**GIT VERSION CONTROL**

<“Ömer Faruk Akyapak” >

**Versiyon Kontrolü :**

Bir dosya veya bir küme dosyadaki değişiklikleri takip edebilmek için uyguladığımız bir yöntem olarak tanımlayabiliriz. Git gibi sistemler tüm bu değişikliklerin tarihçesini ve içeriğini elektronik olarak bizim için takip ederek kayıt altına almamızı sağlayan veri tabanları olarak düşünülebilir.

Dosya vb. yapının kayıt altına alınmış herhangi bir andaki haline versiyon denir.

**Neden İhtiyacımız Var :**

-Uyumlu ekip çalışması

-İzlenebilirlik

-Önceki versiyonlara geri dönebilme

-Değişiklikleri anlama, takip etme

-Yedekleme

**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.

**Kurulum :**

İşletim sisteminiz Windows ise git ile çalışmak için "msysgit" paketini kullanabilirsiniz.

msysgit paketini kurmak için <http://msysgit.github.io/>adresinden kurulum uygulamasını indirip çalıştırmalısınız. Kurulum adımları sırasında karşınıza çıkacak olan ekranlarda varsayılan ayarları seçili olarak bırakarak kurulumunuzu tamamlayabilirsiniz.

Kurulum tamamlandıktan sonra Windows Başlangıç menüsünden Git klasörü altındaki Git Bash uygulamasını çalıştırıp Git'in komut satırı arayüzünü kullanmaya başlayabilirsiniz.

**Git Konfigürasyonu :**

Git'i kurduğumuza göre artık Git ile çalışmak için bazı ayarlar yapabiliriz. Bu ayarlar için Git bize git config isimli bir araç/komut sunar. Git ayarlarını bir defa yapmanız yeterli olacaktır.

Bu ayarları istediğiniz zaman değiştirebilirsiniz.

Git ayarlarınız aşağıda belirtilen üç konumda kaydedilir ve hiyerarşik olarak bu konumlardan yüklenir;

1.Seviye (/etc/gitconfig dosyası) : Tüm kullanıcı ve projeler için geçerli olan ayarlar bu dosyada kaydedilir. git config komutunu --system seçeneği ile çalıştırırsanız ayarlar bu dosyada kaydedilecek ve bu dosyadan okunacaktır

2.Seviye (/.gitconfig dosyası) : Sadece sizin kullanıcınız için tanımlanan ayarların kaydedildiği dosyadır. git config komutunu --global seçeneği ile çalıştırısanız ayarlar bu dosyaya kaydedilecek ve bu dosyadan okunacaktır

3.Seviye : Proje klasörünüzün (projenizin Git ile versiyon kontrolüne alınmış olması gerekiyor) altında yer alan .git/config dosyasında ise proje bazındaki git ayarlarınız yer alır.

Git ayarlarından en önemli olanları kullanıcı adınız ve email adresinizdir. Git, ayar olarak tanımladığınız değerleri **commit** vb işlemlerde otomatik olarak kullanır. Bu ayarların değerini belirlemek için komut satırında aşağıdaki komutları çalıştırıyoruz

git config --global user.name "faruk"

git config --global user.email "faruk@example.com"

Global seviyede tüm ayarları listelemek için

**git config --global -l**

Global seviyede tek bir ayar'ın değerini (örneğimizde user.name anahtarına sahip ayar) görmek için ise

**git config --global user.name**

**İPUCU**

Git'in komutları ve bu komutların seçenek ve parametreleri ile ilgili yardım almak istediğinizde

* git [komut adı] --help (örneğin: git init --help)
* git help [komut adı] (örneğin: git help init)

komutlarını kullanabilirsiniz.

**Untracked Files (İzleri Oluşturulmamış Dosyalar) :**

a1.txt ve a2.txt dosyalarını çalıştığınız klasöre eklediğimizde bunları henüz izleri oluşturulmamıştır. Fonksiyonu ile izi oluşmamış(untracked) dosyaları görebililirsiniz.

git status

**Staging Taşıma :**

git add <filename>

**Staging Alanından Çıkarma :**

git reset HEAD <filename>

**Local (Kendi Çalışma Ortamınızaki) Dosyaları Değiştirmek :**

git status

git diff

git checkout — a1.txt

**Commit (Save to Local Repo) :**

git commit -m ‘msg’

**Push (Save to Remote Repo) :**

git push origin master

**Pull (Get From Remote Repo) :**

Push yapmadan bir pull yapın çünkü başkaları Remote Repo’da değişiklikler yapmış olabilir önce değişiklikleri alıp sonrasında kendi değişikliklerinizi yapmanız gerekir.

git pull

**Conflict/Merge :**

Gerçek dünyada siz bir dosyada bir güncelleme yaparken başka birisi o dosyayı silmiş, ismini değiştirmiş olabilir. Bunun için bu farklılıkları çözmemiz gerekir. Oluşabilecek durumlar.

Sorun: İki kişinin aynı dosyada değişiklik yapması .

Çözüm: Bendeki kaynağı kullan, Ondaki kaynağı kullan, İkimizdekini mantıklı bir şekilde gözle birleştir.

git pull

Sorun: Birisinin çalıştığı dosyaları ,diğer kişinin silmesi

Çözüm: Bendeki kaynağı kullan, Ondaki kaynağı kullan.Değişikliği yapıp direk push etmeye çalıştığımda (fetch first) uyarısını aldım. AutoFix işleminin yapılamadığını görebilirsiniz. Bunun için kendiniz dosyayı local repo’dan çıkarıp tekrar commit yapmanız gerekir.

# Eklediğiniz ornek1.txt dosyasını metin editörünüz ile açıp içine adınızı ve soyadınızı yazın

# Değişikliklerinizi commit edin

git add -A .

git commit -m “Ad ve soyad eklendi”

# Oluşturulan dalı aktif hale getirelim

git checkout <dalınızın adı>

# ornek1.txt dosyasını metin editörünüz ile açıp içine Deneme yazın

# Değişikliklerinizi commit edin

git add -A .

git commit -m “Ad soyad bilgisi Deneme olarak değiştirildi”

git hist --all

git merge master

**Değişikliklerin Tarihçesini ve Detayını Gör :**

Değişikliklerin tarihçesini bu şekilde görebilirisiniz.

git logs

Commit içerisindeki değişiklikleri ise bu şekilde görebilirsiniz.

git shows

**Git Tag Oluşturma :**

# Oluşturduğunuz dosyayı commit edelim

git add -A .

git commit -m “7. gün 1. alıştırma”

# Deponuzun şu anki halini yansıtan v0.1 isimli tag oluşturduk

git tag v0.1

# Depomuzdaki tag’ları listeledik

git tag

# Depomuzdaki değişiklikleri uzak depo ile senkronize ettik

git push

**Git Branch Oluşturma :**

Genelde projelerde master, development, bugfix, release, poc amaçlı dallar(branch) oluşturulur. Bu dalları oluşturmak git ile oldukça basittir .

git branch