WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 2

Geliştiricinin Araçları

Kod editörü ve IDE, terminal ve komutları, derleyici–yorumlayıcı farkı, paket yöneticileri ve Git & GitHub. Özgün diyagramlarla, sıfırdan ve uygulamalı.

VS Code · Terminal · Git & GitHub · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, her geliştiricinin günlük olarak kullandığı temel araçları sıfırdan öğretir. Dört seviye ve yirmi altı bölüm boyunca kod editörü ve IDE'lerden terminale, derleyici–yorumlayıcı farkından paket yöneticilerine ve sürüm kontrolünün kalbi olan Git & GitHub'a kadar tüm araç çantasını kapsar.

Her bölümde konuyu görselleştiren özgün bir diyagram, bir 'acemiye tavsiye' kartı ve gerçek bir alıştırma yer alır. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin ikinci modülüdür; sıradaki modülde (HTML) ilk gerçek kodunu yazmaya başlarsın. Şemalar özgün olarak çizilmiştir; araç arayüzleri sürümle değişebilir, güncel adımlar için aracın resmî sitesini esas al. Bu seri eğitim amaçlıdır.

Web & Yazılım Geliştirme Serisi · Modül 2

İçindekiler

★ Terimler Sözlüğü 61

SEVİYE 1

Çalışma Ortamı

Her geliştiricinin temel aletleri: alet çantasına genel bakış, kod editörü ve IDE, VS Code, terminal ve komutları, dosya yönetimi, derleyici–yorumlayıcı farkı ve paket yöneticileri.

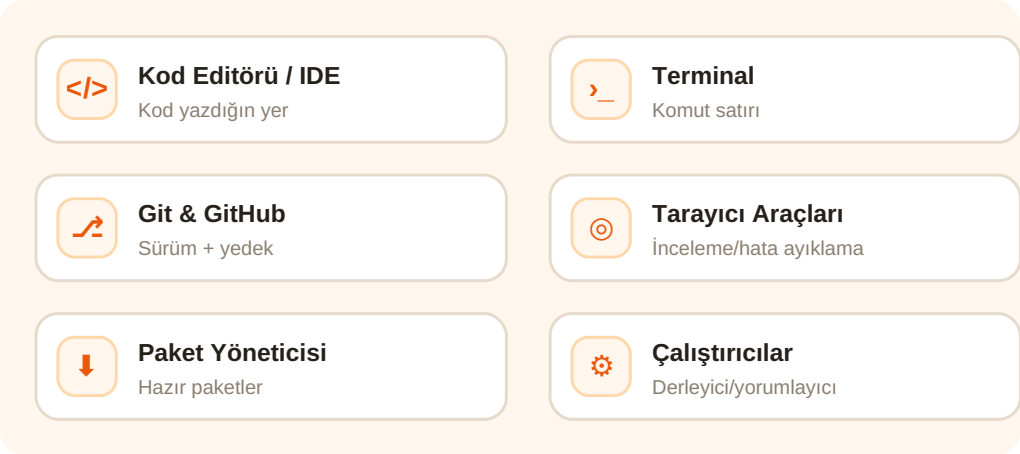
BÖLÜM 01

Geliştiricinin Alet Çantası

Bir marangozun aletleri olduğu gibi, bir geliştiricinin de bir "alet çantası" vardır. Bu araçları erkenden tanımak, öğrenme yolculuğunu çok daha hızlı ve keyifli kılar. Önce büyük resme bakalım.

Temel araçlar

- **Kod editörü / IDE:** kod yazdığın asıl program (örn. VS Code).
- **Terminal:** komutlarla bilgisayarı yönettiğin metin arayüzü.
- **Git & GitHub:** kodunu sürümleme, yedekleme ve sergileme.
- **Tarayıcı geliştirici araçları:** web sayfasını inceleme ve hata ayıklama.
- **Paket yöneticisi:** hazır kod paketlerini kurma (npm, pip, composer).



Şema 1.1 — Geliştiricinin alet çantasına genel bakış.

İPUCU

Hepsini bir anda öğrenmeye çalışma. Önce **bir kod editörü** (VS Code) ve **temel terminal** ile başla; Git/GitHub'ı erken ekle. Diğerleri ihtiyaç doğdukça yerine oturur. Araçlar amaç değil, işini kolaylaştıran yardımcılardır.

Acemiye tavsiye TAVSİYE

Yeni başlıyorsan şu üçü yeterli:

- **VS Code** (kod editörü), **terminal** ve **Git/GitHub**.
- Bu üçü, frontend de backend de mobil de yapsan işine yarar.
- Gerisini (debugger, paket yöneticisi, otomasyon) yolda öğrenirsin.

◎ Alıştırma

8 dk

Çantanı tanı:

1. Yukarıdaki 6 aracı, "ne işe yarar" diye kendi cümlenle yaz.
2. Bilgisayarında hangileri kurulu, kontrol et.
3. İlk kuracağın aracı belirle (öneri: VS Code).

BÖLÜM 02

Kod Editörü ve IDE

Kod, sıradan bir metin dosyasıdır; ama onu doğru araçla yazmak her şeyi değiştirir. Kod editörleri ve IDE'ler; renklendirme, otomatik tamamlama ve hata yakalama ile işini hızlandırır.

Editör mü, IDE mi?

- **Kod editörü:** hafif, hızlı; kod yazmaya odaklı (örn. VS Code). Eklentilerle güçlenir.
- **IDE (Tümleşik Geliştirme Ortamı):** editör + derleyici + hata ayıklayıcı + daha fazlasını bir arada sunan ağır paket (örn. Visual Studio, IntelliJ).
- Sınır bulanıktır: VS Code eklentilerle neredeyse bir IDE gibi çalışır.

Bir editörün bölümleri

- **Kenar çubuğu:** proje dosya ve klasörleri.
- **Sekmeler:** açık dosyalar.
- **Düzenleme alanı:** kodun yazıldığı yer (satır numaralı).
- **Alt panel:** terminal, sorunlar, çıktı.

Şema 2.1 — Tipik bir kod editörü arayüzü: kenar çubuğu (dosyalar) ve düzenleme alanı.

İPUCU

Yeni başlayanlar için **VS Code** en yaygın ve en çok kaynağı olan seçimdir; üstelik ücretsizdir. Bir aracı seçip onda derinleşmek, sürekli araç değiştirmekten daha verimlidir.

Acemiye tavsiye

TAVSİYE

Hangi editörle başlamalı?

- **VS Code** — web, Python, çoğu dil için ideal başlangıç.
- C#/.NET ağırlıklı gideceksen ileride **Visual Studio** (IDE) işine yarar.
- Önce VS Code'da ustalaş; gerisi kolay gelir.

Alıştırma

8 dk

Editörle tanış:

1. VS Code'u aç (kurulu değilse bir sonraki bölümde kuracaksın).
2. Kenar çubuğu, sekme ve düzenleme alanını bul.
3. Editör ile IDE farkını bir cümleyle yaz.

BÖLÜM 03

VS Code'u Kurmak ve Tanımak

En popüler kod editörü VS Code'u kurup temel ayarlarını yapmak, geliştirici yolculuğunun ilk somut adımıdır. Eklentilerle onu kendine göre güçlendirebilirsin.

Kurulum adımları

Şema 3.1 — VS Code kurulum ve ilk ayar akışı.

Faydalı ilk eklentiler

- **Dil desteği:** üzerinde çalışacağın dile göre (HTML/CSS, Python, PHP...).
- **Prettier:** kodu otomatik düzgün biçimler.
- **Live Server:** web sayfanı anında tarayıcıda gösterir.

İPUCU

Eklenti yağmuruna tutulma; başta yalnızca **3-4 temel eklenti** kur. Çok fazla eklenti editörü yavaşlatır ve dikkat dağıtır. İhtiyacını fark ettikçe yenisini eklemek en sağlıklısıdır.

Acemiye tavsiye

TAVSİYE

Acemi kurulumu:

- VS Code + dilinin eklentisi + Prettier + (web yapıyorsan) Live Server.
- Bu kadarı uzun süre yeter; gerisini ihtiyaç doğunca ekle.
- Türkçe arayüz istersen "Turkish Language Pack" eklentisini kur.

