# Table of Contents

# Git Temelleri

## Temel İş Akışı

| Kod | Açıklama |
|---|---|
| `git status` | *Projenin o anki durumunu, değişen veya yeni eklenen dosyaları gösterir.* |
| `git add <DOSYA_ADI>` | *Belirli bir dosyadaki modifikasyonları hazırlık alanına[1] ekler.* |
| `git add .` **veya** `git add –all` | *Klasördeki tüm değişiklikleri hazırlık alanına ekler..* |
| `git commit -m "COMMIT_MESAJI"` | *Hazırlık alanındaki dosyaları kalıcı olarak proje tarihçesine kaydeder.* |
| `git log` | *Geçmiş commit'leri (tarihçeyi) listeler. Çıkmak için q tuşuna basılır..* |

## Uzak Repolara İşlemler

| Kod | Açıklama |
|---|---|
| `git remote add origin <url>` | *Yerel deponuzu GitHub'daki bir depoya bağlar..* |
| `git push -u origin main` | *Kodlarınızı GitHub'a gönderir (İlk seferde `-u` kullanılır, sonra sadece `git push` yeterlidir).* |
| `git pull` | *GitHub'daki güncel kodları bilgisayarınıza indirir ve mevcut kodunuzla birleştirir.* |
| `git fetch` | *Uzak depodaki değişiklikleri bilgisayara indirir ama birleştirmez (sadece ne değişmiş diye bakmak için güvenlidir).* |

## Dallanma

> Ana projenizi (genelde main veya master dalı) bozmadan yeni özellikler denemek veya hataları çözmek için projenin paralel bir kopyasını oluşturmak için kullanılır.

| | |
|---|---|
| `git branch` | *Mevcut dalları listeler. Hangi dalda olduğunu (yanında * olan) gösterir.* |
| `git branch <DAL_ADI>` | *Yeni bir dal oluşturur.* |
| `git checkout <DAL_ADI>` **veya** `git switch <DAL_ADI>` | *Başka bir dala geçiş yapar.* |
| `git checkout -b <DAL_ADI>` | *Hem yeni dal oluşturur hem de o dala anında geçiş yapar.* |
| `git branch -d <DAL_ADI>` | *İşi bitmiş ve birleştirilmiş bir dalı siler.* |

## Dalları Birleştirme

Kendi dalında işini bitirdiğinde ve bunu ana projeye ektarmak istediğinde bu kodları kullanacaksın. İki yöntemi vardır.

## Git Merge

Sen bir dalda çalışırken ana dalda değişiklikler olmuş olabilir. Gerek sen veya bir ekip üyesi değişiklik gerçekleştirmiş olabilir. **Merge** senin dalını ve ana dalını alır, ikisinin son hallerini birleştirip **Merge commit** adında yeni bir kayıt oluşturur.

#### Ne zaman kullanılır?

Takım halinde çalışırken, ana dalları kendi dalına çekerken veya kendi bitmiş özelliğini ana dala atarken.

#### Avantajı?

Tarihçeyi asla silmez veya değiştirmez. Güvenlidir. Kimin ne zaman ne yaptığını tam olarak gösterir.

#### Dezavantajı?

Çok fazla kişi proje üzerinde çalışıyorsa, proje geçmişi(log) “merge commit” çöplüğüne dönebilir ve örümcek ağı gibi karmaşık görünür.

```
git checkout main # Hedef dala geç.
git merge <DAL_ADI> # Birleştir.
```

## Git Rebase

Senin dalının başlangıç noktasını alır, main dalının en son haline taşır. Yani “Ben projeye dün başlamıştım ama sanki bugün, herkesin son kodunun üzerine başlamışım gibi tarihçeyi yeniden yaz” der.

#### Ne zaman kullanılır?

Kendi yerel (henüz push edilmemiş) dalını, ana projenin güncel haliyle senkronize etmek için. “Merge commit” kirliliği yaratmadan temiz bir geçmiş isteniyorsa.

#### Avantajı?

Dümdüz, okuması çok kolay bir proje geçmişi sağlar. Gereksiz merge commit'leri olmaz.

#### Dezavantajı?

Tarihçeyi yeniden yazar. Bu yüzden tehlikeli olabilir.

Başka insanların da kullandığı ortak dallarda (örneğin main dalında) ASLA rebase yapma! Sadece kendi bilgisayarındaki, henüz kimsenin görmediği dallarda yap.

```
git checkout <DAL_ADI> # Kendi dalında ol.
```

```
git rebase main # Temeli güncelle
```

### Git Squash

Kendi dalında çalışırken ufak tefek bir sürü commit atmış olabilirsin (“buton eklendi”, “renk düzeltildi”, “yazım hatası”, “çalışmıyor deneme 1”). Bu gereksiz kalabalığı ana projeye atmadan önce, hepsini tek bir anlamlı commit (“Login Ekranı Tamamlandı”) haline getirme işlemidir.

#### Ne zaman kullanılır?

Özellik geliştirme bittiğinde, PR (Pull Request) açmadan hemen önce geçmişi temizlemek için.

1. Kaç commit geriye gideceğini belirle (örneğin son 3 commit): `git rebase -i HEAD~3`
2. Karşına bir metin editörü açılır. En üstteki commit pick olarak kalır, altındakilerin başındaki pick yazısını silip `squash` (veya sadece `s`) yazıp kaydedersin.
3. Git sana yeni bir birleştirilmiş mesaj girmen için bir ekran daha açar. Mesajı yazar ve kaydedersin. Boom! 3 commit tek commit oldu.

Not: GitHub üzerinden “Squash and Merge” butonuna basmak, bu işlemi otomatik yapmanın en kolay ve popüler yoludur

## Acil Durumlarda Kullanılan Komutlar

| Kod | Açıklama |
|---|---|
| **`git stash`** | Çalışıyorsun ama işin bitmedi. Acilen başka bir dala geçmen gerekti. Mevcut değişikliklerini geçici olarak bir “çekmeceye” atar. Çalışma alanın temizlenir. |
| **`git stash pop`** | Çekmeceye attığın yarım kalan kodları geri getirir. |
| **`git reset HEAD~1`** | Yaptığın son commit'i iptal eder ama kodlarını silmez. (Yanlışlıkla commit atarsan kurtarıcıdır). |
| **`git reset –hard HEAD~1`** |  TEHLİKELİ! Son commit'i VE yazdığın kodları tamamen siler, geri getirilemez. |
| **`git revert <COMMIT_ID>`** | Hatalı bir commit'i silmez, ancak o commit'in yaptıklarının tam tersini yapan yeni bir commit atar. Ortak çalışılan public dallarda hata düzeltmenin en güvenli yoludur.. |

UCH Viki'den alınmıştır.

1)
staging area