AI DANIŞMAN

ÜCRETSİZ SÜRÜM

Claude Code 101

Terminalde çalışan ajanlı asistan: kurulum, süreç tarif etme, CLAUDE.md, bağlam yönetimi, izinler, maliyet, kod dışı kullanımlar ve on tipik hata.

AI Danışman — Ücretsiz Eğitim Serisi

Eylül 2026 · aidanisman.pages.dev · t.me/aidanisman

İÇİNDEKİLER

01 Ne olduğu — ve sohbet arayüzünden farkı

Claude Code, kod tabanını okuyan, dosyaları düzenleyen, komut çalıştıran ve geliştirme araçlarınla bütünleşen bir asistan. Terminalde, editörde, masaüstü uygulamasında ve tarayıcıda çalışıyor.

Sohbet arayüzünden farkı tek cümleyle: **sohbet sana metin verir, Claude Code işi yapar.**

	Sohbet arayüzü	Claude Code
Bağlam	Sen ne yapıştırırsan onu görür	Projeyi kendi okur, gerekli dosyayı kendi bulur
Çıktı	Metin — sen kopyalarsın	Dosyayı doğrudan değiştirir
Doğrulama	Sen çalıştırırsın	Komutu kendi çalıştırır, hatayı görür, düzeltir
Döngü	Sen ↔ pencere arası kopyala-yapıştır	Kendi içinde döngü kurar

> ASIL FARK: GERİ BİLDİRİM DÖNGÜSÜ
>
> Sohbet arayüzünde model kodu yazar ama *çalışıp çalışmadığını göremez*. Claude Code testi çalıştırıp hatayı okuyabildiği için kendi hatasını düzeltebiliyor. Kalite farkının kaynağı daha iyi bir model değil, **kapalı bir geri bildirim döngüsü.**

02 Kimin için

Adında "code" geçtiği için yazılımcılara özel sanılıyor. Kısmen doğru ama eksik.

Kim	Ne için
Yazılımcı	Asıl kitle. Özellik geliştirme, hata ayıklama, test yazma, refactor, kod inceleme
Veri ile çalışan	Dosya dönüştürme, toplu düzenleme, rapor üretme, betik yazdırma
Teknik olmayan ama meraklı	Dosya düzenleme, klasör düzeni, tekrar eden bilgisayar işleri, statik site kurma

Üçüncü satır sürpriz olabilir. Claude Code dosya okuyup yazabildiği ve komut çalıştırabildiği için, **kod yazmayan biri de kullanabiliyor** — yeter ki ne istediğini tarif edebilsin ve çıktıyı doğrulayabilsin.

03 Kurulum ve ilk oturum

Kurulum yolları

Ortam	Komut / yol
macOS, Linux, WSL	`curl -fsSL https://claude.ai/install.sh | bash`
Windows PowerShell	`irm https://claude.ai/install.ps1 | iex`
Homebrew	`brew install --cask claude-code`
Windows WinGet	`winget install Anthropic.ClaudeCode`
VS Code / Cursor	Eklenti mağazasından "Claude Code"
JetBrains	Marketplace eklentisi (CLI ayrıca kurulur)
Masaüstü uygulaması	İndir ve kur — CLI'ı ayrıca kurmaya gerek yok
Tarayıcı	claude.ai/code — hiç kurulum yok

> NEREDEN BAŞLAMALI
>
> Hiç denemediysen **tarayıcıdan** başla: kurulum yok, riski yok. Alışınca terminale geç — asıl güç orada. Homebrew ve WinGet kurulumları kendiliğinden güncellenmiyor; yerel kurulum (install.sh) arka planda güncelleniyor.

İlk oturum

Bir proje klasörüne gir ve başlat:

```
TERMİNAL
cd proje-klasorun
claude
```

İlk kullanımda giriş yapman istenir. Sonrası bir sohbet penceresi gibi görünür — ama değil: yazdığın her şey o klasör bağlamında yorumlanır.

İlk komut: soru sor, değişiklik isteme

İlk oturumda hiçbir şey değiştirtme. Önce anlatsın:

```
İLK ÜÇ İSTEK

Bu projede ne var? Klasör yapısını ve her parçanın
ne işe yaradığını özetle.

---

Bu proje nasıl çalıştırılıyor? Bağımlılıklar ne,
hangi komutlar var?

---

Bu kod tabanında en riskli veya en karışık bölüm neresi
ve neden?
```

Üçüncü soru özellikle değerli. Kendi projende bile fark etmediğin bir şey söyleyebilir — ve sana modelin projeyi gerçekten okuyup okumadığını gösterir.

