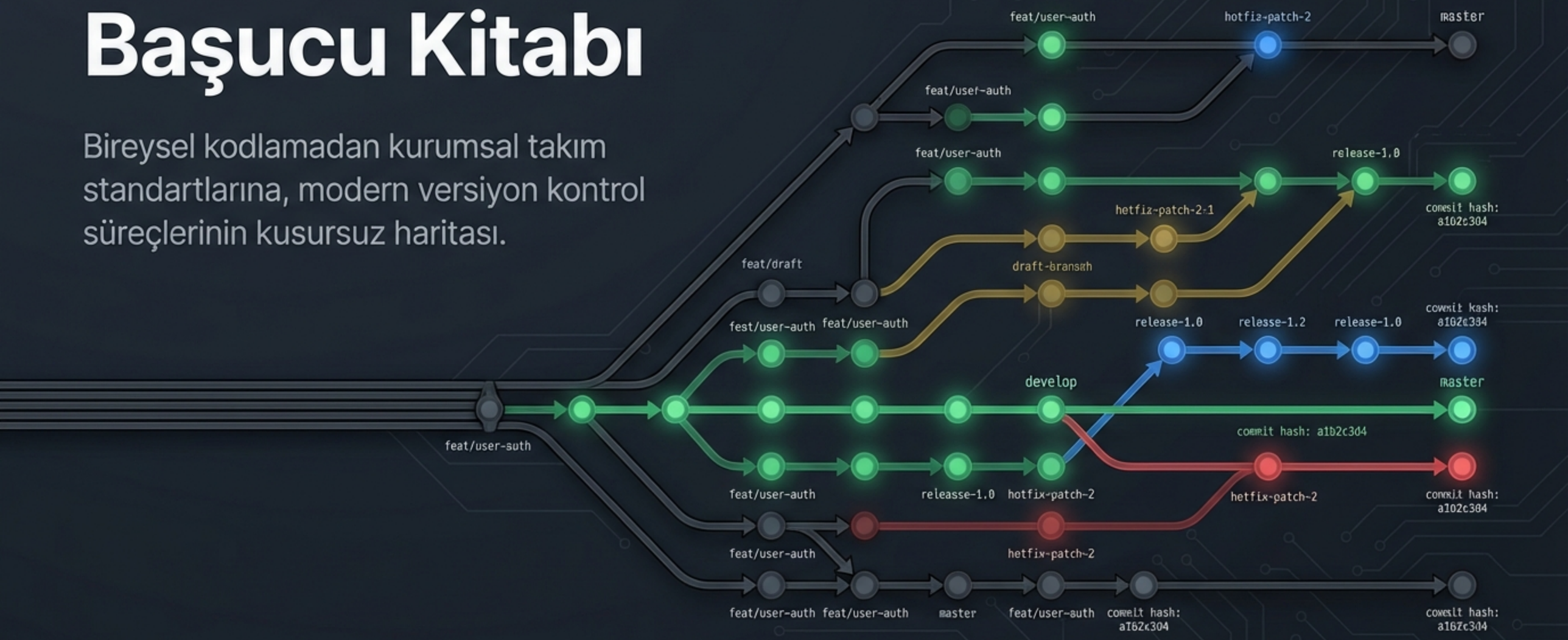
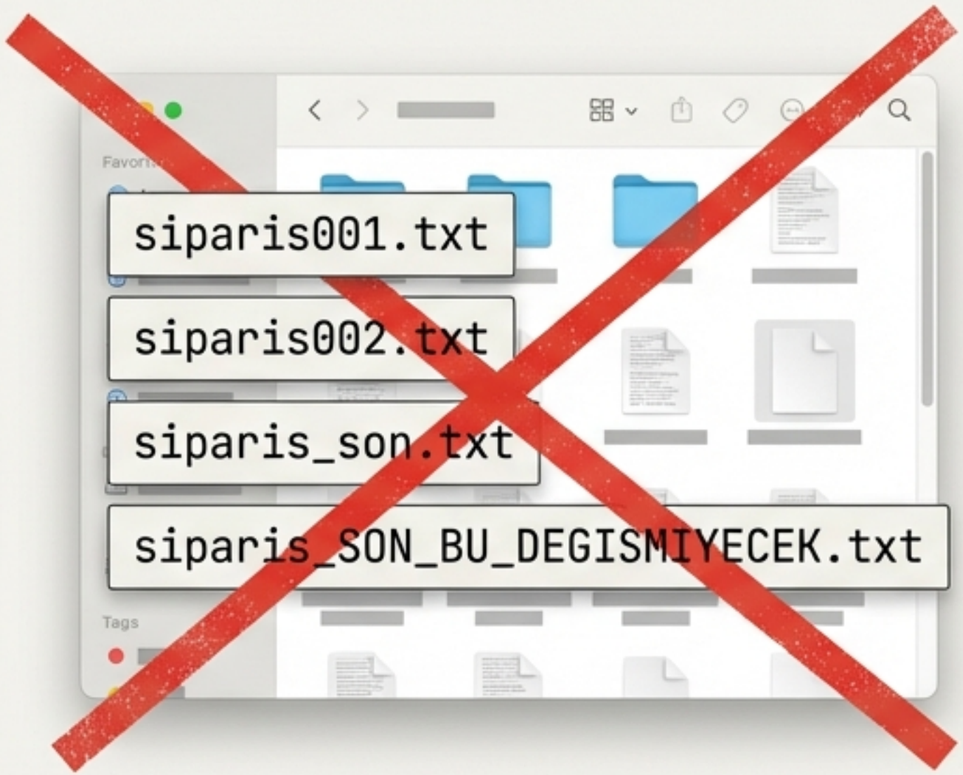
Git Ustalık Başucu Kitabı

