# GIT

Notes :  Tanımlar ve özellikler, ders notları, diğer notlar,  cheet sheet bulunmaktadır.

# TANIMLAR ve ÖZELLİKLER

Git Nedir?

Git free ve opernsource bir versiyon kontrol sistemidir.

kod daki değişiklikleri takip eden dağıtık bir sistemdir.

Git Ne işe yarar?

- Revert files to previous state,
- Revert entire project back to previous state,
- Compare changes over time,
- See who modified what? **And much more...**

Neden?

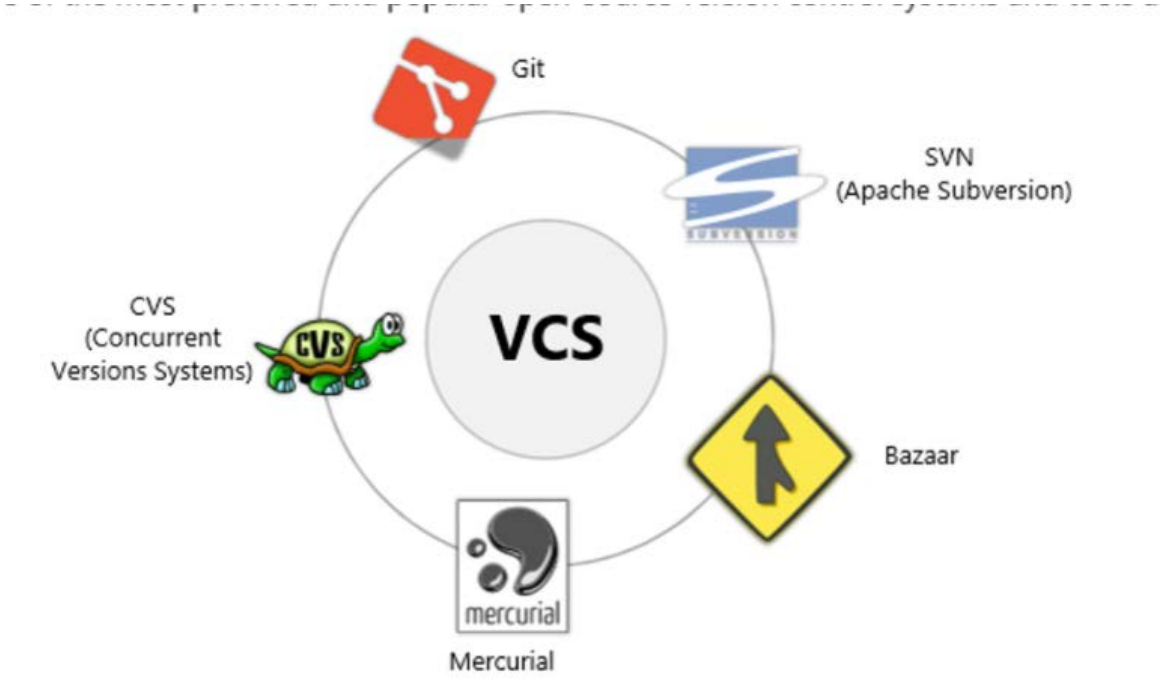
- Everything is local (full history tree available offline),
- Everything is fast,
- Snapshots, not diffs,
- It is distributed not centralized,
- Great for those who hate: CVS/SVN (earlier version control systems).

VCS ne işe yarar?

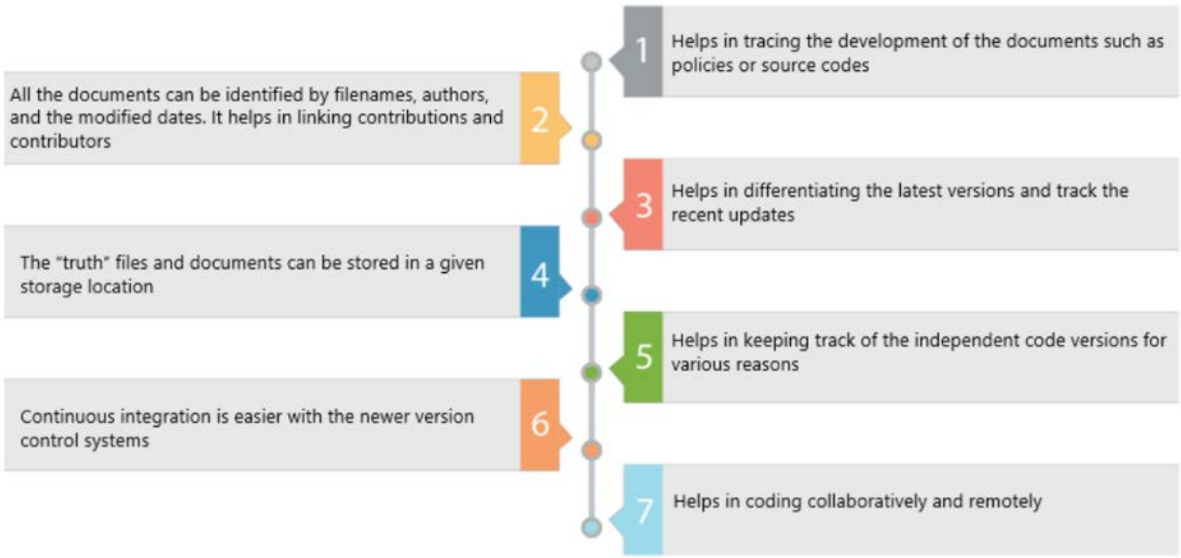
- What had changed
- When it changed
- Why it changed
- Who changed it

Dosyaların saklanması gerekir. VCS kullanırsanız kaybetmezsiniz, tüm versiyonlarına erişebilirsiniz.

Populer versiyon kontrol sistemleri:

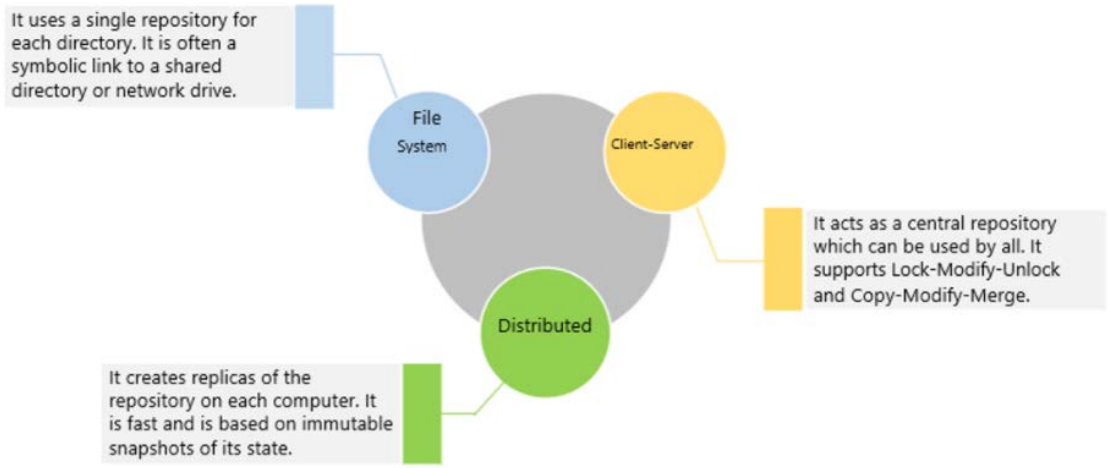



VCS in Önemi:



