Table of Contents

Git Cheat Sheet

1. Başlangıç ve Yapılandırma (Kurulum)

git config –global user.name “Adınız Soyadınız”	Git'e kullanıcı adını tanımlar. –global etiketini kullanmazsan sadece ilgili repoda kullanıcı adını değşiştirir.
git config –global user.email “mail@adresiniz.com”	Git'e e-postanızı tanımlar. –global etiketini kullanmazsan sadece ilgili repoda kullanıcı adını değşiştirir.
git config –global init.defaultBranch main	Varsayılan ana dalı **main** olarak ayarlar.
git config –global merge.tool meld	Çakışmaları (conflict) çözmek için varsayılan bir araç (örneğin Meld) belirler.
git config –list	O anki mevcut tüm Git ayarlarınızı listeler.
git init	İligli klasörü, Git'in takip ettiği bir depoya (repository) dönüştürür.
git clone <url>	Uzak bir sunucudaki (GitHub) mevcut bir projeyi bilgisayarınıza indirir.

2. Temel İş Akışı (Değişiklikleri Kaydetme ve İnceleme)

git status	Projenin o anki durumunu gösterir. (En çok kullanacağın komut).
git diff	Hazırlık alanına (staging area) eklenmemiş tüm değişiklikleri (satır satır) gösterir.
git diff <dosya_adi>	Sadece o dosyadaki değişiklikleri gösterir.
git add <dosya1> <dosya2>	Belirtilen dosyaları hazırlık alanına ekler.
git add .	Klasördeki tüm değişiklikleri hazırlık alanına ekler.
git add -p <dosya_adi>	Değişiklikleri parça parça (interactive) inceleyerek eklemeni sağlar.
git commit	Sadece bu komutu yazarsan detaylı mesaj yazman için varsayılan terminal editörü açılır.
git commit -m “Mesaj”	Değişiklikleri kısa bir mesajla tarihçeye kaydeder.
git commit -a	Değişen tüm dosyaları (yeni oluşturulanlar hariç) otomatik add yapıp commit ekranını açar.
git commit -am “Mesaj”	Hem tüm değişenleri otomatik ekler hem de mesajla kaydeder (add . ve commit -m birleşimi).
git mv <eski_ad> <yeni_ad>	Dosyanın ismini değiştirir veya taşır (bunu Git'e bildirerek yapar, eğer sen kendin dosyanın ismini değiştirirsen git ilgili dosya silinmiş sanar ve yeni bir dosya oluşmuş sanar.)

3. Uzak Depolarla Çalışmak (GitHub)

git remote add origin <url>	Yerel deponuzu GitHub'daki bir depoya bağlar.
git push -u origin main	Kodlarınızı GitHub'a gönderir (İlk seferde -u kullanılır).
git pull	GitHub'daki güncel kodları indirir ve mevcut kodunuzla birleştirir.

git pull –rebase <remote> <branch>	Çekerken gereksiz merge commit'i oluşturmamak için güncellemeleri rebase ile alır (Örn: `git pull –rebase origin main`).
git fetch	Uzak depodaki değişiklikleri bilgisayara indirir ama birleştirmez.

4. Gelişmiş Dallanma (Branching)

git branch	Yerel dalları listeler.
git branch -a	Hem yerel hem de uzak (remote) tüm dalları listeler.
git branch -r	Sadece uzak depodaki dalları listeler.
git branch –merged	Aktif dala başarıyla birleştirilmiş (merge edilmiş) dalları listeler.
git branch <yeni_dal>	Yeni bir dal oluşturur ama o dala geçmez.
git branch -m <yeni_isim>	Bulunduğun dalın adını değiştirir.
git branch -d <dal_adi>	İşi bitmiş ve birleştirilmiş dalı güvenle siler.
git branch -D <dal_adi>	Dalı birleştirilmemiş olsa bile zorla siler.
git branch –track <yeni_dal> <uzak_dal>	Uzak depodaki bir dalı takip eden yerel dal oluşturur.
git checkout <dal_adi>	Başka bir dala geçiş yapar.
git checkout -	Bir önceki bulunduğun dala hızlıca geri döner (TV kumandasındaki “önceki kanal” tuşu gibi).
git checkout -b <dal_adi>	Hem yeni dal oluşturur hem de o dala anında geçiş yapar.
git checkout -b <yeni_dal> <mevcut_dal>	Mevcut bir daldan türeyen yeni bir dal oluşturup geçer.
git checkout <commit-hash> -b <yeni_dal>	Eski bir commit'in olduğu noktadan yeni bir dal başlatır.
git checkout <dal_adi> – <dosya_adi>	Başka bir daldaki belirli bir dosyayı, senin bulunduğun dala kopyalar.
git cherry-pick <commit_hash>	Başka bir daldaki tek bir commit'i (ve sadece onu) alıp senin bulunduğun dala uygular.