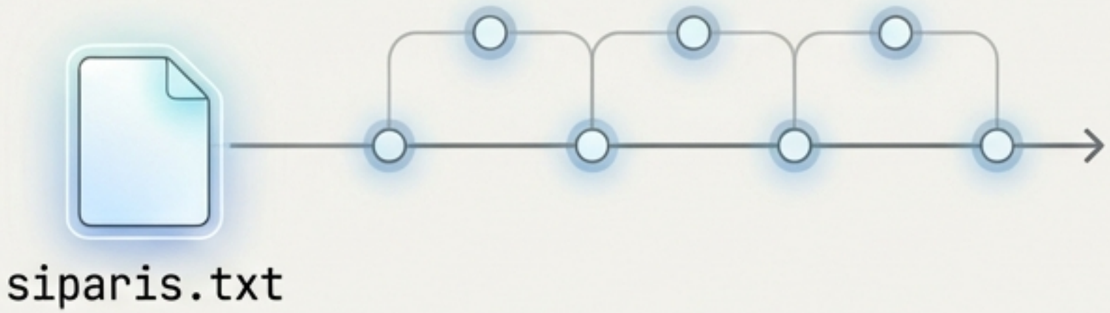
Bireysel kodlamadan kurumsal takım standartlarına, modern versiyon kontrol süreçlerinin kusursuz haritası.



Dosya İsimlendirme Kaosunun Sonu

İnsanoğlunun en ilkel versiyon kontrol yöntemi, dosya adının sonuna numara eklemektir. Git, bu “siparis001.txt” enflasyonunu ortadan kaldırır. Tek bir dosya ismi kalır, zaman içindeki tüm değişimler şifrelenmiş düğümler (commits) halinde deponun tarihçesinde kusursuzca saklanır.





Açık kaynaklı projelerin **%99'u**, bu kaosu önlemek için versiyon kontrol sistemlerini kullanmaktadır.

Versiyon Kontrol Sistemlerinin Evrimi

Yerel (Local)

- Değişiklikler aynı bilgisayarda saklanır.

! Risk: Disk çökmesi = Tam veri kaybı.

Merkezi (Centralized - SVN)

- Tek bir merkezi sunucu. Herkes buradan dosyaları alır (checkout).

! Zayıflık: Sunucu çökerse çevrimdışı kalırsınız. Tek Nokta Hatası.

Dağıtık (Distributed - Git)

- Her bilgisayarda deponun tam ve eksiksiz bir kopyası (clone) vardır.
- Çevrimdışı çalışma mümkündür.
- Hız maksimumdur ve tek nokta hatası yoktur.

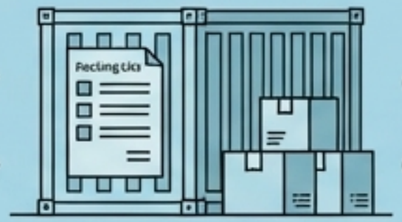
Zihinsel Model: Git’in Üç Öz Alanı

Çalışma Ağacı
(Working Tree)

O an üzerinde çalıştığınız, izlenmeyen (untracked) taslak bölgesi.