Alıştırma

10 dk

Kurulumu yap:

1. VS Code'u resmî siteden indirip kur.
2. Bir dil eklentisi ve Prettier kur.
3. Bir klasör açıp boş bir dosya oluştur ve kaydet.

BÖLÜM 04

Terminal / Komut Satırı Nedir?

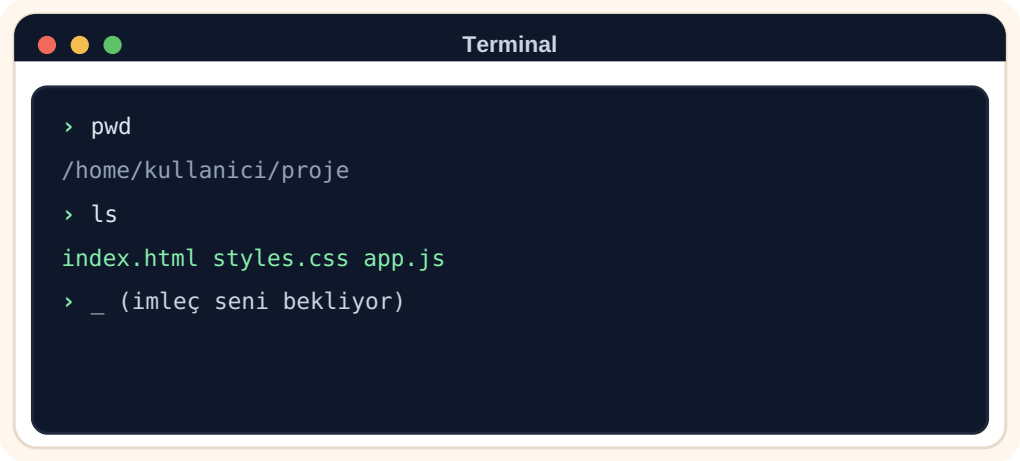
Terminal, bilgisayarla "yazarak" konuştuğun bir arayüzdür. Fareyle tıklamak yerine komut yazarsın. Korkutucu görünür ama birkaç komutla hayatını kolaylaştırır; geliştiricilikte vazgeçilmezdir.

Neden terminal?

- Birçok geliştirici aracı (Git, paket yöneticileri) terminalden çalışır.
- Tek komutla, fareyle dakikalarca süren işleri saniyede yaparsın.
- Uzaktaki sunucuları çoğu zaman **yalnızca** terminalle yönetirsin.

Nasıl görünür?

- Bir **komut istemi** (prompt) seni bekler.
- Bir **komut** yazıp Enter'a basarsın.
- Terminal komutu çalıştırır ve **sonucu/çıktıyı** gösterir.



Şema 4.1 — Terminal: komut yaz, Enter'a bas, çıktıyı gör.

İPUCU

Terminalde yazdığın her komut bir **iş emridir**; dikkatli ol, çünkü bazı komutlar (özellikle silme) geri alınamaz. Emin olmadığın bir komutu çalıştırmadan önce ne yaptığını araştır. Korkma ama saygı duy.

Acemiye tavsiye

TAVSİYE

Acemi için terminal:

- Önce sadece "gezinme" komutlarını öğren: nerede olduğunu gör, klasör değiştir.
- Tehlikeli silme komutlarını acemiyken kullanma.
- VS Code'un içindeki terminali kullanmak en pratiğidir.

Alıştırma

8 dk

Terminali aç:

1. VS Code'da terminali aç (Görünüm → Terminal).
2. Komut istemini (prompt) ve imleci gör.
3. Terminalin ne işe yaradığını bir cümleyle özetle.

BÖLÜM 05

Temel Terminal Komutları

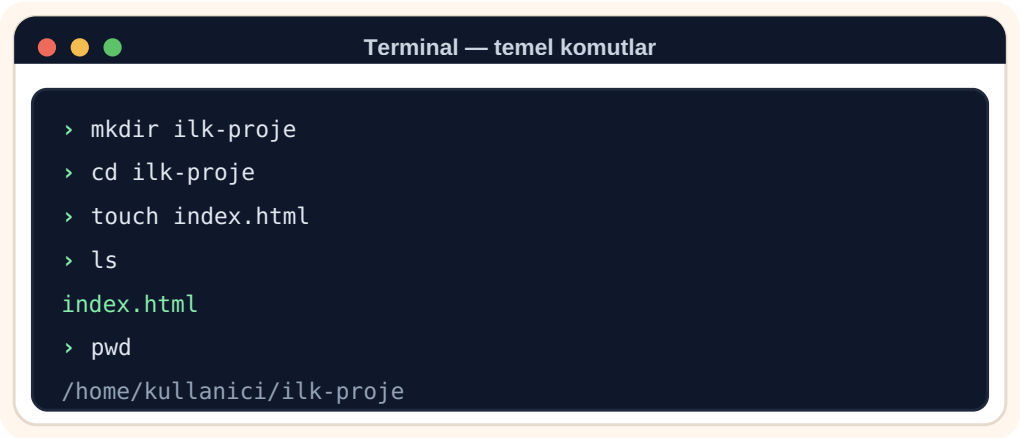
Birkaç temel komut, terminalde rahatça gezinmen için yeterlidir. Bunlar her geliştiricinin ezbere bildiği, günde onlarca kez kullandığı komutlardır.

Gezinme ve listeleme

- `pwd` — şu an hangi klasördeyim? (yolu gösterir)
- `ls` — bu klasörde ne var? (içeriği listeler)
- `cd klasor` — klasöre gir. `cd ..` — bir üst klasöre çık.

Oluşturma

- `mkdir yeni-klasor` — yeni klasör oluştur.
- `touch dosya.txt` — boş dosya oluştur (Linux/Mac).



Şema 5.1 — Klasör oluştur, içine gir, dosya yarat, listele.

İPUCU

Sekme (Tab) tuşu en iyi arkadaşındır: bir dosya/klasör adının başını yazıp Tab'a basınca terminal gerisini tamamlar. Hem hızlandırır hem yazım hatasını önler. Yukarı ok tuşu da önceki komutları geri getirir.

Acemiye tavsiye

TAVSİYE

Önce şu 5 komut yeter:

- `pwd`, `ls`, `cd`, `mkdir`, `touch`.
- Bunlarla dosya sisteminde rahatça gezinir ve oluşturursun.
- Gerisini (kopyalama, taşıma) ihtiyaç oldukça öğren.

Alıştırma

10 dk

Komutları dene:

1. Terminalde bir klasör oluşturup içine gir.
2. İçinde bir dosya oluştur ve `ls` ile listele.
3. `pwd` ile nerede olduğunu doğrula.

BÖLÜM 06

Dosya ve Klasör Yönetimi

Projeler dosyalardan oluşur ve düzenli bir klasör yapısı, işini kolaylaştırır. İyi bir geliştirici, projesinin "nerede ne var" haritasını bilir.

Tipik bir web projesi yapısı

- Ana klasörde `index.html` (giriş sayfası).
- `css/` klasöründe stil dosyaları.
- `js/` klasöründe betikler.
- `images/` klasöründe görseller.

Şema 6.1 — Düzenli bir proje klasör yapısı örneği.

İPUCU

Klasör ve dosya adlarında **boşluk ve Türkçe karakter kullanma**; bunun yerine küçük harf ve tire kullan (örn. `ana-sayfa.html`). Boşluk ve özel karakterler, terminalde ve web'de sorun çıkarır.

Acemiye tavsiye

TAVSİYE

Acemi düzeni:

- Her projeyi kendi klasöründe tut; her şeyi tek klasöre yığma.
- Dosya adları kısa, anlamlı, boşluksuz olsun.
- Bir `README.md` ile projeni kısaca açıkla.

Alıştırma

10 dk

Yapı kur:

1. Bir proje klasörü oluştur.
2. İçine css, js, images klasörleri ve bir index.html ekle.
3. Yapıyı düzgün adlandırma kurallarına uyarak oluştur.

