Sürüm Kontrol Sistemleri (Git) Nedir?

Çalışma Kağıdı

Sürüm Kontrol Sistemleri (Git), yazılım projelerindeki kod değişikliklerini kaydeden ve geçmiş sürümlere kolayca dönülmesini sağlayan bir sistemdir. Git, dağıtık yapısı sayesinde geliştiricilerin çevrimdışı çalışmasına ve değişiklikleri senkronize etmesine olanak tanır.

Sorular

1. Aşağıdakilerden hangisi Git'in temel amacıdır?

A) Sadece kodları yedeklemek
B) Yazılım geliştirme sürecinde değişiklikleri takip etmek, sürümleri yönetmek ve işbirliğini sağlamak
C) Proje belgelerini otomatik olarak oluşturmak
D) Sunucu güvenliğini sağlamak

2. Bir geliştirici, uzak depodaki en son değişiklikleri yerel deposuna çekmek için hangi komutu kullanır?

A) git push
B) git commit
C) git pull
D) git clone

3. `git commit -m "Mesajınız"` komutu ne yapar?

A) Dosyaları uzak depoya gönderir.
B) Yeni bir dosya oluşturur.
C) Hazırlık alanındaki değişiklikleri yerel depoya kaydeder ve açıklayıcı bir mesaj ekler.
D) Depoyu klonlar.

4. Git ile bir projeye yeni başlayan bir geliştirici hangi temel adımları izlemelidir?

5. Farklı geliştiricilerin aynı projede çalışırken Git ile çakışmaları (conflict) nasıl yönetebilir?

6. Bir geliştirici, yaptığı son commit'i geri almak isterse ne yapmalıdır?

7. Tanımla: Git nedir?

8. Tanımla: `git init` komutu ne işe yarar?

9. Tanımla: `git add` komutu ne işe yarar?

Cevap Anahtarı

1. B) Yazılım geliştirme sürecinde değişiklikleri takip etmek, sürümleri yönetmek ve işbirliğini sağlamak — Git'in temel amacı, kod değişikliklerini kaydetmek, geçmiş sürümlere erişim sağlamak ve birden fazla geliştiricinin aynı proje üzerinde verimli bir şekilde çalışmasını mümkün kılmaktır.

2. C) git pull — `git pull` komutu, uzak depodaki (genellikle `origin` olarak adlandırılır) en son değişiklikleri yerel depoya indirir ve mevcut çalışma dalı ile birleştirir.

3. C) Hazırlık alanındaki değişiklikleri yerel depoya kaydeder ve açıklayıcı bir mesaj ekler. — Bu komut, hazırlık alanındaki (staging area) tüm değişiklikleri yerel Git deposuna kalıcı olarak kaydeder. `-m` parametresi, commit için bir mesaj belirtilmesini sağlar, bu mesaj değişikliğin ne hakkında olduğunu açıklar.

4. 1. **Git Kurulumu:** Bilgisayarınıza Git'i kurun. 2. **Depo Oluşturma/Klonlama:** Yeni bir proje için `git init` ile depo başlatın veya mevcut bir depoyu `git clone <url>` ile klonlayın. 3. **Dosya Ekleme:** Değişiklikleri takip etmek istediğiniz dosyaları `git add <dosya_adı>` ile hazırlık alanına ekleyin. 4. **Değişiklikleri Kaydetme:** Hazırlanan değişiklikleri `git commit -m "Mesajınız"` ile yerel depoya kaydedin. 5. **Uzak Depo ile Senkronizasyon:** Değişiklikleri uzak depoya (`git push`) göndermeden veya oradan çekmeden (`git pull`) önce yerel değişikliklerinizi gözden geçirin.

5. 1. **Güncel Depoyu Çekme:** Çalışmaya başlamadan önce uzak depodaki en son değişiklikleri `git pull` komutu ile yerel deponuza çekin. 2. **Çakışma Durumu:** Eğer sizin yaptığınız değişiklikler ile çektiğiniz değişiklikler aynı dosyanın aynı satırlarında ise Git otomatik olarak birleştirme işlemini durdurur ve çakışma (conflict) olduğunu belirtir. 3. **Çakışmaları Çözme:** Çakışan dosyaları açın. Git, çakışan kısımları özel işaretlerle (`<<<<<<<`, `=======`, `>>>>>>>`) belirtir. Bu işaretleri silerek ve kodunuzu istediğiniz gibi düzenleyerek çakışmayı manuel olarak çözün. 4. **Çözülen Dosyaları Ekleme ve Commit:** Çakışmayı çözdüğünüz dosyaları `git add <dosya_adı>` ile hazırlık alanına ekleyin ve ardından `git commit` komutu ile birleştirme işlemini tamamlayın.

6. 1. **Yerel Commit'i Geri Alma:** Eğer commit henüz uzak depoya gönderilmediyse, `git reset --hard HEAD~1` komutu ile son commit'i geri alabilirsiniz. Bu komut, son commit'i hem commit geçmişinden siler hem de yapılan değişiklikleri çalışma dizininden kaldırır. 2. **Sadece Commit Mesajını Değiştirme:** Eğer commit'i geri almak yerine sadece commit mesajını değiştirmek istiyorsanız `git commit --amend` komutunu kullanabilirsiniz. 3. **Uzak Depoya Gönderilmiş Commit'i Geri Alma (Dikkatli Olun!):** Eğer commit uzak depoya gönderilmişse, `git revert HEAD` komutu ile geri almak istediğiniz commit'in tam tersini yapan yeni bir commit oluşturabilirsiniz. Bu, geçmişi değiştirmek yerine yeni bir değişiklik ekleyerek güvenli bir geri alma yöntemidir. `git push` ile bu değişikliği uzak depoya göndermeniz gerekir.

7. Yazılım projelerindeki kod değişikliklerini takip eden, sürümleri yöneten ve işbirliğini kolaylaştıran dağıtık bir sürüm kontrol sistemidir.

8. Mevcut bir dizinde yeni bir Git deposu (repository) başlatır.

9. Yapılan değişiklikleri hazırlık alanına (staging area) ekleyerek bir sonraki commit için işaretler.