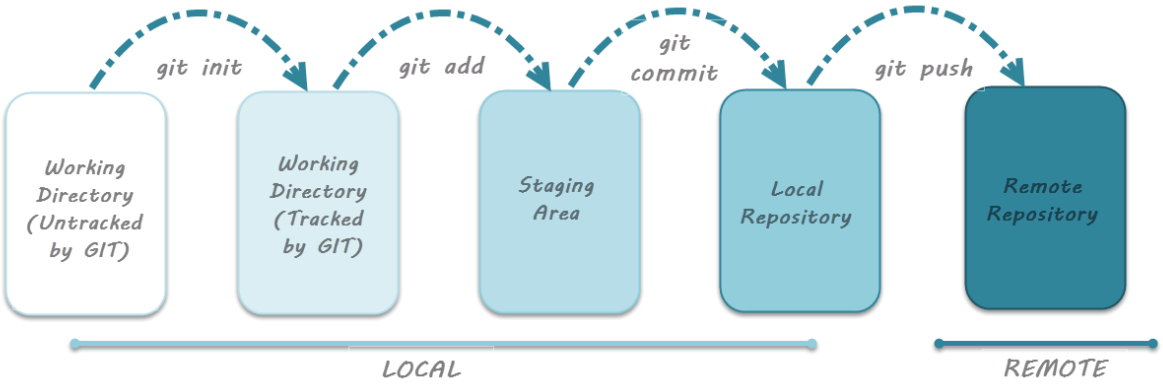
git config

- **git config --global user.email “x@gmail.com” | git config --global user.name “ali”** git cli üzerinde kendinizi tanıtırsınız yapılması gereken bir işlemdir. Atlanamaz.

Local Repo ve Remote Repo Nedir?

- **Local Repo bilgisayar üzerinde .git dosyasına sahip olan klasörler için kullanılan tanımdır. Bu iki şekilde olabilir. Remote Repo’dan klon alınmış olabilir ya da herhangi bir çalışma klasörü ‘git init’ komutu ile tracked aşamasına getirilmiştir.**
- **Remote Repo internet üzerinde Github, Gitlab vs. gibi uygulamalarda tutulan repolar için kullanılır.**

Repo Aşamaları



- **Klasör eğer sadece pc üzerinde açık ve üzerinde git namına hiçbir şey yok ise *untracked* konumdadır.**
- **Klasöre ‘git init’ ile bir git tanımlaması yapılırsa bu artık *tracked* konuma gelir.**
- **Klasör üzerinde ki bir sonraki işlem “git add .” işlemidir. Bu işlemden sonra çalışma klasörümüz *staged/staging* konuma gelir.**
- **Klasörün local bir repo olması için commit atılması gerekir. “git commit –m “first ‘commit’ ”. Artık elimizde local reponun son hali vardır.**
- **Push işlemi sonrası repomuz artık remote repo olarak internette yerini alır.**

git clone

- **“git clone [URI]” komutu ile remote bir repoyu bilgisayarımıza çekebiliriz ve klonlanan bu repo aynı zamanda bizim local repomuzdur artık.**

Remote Repo Güncelleme

- **Local repo adresine gidillir ve komutlar sırasıyla girilir.**
  git add .
  git commit –m “commit example”
  git push –u origin [branch_name]

Branch Nedir?

- **Branch, oluşturulup geliştirilmiş projeye yeni versiyon eklenmek istendiğinde veya büyük bir değişiklik uygulanacağı zaman kullanılır. Her bir branch farklı bir versiyon olarak görülebilir.**
  git branch -> **Local Repo branchlerini listeler.**
  git branch –a -> **Local ve Remote bütün branchleri listeler.**
  git branch [branch_name] -> **Yeni branch ekler.**
  git branch –d [branch_name] -> **Local Branch Siler.**
  git push origin --delete [branch_name] -> **Remote Repoyu siler.**
  git switch [branch_name] -> **Branchler arası geçiş yapar. (Yaygın ve yeni)**
  git checkout -b [branch_name] -> **Yeni branch oluşturup o branche geçer.**

  NOT: Oluşturulan branch ilk başta local branchtir. Üzerinde yapılan değişiklik sonrası push edilirse hem local hem remote branch olur.

```
C:\Users\ahmet\Desktop\repositories\git test>git branch --all
  main
  v2
* v3
  remotes/origin/main
  remotes/origin/v2
```

  Görüldüğü gibi main ve v2 üzerinde işlem yapılıp push edilmiş ama v3 branchi push edilmemiş ve local branch olarak kalmış.

git log

- **Repo üzerindeki işlemlerin kaydını tutar.**

.gitignore Dosyası

- **Uzak sunucuya atılmamasını istediğimi dosyalar için oluşturulur ve her projenin .gitignore dosyası internette mevcuttur, yazmaya gerek yoktur.**