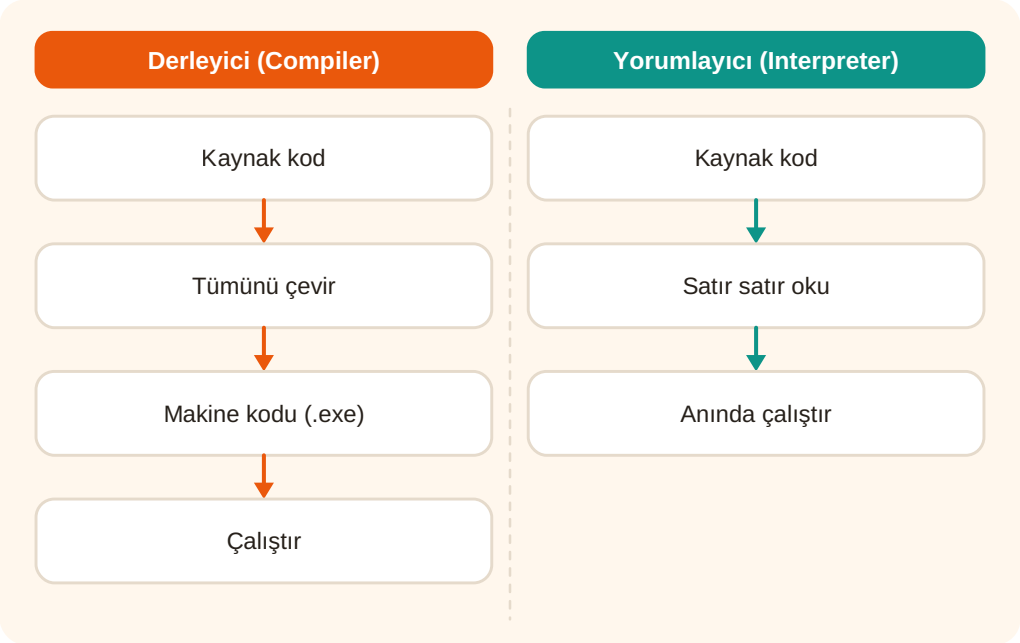
BÖLÜM 07

Derleyici mi, Yorumlayıcı mı?

Bilgisayar yalnızca "makine kodu" anlar; bizim yazdığımız kodun ona çevrilmesi gerekir. Bu çeviri iki yolla olur: derleyici (compiler) ya da yorumlayıcı (interpreter). Bu fark, dilleri anlamanın anahtarıdır.

İki yaklaşım

- **Derleyici:** kodun tamamını önceden makine koduna çevirir; sonra çalıştırılır. (örn. C, C++, C#)
- **Yorumlayıcı:** kodu satır satır okuyup anında çalıştırır. (örn. Python, JavaScript, PHP)
- Bazı diller ikisinin karışımını kullanır.



Şema 7.1 — Derleyici tümünü önceden çevirir; yorumlayıcı satır satır çalıştırır.

Pratikte ne fark eder?

- Derlenen diller genelde **daha hızlı çalışır** ama bir "derleme" adımı gerekir.
- Yorumlanan diller **daha hızlı denenir** (yaz-çalıştır); başlangıç için pratiktir.
- Yeni başlayanların çoğu, yorumlanan bir dille (Python/JavaScript) başlar.

İPUCU

Acemiyken bu farkı derinlemesine bilmen gerekmez; ama "neden bazı dillerde derleme adımı var, bazılarında yok?" sorusunun cevabı budur. Yorumlanan diller hızlı geri bildirim verdiği için öğrenmesi daha keyiflidir.

Acemiye tavsiye

TAVSİYE

Hangisiyle başlamalı?

- Hızlı geri bildirim ve kolay başlangıç için → **yorumlanan** bir dil (Python veya JavaScript).
- Performans ve sistem programlama ilgini çekiyorsa → ileride **derlenen** diller.
- Bu seride: JavaScript ve Python (yorumlanan), C# (derlenen) ayrı modüllerde.

Alıştırma

8 dk

Farkı kavra:

1. Derleyici ve yorumlayıcıyı kendi cümlelerinle anlat.
2. Tanıdığın dilleri (varsa) bu ikisinden birine yerleştir.
3. Neden acemiler genelde yorumlanan dille başlar, yaz.

BÖLÜM 08

Paket Yöneticileri

Hiçbir geliştirici her şeyi sıfırdan yazmaz; başkalarının yazıp paylaştığı hazır "paketleri" kullanır. Paket yöneticileri, bu paketleri tek komutla kurmanı ve yönetmeni sağlar.

Paket yöneticisi nedir?

- Bir **depodan** (kütüphane arşivi) hazır kod paketlerini indirir.
- Tek komutla kurar, günceller, kaldırır.
- Her dilin kendi yöneticisi vardır: **npm** (JavaScript), **pip** (Python), **Composer** (PHP), **NuGet** (C#).

Şema 8.1 — Bir paketin depodan projene yolculuğu.

İPUCU

Paket kurarken dikkatli ol: yalnızca **tanınmış, çok kullanılan** paketleri tercih et. Az bilinen bir paket güvenlik riski taşıyabilir. Ayrıca her paket bir "bağımlılıktır"; ne kadar azsa projen o kadar sağlıklı kalır.

Acemiye tavsiye

TAVSİYE

Acemi için paket yöneticileri:

- Hangi dili öğreniyorsan onun yöneticisini öğren (JS→npm, Python→pip).
- Önce kurulum ve kaldırma komutlarını bil; gerisi sonra gelir.
- Gereksiz paket kurma; her biri bakım yükü demektir.

Alıştırma

8 dk

Paket dünyasını tanı:

1. Öğrendiğin/öğreneceğin dilin paket yöneticisini öğren.
2. O yöneticiyle bir paketin nasıl kurulduğunu araştır.
3. "Bağımlılık" kavramını bir cümleyle açıkla.

SEVİYE 2

Sürüm Kontrolü ve Git

Kodunu güvene almak: sürüm kontrolü nedir, Git'e giriş, temel Git komutları, GitHub, dallar ve birleştirme, iş birliği için pull request ve fork.

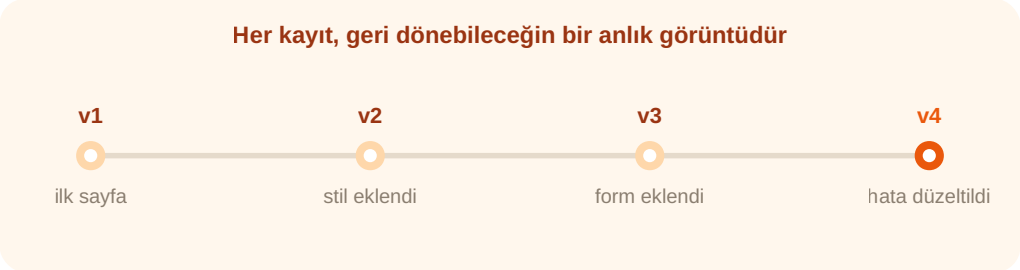
BÖLÜM 09

Sürüm Kontrolü Nedir?

"Az önce çalışıyordu, ne yaptım da bozuldu?" — her geliştiricinin yaşadığı bu kâbusun çözümü sürüm kontrolüdür. Projenin geçmişini kaydeder, istediğin ana geri dönmeni sağlar.

Neden gerekli?

- Projenin her aşamasının bir **anlık görüntüsünü** (kayıt) saklar.
- Bir şeyi bozarsan, çalışan eski hâle **geri dönersin**.
- Kimin neyi ne zaman değiştirdiğini görürsün; **ekip çalışmasının** temelidir.



Şema 9.1 — Sürüm geçmişi: her kayıt, geri dönebileceğin bir anlık görüntü.

"dosya-son-SON-gerçek" dönemi bitti

- Eskiden dosyaları `proje-final-v2-gerçek.zip` diye çoğaltırdık.
- Sürüm kontrolü bunu düzenli, güvenli ve otomatik yapar.
- En yaygın sürüm kontrol sistemi **Git**'tir (sıradaki bölüm).