Evreleme Alanı
(Staging Area)

Paketleme ve depoya gitmeden önceki son seçim alanı.



Paketleme ve depoya gitmeden önceki son seçim alanı.

Yerel Depo
(Local Repository)

`.git` dizini. Değişikliklerin kalıcı ve güvenli olarak şifrelendiği kasa.

`.git` dizini. Değişikliklerin kalıcı ve güvenli olarak şifrelendiği kasa.

`git status` : Bu üç alan arasındaki farkları gösteren radarınızdır.

Bir Kodun Yolculuğu: Günlük Komut Akışı

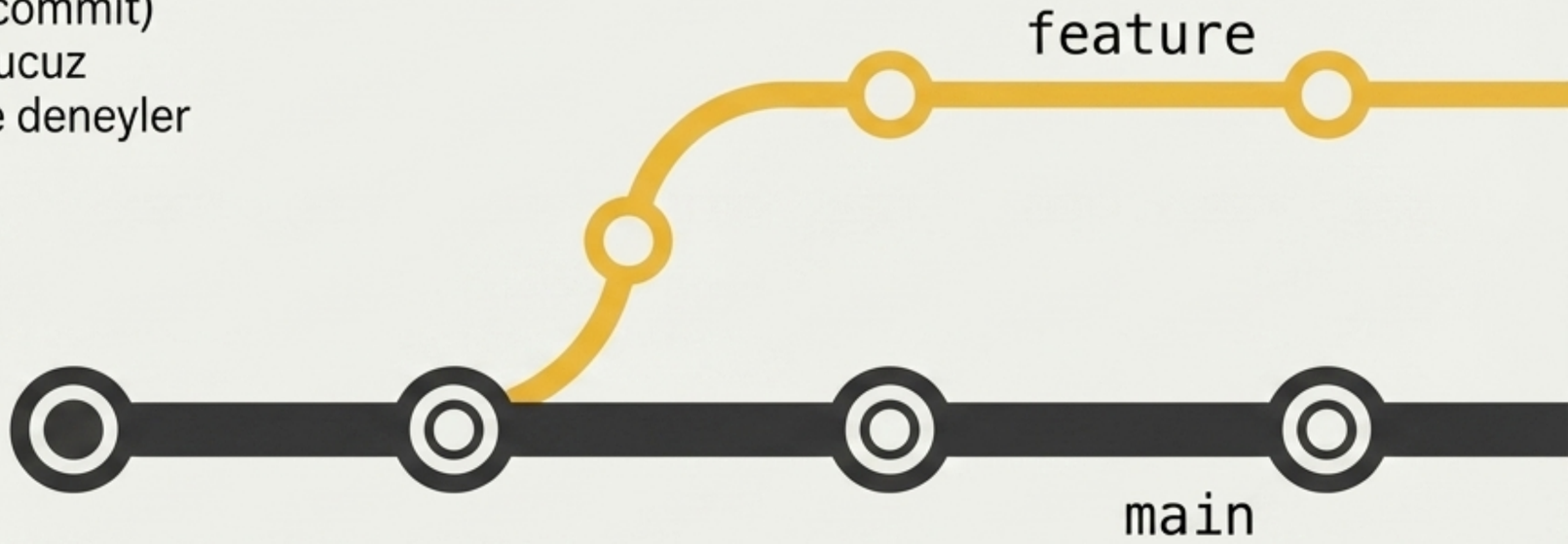
Geri Alma (Undo) Manevraları

- Mavi alandan çıkarmak için: `git reset <dosya>`
- Çalışma alanındaki değişikliği iptal etmek için: `git restore <dosya>`

Paralel Evrenler Yaratmak: Branch Kavramı

Klasik sistemlerde dallanma korkutucudur. Git'te ise şubeler (branches) sadece bir düğümü (commit) işaret eden inanılmaz derecede hafif ve ucuz işaretçilerdir. Ana hattı kirletmeden izole deneyler yapmak, takım çalışmasının temelidir.

