Table of Contents

Git Basics

Migrating the Local Repo to GitHub

```
git init -b main
git add --all
git commit -m "First commit"
git remote add origin <REMOTE_URL>
git remote -v
git push origin main
```

Basic Workflow

Kod	Açıklama
git status	*Displays the current status of the project, as well as any files that have been modified or added.*
git add <DOSYA_ADI>	*Adds the modifications to a specific file to the staging area.*
git add . veya git add –all	*Adds all changes in the folder to the staging area..*
git commit -m "COMMIT_MESAJI"	*Saves the files in the stage area permanently to the project history.*
git log	*Lists past commits (the commit history). Press the "q" key to exit...*

Transactions with Remote Repositories

Kod	Açıklama
git remote add origin <url>	*Connects your local repository to a repository on GitHub..*
git push -u origin main	*Push your code to GitHub (Use "-u" the first time; after that, "git push" is sufficient).*
git pull	*It downloads the latest code from GitHub to your computer and merges it with your existing code.*
git fetch	*It downloads changes from the remote repository to the computer but does not merge them (this is safe just for checking what has changed).*

Branching

> It is used to create a parallel copy of the project so you can test new features or fix bugs without affecting your main project (usually the main or master branch).

git branch	*Lists the current branches. Shows which branch you are on (the one marked with an *)..*
git branch <DAL_ADI>	*Creates a new branch.*
git checkout <DAL_ADI> or git switch <DAL_ADI>	*Switches to another subject.*

git checkout -b <DAL_ADI>	*Hem yeni dal oluşturur hem de o dala anında geçiş yapar.*
git branch -d <DAL_ADI>	*İşi bitmiş ve birleştirilmiş bir dalı siler.*

Joining Branches

Kendi dalında işini bitirdiğinde ve bunu ana projeye ektarmak istediğinde bu kodları kullanacaksın. İki yöntemi vardır.

Git Merge

Sen bir dalda çalışırken ana dalda değişiklikler olmuş olabilir. Gerek sen veya bir ekip üyesi değişiklik gerçekleştirmiş olabilir. **Merge** senin dalını ve ana dalını alır, ikisinin son hallerini birleştirip **Merge commit** adında yeni bir kayıt oluşturur.

Ne zaman kullanılır?

Takım halinde çalışırken, ana dalları kendi dalına çekerken veya kendi bitmiş özelliğini ana dala atarken.

Avantajı?

Tarihçeyi asla silmez veya değiştirmez. Güvenlidir. Kimin ne zaman ne yaptığını tam olarak gösterir.

Dezavantajı?

Çok fazla kişi proje üzerinde çalışıyorsa, proje geçmişi(log) “merge commit” çöplüğüne dönebilir ve örümcek ağı gibi karmaşık görünür.

```
git checkout main # Hedef dala geç.
git merge <DAL_ADI> # Birleştir.
```

Git Rebase

Senin dalının başlangıç noktasını alır, main dalının en son haline taşır. Yani “Ben projeye dün başlamıştım ama sanki bugün, herkesin son kodunun üzerine başlamışım gibi tarihçeyi yeniden yaz” der.

Ne zaman kullanılır?

Kendi yerel (henüz push edilmemiş) dalını, ana projenin güncel haliyle senkronize etmek için. “Merge commit” kirliliği yaratmadan temiz bir geçmiş isteniyorsa.

Avantajı?

Dümdüz, okuması çok kolay bir proje geçmişi sağlar. Gereksiz merge commit'leri olmaz.

Dezavantajı?

Tarihçeyi yeniden yazar. Bu yüzden tehlikeli olabilir.

Başka insanların da kullandığı ortak dallarda (örneğin main dalında) ASLA rebase yapma! Sadece kendi bilgisayarındaki, henüz kimsenin görmediği dallarda yap.

```
git checkout <DAL_ADI> # Kendi dalında ol.
git rebase main # Temeli güncelle
```

Git Squash

Kendi dalında çalışırken ufak tefek bir sürü commit atmış olabilirsin (“buton eklendi”, “renk düzeltildi”, “yazım hatası”, “çalışmıyor deneme 1”). Bu gereksiz kalabalığı ana projeye atmadan önce, hepsini tek bir anlamlı commit (“Login Ekranı Tamamlandı”) haline getirme işlemidir.

Ne zaman kullanılır?

Özellik geliştirme bittiğinde, PR (Pull Request) açmadan hemen önce geçmişi temizlemek için.

1. Kaç commit geriye gideceğini belirle (örneğin son 3 commit): `git rebase -i HEAD~3`
2. Karşına bir metin editörü açılır. En üstteki commit pick olarak kalır, altındakilerin başındaki pick yazısını silip `squash` (veya sadece `s`) yazıp kaydedersin.
3. Git sana yeni bir birleştirilmiş mesaj girmen için bir ekran daha açar. Mesajı yazar ve kaydedersin. Boom! 3 commit tek commit oldu.

Note: Clicking the “Squash and Merge” button on GitHub is the easiest and most popular way to automate this process

Commands Used in Emergencies

Kod	Açıklama
`git stash`	You're working, but you're not done yet. You need to switch to another task immediately. You temporarily set aside your current changes. Your workspace is cleared..
`git stash pop`	It retrieves the incomplete code snippets you saved in the drawer.
`git reset HEAD~1`	It reverts your last commit but doesn't delete your code. (It's a lifesaver if you accidentally commit something.).
`git reset –hard HEAD~1`	It will completely delete the latest commit AND the code you wrote; this action cannot be undone.
`git revert <COMMIT_ID>`	It does not delete a faulty commit, but instead creates a new commit that reverses the changes made by that commit. This is the safest way to fix errors on public branches used for collaborative work...

Taken from UCH Wiki.