İPUCU

Sürüm kontrolü yalnızca ekipler için değildir; **tek başına çalışırken bile** hayat kurtarır. "Geri al" tuşunun proje geneli, kalıcı ve güçlü hâli gibi düşün. Bir kez alışınca onsuz çalışamazsın.

Acemiye tavsiye TAVSİYE

Acemi için sürüm kontrolü:

- İlk projenden itibaren kullan; başlamak için "büyük proje" bekleme.
- Önce mantığını anla: kaydet (commit), geri dön, geçmişi gör.
- Araç olarak Git + GitHub öğreneceksin; gerisi bunun üstüne kurulu.

◎ Alıştırma

8 dk

Mantığı kavra:

1. Sürüm kontrolünün ne işe yaradığını kendi cümlelerinle yaz.
2. Daha önce bir dosyayı "v2, v3" diye çoğalttığın bir anı hatırla.
3. Sürüm kontrolü bunu nasıl çözerdi, açıkla.

BÖLÜM 10

Git'e Giriş

Git, dünyanın en yaygın sürüm kontrol sistemidir ve bilgisayarında çalışır. Çalışmanın kalbinde üç alan vardır; bu üçlüyü anlamak Git'i anlamaktır.

Git'in üç alanı

- **Çalışma dizini:** üzerinde düzenleme yaptığın gerçek dosyalar.
- **Hazırlık alanı (staging):** bir sonraki kayda dahil edilecekler.
- **Depo (.git):** kalıcı kayıtların (commit) saklandığı yer.

Şema 10.1 — Bir değişiklik: çalışma dizini → (git add) hazırlık → (git commit) depo.

Bir depo başlatmak

- Bir proje klasöründe `git init` ile depo başlatırsın.
- Artık o klasördeki değişiklikler Git tarafından izlenir.
- Değişiklikleri `git add` ile hazırlar, `git commit` ile kaydedersin.

> **İPUCU**
>
> `git add` ile `git commit`'i karıştırma: **add** "bunları bir sonraki kayda koy" demektir; **commit** ise o kaydı kalıcı olarak oluşturur. Bu iki adımlı yapı, tam olarak neyi kaydedeceğini kontrol etmeni sağlar.

Acemiye tavsiye

TAVSİYE

Acemi için Git başlangıcı:

- Önce üç alanı (çalışma/hazırlık/depo) zihninde oturt.
- Sonra `init → add → commit` döngüsünü ezberle.
- Git'i VS Code'un içindeki arayüzden de kullanabilirsin (kolaylık).

Alıştırma

8 dk

Üç alanı kavra:

1. Çalışma dizini, hazırlık ve depoyu kendi cümlelerinle açıkla.
2. git add ile git commit farkını yaz.
3. Bir değişikliğin üç alandan nasıl geçtiğini anlat.

BÖLÜM 11

Temel Git Komutları

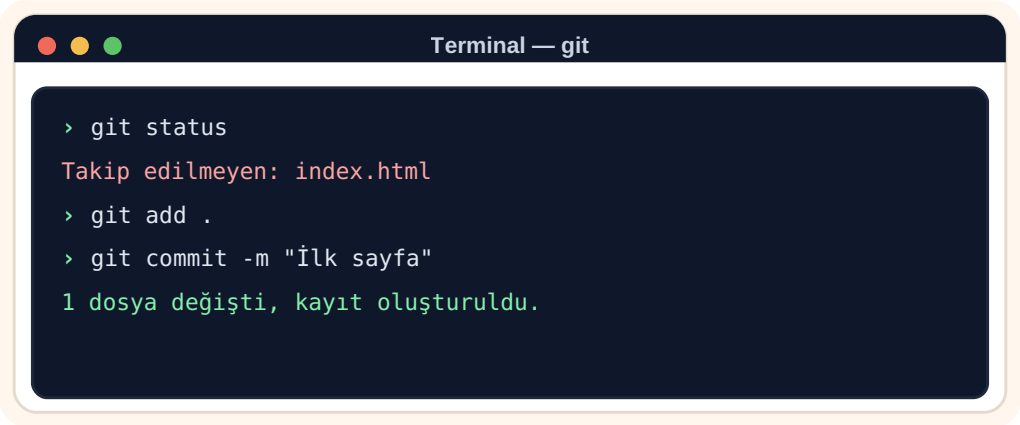
Birkaç komutla Git'in gücünü kullanmaya başlarsın. Bunlar her gün tekrar edeceğin, kısa sürede ezberleyeceğin komutlardır.

Temel döngü

- `git init` — yeni depo başlat.
- `git status` — şu an ne değişti, ne hazır? (en çok kullanacağın komut)
- `git add .` — tüm değişiklikleri hazırlık alanına koy.
- `git commit -m "mesaj"` — bir kayıt oluştur.
- `git log` — geçmiş kayıtları gör.

Tipik bir Git oturumu

```
# yeni depo
git init
# neler değişti?
git status
# değişiklikleri hazırla ve kaydet
git add .
git commit -m "İlk sayfa eklendi"
```



Şema 11.1 — add ve commit ile ilk kaydın oluşması.

İPUCU

Commit mesajların **kısa ve açıklayıcı** olsun: "düzeltme" değil, "giriş formu doğrulaması eklendi". İyi mesajlar, aylar sonra geçmişe baktığında ne yaptığını anlamanı sağlar; bu, gelecekteki sana bir armağandır.

Acemiye tavsiye

TAVSİYE

Acemi için komut seti:

- `status`, `add`, `commit`, `log` — başlangıç için bu kadarı yeter.
- Takıldığında `git status` neredeyse her zaman yol gösterir.
- İleri komutları (rebase, stash) Seviye 4'te öğreneceksin.

Alıştırma

12 dk

İlk deponu oluştur:

1. Bir proje klasöründe `git init` çalıştır.
2. Bir dosya oluşturup `git add` ve `git commit` yap.
3. `git log` ile kaydını gör.

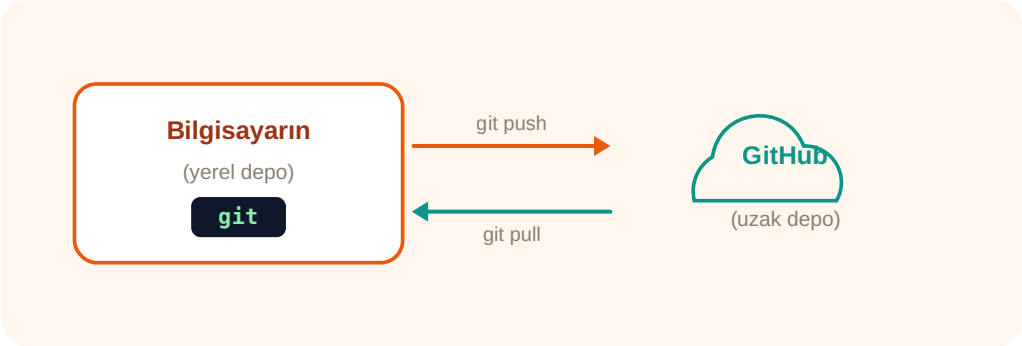
BÖLÜM 12

GitHub'a Giriş

Git bilgisayarında çalışır; GitHub ise depolarını internette saklayan, paylaşan ve ekiple çalıştıran platformdur. Git ile GitHub farklı ama birlikte çalışan iki şeydir.

Git ≠ GitHub

- **Git:** bilgisayarındaki sürüm kontrol aracı.
- **GitHub:** depolarını barındıran çevrimiçi platform (yedek + paylaşım + iş birliği).
- Benzerleri: GitLab, Bitbucket. Mantık aynıdır.



Şema 12.1 — Yerel deponu push ile gönderir, pull ile uzak depodan alırsın.

Temel akış

- GitHub'da bir **depo (repository)** oluşturursun.
- `git push` ile yerel kayıtlarını GitHub'a gönderirsin.
- `git pull` ile GitHub'daki güncellemeleri alırsın.

İPUCU

GitHub aynı zamanda **portfolyondur**: işverenler projelerine buradan bakar. Her küçük projeni GitHub'a koymak, hem güvenli yedek hem de zamanla büyüyen bir vitrin demektir. Düzenli, açıklamalı depolar seni öne çıkarır.