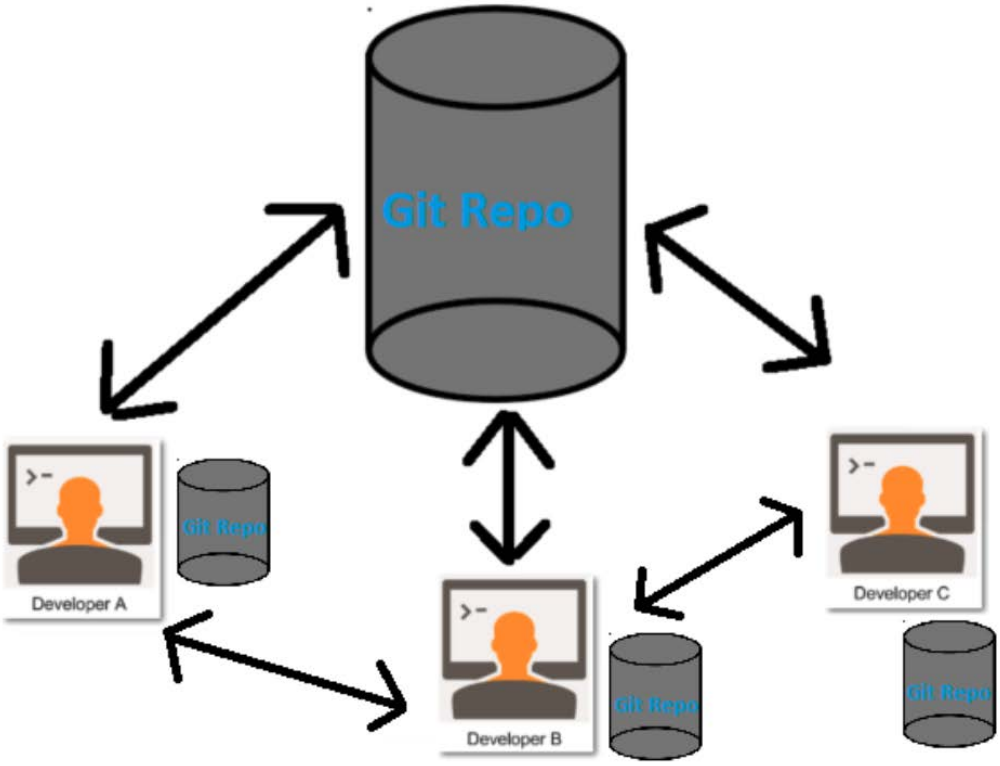

VCS Çeşitleri:

- File-based, eski
- Client-server type, git gibi
- Distributed.




Clien-Server tip iki çeşit:


Distributed sistemlerde her kullanıcı için bir kopya oluşturulur. Ağ bağlantısı kopsa bile devam eder.




Here are some features of Distributed VCS.


Repository:

Projeler için dizin veya depolama alanı

Localde veya bulutta olabilir.

Git Takip Süreci:

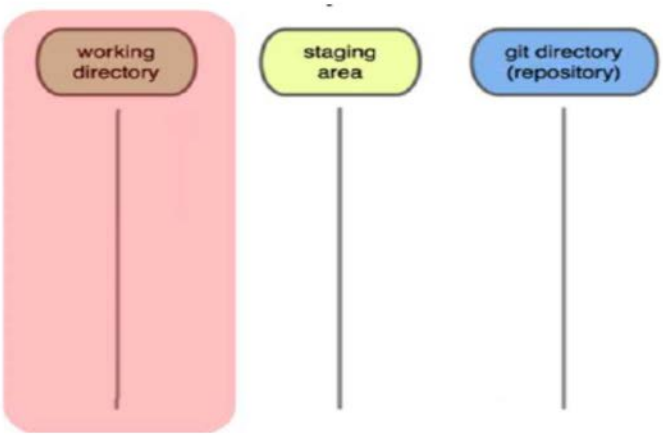

git repoda dosya 3 aşamada bulunabilir. Modified, Staged, and Committed.

- **Modified** means that you have changed the file but have not committed it to your database (repo) yet.
- **Staged** means that you have marked a modified file in its current version to go into your next commit snapshot.
- **Committed** means that the data is safely stored in your local database.

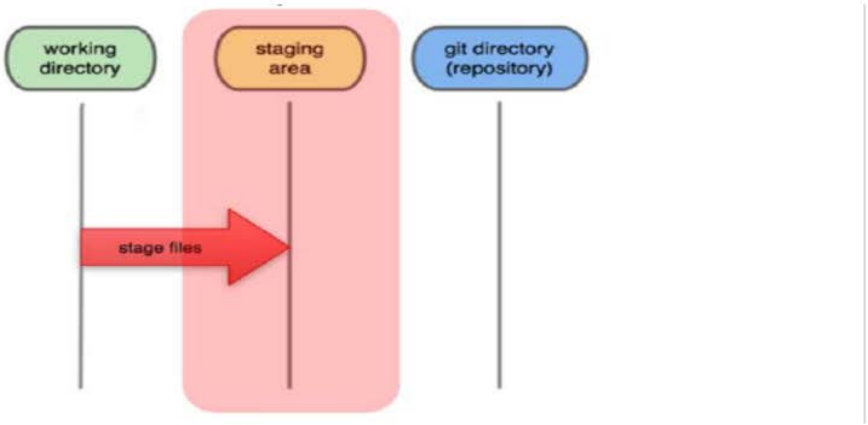
git projesinin 3 ana bölümü

- The working tree,
- The staging area,
- The Git directory.

You modify files in your working directory.

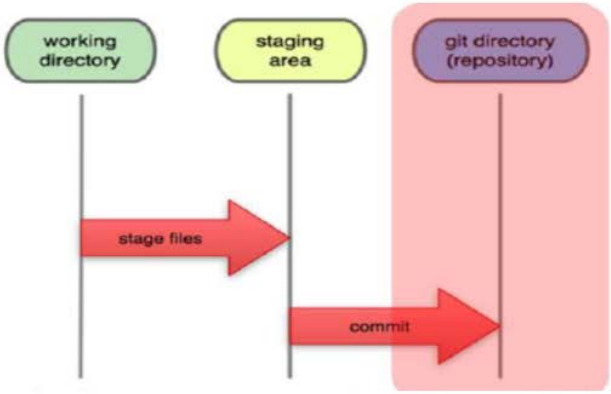



After you are done with your file in your working directory you add them to staging area, you can think of this as a temporary storage for your files. You stage the files, adding snapshots of them to your staging area.