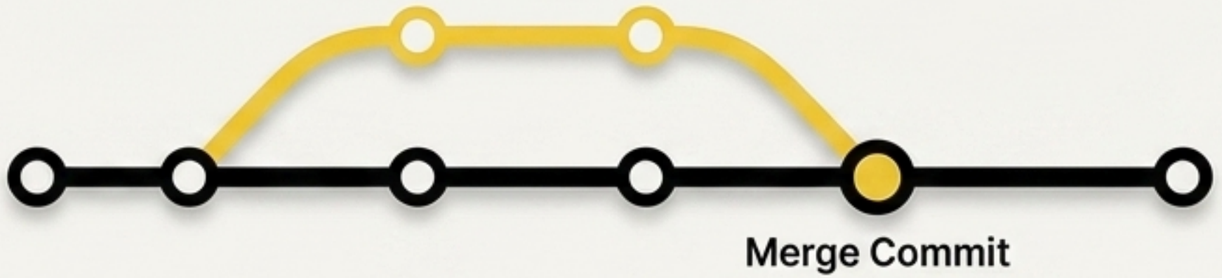
```
Yeni dal oluştur ve geçiş yap:
> git switch -c <isim>

Dallar arası seyahat et:
> git switch <isim>

Dalları listele:
> git branch
```



Entegrasyon Matrisi: Merge ve Rebase

`git merge <dal>`

Şubeleri birleştirir ve özel bir 'Merge Commit' oluşturur. Avantajı tahribatsız olmasıdır, bağlamı korur. Dezavantajı ise tarihçenin karmaşık görünmesidir.

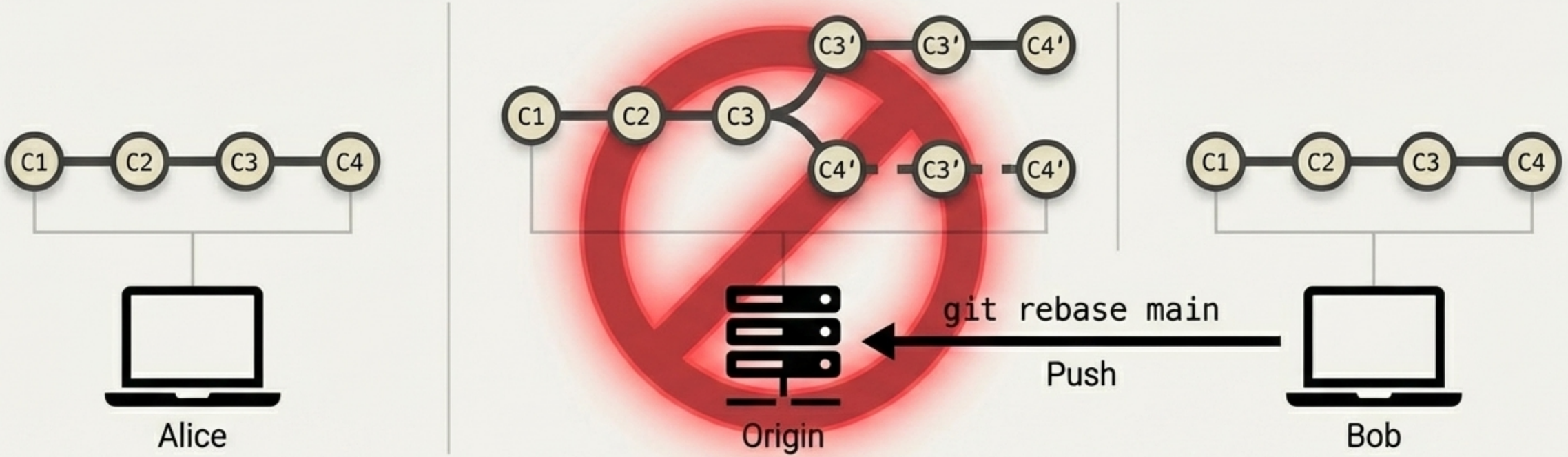


`git rebase <dal>`

Seçili şubeyi koparır, ana şubenin en ucuna yeni komitler yaratarak yapıştırır.

Avantajı kusursuz ve düz bir tarihçe çizgisidir. Dezavantajı ise geçmişi yeniden yazmasıdır.

Altın Kural: Asla Ortak Dalları Rebase Etmeyin



Rebase, komitleri taşımaz; onları yok edip yeni kimliklerle (ID) baştan yaratır. Ortak kullanılan bir ana dalı (main) rebase ederseniz, takım arkadaşlarınızın bilgisayarlarındaki geçmiş ile sunucudaki geçmiş tamamen uyuşmaz hale gelir (Forked History).