Acemiye tavsiye TAVSİYE

Acemi için GitHub:

- Bir hesap aç ve ilk deponu oluştur.
- `push` ve `pull` mantığını oturt; gerisi sonra.
- Her depoya açıklayıcı bir `README.md` ekle.

◎ Alıştırma

12 dk

İlk deponu yayınla:

1. GitHub'da bir depo oluştur.
2. Yerel projeni `git push` ile GitHub'a gönder.
3. GitHub'da dosyalarının göründüğünü doğrula.

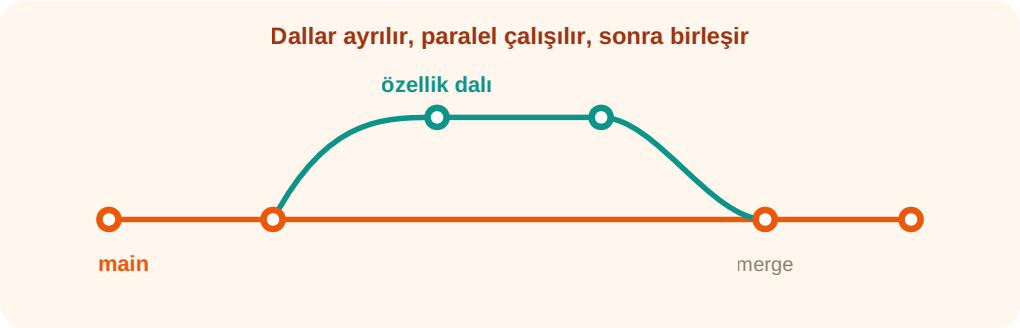
BÖLÜM 13

Dal (Branch) ve Birleştirme

Dallar (branch), ana kodu bozmadan paralel çalışmanı sağlar. Yeni bir özelliği kendi dalında geliştirir, hazır olunca ana dala birleştirirsin. Bu, profesyonel çalışmanın temelidir.

Neden dal?

- **main** (ana dal) her zaman çalışır hâlde kalır.
- Yeni özelliği ayrı bir dalda denersin; bozulursa ana kod etkilenmez.
- Hazır olunca dalı ana dala **birleştirirsin (merge)**.



Şema 13.1 — Özellik dalı ana daldan ayrılır, geliştirilir ve sonra birleştirilir.

Temel komutlar

- `git branch ozellik` — yeni dal oluştur.
- `git switch ozellik` — dala geç.
- `git merge ozellik` — dalı (ana dala) birleştir.

İPUCU

Her yeni özellik/iş için **ayrı bir dal** aç; doğrudan `main` üzerinde çalışma. Böylece ana kodun her zaman güvende kalır ve birden çok işi karıştırmadan yürütürsün. Bu alışkanlık, ekip çalışmasının da olmazsa olmazıdır.

Acemiye tavsiye TAVSİYE

Acemi için dallar:

- Önce mantığı kavra: main güvende, iş ayrı dalda.
- `branch → switch → merge` döngüsünü dene.
- Tek başına çalışırken bile dal kullanmak iyi bir alışkanlıktır.

Alıştırma

10 dk

Dal dene:

1. Bir depoda yeni bir dal oluşturup ona geç.
2. Bir değişiklik yapıp commit et.
3. Dalı ana dala birleştir.

BÖLÜM 14

İş Birliği: Pull Request ve Fork

Birden çok kişi aynı projede çalışırken, değişiklikler kontrollü biçimde birleştirilir. Pull request ve fork, açık kaynak dünyasının ve ekip çalışmasının temel araçlarıdır.

Fork ve Pull Request

- **Fork:** başkasının deposunun kendi kopyanı oluşturmak.
- **Pull Request (PR):** "şu değişiklikleri ana projeye alır mısın?" teklifi.
- Proje sahibi PR'ı inceler, tartışır ve onaylarsa birleştirir.

Şema 14.1 — Bir katkının fork/dal'dan pull request ile ana projeye yolculuğu.

İPUCU

Pull request, sadece kod göndermek değil; bir **iletişim** aracıdır. Ne yaptığını ve neden yaptığını açıklayan iyi bir PR açıklaması yaz. Açık kaynağa küçük katkılar (örn. yazım hatası düzeltmesi) bile portfolyonu güçlendirir.

Acemiye tavsiye

TAVSİYE

Acemi için iş birliği:

- Önce kendi depolarında dal + merge alıştırması yap.
- Sonra küçük bir açık kaynak projeye ufak bir PR göndermeyi dene.
- İyi PR açıklaması yazmak, kodun kadar değerlidir.

Alıştırma

10 dk

İş birliğini keşfet:

1. Fork ve pull request kavramlarını kendi cümlelerinle yaz.
2. Bir açık kaynak projeyi GitHub'da incele.
3. Bir PR'ın hangi aşamalardan geçtiğini sırala.

SEVİYE 3

Üretken Ortam

Daha hızlı çalışmak: editör eklentileri, hata ayıklama araçları, tarayıcı geliştirici araçları, komut satırında verimlilik, ortam yönetimi ve otomasyon.

BÖLÜM 15

Editör Eklentileri ve Verimlilik

Doğru eklentiler ve kısayollar, aynı işi yarı sürede yapmanı sağlar. Editörünü kendine göre güçlendirmek, üretkenliğini görünür biçimde artırır.

İşe yarar eklenti türleri

- **Biçimlendirici (Prettier):** kodu otomatik düzgün hizalar.
- **Linter (ESLint):** hataları sen çalıştırmadan yakalar.
- **Canlı önizleme (Live Server):** web sayfanı anında gösterir.
- **Tema/ikon paketleri:** göz yormayan, okunur bir ortam.

Şema 15.1 — Eklentiler editörü adeta bir IDE'ye dönüştürür.

Kısayollar zaman kazandırır

- Komut paleti (genelde `Ctrl/Cmd + Shift + P`) her şeye erişim verir.
- Çok satırlı düzenleme, hızlı dosya açma, formatla-kaydet alışkanlık edin.

> **İPUCU**
>
> Birkaç temel klavye kısayolu öğrenmek, fareyle uğraşmaktan çok daha hızlıdır. Haftada bir-iki kısayol ekle; birkaç ayda elin klavyeden hiç kalkmaz. **Komut paleti** ile başla — neredeyse her komuta oradan ulaşırsın.

Acemiye tavsiye

TAVSİYE

Acemi için eklentiler:

- Az ama öz: Prettier + dil eklentisi + (web'de) Live Server.
- Tema/ikon paketi göz konforu için iyidir ama gereklilik değil.
- Eklenti çokluğu editörü yavaşlatır; ihtiyaç oldukça ekle.

Alıştırma

8 dk

Ortamını iyileştir:

1. Bir biçimlendirici eklenti kur ve bir dosyayı otomatik biçimle.
2. Komut paletini açıp birkaç komut dene.
3. Öğrendiğin bir kısayolu not et.

BÖLÜM 16

Hata Ayıklama (Debugging) Araçları

Kod ilk seferde nadiren çalışır; bu normaldir. Hata ayıklama, sorunun nerede olduğunu sistemli biçimde bulmaktır. "console.log" ile başlar, debugger ile derinleşir.

İki temel yöntem

- **Çıktı alma:** şüpheli yerlere değerleri yazdır (`console.log` , `print`).
- **Debugger:** kodu bir noktada durdurup (breakpoint) adım adım izleme.

Şema 16.1 — Sistemli hata ayıklama döngüsü.

Hata mesajı dosttur

- Hata mesajı çoğu zaman **dosyayı ve satırı** söyler.
- Mesajı olduğu gibi arama motoruna/yapay zekâya yazmak hızlı çözüm verir.
- Panik yapma; hata, kodun sana "şurada bir sorun var" demesidir.

İPUCU

Hata mesajını "kötü haber" değil, **yol tarifi** olarak gör. İçinde genelde dosya adı, satır numarası ve sorunun türü yazar. Mesajı dikkatle okumak, çoğu hatayı dakikalar içinde çözer; okumadan denemek ise saatler kaybettirir.