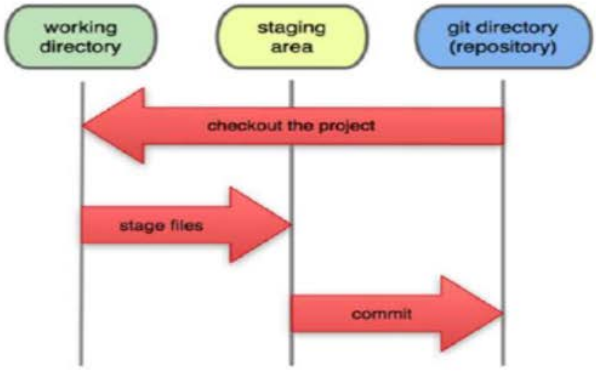



Alışveriş yapıyorsun sepete doldurdun sepet staging area. daha almadın değiştirebilirsin.

Then you are sure that the files you have in your staging area are ready to go next step then you commit your files to your git local repo. You do a commit that stores snapshot permanently to your Git directory.




Once you commit your files to your local repo you can then checkout them at any time you can modify them then add and commit again as you wish.

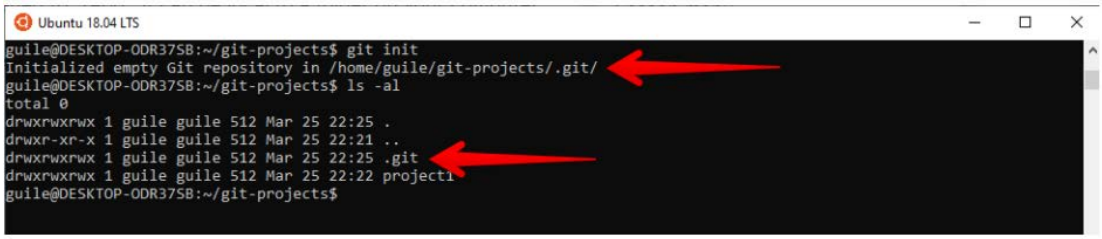



# DERS NOTLARI

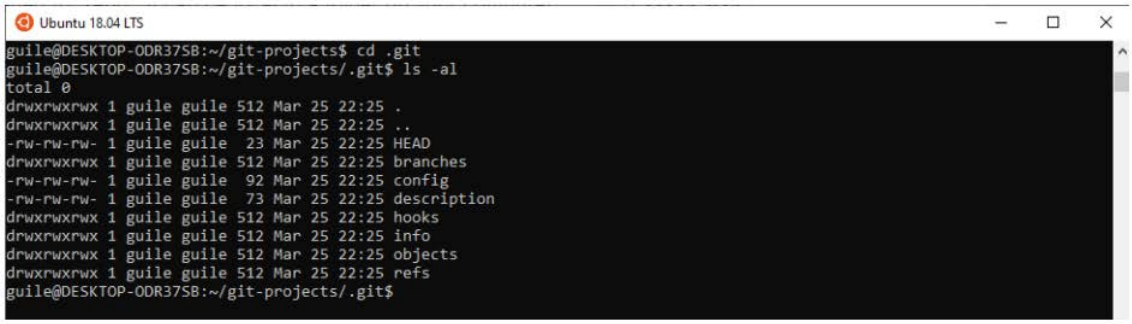
Git başlatma komutu:

```
$ git init
```

```
$ ls -al
```




```
$ cd .git
$ ls -al
```

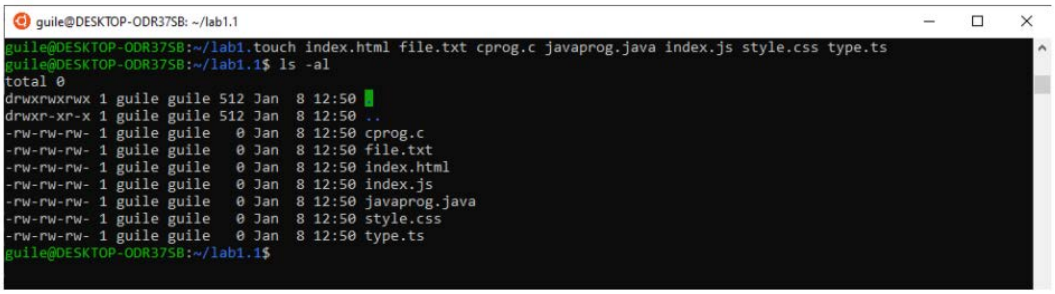



Örnek ;

1-) Takip etmek istediğin klasörü localde oluşturmaya başla.

$ touch index.html file.txt cprog.c javaprog.java index.js style.css type.ts

$ ls –al

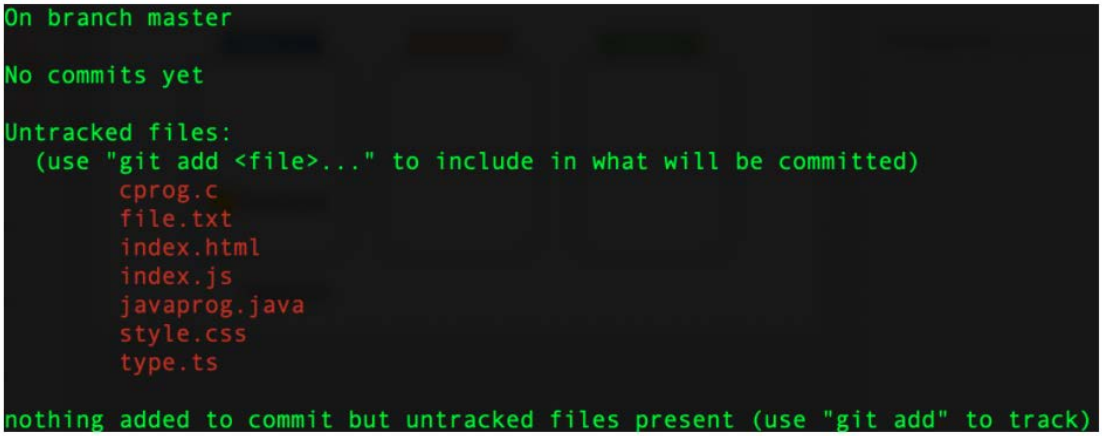



2-) Olışan klasörde takip sistemi yani git başlatma.

$ git init (o klasörde vcs başlatır)


3-) Check git status.

$ git status




kırmızılar çalışma alanında duruyor. git takip etmiyor

git status -s # Short status

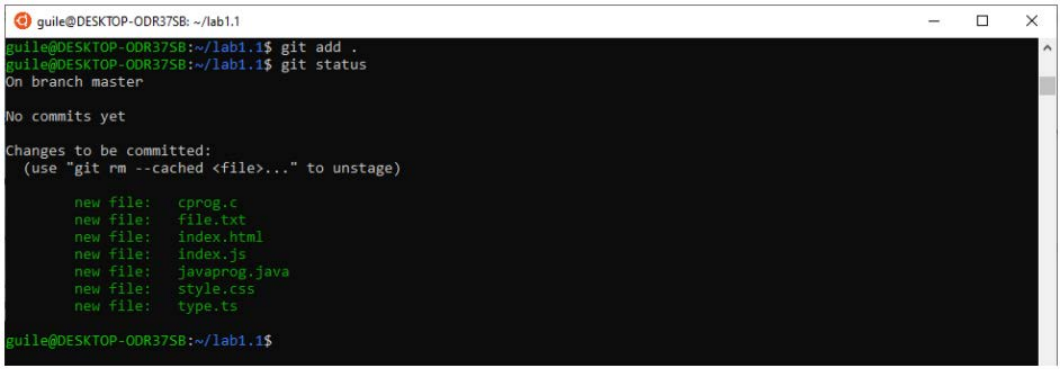
4-) "Git add" to add the files to staging area from the working area.