KURAL: "Bu dala benden başka bakan var mı?" Cevap **EVET** ise **git rebase KULLANMA**. Yerine **git merge** kullan.

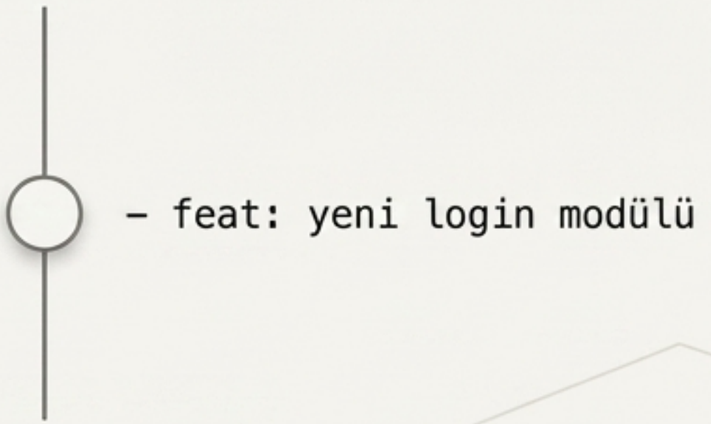
Tarihçeyi Yeniden Yazmak: **İnteraktif Temizlik**

Pull Request açmadan önce, son komitleriniz üzerinde zaman yolculuğu yapın ve projenizi profesyonelleştirin.

Before

`git rebase -i HEAD~3`

After




```
pick 33d5b7a Mesaj 1
squash 9480b3d Hata düzeltme
fixup 5c67e61 Ufak tefek
```

Takım Mimarinizi Seçmek: **İş Akışı Modelleri**

Git Flow

Git Flow (Vincent Driessen)

Git Flow (Vincent Driessen)

Kesin sürümlenmiş yazılımlar (v1.0, v1.1) ve geriye dönük destek gerektiren projeler için. Kuralları ve dallanma yapısı katıdır.