Acemiye tavsiye

TAVSİYE

Acemi için hata ayıklama:

- Önce `console.log` / `print` ile değer yazdırmayı öğren.
- Hata mesajını okumayı alışkanlık edin.
- Debugger'ı (breakpoint) sonra, rahatladıkça ekle.

Alıştırma

10 dk

Hata ayıkla:

1. Küçük bir kodda bilerek bir hata oluştur.
2. Hata mesajını oku ve nerede olduğunu bul.
3. Bir `log` ekleyip değeri kontrol et, sonra düzelt.

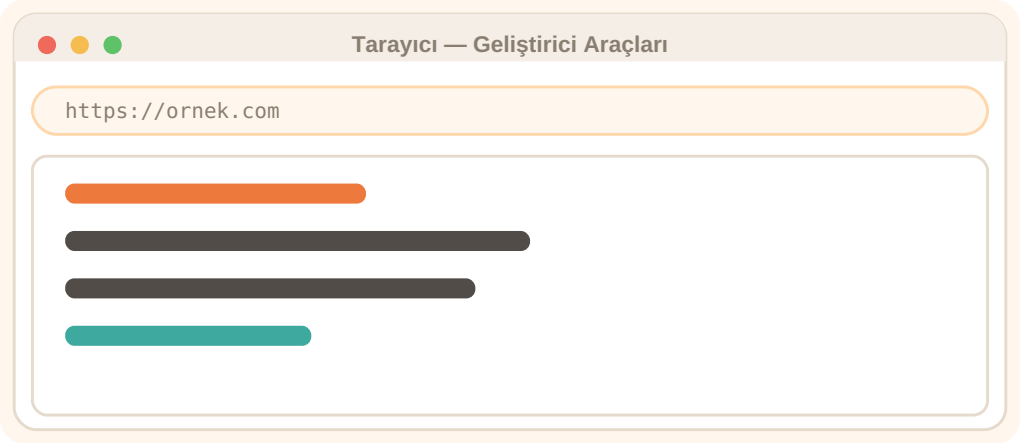
BÖLÜM 17

Tarayıcı Geliştirici Araçları

Her modern tarayıcının içinde güçlü bir geliştirici araç seti vardır (genelde F12 ile açılır). Web geliştirenler için bu araçlar, sayfanın içini görmenin ve düzeltmenin penceresidir.

Başlıca paneller

- **Elements:** sayfanın HTML/CSS yapısını canlı görme ve deneme.
- **Console:** JavaScript çıktıları ve hatalar.
- **Network:** sayfanın yaptığı istekler (yavaşlık avı).
- **Responsive mod:** sayfayı farklı ekran boyutlarında görme.



Şema 17.1 — Tarayıcı geliştirici araçlarıyla sayfanın içini görür ve düzenlersin.

İPUCU

Elements panelinde yaptığın değişiklikler **kalıcı değildir**; sayfayı yenileyince kaybolur. Bu bir özelliktir: gerçek dosyanı bozmadan istediğin kadar deneme yapabilirsin. Beğendiğin değişikliği sonra kod dosyana taşırsın.

Acemiye tavsiye TAVSİYE

Acemi için tarayıcı araçları:

- F12 ile aç; önce Elements ve Console panellerini tanı.
- CSS'i canlı değiştirip sonucu anında görmeyi dene.
- Responsive modla sayfanı telefonda nasıl göründüğünü kontrol et.

◎ Alıştırma

8 dk

Araçları keşfet:

1. Bir web sitesinde geliştirici araçlarını aç (F12).
2. Elements panelinde bir metni geçici olarak değiştir.
3. Console panelini ve responsive modu dene.

BÖLÜM 18

Komut Satırında Verimlilik

Birkaç temel komutu öğrendikten sonra, terminali bir "güç aracı"na dönüştüren küçük hileler vardır. Bunlar zamanla işini ciddi biçimde hızlandırır.

Zaman kazandıran alışkanlıklar

- **Tab** ile otomatik tamamlama; **yukarı ok** ile önceki komutlar.
- Komutları zincirlemek (bir komutun çıktısını diğerine vermek).
- Sık kullandığın komutlara kısaltma (alias) tanımlamak.
- Bir betikle (script) tekrarlı işleri tek komuta indirmek.

Şema 18.1 — Komutları zincirleme ve geçmişi kullanma ile hız.

İPUCU

Aynı komut dizisini sık tekrarlıyorsan, onu bir **betiğe (script)** dönüştür; sonra tek komutla çalıştır. "Kendini tekrar ediyorsan, otomatikleştir" geliştiriciliğin altın kuralıdır. Küçük otomasyonlar zamanla büyük zaman kazandırır.

Acemiye tavsiye TAVSİYE

Acemi için verimlilik:

- Önce Tab ve yukarı-ok alışkanlığını oturt.
- Komut zincirlemeyi (`&&`) öğren.
- Alias ve betikleri rahatladıkça ekle.

Alıştırma

8 dk

Hızlan:

1. Tab ile bir dosya adını otomatik tamamla.
2. İki komutu `&&` ile tek satırda çalıştır.
3. `history` ile önceki komutlarına bak.

BÖLÜM 19

Ortam Yönetimi

Farklı projeler farklı araç sürümleri ve ayarlar gerektirebilir. Ortam yönetimi, projelerin birbirine karışmadan, temiz biçimde çalışmasını sağlar.

Neden ortam yönetimi?

- Bir proje eski, diğeri yeni bir araç sürümü isteyebilir.
- Her projenin **kendi izole ortamı** olursa çakışma olmaz.
- Ayarları (örn. gizli anahtarlar) koddan ayrı, güvenli tutarsın.

Şema 19.1 — Her proje kendi izole ortamında; çakışma olmaz.

Pratik araçlar

- Python'da **sanal ortam** (venv); Node'da sürüm yöneticileri.
- Ayarlar için `.env` dosyaları (gizli bilgiyi koddan ayır).
- Daha ileride **konteyner** (Docker) kavramı.

İPUCU

Gizli bilgileri (şifre, API anahtarı) **asla koda gömme** ve **asla GitHub'a gönderme**. Bunları `.env` gibi ayrı dosyalarda tut ve o dosyayı `.gitignore` ile dışla. Sızan bir anahtar, ciddi güvenlik sorunu yaratır.

Acemiye tavsiye

TAVSİYE

Acemi için ortam:

- Hangi dili kullanıyorsan onun izolasyon yöntemini öğren (Python→venv).
- Gizli bilgileri `.env`'de tut, `.gitignore`'a ekle.
- Docker gibi ileri konuları sonraya bırak.

Alıştırma

8 dk

Ortamı düşün:

1. Neden her projenin ayrı ortamı olmalı, açıkla.
2. Bir `.env` dosyasının ne işe yaradığını yaz.
3. `.gitignore`'un önemini bir cümleyle belirt.

BÖLÜM 20

Otomasyon ve Görev Çalıştırıcılar

Geliştirmede pek çok iş tekrarlanır: kodu biçimle, test et, derle, yayınla. Otomasyon, bu tekrarlı işleri tek komuta ya da otomatik tetiklemeye indirir.

Neyi otomatikleştiririz?

- Kodu biçimleme ve hata kontrolü (linter).
- Testleri çalıştırma.
- Projeyi derleme/paketleme (build).
- Yayına alma (deploy).

Şema 20.1 — Otomatik iş hattı: kod değişince adımlar sırayla çalışır.

İPUCU

Otomasyona **küçük başla**: önce sadece "kaydederken otomatik biçimle" gibi tek bir adımı otomatikleştir. Karmaşık iş hatlarını (CI/CD) sonra, ihtiyaç doğunca kurarsın. Amaç, tekrarlı ve hataya açık işleri makineye devretmektir.

Acemiye tavsiye

TAVSİYE

Acemi için otomasyon:

- Önce "kaydederken biçimle" gibi basit otomasyonlarla başla.
- Test ve derleme komutlarını öğren.
- CI/CD gibi ileri otomasyonu Seviye 4 ve sonraki modüllere bırak.