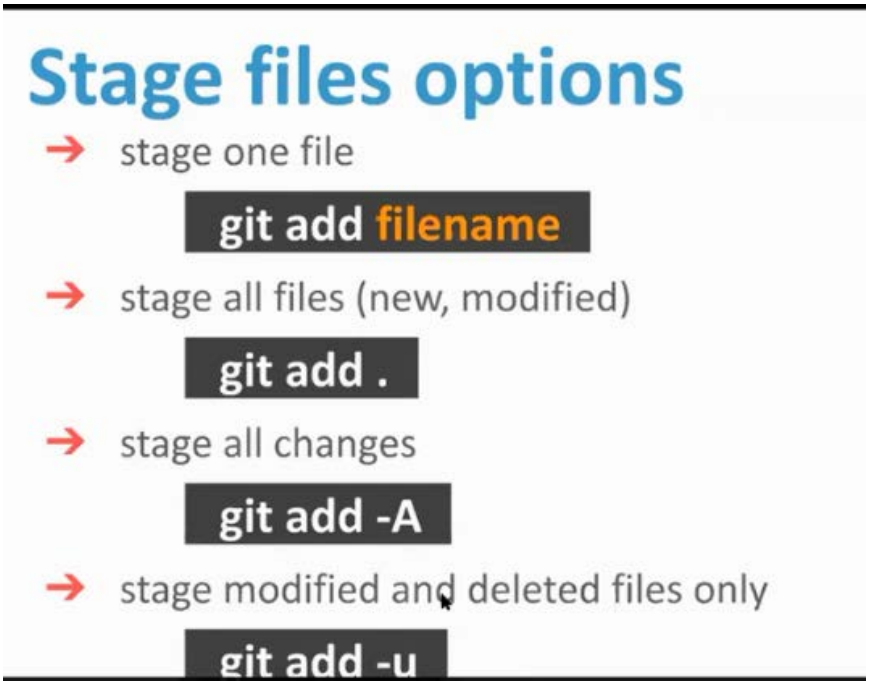
$ git add .

And now we can see the new status of the files.(Notice green color)

git add dosya_adı commite hazır.

$ git status






git  restore —staged dosya_Adı →stage i geri alma

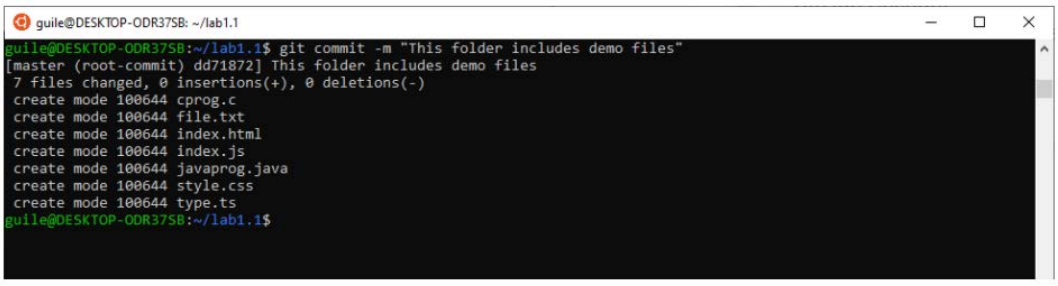
git rm --cached file1.js →stage i geri alma

** git add yerine git rm de kullanılabilir.

5-) Let's commit our files to local repo with the command

`git commit -m "message/explanation"`

$ git commit –m “This folder includes demo files”




git commit -am "message" direk değişiklikleri stage e atmadan commit etme. Dosya daha önce yaratılıp commit edildi ise çalışır. daha sonraki değişiklikler için add yapmadan commit yapılaiblilr.

git commit —amend  commit mesajı değiştirme.

** eğer uzun bir commit mesajı yazılacaksa -m yazılmaz editör açılır istediğiniz uzunlukta mesaj giriliebilir.

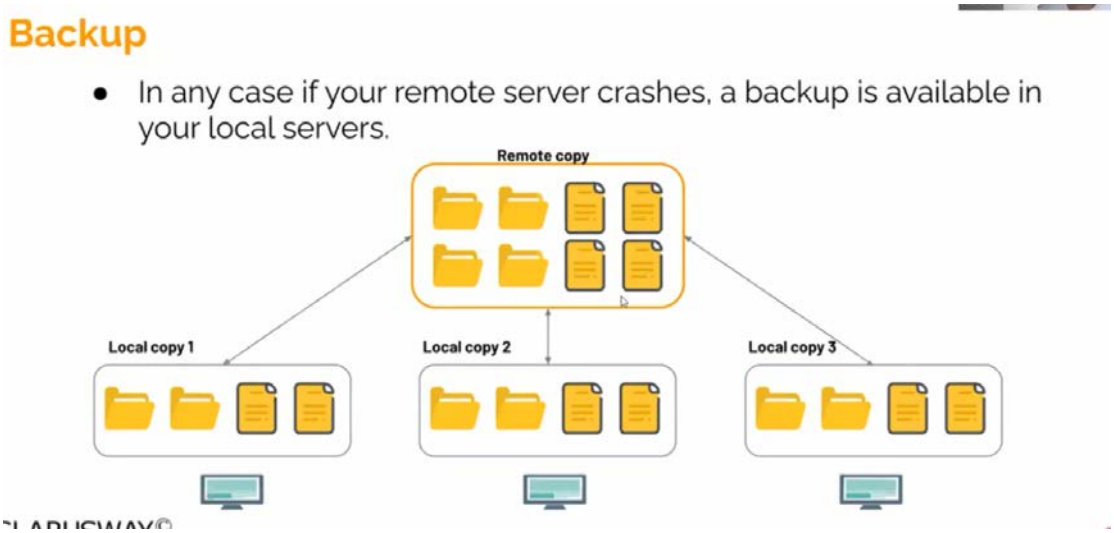
** tamamlanmamış değişikleri commit etmek yerine git stash ile kayıt altına alabilirsiniz.

git stash pop komutu ile yukarıdaki listenin en üstünde yer alan değişiklik geri yüklenecek ve bu değişiklik listeden silinecek.

git stash apply komutu ile istediğiniz değişikliği geri yükleyebilirsiniz. Ancak bu işlem sonrasında yüklediğiniz değişiklik listeden silinmeyecek.

Herhangi bir değişikliği listeden silmek için git stash drop komutunu kullanabilirsiniz.

Backuplama mantığı ve aynı dosya üzerinde birden fazla kişi çalışırsa doğabileccek sorunlar;




1 ve 3 değişiklik yaptı remote a gönderdiğinde proje yöneticisi karar veriyor.

Konfigürasyon ayarları;


Working directory de bir değişiklik yapıldığında ;

git status yazınca kırmızı gözükecek

git add new_file2 yapınca stage area ya alır ve status yapınca yeşil olur.

birden fazla dosyayı aynı anda stage areaya koymak için git add .

git add -u;

değiştirilen ve silinen dosyaları staging areaya atmak için.

git log ;

yapılan tüm commitleri görebilen komut.


sarı hash kodları ile commitleri takip edebilrisniz.

eski komuta gelmek için git checkot hash kodunun ilk 5 hanesini gir

git checkout ae0952e5

** git log -p detaylı logları görebilmeyi sağlar.

git checkout ile hem ileri hem de geri gelebilirsiniz.