---

04 İlk gerçek iş: nasıl konuşulur

Claude Code ile konuşmak, sohbet arayüzüyle konuşmaktan farklı. Üç kural.

1. Sonucu tarif et, adımları değil

"Şu dosyayı aç, 42. satıra git, şunu değiştir" demek yerine ne olmasını istediğini söyle. Adımları o bulur — ve senin bilmediğin bağımlı yerleri de görür.

```
KÖTÜ

utils.js dosyasını aç, formatDate fonksiyonunu bul,
içindeki toLocaleDateString'i değiştir.
```

```
İYİ

Tarihler her yerde Türkçe biçimde görünsün (7 Eylül 2026).
Şu an bazı yerlerde İngilizce çıkıyor. Tüm kullanım
yerlerini bul ve tutarlı hâle getir.
```

2. Doğrulamayı iste

En yüksek kaliteyi veren tek cümle bu. İşin yapıldığını değil, *çalıştığını* görmesini iste.

```
KALIP
[iş tarifi]

Değişikliği yaptıktan sonra testleri çalıştır ve
geçtiğini göster. Test yoksa, değişikliği doğrulayan
küçük bir kontrol yap ve sonucunu göster.
```

3. Belirsizlikte durmasını söyle

```
KALIP
Emin olmadığın bir karar noktasına gelirsen dur ve bana sor.
Varsayımla ilerleme — özellikle veri kaybına yol açabilecek
işlerde.
```

Çıkarım: Sonucu tarif et, doğrulamayı iste, belirsizlikte durdur. Bu üç cümle, çıktı kalitesindeki farkın büyük kısmını açıklıyor.

---

05 Süreç tarif etmek: ajanlı araçta asıl beceri

Sohbet arayüzünde *ne istediğini* tarif ediyorsun. Ajanlı bir araçta buna bir şey daha ekleniyor: **nasıl çalışılacağı.** Bu ayrım, Claude Code'da sonuç kalitesini belirleyen tek en önemli şey.

Katman	Sohbette	Claude Code'da
Ürün	"Şu fonksiyonu yaz"	Aynı
Süreç	Genelde gereksiz	**Kritik** — hangi dosyalara bakılacak, ne sırayla, ne zaman duracak
Performans	Ton, uzunluk	Test geçecek mi, mevcut biçime uyacak mı, hangi kütüphane kullanılmayacak

Süreç tarif etmenin dört öğesi

```
AJANLI İŞ TARİFİ — İSKELET

HEDEF
[Tek cümlede ne olmasını istiyorum]

ÖNCE ANLA
Şu dosyalara bak: [dosyalar/klasörler]
Mevcut [benzer şey] nasıl yapılmış, ona uy.
Anlamadığın bir şey varsa DEĞİŞİKLİK YAPMADAN önce sor.

SONRA YAP
1. [ilk adım]
2. [ikinci adım]
Her adımdan sonra dur ve bana göster.

DOKUNMA
[Şu dosyalar / şu klasör / şu yapılandırma]

BİTTİ SAYILIR
[Test geçiyor / şu komut çalışıyor / şu çıktı geliyor]
```

"ÖNCE ANLA" bölümü en çok atlanan ve en çok fark yaratan bölüm. Bu bölüm olmadan araç doğrudan yazmaya başlıyor ve mevcut kodun biçimine, kurallarına, kalıplarına uymuyor. Sonuç: çalışan ama *yabancı* duran kod.

> **"BİTTİ SAYILIR" SATIRININ GÜCÜ**
>
> Bitiş ölçütü vermezsen araç ne zaman duracağını bilemiyor — ya erken bırakıyor ya gereksiz genişletiyor. Somut bir ölçüt ("`npm test` geçiyor") vermek, aracın kendi kendini kontrol etmesini sağlıyor. Bu tek satır, geri dönüş turlarını belirgin biçimde azaltıyor.

Adım adım ilerletme

Büyük bir işi tek seferde vermek, ajanlı araçta en sık yapılan hata. Araç ilerledikçe hata birikiyor ve nerede yanlış gittiğini bulmak zorlaşıyor.

