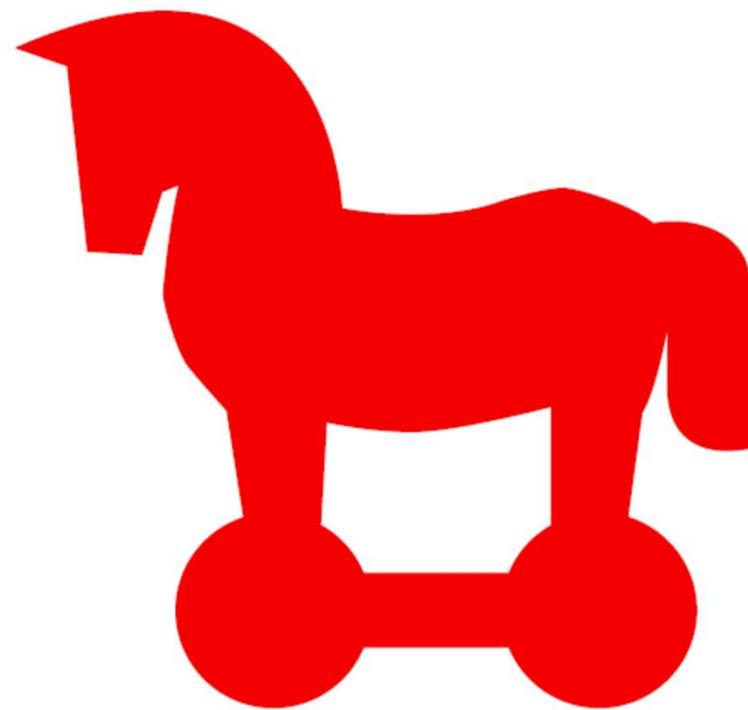


GANTEK
INTEGRATING FUTURE

GANTEK
INTEGRATING FUTURE

GANTEK
INTEGRATING FUTURE



GANTEK ACADEMY

40 YILLIK TECRÜBE İLE

GIT

<https://docs.gitlab.com/ee/topics/git/>

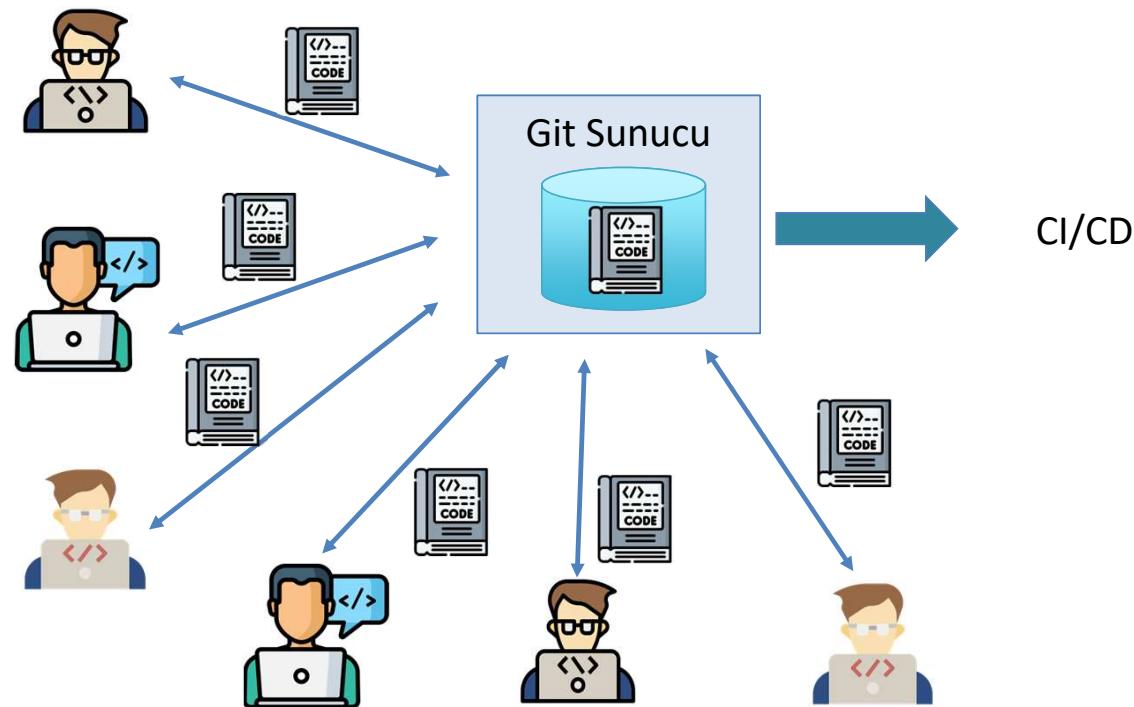
git

- Yazılım geliştirme süreçlerinde kullanılan, kaynak kodu yönetim ve sürüm kontrol sistemidir.
- Git, Linux kaynak kodunun yönetilmesi amacıyla 2005 yılında Linus Torvalds tarafından yazılmıştır.
- Temel olarak bir **DevOps** aracıdır ve küçük büyük fark etmeksizin her türlü çapta projede kullanılmaktadır.
- Git, kaynak kodunda olan değişiklikleri takip etmekte ve bir çok geliştiricinin aynı kaynak kodu üzerinde çalışmasını sağlamaktadır.
- Merkezi git sunucusu sayesinde her geliştirici, projenin kaynak kodunun kopyasını kendi lokal disk sisteminde barındırmakta ve kaynak kodu dosyaları üzerinde yaptığı değişiklikler git tarafından algılanarak ve depoya aktarılırak diğerleri tarafından görülebilmesi sağlanmaktadır.

