



Git & GitHub

İ. H. Parlak

15.05.2024

BAİBÜ B.M.

Git

- <https://git-scm.com/>
- Git, projeleri hızlı ve verimli bir şekilde ele almak için tasarlanmış, ücretsiz ve açık kaynaklı bir **versiyon kontrol sistemidir**.



Git

- Git, lokal bilgisayarımıza kurduğumuz versiyon kontrol sistemidir.
- Kod* geçmişimizi yönetmeye yarar.
- Kodda zaman içinde yapılan değişiklikleri takip edebilmemizi sağlar.
- Sadece bilgisayarın kullanıcısı (biz) Git ile projemizi takip edebiliriz: başkası bilgisayarımızdaki projeye erişemez.



GitHub

- <https://github.com/>
- Dünyanın en büyük yazılım geliştirme platformu.
- Ortaklaşa (bir takım halinde) proje geliştirmek için kullanırız.
- *Git repository hosting.*
- Repository (Repo - Depo): Projemizin kaynaklarını barındıran alan.



Git Komutları (1)

- `git init`: çalışma dizininde git'i başlatır (initialize).
- `git config --global user.email "a@b.com"`
- `git config --global user.name "ismail"`
- `git status`: mevcut git durumunu gösterir.
- `git add dosya1 dosya2`: dosya1 ve dosya2'yi stage alanına ekler.
- `git add .`: tüm değişiklik bulunan dosyaları stage alanına ekler.
- `git commit -m "bir mesaj"`: değişiklikleri stage alanından repo alanına gönderir.

Git Komutları (2)

- `git log`: git log'larını gösterir. *Log ekranından çıkmak için: q*
- `git checkout commit_id`: id'sini girdiğimiz commit'e gideriz.
- `git stash`: commit yapmadan önce üzerinde çalıştığımız dosyaları saklamak istersek.
- `git stash push -m "mesaj"`: değişiklikleri stash'e atarken mesaj yazabiliriz.
- `git stash apply`: stash'teki son durumu mevcut ekrana getirir.
- `git stash list`: stash'leri listeler.
- `git stash apply X`: X'inci stash'i ekrana getirir.
- `git stash drop X`: X'inci stash'i siler.
- `git stash clear`: tüm stash'leri siler.

Branch Kavramı

- Git'te bir projeyi başlattığımızda (git init), Git ön tanımlı olarak master (main) branch (dal)'ında projeyi izlemeye (tracking) başlar.
- Projemizde yeni dal'lar açabilir; bu yeni dallarda proje için yeni özellikler geliştirebilir; istediğimizde yeni oluşturduğumuz bu dalları ana dala (main/master) birleştirebiliriz (merge).
- `git branch`: tüm branch'leri listeler.
- `git branch branch_ismi: branch_ismi` adında yeni branch oluşturur.

Branch Kavramı

- `git checkout branch_ismi`: `branch_ismi` adlı branch'e geçiş yaparız.
- `git checkout -b yeni_branch`: `yeni_branch` isimli branch'i oluşturur ve ona geçiş yaparız.
- `git branch -m eski_ad yeni_ad`: içinde bulunduğumuz branch'ın adını `yeni_ad` olarak değiştirir.
- Git v2.23 ile birlikte sadece branch'ler arası geçiş işlemleri için "switch" komutu kullanıma sunulmuştur.
- `git switch branch_ismi`: `branch_ismi` adlı branch'e geçiş yaparız.
- `git switch -c yeni_branch`: `yeni_branch` isimli branch'i oluşturur ve ona geçiş yaparız.

Merge

- Bir branch'ı başka bir branch ile birleştirmeye merge denir.
- `git merge branch_ismi:branch_ismi` adlı branch'i, şu an bulunduğumuz branch'e birleştirir.

Dosya Silme

- `git ls-files`: aktif branch'te track edilen dosyaları listeler.
- Herhangi bir dosyayı çalışma dizinimizden sildiğimizde Git'ten silinmez.
- `git add/rm dosya_adi`: dosya_adi isimli dosyayı çalışma ağacı ve index'ten çıkarır.