Yanlış	Doğru
"Bu özelliği baştan sona yap"	"Önce sadece veri modelini kur, bana göster"
Bitince kontrol etmek	Her adımda kontrol etmek
10 dosyayı aynı anda değiştirtmek	Bir dosya, bir kontrol
"Hataları da düzelt"	"Hataları listele, hangisini düzeltelim ben söyleyeyim"

Çıkarım: Ajanlı araçlarda hız, *tek seferde çok iş vermekten* gelmiyor — *az geri dönüşten* geliyor. Küçük adımlarla ilerlemek yavaş görünüyor ama toplamda çok daha hızlı, çünkü hata erken yakalanıyor.

06 CLAUDE.md — projeye hafıza vermek

Proje köküne koyduğun `CLAUDE.md` dosyası her oturumun başında okunuyor. Bu, ilk derste anlatılan "bağlam kartı"nın proje düzeyindeki karşılığı.

Ayrıca Claude Code çalışırken kendi kendine **otomatik hafıza** da biriktiriyor — oturumlar arası öğrendiklerini sen yazmadan saklıyor.

ÖRNEK CLAUDE.MD

```
# Proje adı

## Ne olduğu
Bu proje [tek cümle].

## Nasıl çalıştırılır
- Kurulum: ...
- Geliştirme: ...
- Test: ...

## Kurallar
- Yeni bağımlılık ekleme, önce sor
- Bu klasöre dokunma: [yol]
- Değişiklikten sonra testleri çalıştır
- Yorum satırlarını Türkçe yaz

## Bilinmesi gerekenler
- [Sürprizli bir davranış veya geçmiş hata]
```

> EN DEĞERLİ BÖLÜM HANGİSİ
>
> "Bilinmesi gerekenler". Kod okunarak anlaşılmayan şeyler buraya yazılır: neden böyle yapıldığı, hangi çözümün daha önce denenip başarısız olduğu, hangi kısmın görünenden riskli olduğu. Bu bölüm zamanla projenin en pahalı bilgisini biriktirir.

07 İş tipine göre tarif kalıpları

Aşağıdaki dört kalıp, en sık karşılaşılan iş tiplerini kapsıyor. Köşeli parantezleri doldur.

1 · HATA BULMA VE DÜZELTME

```
HEDEF: [hata tarifi — ne oluyor, ne olmalıydı]

TEKRAR ÜRETME
Şu adımlarla oluşuyor: [adımlar]
Hata mesajı: [tam metin]

ÖNCE ANLA
İlgili dosyalar: [dosyalar]
Bana önce 3 olası sebebi olasılık sırasıyla listele.
Her sebep için nasıl doğrulayacağımızı söyle.
DÜZELTME YAPMA — önce hangisi olduğunu birlikte bulalım.

BİTTİ SAYILIR: [test/komut] geçiyor ve hata tekrar etmiyor.
```

2 · YENİ ÖZELLİK

```
HEDEF: [tek cümle]

ÖNCE ANLA
Benzer özellik şurada yapılmış: [dosya]
Aynı yapıyı ve isimlendirmeyi kullan.
Anlamadığın karar varsa sor, varsayma.

ADIMLAR
1. Sadece [ilk parça] — bana göster, onaylayayım
2. Onaydan sonra [ikinci parça]
Adım atlama.

DOKUNMA: [dosyalar/klasörler]
BİTTİ SAYILIR: [somut ölçüt]
```

3 · İNCELEME (DEĞİŞİKLİK YOK)

```
Bu kodu incele. HİÇBİR DEĞİŞİKLİK YAPMA.

Şu sırayla:
1. Hatalı davranış üreten durumlar — somut girdi ve
   beklenen/gerçek çıktıyla
2. Güvenlik riski
3. Okunabilirlik

Her bulgu için: dosya, satır, neden sorun, nasıl tetiklenir.
Stil tercihini gerçek hatadan AYRI başlıkta topla.

Dosyalar: [liste]
```

```
4 · TOPLU DOSYA İŞİ

HEDEF: [klasör] içindeki [dosya tipi] dosyalarında
[işlem] yap.

ÖNCE
Kaç dosya etkileniyor, listele. Bana göster.
Bir tanesinde örnek olarak yap ve sonucu göster.
Onaylamadan diğerlerine geçme.

SONRA
Onay verirsem hepsine uygula.
Sonuçları [yeni klasöre] yaz, orijinalleri BOZMA.

BİTTİ SAYILIR: [n] dosya işlendi, hata yok.
```

Dördünde de ortak olan: "önce göster, sonra yap" ritmi. Ajanlı araçta bu ritim, hız kaybı gibi görünüyor ama gerçekte geri dönüşleri kestiği için toplam süreyi kısaltıyor — ve geri alınamaz hatayı önlüyor.

