 Aslında her şey [Linus Torvalds](http://tr.wikipedia.org/wiki/Linus_Torvalds)'ın Linux çekirdeğini açık kaynak haline getirmesi sonucu dünyanın dört bir tarafındaki geliştiricilerin, kod üzerinde yaptığı değişiklikleri e-posta olarak atması sonucu başladı. Torvalds, günde binlerce e-posta alıp, bir de kod üzerinde gerekli güncellemeler, eklemeler, çıkarmalar, değişiklikler derken hayli zorlu bir hal alan bu geliştirme sürecini yönetebilmek için bir araç yaptı: [Git](http://tr.wikipedia.org/wiki/Git_%28yaz%C4%B1l%C4%B1m%29) bu aracın ismi.

GitHub ise basitçe özetlemek gerekirse, Git’in arayüzlü bir web sürümü olabilir.

(https://gelecegiyazanlar.turkcell.com.tr/blog/github-ile-surum-kontrolu)

**Git Nedir?**

* Proje geliştirirken, kodlarımızı bir klasörde yedekliyoruz. Ancak yedekler çoğaldıkça hem yedekler karışmaya başlıyor hem de alanımız azalıyor. Bu problemleri çözmeye yarayan yardımcı programlar var. Bunlar bizim çalışmalarımızın yedeğini bizim yerimize alarak hem yerden, zamandan hem de kargaşadan tasarruf etmemizi sağlıyor. Bu araçlara sürüm kontrol sistemleri ya da versiyon kontrol sistemleri deniyor.
* Projemiz ya da herhangi bir klasör üzerinde yapılan değişiklikleri zamanları, kullanıcı bilgisi, versiyon numarası gibi bilgilerle birlikte tutuyorlar.
* Projenin yayına çıkacak kısmının beklemesini istiyorsak ve üzerinde de bir şeyler denemek istiyorsak o zaman çalışan kopyamız belli bir dallanmayla (branching) ayrılarak diğer kopyalar üzerinde rahatça çalışmamız sağlanıyor.İstersek bu kopyayı çalışabilir durumdaki kopya ile birleştirebiliyoruz.
* Grup halinde çalışmalarda herkesin yaptığı değişiklikleri görebiliyoruz.
* Git dağıtık bir yapı kullanıyor.Diğer versiyon kontrol sistemleri merkezi bir yapı kullanırken ve bu yapıya sürekli bağlı olmak gerekirken git’te her kullanıcıda bütün bir kopya mevcuttur ve internet olmayan ortamlarda dahi proje geliştirilmeye devam edilebiliyor.
  1. Git gerçekten çok hızlı
  2. 2.Git kaynak depoları disk üzerinde çok az yer kaplıyor.
  3. 3.Git ile dallanma (branching) ve birleştirme(merging) işlemleri kolay ve hızlı
  4. 4.Git offline olarak kullanılabiliyor.
  5. 5.Git dağıtık (distributed) olduğu için her bir geliştiricinin makinesinde proje bütün geçmişi ile birlikte tutulabiliyor. İlk başta bunun işlemleri yavaşlattığı ve disk israfı olduğu düşünülebilir. Fakat git'in oldukça etkin algoritmalar kullanması sayesinde bu sorunlar hiç yaşanmıyor.

( <https://github.com/COMU/bohca/wiki/G%C4%B0T-KULLANIM-KLAVUZU>)

## Uyumlu ekip çalışması

Herhangi bir versiyon kontrol sistemi kullanmadığınızda beraber çalıştığınız diğer kişiler ile aynı dosyalar üzerinde çalışabilmek için muhtemelen herkesin erişimine açık paylaşımlı bir klasör kullanmak zorunda kalacaksınız.

Bu tür bir senaryoda kullanılan yazılımların çoğu değiştirilen dosyaya **kilit** koyar ve başka birisi aynı dosyayı düzenlemek istediğinde

* Kullandığı programa bağlı olarak dosya yazma korumalı olarak salt okunur modda (readonly) açılır veya
* Değişiklikler kaydedilmek istendiğinde hata verir

Versiyon kontrol sistemi kullanıldığında ise ekibinizdeki herkes özgür bir şekilde istediği dosyalar üzerinde güvenli bir şekilde istediği değişikliği yapabilir. Herkes değişikliklerini tamamladıktan sonra da tüm değişiklikler versiyon kontrol sistemi kullanılarak sağlıklı bir şekilde **merge** (birleştirme) edilebilir.

## Versiyonların düzgün bir şekilde takip edilebilmesi

* Belki de canınızı en çok sıkacak şey projenizin iki versiyonu arasında tam olarak ne tür farkların olduğunu sağlıklı bir şekilde bilme şansınız olmaması olacaktır

Versiyon kontrol sistemi kullandığınızda sizin çalıştığınız disk alanında proje dosyalarının sadece bir versiyonu bulunur, bu dosyaların daha önceki halleri versiyon kontrol sisteminin denetimindedir.Bu sayede istediğiniz zaman önceki versiyonlara geri dönebilir, versiyonlar arasındaki farklılıkları rahatlıkla inceleyebilir ve versiyonları kaydederken eklediğiniz ilave bilgileri ve yorumlarınızı rahatlıkla görebilirsiniz.