HEAD hangi commit de olduğumuz ifade eder.

# GITHUB

GitHub is a Git repository hosting (Source Code Hosting) service

**What is the difference between Git and GitHub?**

Git is a version control system that lets you manage and keep track of your source code history locally. GitHub is a cloud-based hosting service that lets you manage Git repositories.

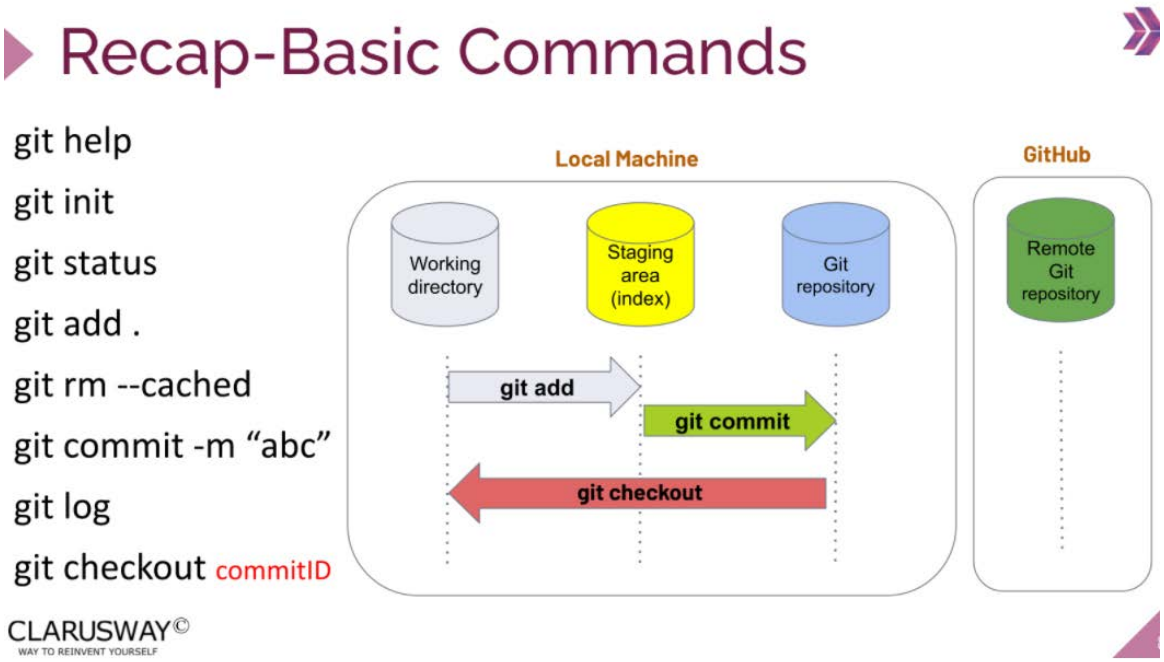
GitHub is the most popular one as you see

| Name | Users | Projects |
|---|---|---|
| GitLab | 100,000 | 546,000 |
| Bitbucket | 5,000,000 | Private |
| SourceForge | 3,700,000 | 500,000 |
| launchpad | 3,965,288 | 40,881 |
| GitHub | 24,000,000 | 69,000,000 |

git config

git config list ile config bilgilerini görebilirsiniz.

