Undoing Changes

git restore ile working tree değişikliklerini geri almak, git reset'in soft/mixed/hard modları arasındaki fark, ve git revert ile paylaşılan commit'leri güvenle geri almak. HEAD, staging area ve working tree arasındaki ilişki, ve git reset --hard'ın veri kaybı riski.

Orta 30 dk

Son güncelleme: 27 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/undoing-changes

---

Hatalar normaldir — yanlış dosyayı stage'lersin, pişman olacağın bir commit mesajı yazarsın, ya da niyetinden önce commit atarsın. Önemli olan, hangi "geri al" komutunun hangi durumu düzelttiğini tam olarak bilmektir, çünkü Git'in birkaç tanesi var ve yanlışını seçmek gerçekten iş kaybına yol açabilir. Bu ders, bunları en güvenliden (yalnızca working tree) en yıkıcıya (`git reset --hard`) doğru, hep zaten bildiğin üç alan cinsinden ele alıyor: working tree, staging area ve commit geçmişi.

Working Tree Değişikliklerini Geri Almak — git restore

Bir dosyayı düzenledin ama henüz stage'lemediysen, ve düzenlemeyi atıp son commit edilmiş sürüme dönmek istiyorsan:

```
git restore UserService.java
```

Bu, commit edilmemiş değişiklikleri **yalnızca working tree'de** atar — staging area'ya ya da commit geçmişine asla dokunmaz. Bu derste en güvenli geri alma komutudur, ama yine de *commit edilmemiş* düzenlemelerin için yıkıcıdır: bir kez atıldığında, o spesifik düzenleme gitmiştir (hiç commit edilmediği için, kurtarılacak bir geçmişi yoktur).

Değişiklikleri Unstage Etmek — git restore --staged

Bir dosyayı `git add` ile stage'ledin ama bir sonraki commit'e dahil etme fikrinden vazgeçtin — düzenlemenin kendisini atmak istemeden:

```
git restore --staged UserService.java
```

Bu, dosyayı staging area'dan geri working tree'ye taşır, unstage edilmiş olarak. Düzenlemelerin tamamen dokunulmamış kalır — yalnızca `git add`'i geri aldın, başka hiçbir şeyi değil. Bu, "yanlış dosyayı stage'ledim" ya da "bunu henüz commit etmeye hazır değilim" için doğru komuttur.

git reset'e Giriş

`git reset`, `restore`'dan daha güçlüdür (ve daha tehlikelidir) — branch'inin HEAD'inin işaret ettiği yeri taşır, ve isteğe bağlı olarak staging area'yı ve working tree'yi de buna uyacak şekilde değiştirebilir. Bir commit referansı alır (genellikle `HEAD~1`, `HEAD~2`, ya da belirli bir hash) ve tam olarak neye ne kadar dokunacağını kontrol eden üç mod:

```
git reset --soft   →  yalnızca HEAD'i taşır
git reset --mixed  →  HEAD'i taşır + staging area'yı sıfırlar   (varsayılan budur)
git reset --hard   →  HEAD'i taşır + staging area'yı sıfırlar + working tree'yi sıfırlar
```

Her mod, kendisinden öncekinden kesinlikle daha yıkıcıdır. Bunları tek tek ele alalım.

git reset --soft

Bu, çok yaygın gerçek bir duruma yanıttır: **"Çok erken commit attım. Değişikliklerimi stage'de tutarken commit'i nasıl geri alırım?"**

```
#!/bin/sh
# "I committed too early. How do I undo the commit while keeping my changes
# staged?" -- git reset --soft HEAD~1.

git commit -m "Add password validation"
# Output:
# [main 3c9d1a2] Add password validation
#  1 file changed, 12 insertions(+)

# Realized the commit was premature -- want to add more before committing again.
git reset --soft HEAD~1

git status
# Output:
# On branch main
# Changes to be committed:
#   (use "git restore --staged <file>..." to unstage)
#         modified:   PasswordValidator.java

# The commit is gone from history, but nothing was lost -- the exact same
# changes are sitting in the staging area, ready to be committed again
# (optionally combined with more work, or a better message).
```

