Versiyon Kontrol Sistemi(VCS)

Versiyon kontrol sistemi bir dosya sistemindeki değişiklikleri yöneten bir yardımcı yazılım programıdır. Ekleme, silme, değiştirme özelliklerine sahiptir. Bir yazılım ekibi için bir çok avantaj sağlayacaktır. Aynı takımdaki üyeler aynı kaynak kodu değiştirmek isteyebilir. VCS kaynak kodundaki farklılaşmayı takip eder ve kodun önceki halinin ulaşılabilirliğini sağlar. Önceki sürümlere geri dönüş eski ve yeni kodların karşılaştırılmasını olanak sağlar. Hata oluşmuşsa hatanın daha hızlı çözülebilmesini yani kodun gelişiminin hızlanmasını sağlar. Süreci kolaylaştırır.

İlk versiyon kontrol sistemi 1972’de SCCS(Source Code Control System) yayınlandı. Özgür Yazılım Vakfı(Free Software Foundation) tarafından alternatif olarak GNU RCS(Revision Control System) yayınlandı. Günümüzdeki versiyon kontrol sistemleri kadar gelişmiş olmamalarından dolayı büyük çaplı projeler yerine küçük çaplı projeler için elverişli sistemlerdi.

İki kişi aynı dosyayı değiştirmek istediğinde çakışıklık yaşanır. Bunu gidermek için iki yöntem bulunur.

Pesimist Sistemler(Senkron)

Pesimist yöntemde bir kişi kaynak kod içerisinde bir dosyada değişiklik yapmak istediğinde o dosyaya bir kilit koyar. Başka biri aynı dosya üzerinde değişiklik yapmak isterse VCS buna izin vermez. Dosyayı ilk değiştirmeye çalışan kişi, checkout işlemini yapan kişi, işi bitince dosyanın son halini VCS ortamına geri gönderir, comit işlemini gerçekleştirmiş olur, ve işlemi bitirir. İşlemin bitmesiyle kilit kalkar ve dosya yeniden değişime açılmış olur. Bu yöntem ekip içinde anlaşmazlıklara sebep olabilir. Sürecin uzamasına ve kavgalara sebep olabilir. CVS(Concurrent Versions Systems) ve SVN(Subversion) bu şekilde çalışır.

Optimist Sistemler(Asenkron)

Pesimist yöntemdeki verimsizliğin nedeni olan kilit mekanizmasını kullanmazlar. Bunun yerine her üyenin aynı dosya üzerinde değişiklik yapılmasına izin verir. Değişikliği VCS ortamına gönderdiği versiyon ile önceki versiyon aynı ise çakışma oluşur. Son gönderen kişi bunu çözerek yeni bir versiyonu aktarır. Projede çalışan üyeler aynı dosya üzerinde çok fazla değişiklik yaparsa çakışıklığı çözmek zorlaşır verim yeniden düşer. Git ve günümüzdeki diğer modern VCSler bu şekilde çalışır.

VCSler ekip çalışmasını sağlar. Bunun için yapılan değişiklikleri ve eylemleri bir sunucuda saklaması şarttır. Birden fazla kaynak dosyasının ve değişikliklerin bulunduğu ortam depo(repository) olarak isimlendirilir.

Yerel Sistemler(Local Version Control Systems)

En eski versiyon kontrol sistemi yaklaşımıdır. Proje ve değişiklikler kullanıcı makinasında veri tabanında tutulur. Her yapılan commit versiyon olarak tutulur ve commit değerine hash atanarak versiyonlar birbirinden ayrılır. Bu sistemde bir kişi etkin şekilde çalışabilir yani ekibe uygun bir sistem değildir.

Merkezi Sistemler(Central Version Control Systems)

Merkezi sistemler bu konudaki en basit çözüm yöntemleridir. Kaynak kod ve repository merkezi bir sunucuda yer alır. İstemcide kaynak kodun kopyası yer almaktadır ama repository yer almaz. Repository sadece merkezi bir sistemde bulunur. Kaynak kod üzerinde değişiklik yapmak için öncelikle checkout işlemi gerçekleştirilir ardından geliştirme yapılır. Geliştirme tamamlandığında dosyanın son hali commit edilir, merkezi sunucuya gönderilir, işlem biter. SVN ve CVS’nin versiyon kontrol sistemi merkezi sistemdir. Bu sistemlerde merkezi sunucu arızalanırsa yedek bulunmazsa tüm repository kaybedilir.