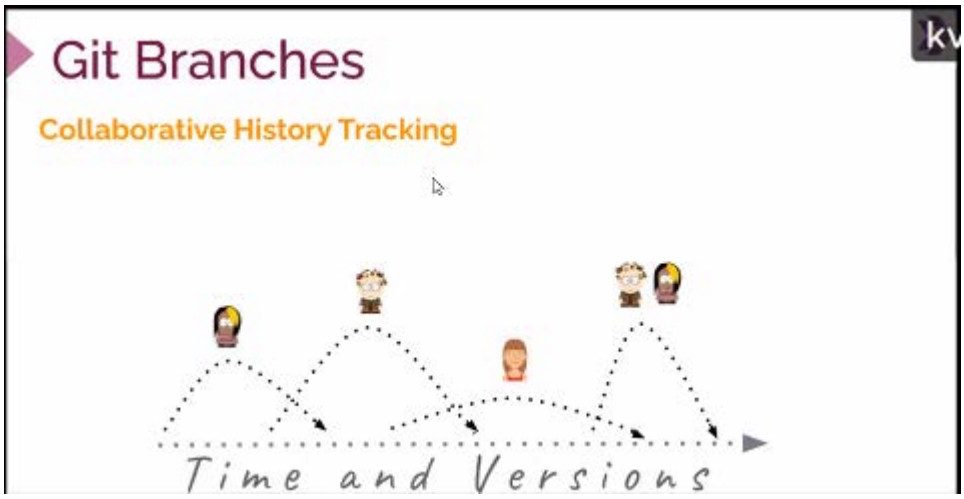
```
Samet USTAOGLU@DESKTOP-SL2E65K MINGW64 ~/Desktop/git-lesson-3 (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=SametUSTAOGLU
user.email=ustaoglusamet05@gmail.com
core.editor=vim
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

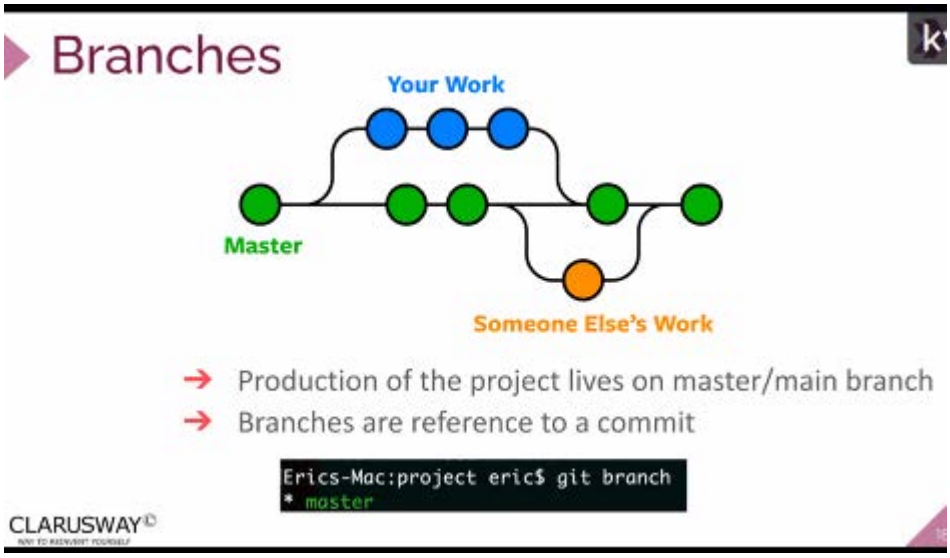



BRANCH

Aynı projede birden fazla grup çalışması için herkes ayrı çalışıyor en sonda birleşecek.

hangi branch de iseniz head orayı gösterir.






Branch komutları:

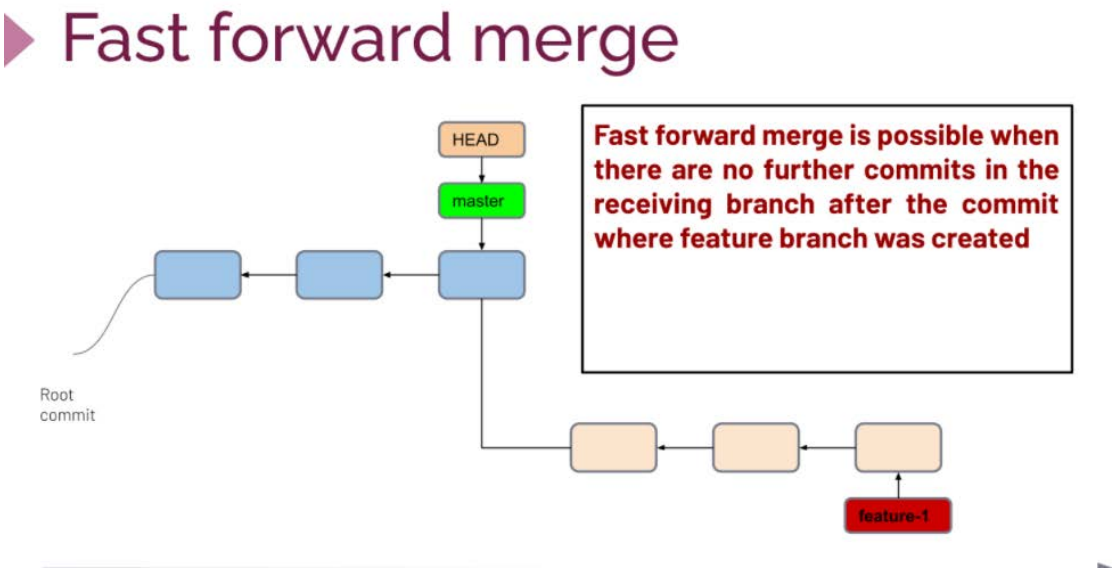

git checkout -b newbranch3 yaratır ve geçer.

** git branch -v ile branchler hakkında ayrıntı gelir.

** Herhangi bir anda bir proje için tek bir branch aktif olabilir. Bu branch'e HEAD denir.

** git branch -dr superyeniozellik ————————-remote daki branch i silme

git push origin :superyeniozellik bu komut ile birlikte siler.




nerde iseniz ordaan birleştirmek istediğiniz yeri seçiyorsun

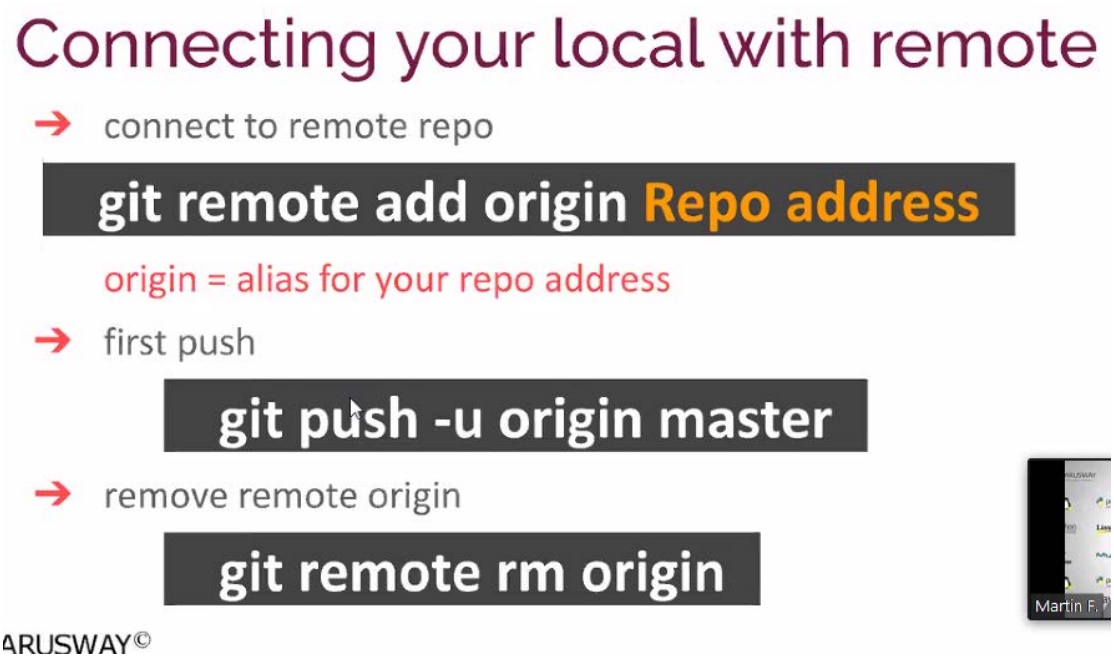
aynı dosya üzerinde farklı değişiklikler varsa conflict verir. bunu proje yöneticisi çözer.

bir grup hiç değişiklik yapmadan diğer kol yaparsa fast forward merge oluyor.

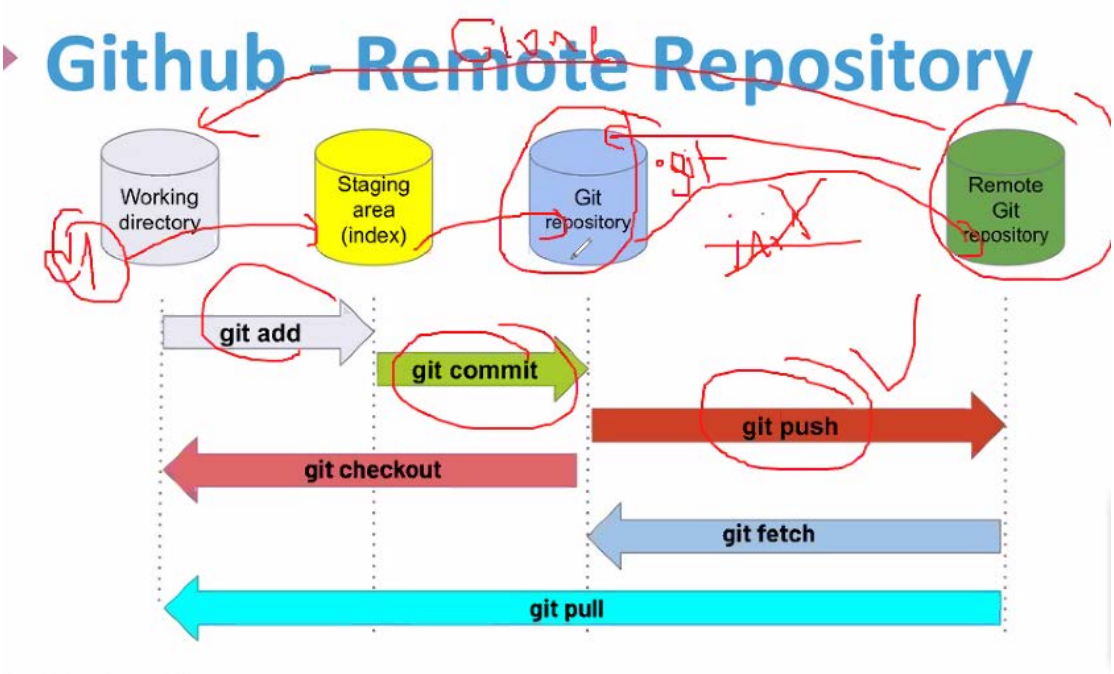
```
git branch branch_name
git branch
git branch -r
git branch -a
git checkout branch_name
git checkout -b branch_name
git branch -d branch_name
git branch -D branch_name
git merge branch_name
```

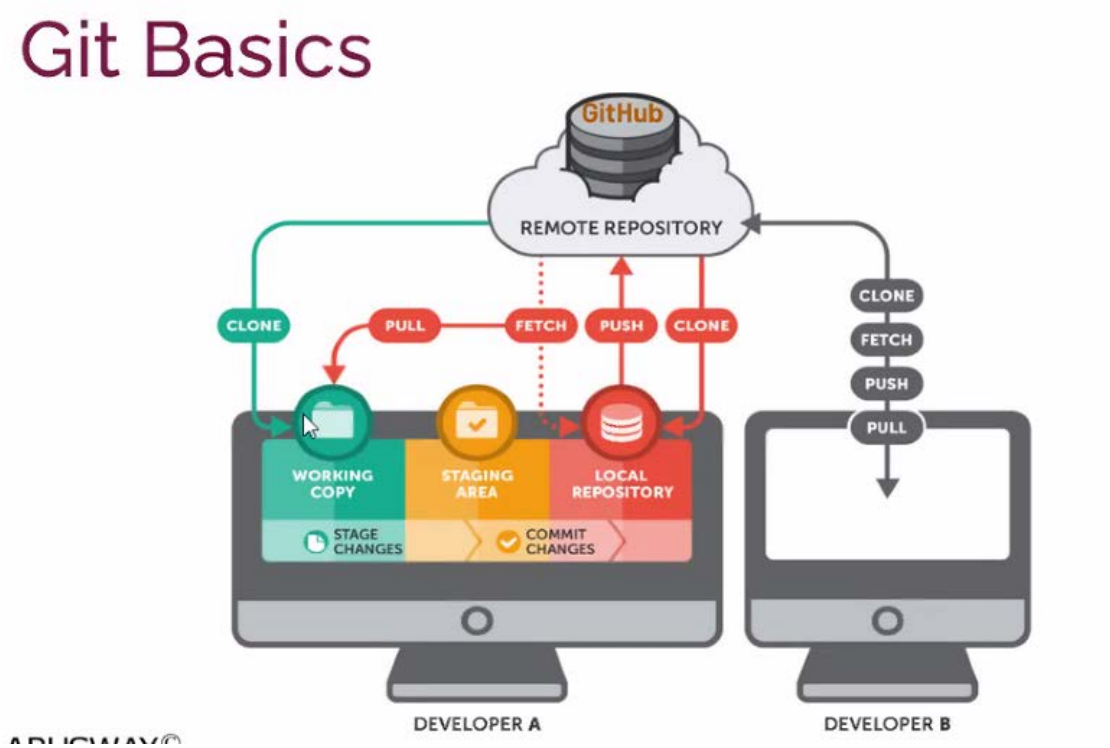
git log --pretty=oneline

git log --oneline




önce clone ile çek daha sonra pull ,le devam




## UZAKTAKİ REPOYU(PRIVATE)  KENDİ GİTHUB IMIZA EKLEME

1-) Öncelikle bilgisayarınızda uygun bir yere klasör oluşturun. O klasöre uzaktaki repoyu clone edeceğiz.

2-)klasörün içinde iken git bash çalıştırıp

git clone <uzak repo linki> komutunu çalıştırın.

dosyalar klasörünüze inmiş olacak.

Bu adımda yüklemek istediğimiz repoyu bilgisayara indirdik.

3-) Kendi Github hesabınızda yeni bir repo create edin. oluşan reponun url sini kopyalayın.

4-) git clone <uzak repo linki> komutunu çalıştırın.

dosyalar klasörünüze inmiş olacak.

Bu adımda boş yeni yarattığımız repoyu bilgisayara indirdik.

5-) 2 numaralı adımda oluşan dolu klasörün içindeki dosyaları .git hariç kopyalayıp 4. adımda oluşturduğumuz kendi  klaörümüze kopyalayın. Dikkat!! .git kopyalanmamalı.

6-) yapıştırdığımız yani kendi klasörümüzde dosyaları göndermeden önce add ve commit adımlarını yapmamız gerekiyor.

git add .

git commit -m "mesaj"

7-) dosya push edilmeye hazır.

git push

8-) sayfayı yenilediğinizde reponun dolduğunu görebilirsiniz.

.gitignore;

git in kontrol etmesini istemediğimiz dosyaları .gitignore klasörüne atılır.

# DİĞER NOTLAR

** md (markdown)  örn: readm.md

!! origin uzaktaki repo demek

git clone yapıldığında git init yapmaya gerek yoktur. Var olan bir repoyu bilgisayara çekme

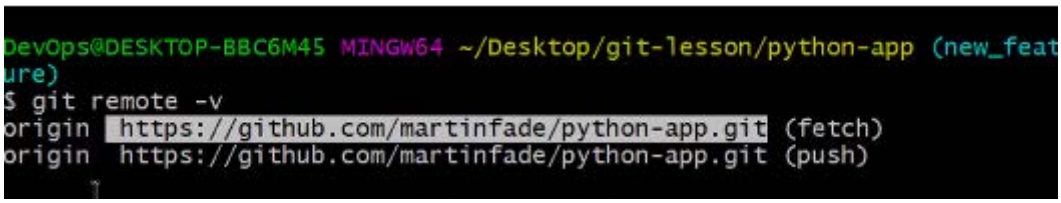
Github genel komutlar;