`git reset --soft HEAD~1`, branch'ini bir commit geriye taşır, ama staging area'yı ve working tree'yi tamamen dokunulmamış bırakır — o commit'te olan her şey artık staging area'da oturuyor, yeniden commit edilmeye hazır (belki daha fazla değişiklikle, ya da daha iyi bir mesajla birleştirilerek). Hiçbir şey kaybolmaz; onu yalnızca "un-commit" ettin.

git reset --mixed

Bu, `git reset`'in **varsayılan modu**dur (hiçbir flag olmadan `git reset HEAD~1` çalıştırmak, `git reset --mixed HEAD~1` ile aynıdır). HEAD'i geri taşır *ve* staging area'yı buna uyacak şekilde sıfırlar — ama working tree dosyalarına dokunmaz.

```
git reset HEAD~1   # şuna eşdeğer: git reset --mixed HEAD~1
```

Pratik etkisi: commit geri alınır, ve değişiklikleri artık working tree'nde **unstaged** düzenlemeler olarak oturur — sanki değişiklikleri yapmışsın ama hiç `git add` çalıştırmamışsın gibi. Bu, bir commit'i geri almak *ve* değişiklikleri sıfırdan yeniden gözden geçirmek/yeniden stage'lemek istediğinde (belki tek yerine birkaç daha küçük commit'e bölmek için) doğru seçimdir.

git reset --hard

Bu, üç moddan en yıkıcı olanıdır: HEAD'i geri taşır, staging area'yı sıfırlar, **ve** working tree dosyalarını hedef commit'e uyacak şekilde üzerine yazar.

```
git reset --hard HEAD~1
```

> `git reset --hard`, hiçbir onay istemi olmadan commit edilmemiş işi **kalıcı olarak atar**. Working tree'nde ya da staging area'nda bir commit'in parçası olmayan her şey, bu komut çalıştığı anda gider. Her zaman önce `git status` çalıştır, ve tutmak isteyebileceğin bir şey varsa `git stash`'i (bu kursta daha sonra ele alınacak) düşün.

Önceki Commit'lere Reset Etmek

Üç mod da yalnızca `HEAD~1` ile değil, herhangi bir commit referansıyla çalışır — birden fazla commit'i tek seferde geriye atlayabilirsin:

```
git reset --soft HEAD~3    # son 3 commit'i geri al, her şeyi stage'de tut
git reset --hard a1b2c3d   # belirli bir commit hash'ine kadar her şeyi at
```

Ne kadar geriye reset edersen, o kadar çok commit aynı şekilde "geri alınır" — bu yüzden birden fazla commit'i tek seferde reset ederken hangi modu kullandığın konusunda bilinçli olmakta fayda var.

git revert

`reset`, HEAD'i geriye taşıyarak geçmişi yeniden yazar — ki bu, onu zaten push edilmiş ve paylaşılmış commit'lerde tam olarak tehlikeli yapan şeydir. `git revert`, aynı sorunu ("bu commit'in yaptığını geri al") tamamen farklı bir şekilde çözer: değişiklikleri hedef commit'in tam tersi olan **yepyeni bir commit** oluşturur, var olan tüm geçmişi dokunulmamış bırakır.

```
git revert HEAD           # en son commit'i geri alan yeni bir commit oluştur
git revert a1b2c3d        # belirli, daha eski bir commit'i geri al
```

`revert` yalnızca her zaman yeni bir commit eklediği için — var olanları asla kaldırmadığı ya da yeniden yazmadığı için — zaten push edilmiş ve başkaları tarafından pull edilmiş commit'lerde kullanmak tamamen güvenlidir.