5. Birleştirme (Merge) & Çakışma (Conflict) Çözme

git merge <dal_adi>	Belirtilen dalı, bulunduğun dala birleştirir.
git mergetool	Birleşme sırasında çakışma (conflict) çıkarsa, yapılandırdığın görsel aracı (örneğin Meld) açarak çözmeni sağlar.
git rebase <dal_adi>	Senin dalının başlangıç noktasını, belirtilen dalın en güncel haline taşır. Doğrusal bir tarihçe sağlar.
git rebase –abort	Rebase sırasında çok fazla çakışma çıkarsa veya işler ters giderse işlemi tamamen iptal edip en başa döner.

- Çakışma (Conflict) çözüldükten sonra Rebase'e devam etme:
- Çakışan dosyaları düzelt.
    - `git add <cozulen_dosya>` (veya gerekmiyorsa `git rm <dosya>`)
    - `git rebase –continue` (Bu işlem bitene kadar tekrarlanır).

6. Geçmişi Düzenleme (Amend) ve Sıkıştırma (Squash)

A) Son Commit'i Değiştirme (Amend)

> Yanlış mesaj yazdın veya bir dosyayı eklemeyi unuttun. (Eğer henüz uzak depoya push etmediysen güvenlidir).

git commit –amend	Hazırlık alanındaki dosyaları son commit'in içine katar ve mesajı değiştirmen için editörü açar.
git commit -a –amend	Tüm değişen dosyaları otomatik ekleyip son commit'i günceller.
git commit –amend –no-edit	Mesajı değiştirmeden sadece unuttuğun dosyaları son commit'e dahil eder.
git commit –amend –date="date"	Son commit'in tarihini değiştirir.
GIT_COMMITTER_DATE="date" git commit –amend	Hem commit hem de committer tarihini geçmiş bir zamana alır. (Örn: `git commit –date="date –date='1 day ago'" -am "Mesaj"`)

B) Sıkıştırma (Interactive Rebase / Squash)

> Çok sayıda küçük commit'i tek, anlamlı bir commit haline getirme.

1. `git rebase -i <birlestirmek_istedigin_en_eski_commitin_bir_oncesi>`
2. Karşına çıkan editörde:

```
# Önceki Hali:
pick 1a2b3c4 İlk yapı
pick 5d6e7f8 Ufak düzeltme
pick 9g0h1i2 Renk ayarı

# Sonraki Hali (İkinci ve üçüncüyü ilkine yediriyoruz):
pick 1a2b3c4 İlk yapı
squash 5d6e7f8 Ufak düzeltme
squash 9g0h1i2 Renk ayarı
```

1. Kaydet ve çık. Yeni bir mesaj ekranı gelecek, birleştirilmiş commit için tek bir mesaj yaz.

7. İnceleme, Arama ve Loglar

git log	Detaylı commit geçmişini listeler.
git log –oneline	Her commit'i tek bir satırda, daha temiz gösterir.
git log –author="isim"	Sadece belirli bir kişinin yaptığı commit'leri gösterir.
git log -p <dosya_adi>	Belirli bir dosyanın geçmişte nasıl değiştiğini (satır satır diff olarak) gösterir.
git log –oneline <origin/master>..<remote/master> –left-right	İki farklı dal arasındaki farkları (kim kimin ilerisinde) oklarla listeler.