```
echo "# starter" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/SametUSTAOGLU/starter.
git push -u origin main

git remote -v  → hangi repoda olduğunu gösterir.
git fetch → uzaktaki değişikliği
git pull  → git fetch + git merge
git push origin newbranch cssnsnn.text → yeni branchi remote a gönderme


vimden çıkış esc- :q!
git config core.autocrlf true uyarıları kapatmak için
git push origin master
```

https://git-school.github.io/visualizing-git/ → görsel commir-t brach çalışma sayfası.




origin aslında sağ taraftali linkin alias ı

git remote remove upstream  pul request için gönderdiğini silme

!! başka repoda olan değişiklikleri nasıl alabiliriz?

git pull upstream main

!! farklı branhe geçip değişiklik yaptınız nasıl push edilecek.

git push yapınca hata alınır sebebi uzaktaki repoda  o branch yok

git push --set-upstream origin newbranch

bu komut ile fork yaptığımız kendi repomuza gönderdik

son olarak repomuza girip yukarıdaki uyarıya tıklayıp creat pull request yapılacak.

** git merge --abort  — merge işlemi  geri alma.

** git rebase Branch-B  — merge gibi

Bu komut ile Git öncelikle Branch-A ile Branch-B'nin ortak en son commit'ini bulup ortak commit sonrasında Branch-A'da yapılan diğer tüm commit'leri geri alır. Aslında bu commitler silinmez sadece geçici olarak farklı bir yerde saklanır. Daha sonra Branch-B'deki tüm commitler Branch-A'ya uygulanır. Son aşamada ise Branch-A'nın geçici olarak farklı bir yerde saklanan commit'leri tekrar uygulanır. Bu işlemler sonrasında tüm değişiklikler sanki sadece Branch-A üzerinde gerçekleşmiş gibi görünür.

