**🚀 Git Rehberi — Başlangıçtan İleriye**

**1. Git Projesi Oluşturma ve Başlangıç Ayarları 🛠️**

• cd — Klasör değiştirmek için kullanılır (Change Directory).  
• pwd — Bulunduğun dizini gösterir (Print Working Directory).  
• git init — Bulunduğun klasörü Git deposu olarak başlatır.  
• git init -b main — Depoyu başlatırken varsayılan dalı main olarak ayarlar.  
• git config --global user.name "İsim" — Commit işlemleri için kullanıcı adını ayarlar.  
• git config --global user.email "mail" — Commit işlemleri için e-posta ayarlanır.  
• git config --global --unset user.email — Daha önce yapılan global e-posta ayarını kaldırır.  
• git config --global --unset user.name — Daha önce yapılan global kullanıcı adı ayarını kaldırır.

**2. Dosyaları Takip Etmek ve Değişiklik Bildirmek 📋**

• git status — Dosya durumlarını gösterir:  
🔴 Kırmızı = Takip edilmeyen dosyalar  
🟢 Yeşil = Takip edilen ve değişen dosyalar  
• git add dosyaadı — Belirtilen dosyayı staging alanına alır.  
• git add . — Tüm değişiklikleri staging alanına alır.  
• git diff — Çalışma dizinindeki değişiklikleri gösterir.  
• git diff dosyaadı — Belirli bir dosyanın farklarını gösterir.  
• git diff --staged — Staging alanındaki değişiklikleri gösterir.

**3. Commit İşlemleri ✅**

• git commit -m "mesaj" — Değişiklikleri kalıcı olarak kaydeder.  
• git log — Commit geçmişini listeler.

✳️ **Yeni:**  
• git commit --amend — Son yapılan commit mesajını düzenlemek veya yeni değişiklikler eklemek için kullanılır.  
Açılan ekranda:

* Commit mesajını düzenle
* ESC ile çık
* :wq ile kaydet ve çık

**4. Uzak Depo (Remote) ile Çalışmak 🌐**

• git remote add origin <repo-url> — Uzak repoya bağlantı ekler.  
• git push -u origin main — Main dalını uzak repoya gönderir ve bağlar.  
• git pull — Uzak değişiklikleri alır ve yerel kodla birleştirir (fetch + merge).  
• git fetch — Değişiklikleri indirir ama birleştirme yapmaz.  
• git clone <repo-url> — Uzak bir repoyu yerel ortama klonlar.  
• git push origin --delete master — Uzak repodaki master dalını siler.

✳️ **Yeni:**  
• git pull origin main — Ana kodla güncel olup olmadığını kontrol etmek için kullanılır.  
• git merge <dal\_ismi> — Yerel bilgisayarda iki dalı birleştirir.

Not: Merge işleminden önce git pull komutu ile son değişikliklerin alınması önemlidir.  
• git push — Merge sonrası yapılan birleşimi uzak repoya gönderir.

**5. Branch (Dal) Yönetimi 🌿**

• git branch — Yerel dalları listeler.  
• git branch -r — Uzak dalları listeler.  
• git branch -a — Tüm dalları gösterir.  
• git branch yeni\_dal — Yeni dal oluşturur.  
• git switch -c yeni\_dal — Yeni dal oluşturup o dala geçer.  
• git switch mevcut\_dal — Belirtilen dala geçiş yapar.  
• git branch -m eski\_yeni — Dal adını değiştirir.  
• git branch --delete dal — Yerel dalı siler.

✳️ **Yeni:**  
• git merge <dal\_ismi> — Mevcut dal ile başka bir dalı birleştirir.  
• git push origin <dal> — Yeni dalı uzak repoya gönderir.  
• git push origin --delete <dal> — Uzak repodaki ilgili dalı siler.  
• git checkout -b <yeni\_dal> origin/<uzak\_dal> — Uzak dalı takip eden yeni bir yerel dal oluşturur.

**6. Commit ve Değişiklikleri Geri Alma Senaryoları ↩️**

**Senaryo 1: Sadece git add ile staging’e alınmış değişiklikleri geri almak**

• Staging’den çıkar:  
git restore --staged dosya.txt  
• Dosyadaki değişiklikleri iptal et:  
git restore dosya.txt  
• Tüm dosyaları eski haline getir:  
git restore .