git log -S 'kelime'	İçinde belirli bir kelimenin eklenip/silindiği commit'leri arar.
git log -S 'kelime' –pickaxe-regex	Yukarıdaki aramayı Regex (Düzenli İfade) ile yapar.
git grep "Aranan"	Tüm projedeki dosyalar içinde "Aranan" metnini bulur.
git grep "Aranan" v2.5	Belirli bir sürümde veya dalda bu kelimeyi arar.
git blame <dosya_adi>	Bir dosyanın her bir satırını en son kimin, ne zaman değiştirdiğini gösterir. (Kodu kimin bozduğunu bulmak için efsanedir).
git reflog show	Git'teki yaptığın her hareketin (silinen dallar, resetlenen commitler dahil) günlüğünü tutar. Kaybettiğin her şeyi buradan bulabilirsin.
git reflog delete	Reflog geçmişini siler.

8. Etiketleme (Tagging) - Sürümlendirme

> Önemli anları (örneğin v1.0.0 sürümü) işaretlemek için kullanılır.

git tag	Mevcut tüm etiketleri listeler.
git tag -n	Etiketleri, yanlarındaki mesajlarıyla birlikte listeler.
git tag <etiket_adi>	Basit, hafif bir etiket oluşturur (Örn: `git tag v1.0`).
git tag -a <etiket_adi>	Ekstra bilgi eklenebilen anotasyonlu etiket oluşturur (Tavsiye edilen).
git tag <etiket_adi> -am 'Mesaj	Anotasyonlu etiketi doğrudan mesajla oluşturur.

9. Zula (Stash) Yönetimi

> Çalışıyorsun ama işin bitmedi. Başka dala geçmen gerektiğinde yarım kalan kodları saklama yöntemidir.

git stash	Mevcut değişiklikleri geçici bir "çekmeceye" (zulaya) atar ve çalışma alanını temizler.
git stash list	Zuladaki tüm kayıtları listeler (örn: `stash@{0}`, `stash@{1}`).
git stash pop	Zuladaki en son attığın kodları geri getirir ve zuladan siler.
git stash apply	Zuladaki en son kodları çalışma alanına getirir ama zuladan silmez.
git stash apply stash@{sayi}	Belirli bir sıradaki zulayı geri getirir.
git stash drop	En son zulayı tamamen çöpe atar.

10. Geri Alma ve Acil Durumlar (Undo / Reset)

git checkout HEAD <dosya_adi>	Belirli bir dosyadaki commit edilmemiş tüm değişiklikleri iptal eder, dosyayı son commit'teki haline döndürür.
git reset HEAD	git add ile hazırlık alanına aldığın dosyaları, hazırlık alanından geri çıkarır (kodlar silinmez).

git reset <commit_hash>	Projeyi belirtilen eski commit'e döndürür ama kodları silmez (Değişiklikleri çalışma alanında bırakır).
git reset –keep <commit_hash>	Geçmiş bir noktaya dönerken, o an çalışma dizininde yaptığın (commitlenmemiş) yerel değişiklikleri korumaya çalışır.
git reset –hard <commit_hash>	TEHLİKELİ! Projeyi o commit'e döndürür ve sonraki tüm kodları tamamen siler.
git reset –hard <uzak_repo/dal_adi>	(Örn `git reset –hard upstream/master`) Kendi yerel kodlarını tamamen silip uzak depodaki dalın birebir aynısı yapar.
git revert <commit_hash>	Hatalı bir commit'in yaptıklarının tam tersini yapan yeni bir commit atar. (Tarihçeyi değiştirmediği için public repolarda hataları geri almanın en güvenli yoludur).

11. Faydalı İpuçları: .gitignore Sonrası Önbellek Temizleme

Diyelim ki bir dosyayı önceden Git'e eklemiştin, sonra .gitignore içine yazdın ama Git hala o dosyayı takip etmeye devam ediyor. Önbelleği temizlemek için sırasıyla şunu çalıştır:

```
git rm -r --cached .
git add .
git commit -m "Git takip önbelleği (.gitignore ayarları) temizlendi"
```

UCH Viki'den alınmıştır.

From:
https://wiki.ulascemh.com/ - **UCH**

Permanent link:
https://wiki.ulascemh.com/doku.php?id=tr:cs:devtools:git:cheatsheet&rev=1775348016

Last update: **2026/04/05 00:13**