```
git config --global core.eol native
git config --global core.autocrlf true
```

İPUCU: core.eol ve core.autocrlf değerleri Windows platformu için önerilen değerler. Git'in windows'da End-Of-Line karaketerlerini (Windows EOL için CRLF kullanıyor, Linux, Unix ve OSX ise LF kullanıyor) düzgün çalışması için global git konfigürasyonunuzda core.eol ve core.autocrlf ayarlarının yapılması gerekiyor. Ancak, proje depolarınızda .gitattributes dosyasının ilk satırında text=auto değerini girerseniz bu ayar yukarıda yapılan core.eol ve core.autocrlf ayarlarını ezecektir.

# GIT CHEATSHEET

## Initializing a repository

```
git init
Staging files
git add file1.js # Stages a single file
git add file1.js file2.js # Stages multiple files
git add *.js # Stages with a pattern
git add . # Stages the current directory and all its content
```

## Viewing the status

```
git status # Full status
git status -s # Short status
```

## Committing the staged files

```
git commit -m “Message” # Commits with a one-line message
```

```
git commit # Opens the default editor to type a long message
```

## Skipping the staging area

```
git commit -am “Message”
```

## Removing files

```
git rm file1.js # Removes from working directory and staging area
git rm --cached file1.js # Removes from staging area only
```

## Renaming or moving files

```
git mv file1.js file1.txt
```

## Viewing the staged/unstaged changes

```
git diff # Shows unstaged changes
git diff --staged # Shows staged changes
git diff --cached # Same as the above
```

## Viewing the history

```
git log # Full history
git log --oneline # Summary
git log --reverse # Lists the commits from the oldest to the newest
```

## Viewing a commit

```
git show 921a2ff # Shows the given commit
git show HEAD # Shows the last commit
git show HEAD~2 # Two steps before the last commit
git show HEAD:file.js # Shows the version of file.js stored in the last commit
```

## Unstaging files (undoing git add)

```
git restore --staged file.js # Copies the last version of file.js from repo to index
```

## Discarding local changes

```
git restore file.js # Copies file.js from index to working directory
git restore file1.js file2.js # Restores multiple files in working directory
git restore . # Discards all local changes (except untracked files)
git clean -fd # Removes all untracked files
```

## Restoring an earlier version of a file

```
git restore --source=HEAD~2 file.js
```

# Browsing History

## Viewing the history

```
git log --stat # Shows the list of modified files
git log --patch # Shows the actual changes (patches)
```

## Filtering the history

```
git log -3 # Shows the last 3 entries
git log --author="Mosh"
git log --before="2020-08-17"
```

git log --after=“one week ago”
git log --grep=“GUI” # Commits with “GUI” in their message
git log -S“GUI” # Commits with “GUI” in their patches
git log hash1..hash2 # Range of commits
git log file.txt # Commits that touched file.txt

## Formatting the log output

git log --pretty=format:”%an committed %H”

## Creating an alias

git config --global alias.lg “log --oneline"

## Viewing a commit

git show HEAD~2
git show HEAD~2:file1.txt # Shows the version of file stored in this commit

## Comparing commits

git diff HEAD~2 HEAD # Shows the changes between two commits
git diff HEAD~2 HEAD file.txt # Changes to file.txt only

## Checking out a commit

git checkout dad47ed # Checks out the given commit
git checkout master # Checks out the master branch

## Finding a bad commit

git bisect start
git bisect bad # Marks the current commit as a bad commit
git bisect good ca49180 # Marks the given commit as a good commit
git bisect reset # Terminates the bisect session

## Finding contributors

git shortlog

## Viewing the history of a file

git log file.txt # Shows the commits that touched file.txt
git log --stat file.txt # Shows statistics (the number of changes) for file.txt
git log --patch file.txt # Shows the patches (changes) applied to file.txt

## Finding the author of lines

git blame file.txt # Shows the author of each line in file.txt

## Tagging

git tag v1.0 # Tags the last commit as v1.0
git tag v1.0 5e7a828 # Tags an earlier commit
git tag # Lists all the tags
git tag -d v1.0 # Deletes the given tag

# Branching & Merging

## Managing branches

git branch bugfix # Creates a new branch called bugfix
git checkout bugfix # Switches to the bugfix branch
git switch bugfix # Same as the above
git switch -C bugfix # Creates and switches
git branch -d bugfix # Deletes the bugfix branch

## Comparing branches

git log master..bugfix # Lists the commits in the bugfix branch not in master
git diff master..bugfix # Shows the summary of changes

## Stashing

git stash push -m “New tax rules” # Creates a new stash
git stash list # Lists all the stashes
git stash show stash@{1} # Shows the given stash
git stash show 1 # shortcut for stash@{1}
git stash apply 1 # Applies the given stash to the working dir
git stash drop 1 # Deletes the given stash
git stash clear # Deletes all the stashes

## Merging

git merge bugfix # Merges the bugfix branch into the current branch
git merge --no-ff bugfix # Creates a merge commit even if FF is possible
git merge --squash bugfix # Performs a squash merge
git merge --abort # Aborts the merge

## Viewing the merged branches

git branch --merged # Shows the merged branches
git branch --no-merged # Shows the unmerged branches

## Rebasing

git rebase master # Changes the base of the current branch

## Cherry picking

git cherry-pick dad47ed # Applies the given commit on the current branch

# Collaboration

## Cloning a repository

git clone url

## Syncing with remotes

git fetch origin master # Fetches master from origin
git fetch origin # Fetches all objects from origin
git fetch # Shortcut for “git fetch origin”
git pull # Fetch + merge
git push origin master # Pushes master to origin
git push # Shortcut for “git push origin master”

## Sharing tags

git push origin v1.0 # Pushes tag v1.0 to origin
git push origin —delete v1.0

## Sharing branches

git branch -r # Shows remote tracking branches
git branch -vv # Shows local & remote tracking branches
git push -u origin bugfix # Pushes bugfix to origin
git push -d origin bugfix # Removes bugfix from origin

## Managing remotes

git remote # Shows remote repos
git remote add upstream url # Adds a new remote called upstream
git remote rm upstream # Remotes upstream

# Rewriting History

## Undoing commits

git reset --soft HEAD^ # Removes the last commit, keeps changed staged
git reset --mixed HEAD^ # Unstages the changes as well
git reset --hard HEAD^ # Discards local changes

## Reverting commits

git revert 72856ea # Reverts the given commit
git revert HEAD~3.. # Reverts the last three commits
git revert --no-commit HEAD~3..

## Recovering lost commits

git reflog # Shows the history of HEAD
git reflog show bugfix # Shows the history of bugfix pointer

## Amending the last commit

git commit --amend

## Interactive rebasing

git rebase -i HEAD~5