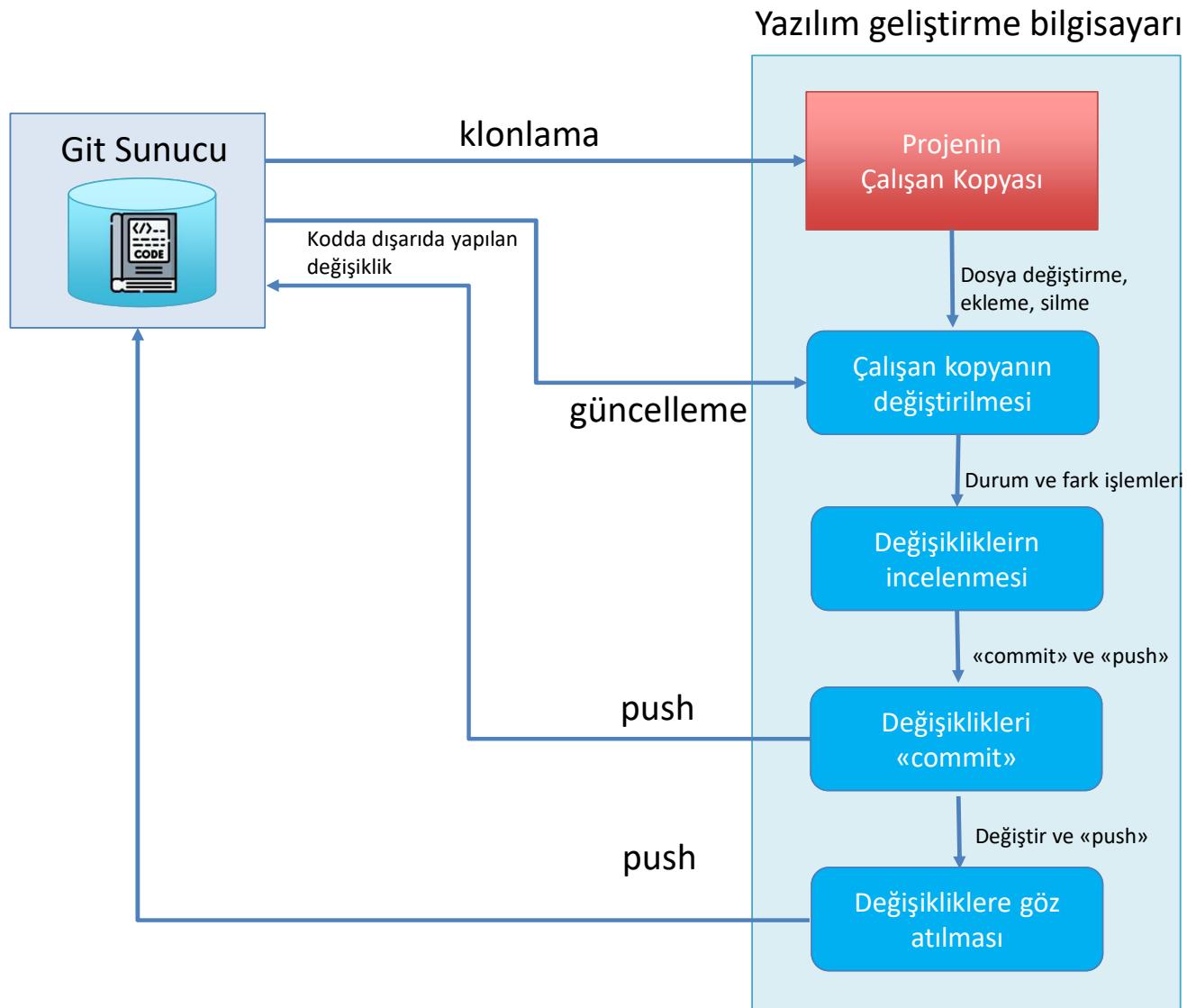
git

- Eğer istenirse, ana dalda (master branch) değişiklik yapılmadan, ana daldaki kodun bir kısmı/tamamı kopyalanarak farklı bir dal üzerinde (feature branch) kaynak kodunda değişiklik yapılması ve başarılı olursa ana dalla birleştirilmesi (veya ana kol haline getirilmesi) sağlanabilmektedir. Bu da doğrusal olmayan bir geliştirme imkanı yaratmaktadır.
- Kod üzerinde yapılan değişiklikler zaman etiketleriyle kaydedilmektedir.
- Yedekleme imkanı bulunmaktadır.
- İstenen sürümе dönme imkanı sağlanmaktadır.
- Ölçeklenebilir bir sistemdir.
- Yazılımlar dağıtık şekilde geliştirilebilmektedir.