Dağıtık Sistemler(Distributed Verison Control Systems)

Merkezi sistemlerin aksine repository ve kaynak kodu bir yerde toplamak yerine dosya üzerinde değişiklik yapan her istemcinin sitemine dağıtır. Tek sunucu değil geliştiricilerinde sunucu olduğu çoklu sunucu sistemine dayanır. Her geliştirici yerel sunucudakilerin kopyalarına sahiptir. Bu sistemde kendi bilgisayarınızda bir repository olduğundan commit işlemi uzak sunucuya değil kendi yerele yapılır. Yerele yapılan işlemler uzak sunucuya gönderilir. Ekibin üyeleri uzak sunucudan gönderdiğiniz dosyaya ulaşıp kendi yerellerine yaptıkları değişikliklerle aktarabilir. Dağıtık sistemlerin avantajı merkezi sunucuya bağlılık azaldığı için işlemlerin daha hızlı gerçekleşmesine olanak sağlayabilmesidir. Kaynak kodun farklı versiyonlarını farklı repository’de saklama imkanı sunar. Ayrıca bu yapıda birden fazla uzak sunucu ile çalışılabilir. Uzak sunucudaki repository kaybedilirse ekipteki herhangi bir üyenin bilgisayarındaki aynı repository’i kullanarak kurtarma işlemi yapılabilir. Git, Mercurial ve Bitkeeper bu sistemlere örnek gösterilebilir.

Git

Özellikleri: Dağıtılmış depo(repository) modeli sunar. HTTP, FTP ve ssh gibi mevcut sistemler ve protokollerle uyumludur. Küçük ve büyük ölçekli projelerin verimli bir şekilde hazırlanmasına olanak sağlar. Kullanışlı merge stratejileri sunar. Araç kitlerini kullanıma sunar. Çapraz platform desteği sunma. Kod değişiklikleri kolay takip edilebilir dolayısıyla verimlilik sağlar. Git bash olarak bilinen yardımcı programı sunar. Farklı eylemleri yapabilmeye olanak sağlayan GIT GUI sunar. Karmaşık ve daha geniş geçmiş günlüğünün anlaşılması zorlaşır. Anahtar kelime genişletmeyi ve zaman damgası korumasını desteklemez.

CVS

Özellikleri: İstemci-sunucu mimarisiyle çalışır. Birden çok geliştirici aynı proje üzerinde paralel olarak çalışabilir. CVS istemcisi dosyanın çalışan kopyasını güncel tutar ve yalnızca bir çakışma meydana gelirse manuel müdahale gereklidir. Projesinin tarihsel görüntüsünü tutar. Anonim okuma erişimi sağlar. Bir projenin farklı dallarını destekleyebilir. Verimli depolama için delta sıkıştırma tekniğini kullanır. Çapraz platform desteği sunar. Sağlam komut satırı istemcisi güçlü komut dosyası oluşturmaya olanak sağlar. Kaynak kod deposunda geniş web taramasına izin verir. İyi bilinen ve anlaşılan bir araçtır. Açık kaynak dünyasının işbirlikçi doğasına uyar. Kaynak kod deposu için bütünlük denetimi yoktur. Atomik kontrolleri ve taahhütleri desteklemez. Dağıtılmış kaynak desteği için zayıf destek sunmaktadır.

