

◆ Örnek 1: Sıfırdan Git Projesi Oluşturma

1 Git Bash'ı aç

Sağ tık → **Git Bash Here**

2 Kullanıcı dizinine git

cd ~

📌 Git projeleri **her zaman kullanıcı dizininde** başlatılır.

3 Proje klasörü oluştur

mkdir git-ogreniyorum

4 Proje klasörüne gir

cd git-ogreniyorum

5 Git başlat

git init

📌 Açıklama:

Bu komut klasörü bir **Git repository** haline getirir.

◆ Örnek 2: Dosya Takibine Alma (add & commit)

6 Dosya oluştur

touch index.html

7 Git durumunu kontrol et

git status

📌 Untracked file → Git henüz dosyayı izlemiyor.

8 Dosyayı staging alanına ekle

git add index.html

9 Commit at

git commit -m "index.html dosyası eklendi"

📌 Commit = değişiklikleri kalıcı kaydetme

◆ Örnek 4: Commit Geçmişini Görüntüleme

git log

Kısa görünüm:

git log --oneline

Branch = Ana kodu bozmadan ayrı bir yolda geliştirme yapma

Branch, ana koddan bağımsız geliştirme yapmayı sağlayan Git mekanizmasıdır.


ADIM ADIM BRANCH KULLANIMI

◆ 1 Mevcut Branch'leri Görüntüle

git branch

 Örnek çıktı:


* main

 * işareti → şu an bulunduğun branch

◆ 2 Yeni Branch Oluştur

 Senaryo: **Login özelliği geliştirilecek**

git branch feature-login


 Bu komut:


- Yeni branch oluşturur
- Ama **o branch'e geçmez**

◆ 3 Yeni Branch'e Geç

git checkout feature-login

 Artık:

main 

feature-login 

◆ KISA YOL (En Çok Kullanılan)

git checkout -b feature-login

✓ Branch oluşturur

✓ Branch'e geçer

◆ 4 Branch'te Geliştirme Yap

touch login.html

git add login.html

git commit -m "Login sayfası eklendi"

✂ Bu commit:

- Sadece feature-login branch'ine aittir
- main branch **etkilenmez**

◆ 5 Branch'ler Arası Farkı Gör

git checkout main

ls

✂ login.html yoktur

git checkout feature-login

ls

✂ login.html vardır

👉 Branch'lerin bağımsızlığı burada netleşir.

◆ 6 Branch'leri Birleştirme (Merge)

✂ Senaryo: Özellik tamamlandı.

Ana branch'e geç

git checkout main

Branch'i merge et

git merge feature-login

✂ Bu komut:

- feature-login branch'indeki commit'leri
- main branch'e ekler

◆ 7 Merge Sonrası Durum

git branch

feature-login hâlâ vardır.

✚ İstersen silebilirsin:

git branch -d feature-login

◆ 8 Branch Silme Neden Önemli?

- Projeyi temiz tutar
- Bitmiş işler tekrar kullanılmaz
- Gerçek projelerde **zorunlu** sayılır

◆ Örnek 6: Dosya Değişikliğini Geri Alma (Checkout / Restore)

✚ Senaryo

Bir dosyada değişiklik yaptın ama **kaydetmek istemiyorsun**.

1 Dosyayı değiştirdin

echo "hatalı kod" >> index.html

2 Durumu kontrol et

git status

3 Değişikliği iptal et

git restore index.html

✚ Sonuç:

Dosya **son commit haline geri döner**.

● git reset vs ● git revert FARKI

🧠 En Kısa Tanım

Komut **Ne Yapar?**

git reset Commit'i **geçmişten siler / geri sarar**

git revert Commit'i **iptal eden yeni bir commit oluşturur**

◆ Örnek 8: Commit'i Geri Alma ama Geçmiş Silmeden (Revert)

📌 Senaryo

Canlıya giden bir commit'i **iptal etmek istiyorsun**.