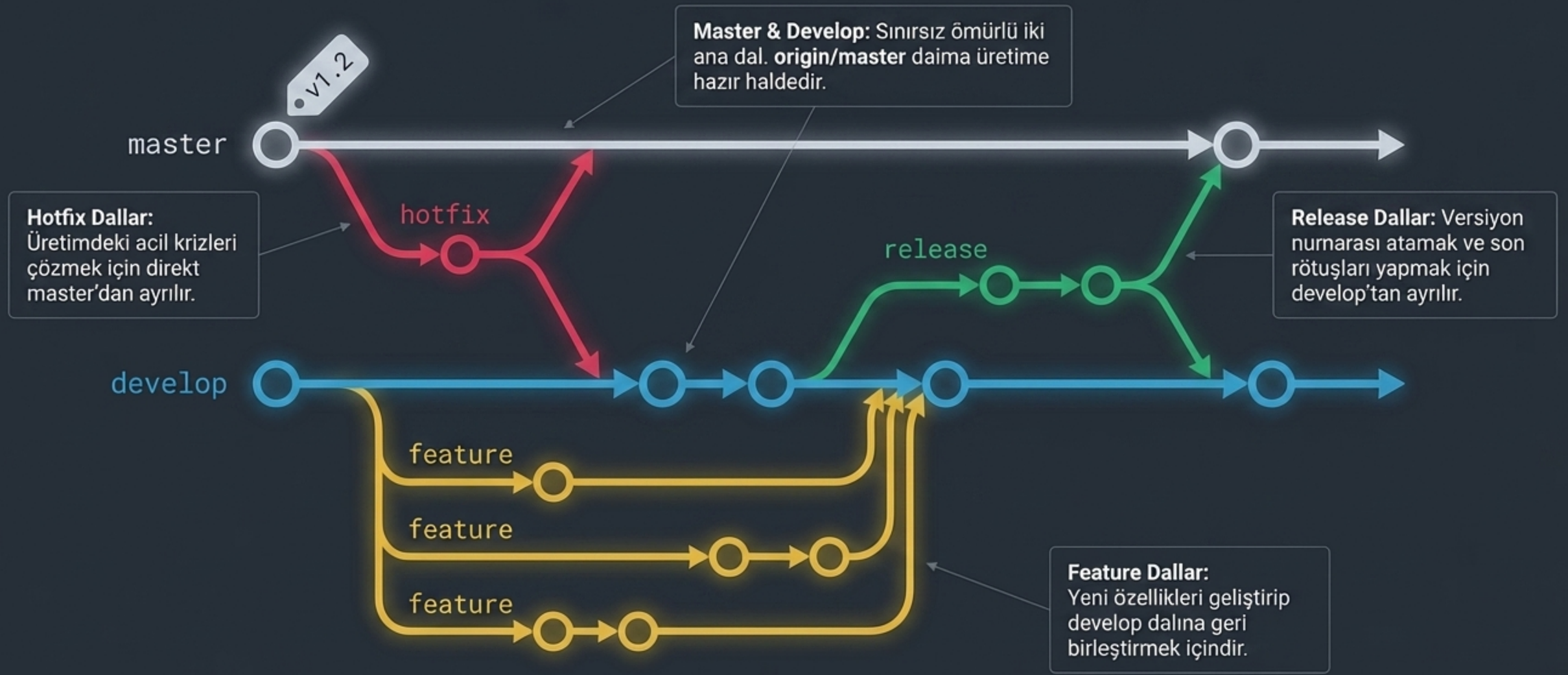
GitHub Flow

GitHub Flow

GitHub Flow

Sürekli teslimat (Continuous Delivery) yapan, anında üretim (Production) ortamına çıkılan modern Web Uygulamaları (Web Apps) için idealdir. Sadece 'main' dalı ve özellik dalları vardır. Karmaşıklıktan uzaktır.

