Git Fundamentals

Git nedir, GitHub'dan nasıl farklıdır, ve working tree/staging area/repository zihinsel modeli -- git init, status, add, commit, diff ve log ile. Git & GitHub kursunun ve Git Fundamentals kategorisinin 1.'si.

Başlangıç 30 dk

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

---

Her geliştirici er ya da geç aynı soruyu sorar: "bu değişikliği batırırsam nasıl geri alırım?" Git, endüstrinin bu soruya verdiği yanıttır — seni dikkatli olman konusunda güvenmek yerine, her dosyanın her sürümünü takip ederek işini asla gerçekten kaybetmemeni sağlar. Bu ders, kursun geri kalanının üzerine kurulacağı zihinsel modeli oluşturuyor: Git tam olarak nedir, GitHub'dan nasıl farklıdır, ve her Git komutunun üzerinde çalıştığı üç aşamalı akış — working tree, staging area, repository.

Git Nedir?

Git, bir **dağıtık (distributed) versiyon kontrol sistemi**dir: bir projenin dosyalarının zaman içindeki anlık görüntülerini (snapshot) kaydeden, böylece neyin değiştiğini, kimin değiştirdiğini görmeni ve istediğin herhangi bir önceki anlık görüntüye dönebilmeni sağlayan bir araç. "Dağıtık" burada anahtar kelime — bir repository'yi clone'layan her geliştirici, yalnızca en son dosyaları değil, **tüm geçmişi** alır. Geçmiş sürümleri görmek için bağımlı olduğun merkezi bir sunucu yoktur; kendi makinende zaten her şey vardır.

Bunun pratikte anlamı şu: tamamen çevrimdışıyken commit atabilir, geçmişe göz atabilir, branch oluşturabilir ve hataları geri alabilirsin. Ağ bağlantısına yalnızca başka birinin kopyasıyla *senkronize* olmak istediğinde ihtiyacın olur (ki GitHub'ın var olma sebebi tam olarak budur — az sonra geleceğiz).

Git vs GitHub

Bu, yeni başlayanlar için en yaygın kafa karışıklığı noktasıdır, o yüzden net olalım:

- **Git**, versiyon kontrol aracının kendisidir — makinende çalışan ve bir projenin geçmişini yöneten bir program.
- **GitHub**, Git repository'lerini internette barındıran (host eden) ve üzerine işbirliği özellikleri ekleyen bir web sitesi (ve şirket)dir: Pull Request'ler, kod incelemesi, issue takibi, CI entegrasyonu.

GitHub'a hiç dokunmadan Git kullanabilirsin — birçok tekil (solo) proje tamamen bir geliştiricinin dizüstü bilgisayarında yaşar. GitHub, repository'nin paylaşılan, her zaman erişilebilir bir kopyasına ihtiyaç duyduğun anda faydalı hale gelir — başka insanların (ya da senin kendi diğer makinelerinin) push/pull yapabileceği bir kopya. Başka şirketler de aynı türden hosting sunar (GitLab, Bitbucket) — hepsi aynı temel Git üzerine kuruludur.

Git'i Kurmak ve Yapılandırmak

Git, platformuna göre diğer geliştirici araçlarıyla aynı şekilde kurulur (bir paket yöneticisi, ya da git-scm.com'dan bir kurulum programı). Kurulduktan sonra, Git'e kim olduğunu söyle — bu kimlik, attığın her commit'e eklenir:

```
git --version
git config --global user.name "Ada Lovelace"
git config --global user.email "ada@example.com"
```

`--global`, bunun makinendeki her repository için geçerli olacağı, yalnızca mevcut olan için değil anlamına gelir. Aynı komutu `--global` olmadan bir projenin içinde çalıştırarak proje bazında override edebilirsin — iş ve kişisel projeler için farklı e-posta kullanıyorsan işine yarar.

> Herhangi bir anda `git config --list` çalıştırarak, her ayarın nereden geldiği dahil, o an geçerli olan her ayarı görebilirsin.

Bir Repository Oluşturmak — git init

Bir **repository** ("repo"), Git'in takip ettiği bir proje klasörüdür. Herhangi bir klasörü tek bir komutla repository'ye dönüştürürsün:

```
#!/bin/sh
# Turning a plain folder into a Git repository, then confirming it worked.

mkdir task-tracker
cd task-tracker
git init

# Output:
# Initialized empty Git repository in /home/ada/task-tracker/.git/

ls -la

# Output (trimmed):
# drwxr-xr-x  3 ada  staff   96 Aug 24 10:00 .
# drwxr-xr-x  4 ada  staff  128 Aug 24 10:00 ..
# drwxr-xr-x  9 ada  staff  288 Aug 24 10:00 .git

# The .git folder is the repository itself -- it did not exist a moment ago,
# and the "task-tracker" folder had no history until git init created it.
```

`git init`, projenin içinde gizli bir `.git` klasörü oluşturur. O klasör repository'nin *kendisidir* — her anlık görüntünün, her commit'in ve tüm geçmişin yaşayacağı yer. `.git`'i silmek, projenin Git geçmişini siler (gerçek dosyalarını değil) — bunu hatırlamakta fayda var, çünkü bir projenin Git geçmişini tamamen sıfırdan başlatmak istediğinde de yapman gereken şey tam olarak budur.