SVN

Özellikleri: İstemci-sunucu veri havuzu modeli sunar. Dizinlerde versiyonlanabilir. Kopyalama, silme, taşıma ve yeniden adlandırma işlemleri de versiyonlanmıştır. Atomik comitleri destekler. Serbest biçimli sürümlü meta verileri destekler. Birleştirme izleme, tam MIME desteği, yola dayalı yetkilendirme, dosya kilitleme, bağımsız sunucu çalışması gibi bir çok özelliği destekler. Tortoise SVN gibi GUI araçlarının sahiptir. Boş dizinleri destekler. Git’e kıyasla daha iyi Windows desteğine sahiptir. Kurulumu ve yönetimi kolaydır. Dosyaların değişiklik zamanını saklamaz. Dosya adı normalleştirme ile iyi ilgilenmez. İmzalı revizyonları desteklemez.

Git Kullanımı

Yapılandırma: Kimliği git sistemine tanıtmak yapılması gerek ilk işlemlerdendir. Bu yapılan değişikliklerin kimin tarafından yapıldığını belirtir ve önemlidir. Yapılandırmayı git bash üzerinde git config --global user.name “” (tırnaklar içinde kullanıcının yani dosyayı değiştirenin adı soyadı olmalı) ve e posta adresi için yine git config --global user.email “” yoluyla e posta adresimizi de belli etmiş olduk. Sadece git config --global user.name veya user.email diyerek girilen ad-soyad ve e posta adreslerini görmüş oluruz. ls komutu bulunduğumuz dizindeki dosyaları ve diğer dizinleri listeleyerek gösterir. (cd ……..) cd yazıp yanına çalışmak istediğimiz dizini yazarak o dizine geçiş yapabiliriz. Ardından o dizinde git init komutunu kullanarak projeyi bir git projesi haline getirebiliriz. Git dosyaları gizli dosya formunda yüklendiğinden ls komutu ile kontrol etsekte boş gözükecektir. Gizli dosyaları listelemek için ls -a komutunu kullanmamız lazım. .git klasörünü bu şekilde görebiliriz. Bu klasör git deposu olarak da adlandırılabilir. Dosya hazırlandıktan sonra depoya ekleme yapmak için git add . komutu ile bütün dosyalar geçiş bölgesine aktarılır eğer belli bir dosyayı ya da dizini aktarmak istiyorsak git add …..(dosya adı veya dizin) komutunu kullanmalıyız. Geçiş bölgesinden git deposuna aktarmak içinse git commit -m “” (tırnaklar arasında versiyonun açıklaması yazılmalıdır.) git log komutu ile şimdiye kadar depoya aktarılmış dosyaları listeler. git status çalışma dizini ile depo arasında farklılık olup olmadığını kontrol ederek geçiş bölgesinde kalan, silinmek istenen ama silinmeyen dosyalar hakkındaki bilgileri gösterir. Eğer dosyalarda bir değişiklik yapılırsa git status ile dosyaların durumunu incelediğimizde dosyayı modifiye edilmiş dosya olarak tanımlayacaktır. git diff komutu ile değişiklik ekleme tarzındaysa eklenen kısmın başına ve sonuna + işareti koyarak eklenen kısmı belirtir. Dosyadan çıkarma işlemi yapılmışsa yani kod blokları silinmişse - işareti ile gösterir. Değiştirilen dosyayı tekrardan git add …..(dosya adı) komutu ile geçiş bölgesine ardından git commit -m “” ile de git deposuna aktarıyoruz. git diff çalışılan dizin ile git deposu arasındaki farklılıkları gösterir, geçiş bölgesi ve git deposu arasındaki farklılıkları gösterebilmek için git diff --staged komutu kullanılır. Dosya silmek için git rm ……(dosya adı) yazılarak istenilen dosya silinebilir ya da git rm -r ……/(dizin adı) yazılarak dizini ve içindekileri siler. Bu işlemlerin ardından commit işlemini tekrarlamak gereklidir. Dosyaların ismini değiştirmek için git mv …..(eski ad) …..(yeni ad) komutunu kullanabilirsiniz. Çalışma dizininde herhangi bir dosyadaki değişimi geri almak için git checkout -- ….(dosya adı) komutundan yararlanılabilir. Geçiş bölgesindeki bir dosyadaki değişiklikleri geri almak için git reset HEAD …...(dosya adı) ardından git checkout --(dosya adı) komutları girilmelidir.