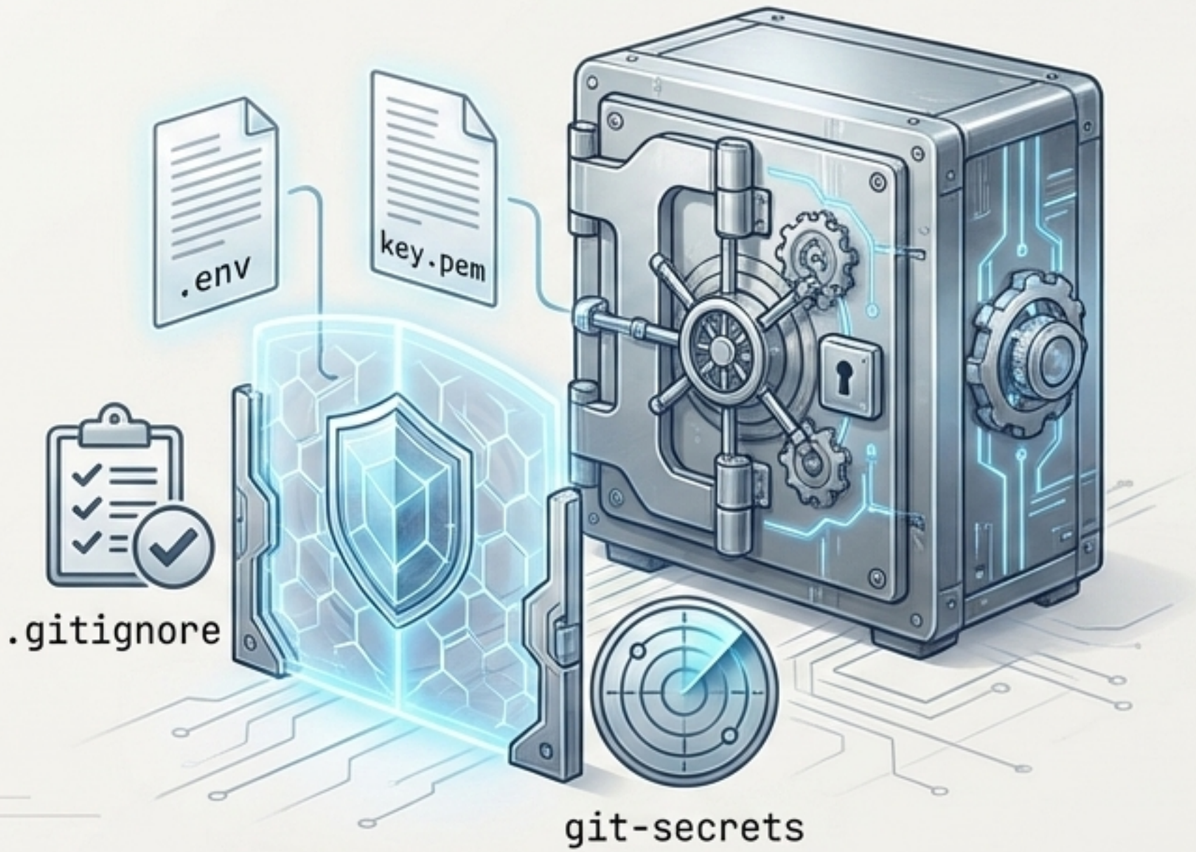
Anatomi: Git Flow Mimarisi



Kusursuz Bir Commit Mesajının Anatomisi

Güvenlik I: Hassas Veri İzolasyonu

Bir API anahtarının ya da veritabanı parolasının depoya sızması saniyeler sürer, hasarı devasa olur.



Önleyici Kalkan: .gitignore

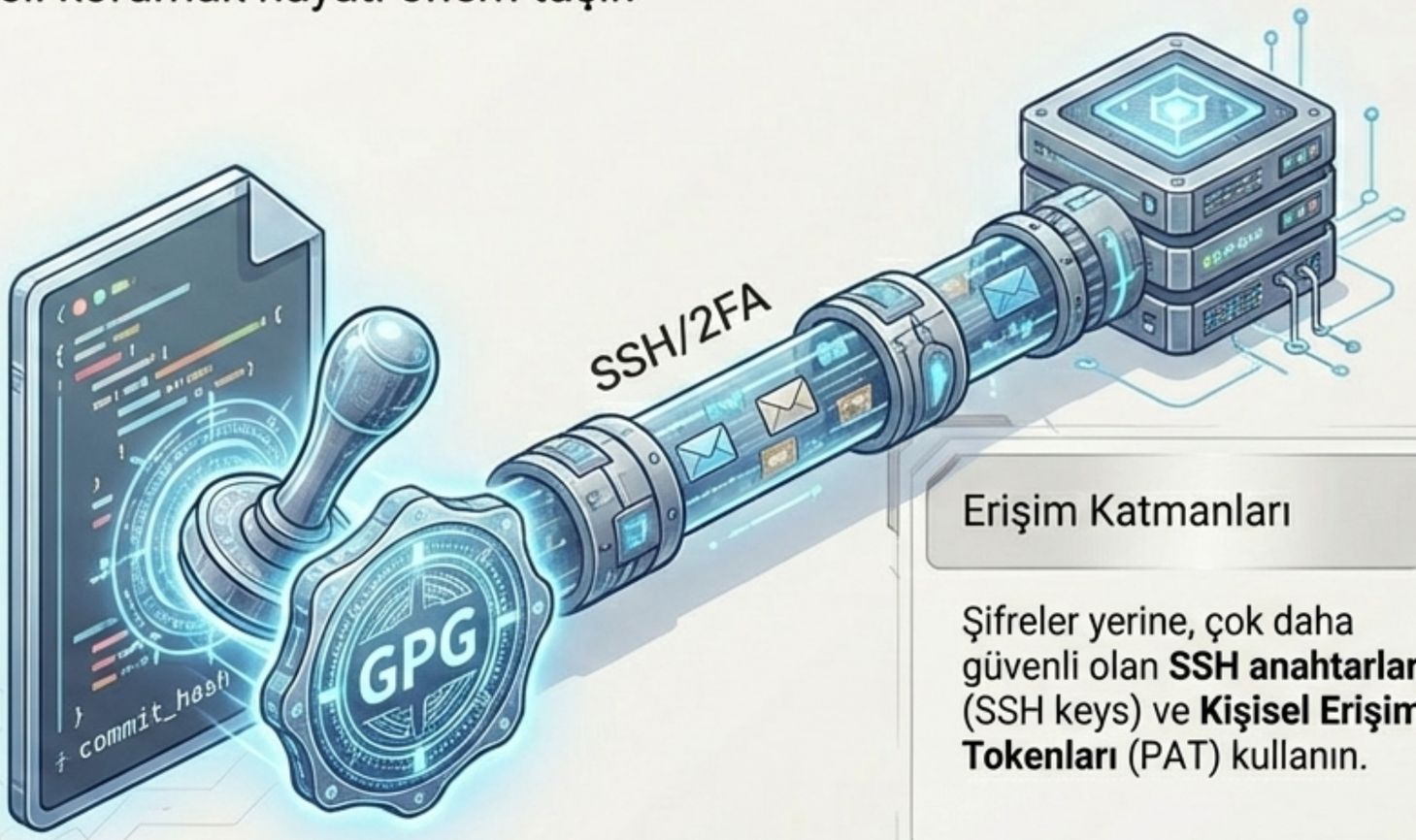
Depoya girmemesi gereken dosyaları yapılandırın (örneğin **.env** ve ***.pem** dosyalarını dışarıda bırakın).