08 İzinler ve güvenlik

Claude Code dosya değiştirebildiği ve komut çalıştırabildiği için, izin modeli öğrenilmesi gereken ilk konulardan.

Üç temel alışkanlık

- **Git kullan.** En önemli güvenlik ağı. Değişiklik beğenilmezse geri alınır. Git olmayan bir klasörde çalıştırmadan önce yedek al.
- **İzin isteklerini okuyup onayla.** Otomatik onayı ancak ne yaptığını tam bildiğin, geri dönülebilir işlerde aç.
- **Sırları dosyaya yazdırma.** API anahtarı, şifre ve token kod içine değil ortam değişkenine girer — bunu `CLAUDE.md`'ye kural olarak yaz.

> **SİLME VE ÜZERİNE YAZMA**
>
> Geri dönüşü olmayan işlerde ("şu klasörü temizle", "eskileri sil") komutu onaylamadan önce ne sildiğini kendin gör. Git izlemeyen dosyalar (build çıktıları, yerel yapılandırma) geri gelmiyor.

09 Bağlam yönetimi — uzun oturumların sorunu

Claude Code'da en sık yaşanan ve en az anlaşılan sorun şu: oturum uzadıkça araç başta söylediklerini "unutuyor" gibi davranıyor. Unutmuyor — **artık göremiyor.**

Neden oluyor

Modelin bir seferde görebildiği metin miktarı sınırlı. Uzun bir oturumda okunan dosyalar, çalıştırılan komutların çıktıları ve konuşma geçmişi bu alanı dolduruyor. Dolduğunda en eskiler görüş alanından çıkıyor.

Beş pratik

Pratik	Neden işe yarıyor
Kalıcı kuralları CLAUDE.md'ye yaz	Her oturumda otomatik okunuyor; sohbette söylenen kaybolabiliyor
Yeni iş = yeni oturum	Alakasız geçmiş, bağlamı boş yere dolduruyor
Gereksiz dosya okutma	"Tüm projeyi oku" pahalı ve çoğu zaman gereksiz
Uzun çıktıları filtrele	Bin satırlık log yerine ilgili kısmı ver
Önemli kararı tekrar hatırlat	Uzun oturumda bir cümlelik hatırlatma ucuz

> BELİRTİ: ARAÇ KENDİ YAZDIĞINI BOZUYOR
>
> Uzun bir oturumda araç, kendi bir saat önce yazdığı kodu bozabiliyor — çünkü onu artık göremiyor. Bunu gördüğünde çözüm daha fazla açıklama yapmak değil: **oturumu bitir, yeni oturum aç ve mevcut durumu kısaca özetle.** Temiz bağlam, uzun açıklamadan iyidir.

10 Maliyet ve hız

Ajanlı araçlar sohbet arayüzünden farklı bir maliyet yapısına sahip: iş başına maliyet, ne kadar dosya okunduğuna ve kaç tur döndüğüne bağlı.

Maliyeti artıran dört şey

1. **Gereksiz dosya okuma.** "Projeyi incele" demek, bütün dosyaları okutmak demek olabiliyor.
2. **Uzun oturumlar.** Bağlam her turda yeniden işleniyor.
3. **Belirsiz talimat.** Yanlış anlaşılan iş, geri dönüş turu demek.
4. **Bitiş ölçütü olmaması.** Araç ne zaman duracağını bilmiyor.

Maliyeti düşüren dört alışkanlık

Alışkanlık	Etkisi
Hangi dosyalara bakılacağını söylemek	Arama turlarını kesiyor
İşi küçük adımlara bölmek	Hatalı yolda uzun gidilmiyor
Somut bitiş ölçütü vermek	Gereksiz genişleme olmuyor
Yeni iş için yeni oturum	Bağlam küçük kalıyor

Çıkarım: Bu dört alışkanlığın hepsi aynı zamanda *kaliteyi* de artırıyor. Maliyeti düşüren şeyle sonucu iyileştiren şey aynı: **net tarif.**