1 Commit listesini gör

```
git log --oneline
```

2 Commit'i geri al

```
git revert a1b2c3d
```

📌 Sonuç:

- Yeni bir commit oluşur
- Eski commit **iptal edilmiş olur**
- Güvenlidir (takım projelerinde önerilir)

Git'te bir şey kaybolduysa **önce reflog'a bak**

git reset --hard SONRASI COMMIT KURTARMA

✅ 1 Reflog'u Gör

```
git reflog
```

📌 Örnek çıktı:

```
a3f5c2d HEAD@{0}: reset: moving to HEAD~1
```

```
9b7d1e4 HEAD@{1}: commit: login sayfası eklendi
```

📌 Aradığın commit:

```
9b7d1e4
```

✅ 2 Commit'e Geri Dön

```
git reset --hard 9b7d1e4
```

🎉 Commit GERİ GELDİ

✅ Yöntem 1: Sadece Git'i Sil (Dosyalar Kalsın)

📌 En güvenli

Sadece .git klasörünü siler → Git geçmişi gider, dosyalar kalır.

Git Bash'te:

```
cd proje-klasoru
```

```
rm -rf .git
```

📌 Sonuç:

- Repo ❌
- Dosyalar ✅
- Git geçmişi ❌

Kontrol:

```
git status
```

➡ fatal: not a git repository

⚠ Yöntem 2: Repo + Dosyaları Tamamen Sil

📌 DİKKAT: Geri dönüş yok

```
cd ..
```

```
rm -rf proje-klasoru
```

🧠 HANGİSİ NE ZAMAN KULLANILIR?

Durum	Komut
Değişikliği iptal	git restore
Yanlış commit	git commit --amend
Güvenli geri alma	git revert
Yarım işi sakla	git stash
Versiyon	git tag

Durum	Komut
Tek commit al	git cherry-pick

GitHub

◆ 1 GitHub Hesabı & Repo Oluşturma

✚ GitHub'da repo açma

1. github.com → **New repository**
2. Repository name: proje-adi
3. Public / Private seç
4. README.md ✓
5. **Create repository**

✚ Artık bir **remote (uzak) repo**'n var.

◆ 2 GitHub Repo'yu Bilgisayara İndirme (clone)

cd ~

git clone https://github.com/kullanici-adi/proje-adi.git

✚ Ne oldu?

- Repo bilgisayarına indi
- Git otomatik kuruldu
- origin adlı remote eklendi

Kontrol:

git remote -v

◆ 3 Dosya Ekleme → Commit → GitHub'a Gönderme

◆ Dosya oluştur

cd proje-adi

touch index.html

◆ **Commit hazırla**

git add index.html

git commit -m "index.html eklendi"

◆ **GitHub'a gönder**

git push origin main

✦ Artık dosya GitHub'da 🎉

◆ **5 Branch ile Çalışma**

✦ Ana kural:

main branch'e direkt kod yazılmaz

◆ **Yeni branch oluştur**

git checkout -b feature-login

◆ **Geliştirme yap**

touch login.html

git add .

git commit -m "Login sayfası eklendi"

◆ **Branch'i GitHub'a gönder**

git push origin feature-login

✦ **EN ÇOK KULLANILAN GITHUB KOMUTLARI**

Komut	Amaç
git clone	Repo indir
git push	GitHub'a gönder
git pull	Güncelle
git fetch	Sadece al

Komut	Amaç
-------	------

<code>git remote -v</code>	Bağlantıları gör
----------------------------	------------------

◆ Var Olan Bir Repo'nun GitHub Bağlantısını Güncelleme

Senaryo

- Lokalinde **git repo zaten var**
- Ama:
 - Repo URL değişti
 - Eski GitHub repo silindi
 - Yeni repo açtın
 - Yanlış GitHub hesabına bağlı

ADIM ADIM ÇÖZÜM

◆ 1 Repo klasörüne gir

```
cd ~/git-ogreniyorum
```

◆ 2 Mevcut remote'ları kontrol et

```
git remote -v
```

 Örnek çıktı:

```
origin https://github.com/eski-kullanici/eski-repo.git (fetch)
```

```
origin https://github.com/eski-kullanici/eski-repo.git (push)
```

YÖNTEM 1 (ÖNERİLEN): Mevcut origin URL'sini Güncelle

◆ 3 Remote URL'yi değiştir

```
git remote set-url origin https://github.com/yeni-kullanici/yeni-repo.git
```

◆ 4 Kontrol et

```
git remote -v
```

 Artık yeni repo görünmeli.

◆ 5 GitHub'a gönder

```
git push -u origin main
```

 -u:

- main ↔ origin/main eşleştirir
 - Sonraki push'larda sadece git push yeter
-