Reset vs Revert

- **Nasıl geri alır** — `git reset`, HEAD'i geriye taşır, geçmişi yeniden yazar; `git revert`, eskisini iptal eden yeni bir commit ekler.
- **Push edilmiş/paylaşılan commit'lerde güvenli mi?** — `git reset`: hayır, paylaşılan geçmişi yeniden yazar. `git revert`: evet, var olan geçmişi asla yeniden yazmaz.
- **Commit sayısına etkisi** — `git reset`'te commit'ler branch'ten kaybolur; `git revert`'te sayı artar (bir "geri alma" commit'i ekler).
- **Tipik kullanım** — `git reset`, yerel, henüz push edilmemiş commit'ler için; `git revert`, herhangi bir commit için, özellikle zaten paylaşılmış olanlar için.

Doğru Geri Alma Komutunu Seçmek

En hafiften en köklüye, hızlı bir karar rehberi:

- Commit edilmemiş bir düzenlemeyi at → `git restore <file>`
- Bir dosyayı unstage et, düzenlemeyi koru → `git restore --staged <file>`
- *Yerel, push edilmemiş* bir commit'i geri al, değişiklikleri stage'de tut → `git reset --soft HEAD~1`
- *Yerel, push edilmemiş* bir commit'i geri al, değişiklikleri yeniden gözden geçir → `git reset --mixed HEAD~1` (ya da yalnızca `git reset HEAD~1`)
- Yerel bir commit'i *ve* değişikliklerini tamamen at → `git reset --hard HEAD~1`
- Zaten **push edilmiş ve paylaşılmış** bir commit'i geri al → `git revert <hash>`

Yaygın Hatalar

- **Alışkanlıktan `git reset --hard`'a başvurmak** halbuki `--soft` ya da `--mixed` işini korurdu. `--hard` çalıştığında, commit edilmemiş değişiklikler gider.

- **Başka birinin zaten pull ettiği bir commit'te `git reset` kullanmak**, sessizce paylaşılan geçmişi yeniden yazmak — o durumda yerine `git revert` kullan.
- **`restore` ile `reset`i karıştırmak** — `restore` yalnızca working tree'ye (ve `--staged` ile staging area'ya) dokunur, `reset` ise ayrıca HEAD'i de taşır.

En İyi Pratikler

- Herhangi bir reset'ten önce, özellikle `--hard`'dan önce, tam olarak neyi kaybetmek üzere olduğunu bilmek için `git status` çalıştır.
- Zaten push edilmiş herhangi bir şey için varsayılan olarak `git revert`'i kullan — paylaşılan geçmiş için güvenli seçim budur.
- "Commit'i geri al ama her şeyi koru" istediğinde özellikle `git reset --soft`'a başvur — erken bir commit'i düzeltmenin en az yıkıcı yoludur.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `git restore`, working tree değişikliklerini geri alır (ve `--staged` ile dosyaları unstage eder) — HEAD'i asla taşımaz.
- `git reset --soft/--mixed/--hard`, HEAD'i artan yıkıcılık seviyeleriyle geriye taşır — soft her şeyi stage'de tutar, mixed unstage eder, hard tamamen atar.
- `git revert`, bir commit'in etkisini yeni bir commit ekleyerek geri alır, geçmişi asla yeniden yazmaz — zaten paylaşılmış commit'ler için tek güvenli geri alma seçeneği.

Cheat Sheet

```
git restore <file>              # commit edilmemiş working tree değişikliklerini at
git restore --staged <file>     # bir dosyayı unstage et, düzenlemeyi koru
git reset --soft HEAD~1         # son commit'i geri al, değişiklikleri stage'de tut
git reset --mixed HEAD~1        # son commit'i geri al, değişiklikleri unstage et
git reset --hard HEAD~1         # son commit'i geri al, her şeyi at (tehlikeli!)
git revert HEAD                 # son commit'i güvenle, yeni bir commit'le geri al
```

Terimler Sözlüğü

- **Reset** — HEAD'i (ve isteğe bağlı olarak staging area'yı/working tree'yi) geriye taşır, geçmişi yeniden yazar.
- **Revert** — yeni, tersine bir commit ekleyerek bir commit'in değişikliklerini geri alır; geçmişi asla yeniden yazmaz.
- **Soft / Mixed / Hard** — artan yıkıcılık sırasına göre üç `git reset` modu.