Working Tree, Staging Area ve Repository

Bu kursta öğreneceğin her Git komutu bu üç alandan birinin üzerinde çalışır, ve aralarındaki akışı anlamak bu kursun en önemli zihinsel modelidir:

```
Working Tree              Staging Area                Repository
(gerçek dosyaların)   →   (bir SONRAKİ commit'e   →    (kalıcı, commit
                            girecek olanlar)             edilmiş geçmiş)
              git add                 git commit
```

- **Working Tree** — editöründe gördüğün haliyle, diskteki gerçek dosyalar. Değişiklikleri burada yaparsın.
- **Staging Area** (aynı zamanda "index" olarak da adlandırılır) — bir sonraki commit'in parçası *olması gerektiğine* karar verdiğin değişiklikler için bir bekleme alanı. Staging area'yı `git add` ile inşa edersin.
- **Repository** — kalıcı, commit edilmiş geçmiş. Bir şey commit edildiğinde, her zaman geri dönebileceğin kalıcı bir anlık görüntü olur.

Staging area, Git'i daha basit "değişikliklerimi takip et" araçlarından ayıran şeydir: beş dosyayı değiştirebilir ama yalnızca ikisini staging'e alarak sadece o ikisini commit etmeyi seçebilirsin. Bu, ilgisiz değişiklikleri karıştıran tek bir dev commit yerine, odaklı ve anlamlı commit'ler oluşturmana olanak tanır.

Değişiklikleri Kontrol Etmek — git status

Herhangi bir anda, `git status` her dosyanın bu üç alana göre tam olarak nerede durduğunu söyler:

```
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        UserService.java

nothing added to commit but untracked files present (use "git add" to track)
```

`git status` çıktısını dikkatlice oku — sana, düz bir dille, bir sonra hangi komutu çalıştırman gerektiğini tam olarak söyler. Bu alışkanlık (neredeyse her şeyden önce ve sonra `git status` çalıştırmak), bu kurstaki başka herhangi bir tek pratikten daha fazla kafa karışıklığını önleyecek.

Değişiklikleri Staging'e Almak — git add

`git add`, bir değişikliği working tree'den staging area'ya taşır:

```
git add UserService.java      # belirli bir dosyayı stage'le
git add src/                  # bir klasörün altındaki her şeyi stage'le
git add .                     # mevcut dizindeki her şeyi stage'le
```

Stage'lemek, kaydetmekle aynı şey değildir — dosyan, editörün onu yazdığı anda zaten diske kaydedilmiştir. Stage'lemek yalnızca Git'e "bu, bir sonraki commit'imde istediğim değişikliklerden biri" der.

> `git add .`, mevcut dizindeki *her şeyi* stage'ler — commit etmek istemediğin dosyalar da dahil (geçici dosyalar, yerel yapılandırma, build çıktısı). `git add .`'dan hemen önce `git status` çalıştırıp tam olarak neyi stage'lemek üzere olduğunu görme alışkanlığı edin.

Commit Oluşturmak — git commit

Bir **commit**, o an staging area'da olan her şeyi alır ve repository'de kalıcı bir yeni anlık görüntü olarak kaydeder:

```
#!/bin/sh
# The full Working Tree -> Staging Area -> Repository flow for a small Java
# class, from writing the file to committing it.

# 1. A new file exists on disk, but Git isn't tracking it yet.
echo 'public class UserService {
    public User findById(Long id) {
        return null; // not implemented yet
    }
}' > UserService.java

git status
# Output:
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#         UserService.java

# 2. Move it into the staging area.
git add UserService.java

git status
# Output:
# Changes to be committed:
#   (use "git restore --staged <file>..." to unstage)
#         new file:   UserService.java

# 3. Commit what's staged -- this creates a permanent snapshot.
git commit -m "Add UserService with findById stub"

# Output:
# [main (root-commit) 4f2a1c9] Add UserService with findById stub
#  1 file changed, 5 insertions(+)
#  create mode 100644 UserService.java

git status
# Output:
# nothing to commit, working tree clean
```

Her commit, neyin ve neden değiştiğini açıklayan bir mesaj gerektirir (iyi bir commit mesajını neyin oluşturduğunu bir sonraki derste ele alacağız). `-m` olmadan, Git varsayılan metin editörünü açar, böylece daha uzun bir mesaj yazabilirsin — tek satırdan daha fazla açıklama gerektiren commit'ler için işe yarar.