Alıştırma

8 dk

Otomasyonu kavra:

1. Tekrar tekrar yaptığın bir işi (örn. biçimleme) belirle.
2. Onu nasıl otomatikleştirebileceğini düşün.
3. Otomatik bir iş hattının adımlarını sırala.

SEVİYE 4

Profesyonel Akış

Ustalık: ileri Git (rebase, stash, etiket), çatışma çözme, sürüm ve yayın akışı, kod kalitesi araçları, kişisel ortam kurulumu ve uçtan uca geliştirici akışı.

BÖLÜM 21

Git İleri: Rebase, Stash, Etiketler

Temel Git'i öğrendikten sonra, işini düzenli ve temiz tutan ileri komutlar gelir. Bunlar gücü artırır ama dikkatli kullanılmalıdır.

İleri araçlar

- **stash:** yarım işini geçici bir kenara koyma (sonra geri alırsın).
- **rebase:** geçmişi düzenleyip daha temiz, doğrusal bir tarih oluşturma.
- **tag (etiket):** önemli bir noktayı (örn. `v1.0`) işaretleme.

Şema 21.1 — stash ile yarım işini güvenle bir kenara koyup geri alırsın.

İPUCU

rebase güçlüdür ama geçmişi değiştirdiği için dikkat ister; **başkalarıyla paylaştığın** dallarda rebase yapmaktan kaçın. Acemiyken stash ve tag yeterli; rebase'i rahatladıkça ve mantığını anlayınca kullan.

Acemiye tavsiye TAVSİYE

Acemi için ileri Git:

- Önce `stash` (yarım işi sakla) ve `tag` (sürüm işaretle) öğren.
- `rebase`'i tek başına, deneme deposunda öğren.
- Paylaşılan dallarda geçmişi değiştirmekten kaçın.

◎ Alıştırma

10 dk

İleri Git dene:

1. Yarım bir değişikliği `git stash` ile sakla.
2. Başka bir iş yapıp sonra `git stash pop` ile geri dön.
3. Bir kaydı `git tag v1.0` ile etiketle.

BÖLÜM 22

Çatışma Çözme (Merge Conflicts)

İki kişi (ya da iki dal) aynı satırı farklı değiştirirse, Git "hangisi doğru?" diye sana sorar. Buna çatışma (conflict) denir; korkutucu görünür ama çözmesi öğrenilebilir bir iştir.

Çatışma neden olur?

- Aynı dosyanın **aynı satırı** iki yerde farklı değiştirilmiştir.
- Git hangisini seçeceğini bilemez ve sana bırakır.
- Çatışan bölümü özel işaretlerle gösterir; sen düzeltirsin.

Bir çatışma dosyası böyle görünür

```
<<<<<<< HEAD
"Hoş geldiniz"   // senin değişikliğin
=======
"Merhaba"        // diğer dalın değişikliği
>>>>>>> ozellik
```

Nasıl çözülür?

1. İşaretli bölümü bul (`<<<` , `===` , `>>>`).
2. Hangi hâli kalsın karar ver (veya ikisini birleştir).
3. İşaretleri sil, doğru hâli bırak.
4. `git add` + `git commit` ile çözümü kaydet.

İPUCU

Çatışma bir hata değil, **normal bir durumdur**; özellikle ekip çalışmasında sık olur. VS Code çatışmaları renkli gösterir ve "şunu kabul et / bunu kabul et" düğmeleri sunar; bu, çözümü çok kolaylaştırır. Acele etmeden, hangi hâlin doğru olduğunu düşünerek çöz.

Acemiye tavsiye TAVSİYE

Acemi için çatışma çözme:

- Panik yapma; çatışma işaretlerini tanı ve sakince düzelt.
- VS Code'un görsel çatışma çözücüsünü kullan.
- Çözüm sonrası mutlaka test et ve commit et.

Alıştırma 12 dk

Çatışmayı dene:

1. İki dalda aynı satırı farklı değiştir ve birleştirmeyi dene.
2. Çatışma işaretlerini bul.
3. Doğru hâli bırakıp çözümü commit et.

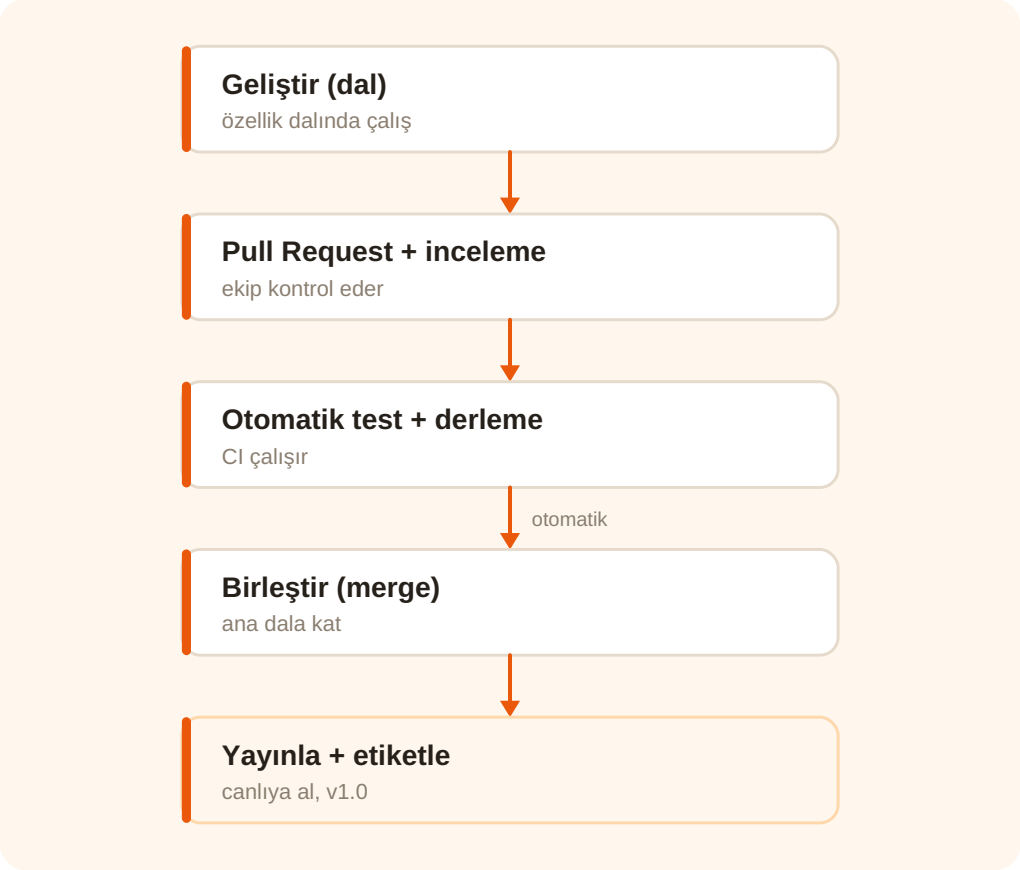
BÖLÜM 23

Sürüm ve Yayın Akışı

Kod yazmak işin yarısı; onu güvenle yayına almak diğer yarısı. Profesyonel ekiplerde kod, belirli adımlardan geçerek (genelde otomatik) canlıya çıkar.

Tipik yayın akışı

- Kod bir dalda geliştirilir ve **pull request** ile incelenir.
- Otomatik **testler** ve **derleme** çalışır.
- Onaylanınca ana dala birleşir ve **yayına** alınır.
- Önemli sürümler **etiketlenir** (v1.0, v1.1…).



Şema 23.1 — Bir değişikliğin geliştirmeden canlıya yolculuğu.

İPUCU

"Cuma akşamı yayın yapma" sektörde bir şakadır ama gerçeği yansıtır: önemli yayınları, sorun çıkarsa müdahale edebileceğin bir zamanda yap. Otomatik testler ve aşamalı yayın, riski büyük ölçüde azaltır.

Acemiye tavsiye

TAVSİYE

Acemi için yayın akışı:

- Önce mantığı kavra: geliştir → incele → test → birleştir → yayınla.
- Kişisel projelerde bu akışı küçük ölçekte uygula.
- Otomatik test/CI'yı ilerleyen modüllerde derinleştireceksin.