Commit Silme

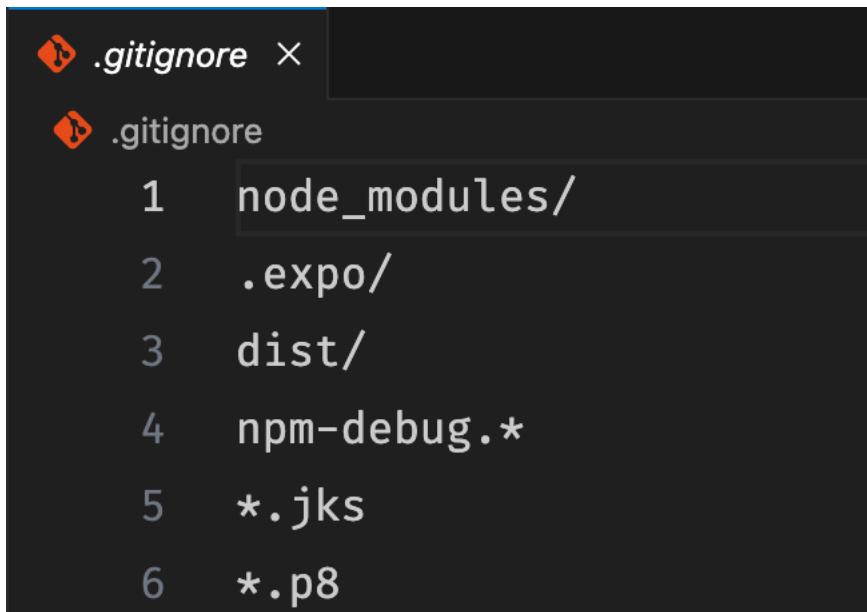
- `git reset --hard HEAD~1`: HEAD'i 1 commit geriye alır.

Branch Silme

- `git branch -d branch_ismi`: `branch_ismi` adlı branch merge edildiyse silinir.
- `git branch -D branch_ismi`: `branch_ismi` adlı branch merge edilmediyse de silinir.

.gitignore

- Git tarafından izlenmesini istemediğimiz dosyaları, örn: kayıtlı şifreler, IDE tarafından oluşturulan dosyalar, işletim sistemi dosyaları, vb., ".gitignore" dosyasına ekleyebiliriz.

A screenshot of a code editor showing a file named ".gitignore". The file contains six lines of text, each preceded by a line number from 1 to 6. The text on the lines is: "node_modules/", ".expo/", "dist/", "npm-debug.*", "*.jks", and "*.p8". The editor has a dark theme and a tab at the top labeled ".gitignore".

```
.gitignore
1 node_modules/
2 .expo/
3 dist/
4 npm-debug.*
5 *.jks
6 *.p8
```

Merge Conflict

- Bir ana branch'tan yeni bir çocuk branch oluşturulduktan sonra, ana branch ve çocuk branch'te farklı değişiklikler yapıp, çocuk branch tekrar ana brach'e birleştirilmeye çalışılırsa merge conflict (birleştirme uyuşmazlığı) ortaya çıkar.
- `git merge --abort`: merge işlemini iptal eder.

GitHub Repository

- `git remote add origin URL`: URL adresindeki repo'yu lokaldeki projeye bağlar.
- `git branch -M main`: lokalimizdeki ana branch'in adını main yapar.
- `git push -u origin main`: main (lokal) branch'i origin'e (uzak branch) push eder.
- `git push --set-upstream origin yeni_branch`: lokaldeki yeni_branch içeriğini uzağa yeni_branch oluşturarak push eder.
- `git push -u origin lokal_bir_branch`: origin'de local_bir_branch adında yeni bir branch oluşturup push yapar.
- `git ls-remote`: GitHub'daki branch'leri listeler.

GitHub Repository

- `git clone URL:URL` adresindeki repo'yu lokale klonlar.
- `git push --set-upstream origin test:origin`'de test adında yeni bir branch açıp lokal test'i origin/test'e push eder.
- `git branch --track branch_ismi remotes/branch_ismi`: Uzaktaki branch_ismi branch'i için lokalde branch_ismi adında tracking branch oluşturur.
- `git push origin --delete branch_ismi`: Uzaktaki branch_ismi branch'ini siler.
- `git pull origin master`: Origin'i, master'a çeker.

Fork & Pull Request

- GitHub'daki open source projelere - proje ekibine dahil olmadan - katkıda bulunmak için **fork** ile projeyi kendimize kopyalayabiliriz.
- Fork edilmiş proje üstünde çalışmalarımızı tamamlayınca, asıl projenin sahibine merge işlemini yapması için **pull request** gönderebiliriz.

Ekstra

- <https://git-scm.com/downloads/guis/>
- .env dosyaları
- read me dosyası
- merge çeşitleri
- rebase
- cherry-pick
- local / remote tracking branches
- GitLab, Beanstalk, Bitbucket

O Zaman...