11 Git ile çalışmak

Claude Code git ile doğrudan çalışıyor: değişiklikleri hazırlıyor, commit mesajı yazıyor, dal açıyor, pull request gönderiyor.

```
YAYGIN İSTEKLER
Değişikliklerimi anlamlı bir commit mesajıyla commit'le.

---

Yeni bir dal aç, şu değişikliği orada yap ve
pull request açıklaması hazırla.

---

Son commit'ten bu yana ne değişti? Özetle.
```

Önerilen akış

1. Yeni bir dalda çalış — ana dalda değil
2. İşi küçük parçalara böl, her parçadan sonra commit'le
3. Commit mesajını okumadan onaylama; neyin neden değiştiğini anlatmalı
4. Beğenmediğin değişikliği tartışma — geri al, yeniden tarif et

12 Yazılımcı olmayanlar için: kod dışı kullanımlar

Claude Code'un adı yanıltıcı: dosyalarla çalışan ve komut çalıştırabilen bir ajan, kod dışında da işe yarıyor. Yazılım geliştirmeyenler için en verimli kullanımlar:

İş	Neden burada daha iyi
Çok dosyalı belge işleri	Onlarca dosyayı okuyup birleştirebiliyor; sohbette tek tek yapıştırmak gerekiyor
Veri temizleme ve dönüştürme	Tabloyu okur, dönüştürür, dosyaya yazar — kopyala-yapıştır yok
Toplu yeniden adlandırma / düzenleme	Yüzlerce dosyada aynı işlem
Rapor derleme	Farklı kaynaklardan veri toplayıp tek belgeye
Arşiv arama ve özetleme	Klasördeki her belgeyi tarayıp bulgu çıkarma
Site ve doküman üretimi	Şablondan çok sayfa üretmek

> ORTAK PAYDA: DOSYA SİSTEMİ
>
> Yukarıdakilerin hepsi aynı sebeple burada daha iyi: **araç senin dosyalarını görüyor ve onlara yazabiliyor.** Sohbet arayüzünde her şeyi yapıştırıp çıktıyı kopyalaman gerekiyor. Fark, iş sayısı arttıkça büyüyor: üç dosyada önemsiz, üç yüz dosyada belirleyici.

Kod yazmayan biri için ilk üç iş

1. **Bir klasördeki belgeleri özetlet.** "Bu klasördeki tüm PDF'leri oku, her biri için tek paragraf özet çıkar ve `ozetler.md` dosyasına yaz."
2. **Bir tabloyu temizlet.** "Bu CSV'de tarih biçimlerini standartlaştır, boş satırları at, sonucu yeni dosyaya yaz."
3. **Şablondan çoklu belge üret.** "Bu şablonu al, listedeki her müşteri için doldurup ayrı dosya olarak kaydet."

Üçünün ortak yanı: **çıktı bir dosya.** İşin sonunda elinde kullanılabilir bir şey oluyor, bir sohbet geçmişi değil.

> İZİN AYARINI BURADA DA CİDDİYE AL
>
> Araç dosya silebiliyor ve komut çalıştırabiliyor. Kod yazmıyor olman riski azaltmıyor — aksine, kodla çalışmayan kullanıcılar genelde sürüm kontrolü (git) kullanmadığı için **geri alma imkânı da olmuyor.** Önemli klasörlerde çalışmadan önce yedek al ve izinleri dar tut.

13 Ne zaman kullanılmaz

Durum	Neden
Ne istediğini bilmiyorsun	Araç hızlı ama yön senden gelir. Belirsizlik hızla çoğalır.
Çıktıyı hiç doğrulayamıyorsun	Bilmediğin bir dilde, denetleyemeyeceğin kod üretmek risk biriktirir
Canlı sistemde doğrudan	Önce yerelde veya test ortamında. Geri alınamayan işlem yaptırma.
Yedeği olmayan klasörde	Git yoksa geri dönüş yok
Mimari karar	Seçenekleri sıralatabilirsin ama kararı sen ver — sonucu sen taşıyacaksın

14 İlk yedi gün

Gün	İş
1	Kur ve sadece soru sor. Hiçbir şey değiştirtme. Projeyi anlattır.
2	Küçük ve geri alınabilir bir iş: yorum ekleme, isim düzeltme, biçimlendirme. Git'le geri almayı dene.
3	`CLAUDE.md` yaz. Nasıl çalıştırılır, kurallar, bilinmesi gerekenler.
4	Gerçek ama küçük bir iş. "Doğrulamayı iste" kalıbını kullan.
5	Bir hata ayıkla. Hata mesajını yapıştır, kök sebebi buldur.
6	Git akışını dene: dal aç, commit'let, farkı özetlet.
7	Bir şeyi kasten kötü tarif et ve ne olduğunu gör. Sınırı öğrenmenin en hızlı yolu.