Değişiklikleri Görüntülemek — git diff

`git diff`, commit etmeden önce tam olarak neyin değiştiğini, satır satır gösterir:

```
git diff             # stage edilmemiş değişiklikler: working tree vs. staging area
git diff --staged    # stage edilmiş değişiklikler: staging area vs. son commit
```

`-` ile başlayan satırlar kaldırılanlar, `+` ile başlayanlar eklenenlerdir. Commit etmeden hemen önce `git diff --staged` çalıştırmak, basit ama son derece etkili bir alışkanlıktır —

kalıcı geçmişin bir parçası olmadan önce (unutulmuş bir debug satırı gibi) yanlışlıkla yapılmış bir değişikliği yakalamak için son şansındır.

Geçmişi Görüntülemek — git log

`git log`, commit geçmişini gösterir — en yeniden en eskiye, kaydedilmiş her anlık görüntü:

```
$ git log
commit 4f2a1c9e8b3d5a6f7e8d9c0b1a2f3e4d5c6b7a8f
Author: Ada Lovelace <ada@example.com>
Date:   Mon Aug 24 10:15:00 2026 +0300

    Add UserService with basic CRUD methods
```

Her commit benzersiz bir tanımlayıcı alır (o uzun onaltılık dizi, **hash** ya da **SHA** olarak adlandırılır) — Git'in o tam anlık görüntüye her yerde referans verme şeklidir bu, ve sonraki derslerde belirli commit'lere nasıl referans vereceğindir (değişiklikleri geri almak, cherry-pick yapmak ve daha fazlası hash ile çalışır).

Yaygın Hatalar

- **Önce hiçbir şeyi stage etmeden `git commit` çalıştırmak.** Git yalnızca staging area'da olanı commit eder — diskte kaydedilmiş bir dosya otomatik olarak dahil edilmez.
- **`.git`'in var olduğunu unutmak.** Bazı yeni başlayanlar yanlışlıkla yanlış klasörün içinde commit atar, ya da `.git`'i sadece bir önbellek sanıp siler. O, tüm proje geçmişidir.
- **`git status` çıktısını okumamak.** Neredeyse her zaman bir sonra ne yapman gerektiğini tam olarak söyler — bunu atlamak, 5 saniyelik bir okumanın önleyebileceği kafa karışıklığına yol açar.

En İyi Pratikler

- Git'i kurduktan hemen sonra `user.name` / `user.email`'i bir kez, global olarak yapılandır — her commit buna bağlıdır.
- `git status`'u cömertçe çalıştır. Ücretsizdir, hızlıdır ve hiçbir şeyi asla değiştirmez — sürekli kontrol etmemek için hiçbir sebep yok.
- Her commit'ten önce, son bir kontrol olarak `git diff --staged`'ı gözden geçir.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Git dağıtık bir versiyon kontrol aracıdır; GitHub Git repository'lerini barındıran ve üzerine işbirliği özellikleri ekleyen bir web sitesidir.
- Her projenin geçmişi, `git init` ile oluşturulan bir `.git` klasöründe yaşar.

- Değişiklikler üç alandan akar: **Working Tree** → (`git add`) → **Staging Area** → (`git commit`) → **Repository**.
- `git status` neyin nerede durduğunu söyler; `git diff` tam satır değişikliklerini gösterir; `git log` commit geçmişini gösterir.

Cheat Sheet

```
git init                      # yeni bir repository oluştur
git config --global user.name "..."
git config --global user.email "..."
git status                    # mevcut durumu gör
git add <file>                # bir değişikliği stage'le
git diff                      # stage edilmemiş değişiklikleri gör
git diff --staged             # stage edilmiş değişiklikleri gör
git commit -m "message"       # stage edilmiş değişiklikleri commit et
git log                       # commit geçmişini görüntüle
```

Terimler Sözlüğü

- **Repository (repo)** — Git tarafından takip edilen bir proje klasörü (`.git` klasörüyle tanımlanır).
- **Working Tree** — diskteki gerçek dosyalar.
- **Staging Area (index)** — bir sonraki commit'in parçası olmak üzere seçilmiş değişiklikler.
- **Commit** — stage edilmiş değişikliklerin kalıcı, adlandırılmış bir anlık görüntüsü.
- **Hash (SHA)** — belirli bir commit'in benzersiz tanımlayıcısı.