Otomatize Edilmiş Denetim

Komitleri taramak ve sızıntı tespit edilirse engellemek için **git-secrets** gibi yardımcı araçlar kullanın. Hassas bilgileri daima dışarıda tutun.

Güvenlik II: Kimlik, Erişim ve İmzalar

Yazdığınız kodu kimin yazdığını kanıtlamak ve depoya giden tüneli korumak hayati önem taşır.



GPG ile İmzalama

Komitlerinize "doğrulanmış" rozeti kazandırın. Proje bütünlüğünü koruyun.

```
git config --global
user.signingkey <KEY_ID>
```

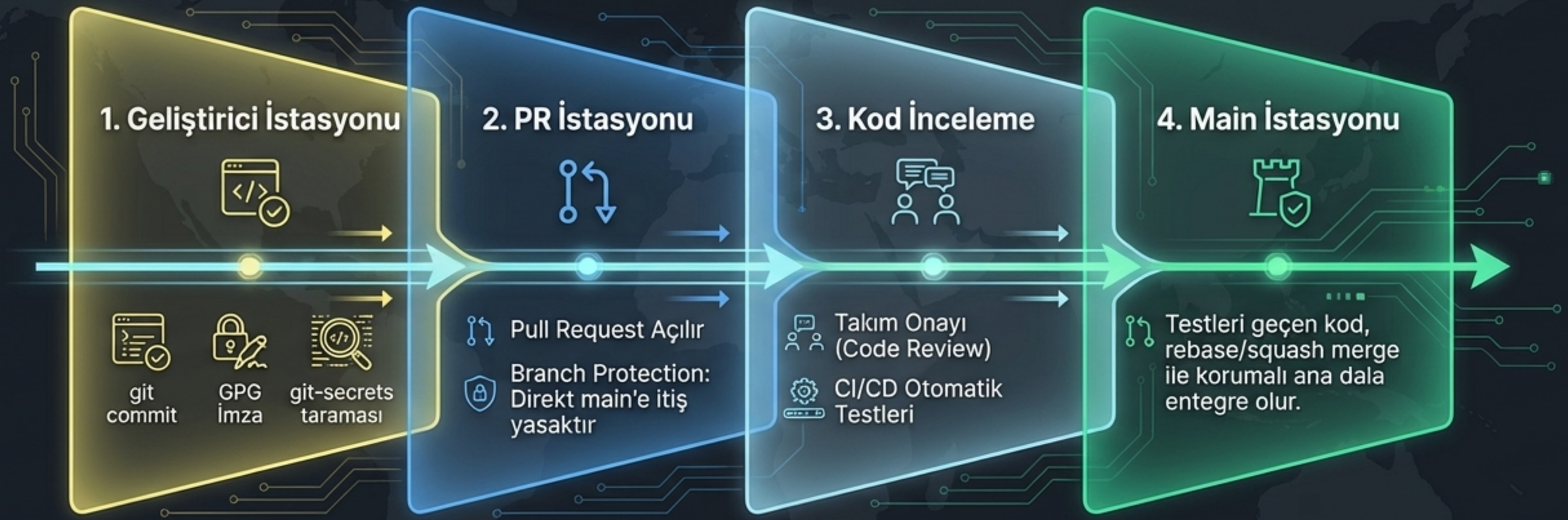
Erişim Katmanları

Şifreler yerine, çok daha güvenli olan **SSH anahtarları** (SSH keys) ve **Kişisel Erişim Tokenları** (PAT) kullanın.

Sıfır Güven (Zero Trust)

Depoya erişim yetkisi olan herkes için **Çift Aşamalı Doğrulama** (2FA) zorunluluğu getirin.

Modern Ekibin Kusursuz Blueprint'i



Mükemmel kod yoktur, mükemmel süreçler vardır. Güvenli komitlerden **takım onaylarına** uzanan bu kalite kapısı; riskleri sıfıra indirir, tarihçeyi tertemiz tutar ve yazılımı bir zanaat eserine dönüştürür. **Git**, bu mimarinin atmakta olan kalbidir.