu, tüm branch'i merge etmeden ya da rebase etmeden gereken durumlarda işine yarar:

```
git cherry-pick a1b2c3d
```

Yaygın gerçek bir senaryo: kritik bir bug düzeltmesi `main`'e iner, ve o tam düzeltmeye bir `release` branch'inde de ihtiyacın vardır, release branch'ten ayrıldığından beri `main`'de olan başka her şeyi çekmeden.

git reflog

HEAD her hareket ettiğinde — bir commit, bir checkout, bir reset, bir amend, bir rebase adımı — Git bunu sessizce **reflog**'a kaydeder, HEAD'inin işaret ettiği her yerin yerel, kişisel bir günlüğü:

```
git reflog
```

```
a9f3e21 HEAD@{0}: commit: Add password validation
8a1b2c3 HEAD@{1}: reset: moving to HEAD~1
3c9d1a2 HEAD@{2}: commit: Add password validation (first attempt)
```

Reflog **yalnızca yereldir** — asla push edilmez, asla paylaşılmaz, ve senin kendi clone'una özeldir. Ama bu, Git'te yaptığın hiçbir şeyin gerçekleştiği an gerçekten, kalıcı olarak gitmiş olmadığı anlamına gelir; yalnızca `log` gibi normal komutlarla bulunması zorlaşır.

Kayıp Commit'leri reflog ile Kurtarmak

Bu, bu kurstaki en değerli tek acil durum becerisidir. Diyelim ki `git reset --hard` çalıştırdın ve hemen gerçek bir commit'i attığını fark ettin:

```
#!/bin/sh
# Recovering a commit that git reset --hard accidentally discarded.

git log --oneline -1
# Output:
# a9f3e21 Add password validation

# Meant to undo the LAST commit before this one -- typed one too many resets.
git reset --hard HEAD~2
# Output:
# HEAD is now at 3c9d1a2 Add UserService with findById stub

# "Add password validation" (a9f3e21) is nowhere in `git log` anymore.
git log --oneline -3
# Output:
# 3c9d1a2 Add UserService with findById stub
# ... (a9f3e21 is gone from this view)

# But it's not actually gone -- the reflog remembers every place HEAD has been.
git reflog
# Output:
# 3c9d1a2 HEAD@{0}: reset: moving to HEAD~2
# a9f3e21 HEAD@{1}: commit: Add password validation
# 8a1b2c3 HEAD@{2}: commit: Add UserService with findById method

# HEAD@{1} is exactly the commit we wanted back. Reset onto it directly.
git reset --hard a9f3e21

git log --oneline -1
# Output:
# a9f3e21 Add password validation
```

Bunu vurgula: `git reflog`, yanlışlıkla kaybedilen commit'ler için kurtarma mekanizmandır — kötü bir `reset --hard`, pişman olduğun bir amend, hatta çok erken sildiğin bir branch. Commit makinende gerçekten commit edildiği sürece, hash'i, garbage collection eninde sonunda temizleyene kadar (normalde haftalar sürer) Git'in obje veritabanında hâlâ vardır, ve reflog o hash'i yeniden bulma şeklindir.

git blame

`git blame`, bir dosyanın her satırını en son kimin değiştirdiğini, ve hangi commit'te değiştirdiğini gösterir:

```
git blame PasswordValidator.java
```

```
a1b2c3d4 (Ada Lovelace  2026-08-10 14:22:03 +0300  12) if (password.length() < 8) {
e5f6a7b8 (Grace Hopper  2026-08-15 09:10:41 +0300  13)     throw new
WeakPasswordException();
```

Belirli bir satırın *neden* var olduğunu anlamaya çalışırken paha biçilmezdir — adını verdiği commit'te `git show <hash>` ile eşleştir, o değişikliğin arkasındaki tam bağlamı ve commit mesajını görmek için.

Conventional Commits (Hatırlatma)

Conventional Commits ile "Working With Commits"te zaten tanıştın — mesajları `feat:` / `fix:` / `docs:` / `refactor:` gibi bir tip önekiyle başlatmak. Burada özellikle bir **takım pratiği** olarak tekrarlanmaya değer: bir takımdaki herkes bunu tutarlı bir şekilde takip ettiğinde, `git log` bir bakışta taranabilir hale gelir, ve araçlar changelog'ları otomatik üretebilir ya da sürüm artışlarını yalnızca commit geçmişinden belirleyebilir.

Faydalı Git Alias'ları

Git, sürekli yazdığın komutlar için kısayollar tanımlamana izin verir:

```
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.lg "log --oneline --graph --all"
```

Bundan sonra, `git st`, `git status`'u çalıştırır, ve `git lg` sana kompakt, görsel bir branch grafiği verir. Alias'lar tamamen kişiseldir — projede değil, kendi Git yapılandırmanda yaşarlar, bu yüzden onları takım arkadaşlarına dayatma riski yoktur.

İyi Branch Adlandırma

Tutarlı bir branch adlandırma konvansiyonu, `git branch -a`'yı taranabilir yapar ve genellikle araçlarla entegre olur (bazı CI sistemleri ya da issue tracker'ları branch isimlerini ayrıştırır):

```
feature/add-password-reset
fix/login-null-pointer
chore/upgrade-spring-boot
```

Kesin önekler takıma göre değişir, ama desen — bir tip, ardından kısa, tire ile ayrılmış bir açıklama — neredeyse evrenseldir.