## Önceki Versiyonlara Geri Dönebilme

Dosyalarınızın veya aslında tüm projenizin daha önceki versiyonuna geri dönebilme imkanın size ciddi anlamda özgürlük sağlar; istediğiniz zaman projenizin önceki temiz haline çok hızlı ve rahat bir şekilde dönebilirsiniz.

## Dosyalarınızın neden değiştiğini anlama

Versiyon kontrol sistemleri değişikliklerinizi tamamlayıp **commit** etmek istediğinizde **comment** adı verilen açıklamalar girmenizi isterler. Bu commentler sayesinde projenizin herhangi bir versiyonundaki değişikliklerin nedenlerini de kayıt altına alıp ihtiyaç halinde geri dönüp inceleyebilirsiniz.

Git'de commit işlemi yapılırken comment (yorum metni) girilmesi zorunludur

## Yedekleme

Git gibi dağıtık versiyon kontrol (DVCS) sistemlerinin yan etki olarak sağladığı faydalardan birisi de yedeklemedir. Git sayesinde aynı projede çalışan herkesin kendi bilgisayarında projenin tam bir tarihçesi tutulur. Merkezi versiyon kontrol sistemi sunucusunda bir sorun oluştuğunda takımdaki herhangi birinin kendi diskindeki projeyi sunucuya geri yüklemesi yeterlidir. Diğerleri de kendi bilgisayarlarındaki proje dosyalarını geri yüklenen proje dosyaları ile senkronize edebilirler.

# Kısa Git Tarihçesi

Git 2005 yılında, başta Linus Torvalds olmak üzere Linux çekirdeğini de kodlayan ekip tarafından Linux kaynak kodunu versiyon kontrolü altında tutmak ve kendi iş akışlarını düzenlemek için geliştirilmiştir

Linux'un kaynak kodu 1991-2002 yılları arasındaki dönemde manuel olarak dosyaların paylaşılması şeklinde yönetiliyordu. 2002 yılında Linux geliştiricileri normalde ücretli olan ancak açık kaynak projeler için ücretsiz lisanslama modeli sunan BitKeeper isimli dağıtık versiyon kontrol sistemini kullanmaya başladılar. 2005 yılında BitKeeper'ın ücretsiz sağladığı lisansı geri çekmesi üzerine Linus Torvalds ve Linux ekibi kendi dağıtık versiyon kontrol sistemini geliştirmeye karar verdiler.

# Git İle Çalışmaya Başlamak

## Komut satırı mı yoksa görsel arayüz mü?

Git ile çalışmak için git'in kendi **komut satırı arayüzünü** (Git Command Line Interface) veya görsel kullanıcı arayüzü olan masaüstü uygulamalar (SourceTree, Tortoise Git, Tower veya GitHub) kullanabilirsiniz.

Git ile çalışırken görsel arayüzü olan bir uygulama kullanmanız üretkenliğinizi arttırıp Git'in çok sayıdaki karmaşık komutuna daha hızlı ve kolay erişmenizi sağlar. Diğer yandan Git'in komut satırı arayüzünü kullanmanız Git ile ilgili daha ayrıntılı bilgilenmenizi ve 3. parti uygulamalara bağımlı kalmadan Git ile çalışabilmenizi sağlar. (https://aliozgur.gitbooks.io/git101)

### Git kavram – terimler:

Genel olarak kullanacağımız Git komut ve kavramlarına aşina olmak adına aşağıda yer verdim.

repository: **Depo.** Projenin ana klasörü gibi düşünebilirsiniz. Local ve Remote depolar olacaktır.

init: Bulunulan dizinde yerel bir kod deposu oluşturur. **Proje başlatmak için ilk aşamadır.**

clone: Kopyalamaya, klonlama. Uzak kod deposunun kopyasını almayı sağlar

pull: Uzak depodaki değişikliklerin ardından son halini almayı sağlar

push: **Yerel deponuzda yaptığınız değişiklikleri uzak depoya göndermek için kullanılır.**

branch: **Dal. Projeyi parçalara ayırarak çalışma imkanı verir.**Farklı özellikler geliştirm

commit: **Yaptığımız çalışmaları depoya eklerken verdiğimiz etiket, mesaj, açıklayıcı yazı.**

merge: **Birleştirme**. 2 farklı dal da çalışıyorsak değişiklikleri birleştirmek için kullanılır.

checkout: Dal seçimi yapmamızı sağlar. Dallar arasında geçiş, seçim yaparız.

.gitignore dosyası: Çalışmamızda bazı dosyaların veya dizinlerin versiyonlamasını yani depoya dahil olmasını istemeyiz. Bazen önemli kritik bilgiler içeren dosyalar olabilir bazende programın ürettiği dosyalar olabilir. Bunları depodan devre dışı etmemizi sağlar.  “.gitignore ” dosyası içinde versiyonlamasını istemediğimiz dosya veya dizinleri belirtiriz.