**Senaryo 2: Commit yapıldı ama push yapılmadı**

• Commit’i geri al, değişiklikler staging’de kalsın:  
git reset --soft HEAD~1  
• Commit’i geri al, değişiklikler çalışma dizininde kalsın:  
git reset --mixed HEAD~1  
• Commit’i ve tüm değişiklikleri tamamen sil:  
git reset --hard HEAD~1

**Senaryo 3: Commit ve push yapıldı**

• Son değişiklikleri al, kendi commitlerini yeniden sırala:  
git pull --rebase  
• Hatalı commit’i geri al, yeni bir commit ile düzelt:  
git revert HEAD  
• Zorla geri al (dikkatli olunmalı):

git reset --hard HEAD~1

git push --force

✳️ **Yeni:**

**7. Silme / Geri Alma ile İlgili Ek Komutlar & Senaryolar 🧹**

• git reset --soft HEAD~1 → Commit silinir, değişiklikler staging alanında kalır.  
• git reset --mixed HEAD~1 → Commit silinir, değişiklikler çalışma dizinine iner.  
• git reset --hard HEAD~1 → Commit ve tüm değişiklikler tamamen silinir.

**Olası Yeni Senaryolar:**

🔸 **Yanlış dosyaları staging’e ekledin**  
git reset → Tüm staging alanı boşaltılır (çalışma dizininde kalır).

🔸 **Yanlış dosyada değişiklik yaptın ve eski haline dönmek istiyorsun**  
git checkout -- dosya.txt *(Git 2.23 öncesi)*  
veya  
git restore dosya.txt *(Yeni sürüm)*

🔸 **Son commit mesajında yazım hatası var**  
git commit --amend → Mesajı düzelt, :wq ile kaydet.

🔸 **Push ettin ama uzak repoda hatalı bir commit var**

git reset --hard HEAD~1

git push --force

⚠️ Takım çalışmasında kullanılacaksa dikkatli olun, başkalarının commitlerini de etkileyebilir.

✳️ **Yeni: reflog ile eski commit'lere ulaşmak**

• git reflog — HEAD’in geçmişte bulunduğu tüm konumları listeler (commit ID’leri dahil).  
• git reset --hard <commit\_id> — reflog ile bulunan bir eski commit’e geri döner.

⚠️ Bu işlem çalışma dizinindeki değişiklikleri silebilir. Geri alınamazsa veri kaybı yaşanabilir.

✳️ **Yeni:**  
🔸 **Bir dalı silmek istiyorsun ama uzak repoda da varsa**  
• Yerel dalı sil: git branch -d dal\_adi  
• Uzak dalı sil: git push origin --delete dal\_adi

🔸 **Uzak repodaki bir dalı yerelde takip etmek istiyorsun**  
git checkout -b yeni\_dal\_adi origin/uzak\_dal

**8. Ek Komut ve Bilgiler ℹ️**

• git diff master..origin/master — Yerel ve uzak dallar arası farkları gösterir.  
• git push -u origin master — İlk push işleminde upstream bağlantısı kurar.  
• git pull origin main --rebase — Commit geçmişini temiz tutarak günceller.  
• git reset — Staging alanını sıfırlar.  
• git restore — Dosyaları eski haline döndürür.

**9. İpucu ve Öneriler 💡**

• git status komutu ile sık sık kontrol yap.  
• Main branch adı main olarak kullanılması önerilir.  
• git fetch kullanarak değişiklikleri kontrol et, ardından merge/rebase kararını ver.  
• git rebase geçmişi daha temiz tutar ancak dikkatle kullanılmalı.  
• Takım çalışmasında git revert, bireysel çalışmada git reset daha güvenlidir.

✳️ **Yeni:**

**10. git reflog — Geçmiş Adımları Kurtarma Günlüğü 🧭**

git reflog, Git’te yapılan *tüm HEAD hareketlerini* (dal değiştirme, commit, reset, merge, rebase vb.) gösterir. Böylece geri dönüşü zor gibi görünen hatalardan kurtulma imkânı sağlar.

**Ne işe yarar?**

• Yanlışlıkla yaptığın reset --hard, rebase, checkout, commit gibi işlemlerden sonra, HEAD’in önceki konumlarını bulup geri dönmene olanak tanır.

**✅ Kullanımı:**

• Git içindeki tüm hareket geçmişini gösterir:

git reflog

Örnek çıktı:

a3f7e23 HEAD@{0}: reset: moving to HEAD~1

c7e9b81 HEAD@{1}: commit: önemli düzeltme

2f3bdc1 HEAD@{2}: commit (amend): mesaj güncellendi

• Her bir satır bir “geri dönüş noktasıdır”. İstediğin noktaya şu şekilde dönebilirsin:

git checkout HEAD@{1}

veya o commit’i bulunduğun dala taşı:

git reset --hard HEAD@{1}

**💡 Notlar:**

* git log sadece kalıcı commit geçmişini gösterir, ama git reflog geçici, bazen silinen commitleri bile gösterir.
* reflog bilgileri .git/logs/HEAD içinde saklanır ve zamanla otomatik temizlenir.

**🚨 Tehlikeli bir reset sonrası geri dönüş:**

Senaryo: git reset --hard HEAD~1 yaptın ama fark ettin ki yanlış commit silindi.

git reflog

git reset --hard HEAD@{2}

Böylece sildiğin commitleri geri getirebilirsin!