İyi Commit Pratikleri (Hatırlatma)

"Working With Commits"ten: küçük, odaklı commit'ler; emir kipinde özet satırları; *ne*yi değil *neden*i açıklamak. Burada yalnızca bireysel değil, takım seviyesi bir alışkanlık olarak tekrarlanmaya değer — iyi commit mesajları tutarlı bir şekilde yazan bir takım, kendi geçmişini dokümante eden bir kod tabanı üretir, ki bu birinin yıllar önceki bir kararı anlaması gerektiği ilk seferde muazzam bir şekilde işe yarar.

Merge vs Rebase vs Squash vs Revert

Bu kurstaki geçmişi değiştiren her işlemi bir araya getiren tek bir tablo:

- `git merge` — branch'leri birleştirir, tüm commit'leri korur. Paylaşılan geçmişi yeniden yazar mı? Hayır — paylaşılan branch'lerde güvenli.
- `git rebase` — commit'leri yeni bir taban üzerine yeniden oynatır. Paylaşılan geçmişi yeniden yazar mı? Evet — yalnızca paylaşılmamış, yerel commit'lerde.
- **Squash** (`rebase -i` ya da GitHub ile) — birden fazla commit'i bire birleştirir. Paylaşılan geçmişi yeniden yazar mı? Evet — rebase ile aynı kural.
- `git revert` — yeni, tersine bir commit ekleyerek bir commit'i geri alır. Paylaşılan geçmişi yeniden yazar mı? Hayır — paylaşılan/push edilmiş commit'lerde güvenli.

Bir Geliştirme Takımı için Git En İyi Pratikleri

Bu kurstaki her dersi bir araya getirirsek, sorunsuz takım işbirliğini sürekli Git sürtüşmesinden ayıran alışkanlıklar:

- *Neden*i açıklayan mesajlarla küçük, odaklı commit'ler.
- Her şey için feature branch'ler — asla doğrudan `main`'e commit etme.
- Branch protection rule'larıyla zorunlu kılınan, merge etmeden önce gerçek incelemeye sahip Pull Request'ler.
- Geçmişi temiz tutmak için kendi paylaşılmamış branch'lerini rebase et; paylaşılanları asla rebase etme.
- Bir force push gerçekten gerektiğinde, nadir durumda, düz `--force` değil `--force-with-lease`.
- Projenin başından itibaren yapılandırılmış `.gitignore` (build çıktısı, IDE dosyaları, `.env` /secret'lar — asla commit edilmez).

Yaygın Hatalar

- **Düzgün bir merge ya da rebase daha net olacakken `cherry-pick`'e başvurmak** — cherry-pick, gerçekten izole, tek seferlik düzeltmeler içindir, merge etmenin genel bir yerine geçeni değil.

- `reflog`'un var olduğunu bilmemek, ve kötü bir `reset --hard`'ın ya da yanlışlıkla branch silmenin kurtarılamaz olduğunu varsaymak — reflog'un tutma penceresi içinde genellikle öyle değildir.
- **Kişisel alias'ların proje dokümantasyonuna sızması** — alias'ların makinene özel olduğunu hatırla; bir takım arkadaşının senin `git lg`'ine sahip olduğunu varsayan kurulum dokümanları yazma.

En İyi Pratikler

- `git reflog`'u ihtiyacın olmadan önce öğren — gerçekten ihtiyacın olduğu an, dokümantasyonu ilk kez okuyacağın an değildir.
- Commit mesajlarını ve branch isimlerini takım genelinde tutarlı tut — küçük konvansiyonlar, gerçekten daha kolay gezinilir bir geçmişe dönüşür.
- Sorunu çözen en güvenli işlemi varsayılan yap — paylaşılan commit'lerde `reset` yerine `revert`, `--force` yerine `--force-with-lease`, paylaşılan branch'lerde rebase yerine merge.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `git cherry-pick`, belirli tek bir commit'i mevcut branch'ine kopyalar.
- `git reflog`, HEAD'in işaret ettiği her yerin yerel, kişisel bir kaydıdır — bir reset, amend ya da rebase ters gittiğinde kurtarma mekanizmandır.
- `git blame`, her satırı en son kimin değiştirdiğini ve hangi commit'te değiştirdiğini gösterir, kodun neden şu anki hâline geldiğini anlamak için işe yarar.
- Takım seviyesi Git pratikleri — tutarlı commit stili, feature branch'ler, incelenen PR'lar, dikkatli rebase/force-push disiplini — Git işbirliğini pratikte gerçekten sorunsuz yapan şeydir.

Cheat Sheet

```
git cherry-pick <hash>          # belirli tek bir commit'i mevcut branch'e kopyala
git reflog                      # HEAD'in işaret ettiği her yeri gör
git reset --hard <reflog-hash>  # reflog'da bulunan bir commit'e kurtar
git blame <file>                # her satırı en son kimin değiştirdiğini gör
git config --global alias.<name> "<command>"   # kişisel bir kısayol tanımla
```

Terimler Sözlüğü

- **Reflog** — HEAD'in işaret ettiği her yerin yerel, kişisel bir günlüğü; "kayıp" commit'ler için birincil kurtarma mekanizması.
- **Cherry-pick** — başka bir yerdeki tek bir commit'i mevcut branch'e kopyalamak.

- **Blame** — bir dosyanın her satırını en son hangi commit'in (ve yazarın) değiştirdiğini göstermek.