Alıştırma

10 dk

Akışı tasarla:

1. Bir özelliğin geliştirmeden yayına yolculuğunu adım adım yaz.
2. Hangi adımlar otomatikleştirilebilir, belirle.
3. Neden doğrudan ana dala yayın yapılmaması gerektiğini açıkla.

BÖLÜM 24

Kod Kalitesi Araçları

Çalışan kod yetmez; okunabilir, tutarlı ve bakımı kolay kod gerekir. Linter ve biçimlendirici gibi araçlar, kod kalitesini otomatik olarak korur.

İki temel araç

- **Biçimlendirici (formatter):** kodun görünümünü (girinti, boşluk) otomatik düzenler — örn. Prettier.
- **Linter:** olası hataları ve kötü alışkanlıkları yakalar — örn. ESLint.
- İkisi birlikte, tutarlı ve temiz bir kod tabanı sağlar.

Şema 24.1 — Linter ve biçimlendirici ile kod otomatik olarak temizlenir.

İPUCU

Kod kalitesi araçlarını **kaydetme anında otomatik** çalışacak şekilde ayarla; böylece hiç uğraşmadan kodun hep temiz kalır. Tutarlı bir kod stili, özellikle ekip çalışmasında "bu kodu kim yazmış?" tartışmalarını ortadan kaldırır.

Acemiye tavsiye

TAVSİYE

Acemi için kod kalitesi:

- Bir biçimlendirici (Prettier) kur ve kaydederken çalıştır.
- Dilinin linter'ını ekle; uyarıları ciddiye al.
- Temiz kod, gelecekteki sana ve ekibe saygıdır.

Alıştırma

8 dk

Kaliteyi artır:

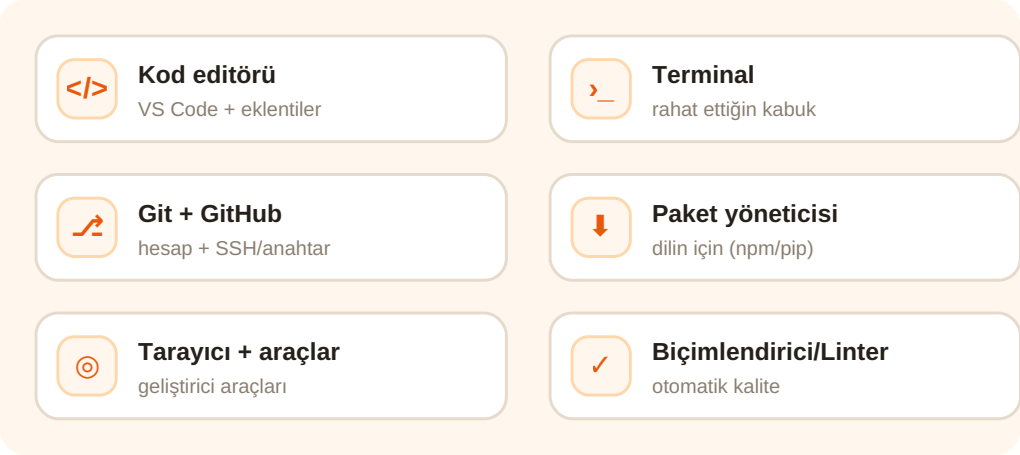
1. Dağınık bir kodu biçimlendiriciyle düzelt.
2. Bir linter'ın verdiği uyarıyı oku ve gider.
3. Kaydederken otomatik biçimlemeyi etkinleştir.

BÖLÜM 25

Kişisel Geliştirme Ortamını Kurmak

Tüm öğrendiklerini bir araya getirip kendine verimli, kalıcı bir çalışma ortamı kurarsın. İyi kurulmuş bir ortam, her gün sana zaman ve huzur kazandırır.

Bir geliştirici kurulumu



Şema 25.1 — Dengeli bir kişisel geliştirme ortamının parçaları.

Kurulum kontrol listesi

- Kod editörü kurulu ve temel eklentiler hazır.
- Git kurulu, GitHub hesabı bağlı.
- Dilinin çalıştırıcısı/paket yöneticisi kurulu.
- Biçimlendirici kaydederken otomatik çalışıyor.

İPUCU

Kurulumunu bir **not** hâline getir (hangi araçlar, hangi ayarlar). Yeni bir bilgisayara geçtiğinde ya da formatladığında, bu not sayesinde ortamını dakikalar içinde yeniden kurarsın. İleride bunu otomatik bir betiğe bile dönüştürebilirsin.

Acemiye tavsiye TAVSİYE

Acemi için kurulum:

- Önce dört temel: editör, terminal, Git/GitHub, dilin çalıştırıcısı.
- Sonra kalite araçları (biçimlendirici/linter) ekle.
- Kurulumunu yaz; ileride çok işine yarayacak.

Alıştırma 12 dk

Ortamını kur:

1. Kontrol listesindeki maddeleri kendi bilgisayarında doğrula.
2. Eksik olanları kur.
3. Kurulumunu kısa bir not hâline getir.

BÖLÜM 26

Uçtan Uca Geliştirici Akışı

Şimdi tüm araçları tek bir akışta birleştiriyorsun: editörde yaz, terminalde çalıştır, Git ile kaydet, GitHub'a gönder, incele ve yayınla. Bu, gerçek bir geliştiricinin günlük döngüsüdür.

Günlük döngü

Şema 26.1 — Editörden yayına: bir geliştiricinin uçtan uca araç akışı.

Akışı sürdürmek

- Küçük ve sık commit'ler yap; her biri anlamlı bir adım olsun.
- Her özelliği ayrı dalda geliştir, PR ile birleştir.
- Kodu temiz tut (biçimlendirici/linter) ve düzenli yedekle (push).
- Takıldığında hata mesajını oku, araştır, sor.

İPUCU

Bu akış zamanla **refleks** hâline gelir; düşünmeden yaparsın. Anahtar, araçları "ezbere komut" olarak değil, **ne işe yaradıklarını anlayarak** kullanmaktır. Mantığı kavradığında, yeni bir araç çıktığında onu da hızlıca öğrenirsin.

Acemiye tavsiye

TAVSİYE

Bundan sonra:

- Bu araçlar her teknik modülde (HTML, CSS, JS, backend...) yanında olacak.
- Yazdığın her kodu Git ile yönet, GitHub'da sakla.
- Sıradaki modül: **HTML** — ilk gerçek kodunu yazmaya başlıyorsun.

Alıştırma

15 dk

Tüm akışı uygula:

1. Küçük bir proje oluştur (editörde bir dosya yaz).
2. Git ile kaydet ve GitHub'a gönder.
3. Bir dalda küçük bir değişiklik yapıp PR ile birleştir.
4. Tüm akışı baştan sona bir kez yaşa.

EK

Terimler Sözlüğü

Bu modülde geçen temel araç ve Git terimleri. Bir başvuru kaynağı olarak saklayabilirsin.

Editör / IDE	`Kod yazılan program (VS Code)`	Terminal	`Komut satırı arayüzü`
Komut istemi (prompt)	`Komut beklenen satır`	Derleyici	`Kodu önceden makine koduna çevirir`
Yorumlayıcı	`Kodu satır satır çalıştırır`	Paket yöneticisi	`npm / pip / Composer / NuGet`
Bağımlılık	`Projenin kullandığı dış paket`	Git	`Sürüm kontrol sistemi`
Depo (repo)	`Projenin sürüm geçmişi`	Commit	`Bir değişiklik kaydı`
Branch (dal)	`Paralel geliştirme hattı`	Push / Pull	`GitHub ile gönder / al`

Minimum kurulum ÖZET

Yeni başlıyorsan üç şey yeter: **VS Code** (kod editörü), **terminal** (VS Code içinden) ve **Git + bir GitHub hesabı**. Bu üçüyle kod yazar, çalıştırır ve projeni güvenle saklayıp sergilersin. Debugger, paket yöneticisi ve otomasyon araçlarını yolda, ihtiyaç doğdukça eklersin.