Yedinci günün sonunda aracın ne yaptığını, nerede güçlü olduğunu ve nerede durdurulması gerektiğini bilirsin. Bu, çoğu kullanıcının aylarca ulaşamadığı yer — çünkü çoğu ilk günden büyük iş yaptırmayı deneyip hayal kırıklığına uğruyor.

15 Sık yapılan on hata

#	Hata	Sonucu	Düzeltme
1	Bağlam vermeden iş vermek	Projeye yabancı kod	"Önce şu dosyalara bak" satırı
2	Tek seferde büyük iş	Hata birikiyor, nerede yanlış gittiği bulunamıyor	Adımlara böl, her adımda kontrol et
3	Bitiş ölçütü vermemek	Ya erken bırakıyor ya gereksiz genişletiyor	"Bitti sayılır: [somut ölçüt]"
4	CLAUDE.md yazmamak	Her oturumda aynı şeyleri anlatmak	Kalıcı kuralları dosyaya taşı
5	Tüm izinleri açmak	Geri alınamaz işlem riski	Dar izinle başla, gerektikçe aç
6	Sürüm kontrolü olmadan çalışmak	Geri dönüş imkânı yok	Önce git deposu kur ya da yedek al
7	Çıktıyı okumadan onaylamak	Sessiz hata birikiyor	Her değişikliği gözden geçir
8	Aynı oturumda alakasız işler	Bağlam doluyor, araç kendi yazdığını bozuyor	Yeni iş = yeni oturum
9	Sırları dosyaya yazdırmak	Anahtar ve şifre sürüm geçmişine giriyor	Ortam değişkeni kullan; .gitignore kontrol et
10	"Hataları da düzelt" demek	Ne düzeltildiği görünmüyor	Önce listelet, sonra seçerek düzelt

> **DOKUZUNCU HATA EN PAHALISI**
>
> Bir API anahtarı yanlışlıkla depoya girdiğinde, dosyayı silmek yetmiyor — **sürüm geçmişinde kalıyor.** Depo herkese açıksa anahtar dakikalar içinde taranıp bulunabiliyor. Kural: sırlar hiçbir zaman koda yazılmaz, ortam değişkeninde tutulur; ve depoyu paylaşmadan önce geçmiş taranır.

Kendini kontrol listesi — her oturumdan önce

1. Bu iş için doğru araç mı, yoksa sohbet arayüzü yeter mi
2. Sürüm kontrolü ya da yedek var mı
3. Hangi dosyalara bakılacağını söyledim mi
4. Bitiş ölçütünü yazdım mı
5. Dokunulmaması gerekenleri söyledim mi

16 Yedi günün sonunda elinde ne olacak

Çıktı	Ne işe yarar
Kurulmuş ve çalışan araç	Kendi projende, kendi izin ayarlarınla
Yazılmış bir CLAUDE.md	Her oturumda aynı şeyleri anlatmayı bitirir
İş tarifi iskeleti	Hedef, önce anla, sonra yap, dokunma, bitti sayılır
Tamamlanmış üç küçük iş	Aracın nerede güçlü nerede zayıf olduğunu deneyimle görmek
İzin ve güvenlik kararı	Neyin otomatik, neyin onaylı olduğu
Kendi hata listen	On hatadan hangilerini yaptığın

Son çıkarım: Bu araçta ustalık, komut ezberlemekten gelmiyor. Sadece iki şeyden geliyor: **süreci tarif edebilmek** ve **çıktıyı denetleyebilmek.** İkisi de bu serinin ilk dersinde anlatılan becerilerin aynısı — burada sadece bahis daha yüksek, çünkü araç senin dosyalarına gerçekten yazıyor.

Buradan sonrası

Bu rehber ilk çalışan işe kadar götürüyor. Alt ajanlar, MCP ile dış araçlara bağlanma, kendi komutlarını yazma (skills), otomatik kapılar (hooks), CLI'ı boru hattında kullanma, zamanlanmış görevler ve ekip kurulumu 'nda.

Türkçe Claude Code içeriklerini Telegram kanalında paylaşıyorum.

t.me/aidanisman · aidanisman.pages.dev