**Branching İş Akışları**

Nasıl kullanıldıklarına bağlı olarak branch'leri iki ana grup altında toplayabiliriz.

Bu gruplama sadece anlamsal ve kullanım pratikleri ile ilgili bir gruplandırmadır, sonuç itibariyle branch kavramı daha önceki bölümlerde anlattığımız kadar basit bir Git aracıdır

**Kısa Vadeli / Konu Bazlı Branch'ler**

Daha önceki bölümlerde branch kullanımı noktasında elinizi korkak alıştırmamanız ile ilgili tavsiyelerde bulunduk. Örneğin yeni özellikleri kodlarken, bug fix yaparken veya deneysel özellikler ile ilgili çalışırken istediğiniz şekilde kolayca ve hızlı bir şekilde üstelik düşük maliyetli branch'ler oluşturabilirsiniz. Bu tür amaçlar için oluşturulan branch'lerin iki ortak özelliği vardır

* Bu branch'ler tek konu veya değişiklik için oluşturulur. Örneğin size bildirilen bir hata için oluşturduğunuz branch üzerinde "GitHub İle Sisteme Giriş" benzeri yeni bir özelliği kodlamayız.
* Bu branch'ler üzerindeki çalışmanız göreceli kısa sürmektedir. Çalışmamız tamamlandığında bu branch'leri **master** veya daha geniş kapsamda tarif edilen bir branch'e merge edip sileriz.

**Uzun Soluklu Branch'ler**

İkinci türdeki branch'ler ise daha üst seviyede anlam taşırlar ve yeni özellikler, bug fix ve deneysel çalışmalar gibi odaklanmış konular yerine projenizi stabil, test ve development gibi aşamalarını temsil ederler. Bu tür branch’ler projeniz üzerinde geliştirme yaptığınız sürece varlıklarını sürdüreceklerdir. Tipik olarak bu tür branch'ler ile ilgili aşağıdaki kurallar geçerlidir

* Genelde bu tür uzun soluklu branch’ler üzerinde doğrudan değişiklik yapmazsınız. Çalışmalarınızı kısa vadeli branch’ler üzerinde yaparak değişiklikleri bu branch'lere entegre edersiniz.
* Uzun soluklu branch'ler arasında bir hiyerarşi vardır. Genellikle **master** branch projenizin stabil versiyonudur ve hiyerarşik olarak bir altında geliştirmelerinizi entegre ettiğiniz ve daha az stabil olabilen **development** branch'i yer alır.

Uzun soluklu branch'lerin hangi kriterlere göre oluşturulacağı, nasıl yönetileceği ve isimlerinin ne olacağı genellikle çalışan ekibe ve projeye göre değişebilir. Ancak her hâlükârda nasıl bir branching stratejisinin izleneceğine ekip olarak fikir birliği içinde karar verilmelidir.

**Basit ve Faydalı Branching Stratejileri**

Yukarıda da belirttiğimiz gibi branch'in stratejileri ekibe ve projeye göre değişebilir, ancak aşağıda çoğu ekip tarafından kullanılabilecek basit bir iş akışı kullanabilirsiniz

**Sadece bir tane uzun soluklu branch kullanın**

Daha önce de belirttiğimiz gibi birden fazla uzun soluklu branch kullanabilirsiniz ancak çoğu zaman bu tip bir yaklaşım karışıklıklara ve fazladan efor sarf etmek gibi zorluklara sebep olabilir. Tek bir uzun soluklu branch kullanmanız durumunda (genelde **master** ismi kullanılır) işiniz önemli miktarda sadeleşip kolaylaşacaktır.

Bu yaklaşım ile çalışmanız durumunda **master** branch'iniz projenizin stabil kodunu barındırmalıdır. Kodunuzun stabil olmasını garantilemek için **master** branch'e entegre edilen (merge) tüm değişikliklerin testler, kod okuma vs gibi kalite kontrol yöntemleri ile denetlenmesi gerekecektir. Bunun bir yansıması olarak değişikliklerin doğrudan **master** branch üzerinde yapılmaması gibi bir zorunluluk da doğacaktır. Eğer *git checkout master* ve sonrasında *git commit* komutlarını çalıştırıyorsanız bilin ki stabilite kuralını ihlal ediyorsunuz.

**Konu Bazlı Branchler'i Bolca Kullanabilirsiniz**

Projeniz için yeni bir özellik üzerinde çalışmak için, bug fix yapmak için veya deneysel özellikleri ve iyileştirmeleri kodlamak için ayrı birer branch oluşturmaktan ve bu branch'ler üzerinde değişikliklerinizi yapmaktan imtina etmeyin. Bu yaklaşım nerdeyse tüm branching iş akışlarında çok sık kullanılan ve sizin de alışkanlık haline getirmeniz gereken bir yaklaşımdır.

Sadece bir tane uzun soluklu ve stabil branch'iniz (master) olduğu için konu bazlı branch’lerinizin hepsini bu ana branch'i baz alarak oluşturup değişikliklerinizi tamamlayıp kalite kontrol sürecinizi (testler, kod okuma vs) de işlettikten sonra bu değişiklikleri tekrar ana branch'iniz olan master'a entegre etmelisiniz.

Diğer yandan siz kendi konu bazlı branch'inizde değişiklikleri yaparken ekip arkadaşlarınız da arada kendi değişikliklerini master branch'e entegre ediyor olacaklarıdır. Bu durumda da kendi branch'inizi master branch'deki değişiklikler nedeniyle güncel tutmak için master'daki değişiklikleri de kendi konu bazlı branch'inize sıkça entegre etmelisiniz.

Bu basit akışta unutmamanız gereken tek bir altın kural var; değişikliklerinizi kalite kontrol süreçlerinizi işletmeden ana branch'iniz olan ve her zaman stabil olması gereken master'a entegre etmeyin aksi durumda master branch'inizin stabilitesini bozabilirsiniz.

**Remote ve Yerel Branch'lerinizi Senkronize Edin**

Git'de remote ve yerel branch'leriniz pratik olarak birbirinden tamamen bağımsızdırlar. Ancak gündelik çalışmanız sırasında kendi bilgisayarınızda oluşturduğunuz branch'lerin uzaktaki sunucudaki eşleniğinin de olmasını sağlamalısınız.

**Değişikliklerinizi Sıkça Remote Branch'lere Yükleyin (Push)**

Remote branch’ler ile yerel branch’leri sadece yapısal olarak değil yaptığınız değişiklikler anlamında da senkronize etmelisiniz. Bu şekilde ekibinizin geri kalanı da sizin yaptığınız güncel değişikliklerden haberdar olacak ilave olarak yerel branch'lerinizi yedeğini almış olacaksınız.

**Diğer Branching Stratejileri**

Bu bölümde bahsettiğimiz basit stratejiler ve iş akışları genelde küçük ve çevik (agile) takımlar tarafından kullanıma uygundur. Daha büyük projelerde ve farklı takım kurgularında daha sıkı kurallar ve daha farklı branching yaklaşımlarının kullanımını gerektirebilir.

Gitflow, Forking ve Pull Request adı verilen alternatif iş akışları ile ilgili arama yaparak farklı yaklaşımları kendiniz inceleyebilirsiniz