Commit İşlemleri

- **git commit --amend -m "new commit" -> Son commit mesajı düzenlenir.**
- **git revert [commit_id] -> Yeni commit ile son commiti geri alır.**
- **git diff [commit_id1]..[commit_id2] -> 2 commiti eklenen eksilen olarak kıyaslar.**
- **git diff [commit_id1]..[commit_id2] x.js -> 2 commiti belirtililen dosya için kıyaslar.**

git stash

- **Bu işlemi şöyle düşünelim bir branchteyiz ve proje geliştiriyoruz ama proje commit atılacak seviyede değil fakat bizim işimizi bırakıp farklı branche geçmemiz lazım. İşte bu noktada stash işlemini gerçekleştiriyoruz. Bunu bir ara belleğe saklamak olarak düşünebilirsin. Şimdi işlemlere bakalım.**
- **Staged konumdaki klasörlere yapılabilir. Klasör içinde "git stash" demek yeterli.**
  git stash list -> Stash olan verileri listeler.
  git stash pop -> En son stash verisini alır.
  git stash apply stash@{2} -> 2 nolu indexe sahip stashı alırız.
  git stash clear -> Bütün stashler silinir.

git restore

- **git restore index.html**
- **Bu komutu üzerinde değişiklik yaptığınız ama geri almak istediğiniz durumlarda kullanırsınız.**
- **Kullanıldığı takdirde sizi en son commite götürür.**

Istenen commite gitme

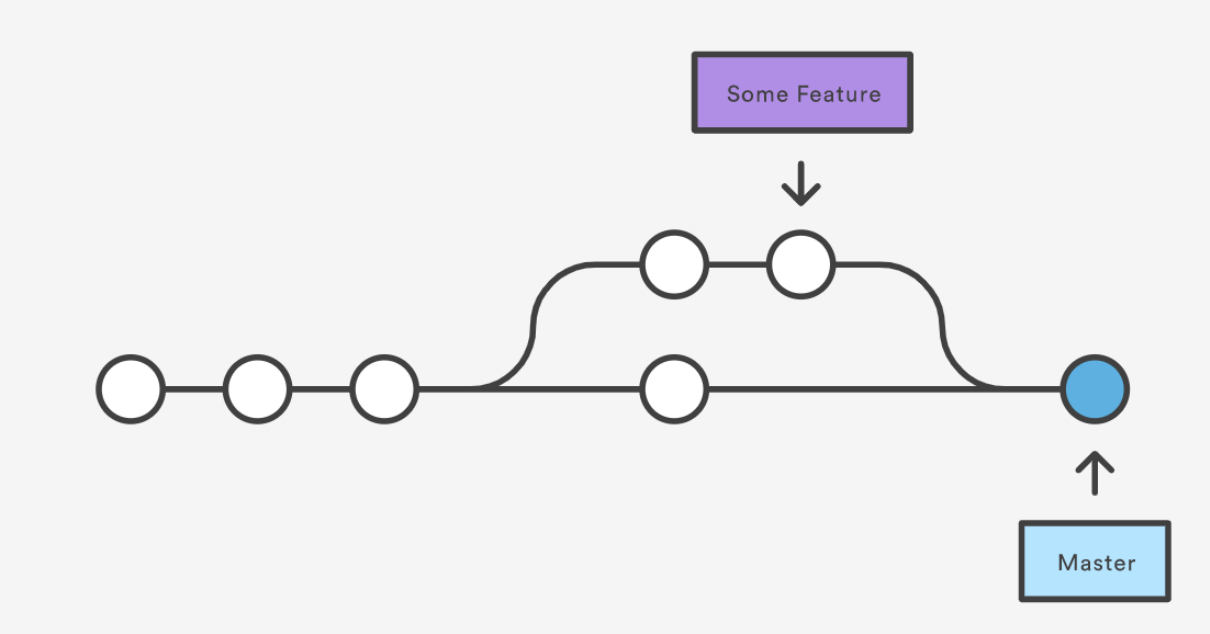
- **Git checkout [commit_id]**
- **Bu komut ile proje belirtilen committe ki haline döner.**

git pull

- **Diyelim ki bizim remote repomuzun master branchinde değişiklik var ve localimiz bundan habersiz bu durumda bu komut işimize yarar.**
- **Bunun alternatifi olarak "git fetch origin" ve "git merge origin/[branch_name]" yapılabilir. "git pull origin [branch_name]" komutu bu komutların ikisini de yapar. git fetch [a], a branch'inin bilgilerini çeker. git merge ise birleştirir.**

git merge

- **Master bizim her zaman ana branchimizdir ve son hal her zaman orada olmalıdır. Bu durumda diğer branchlerde gerçekleşenler mutlaka master'a dönmelidir, veya birleşmelidir.**



- **Görselde gördüğünüz master branchinin son halini alması için bir noktada birleşme gereklidir.**
- **master branchinin içinde olduğumuzu varsayarsak "git merge [xBranch]" yaparız ve bu durumda master branchi artık xBranchi ile birleşmiş ve master branch adı altında yer alır ama xBranchi kendi branchinde aynı olarak kalır.**

Fast Forwarding

- **Merge işlemleri yapılırken aynı yere farklı şeyler yazılmış olabilir, bir tarafta olan dosya diğer tarafta silinmiş olabilir bu durumda *merge conflict* olur yani git ortada bir tutarsızlık olduğunu hangisinin doğru olduğunu bilemez. Bunu engellemek için fast forwarding yapabiliriz yani branch oluşturup geliştiririz ve birleştirme anına kadar mastera hiçbir şey eklemeyiz bu durumda tutarsızlık oluşmaz.**

REBASE KONUSU EKSIKTIR.