Git İş Akışı

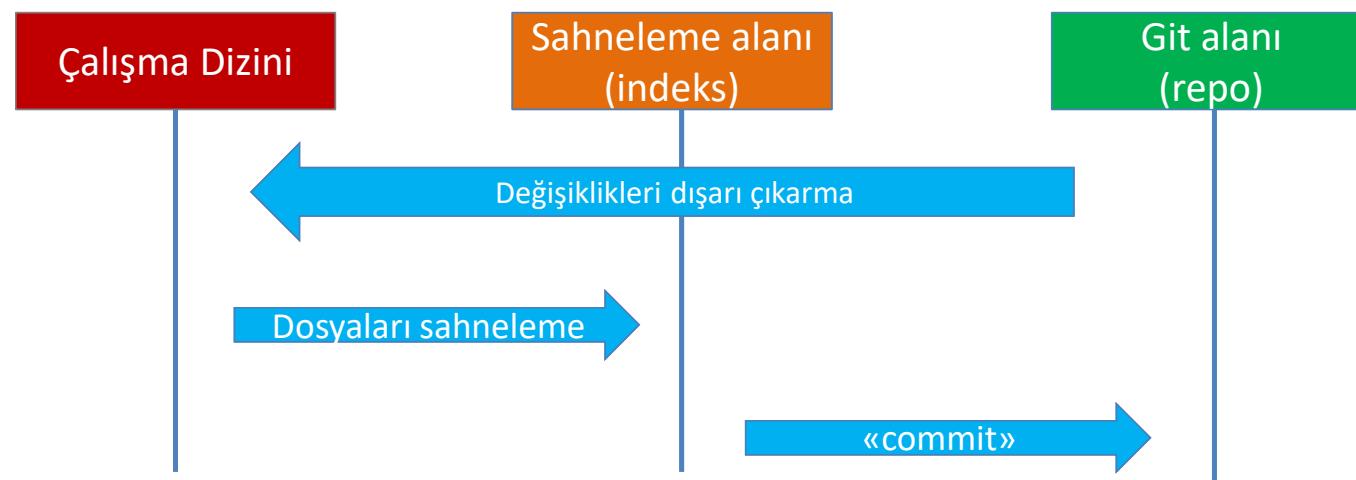


git iş akışı

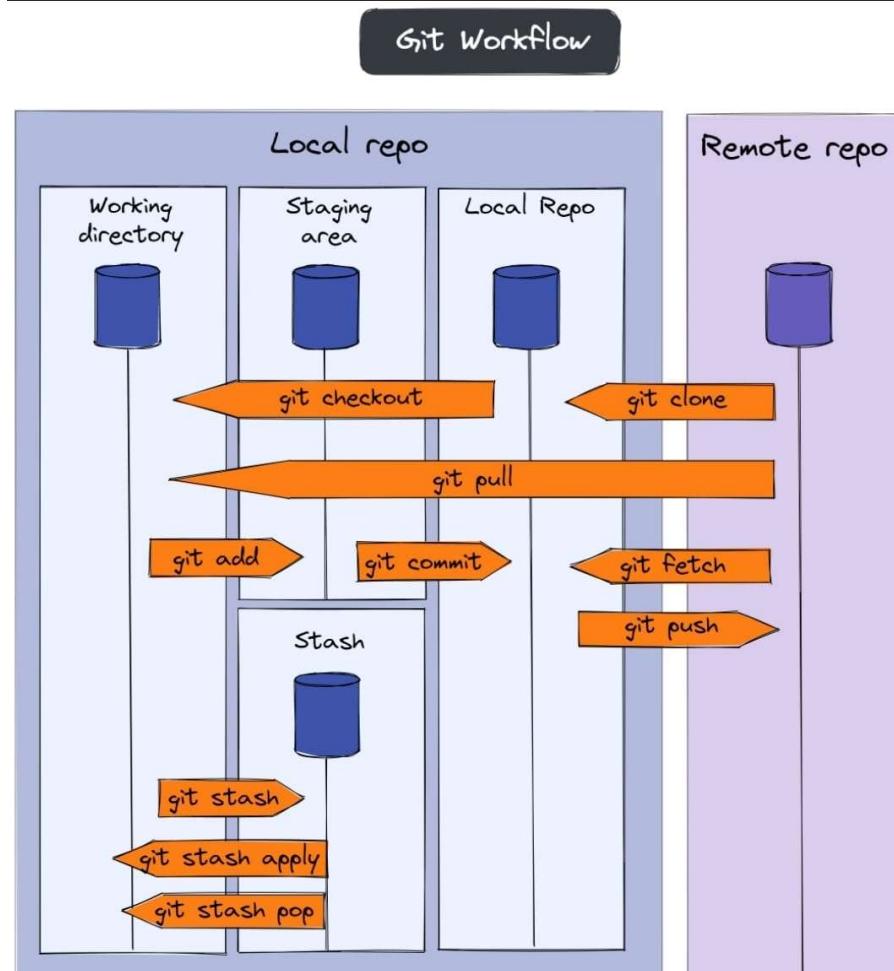


git iş akışı

- Git'in iş akışı üç aşamalıdır:
 - Çalışma dizini: kod dosyaları üzerinde yapılan değişikliklerin tutulduğu çalışma dizini
 - Sahneleme (staging) alanı (indeks): dosyaların evreleriyle ve anlık görüntülerini alarak sahneleme alanına aktarmak
 - Git dizini (repository – repo): Koddaki belirli bir andaki anlık durumun - görüntünün (snapshot) git dizinine aktarılması (commit). Eğer depodaki kodda başkaları tarafından yapılmış değişiklikler varsa, bunların çıkışının yapılması



git komutlarının işleyışı

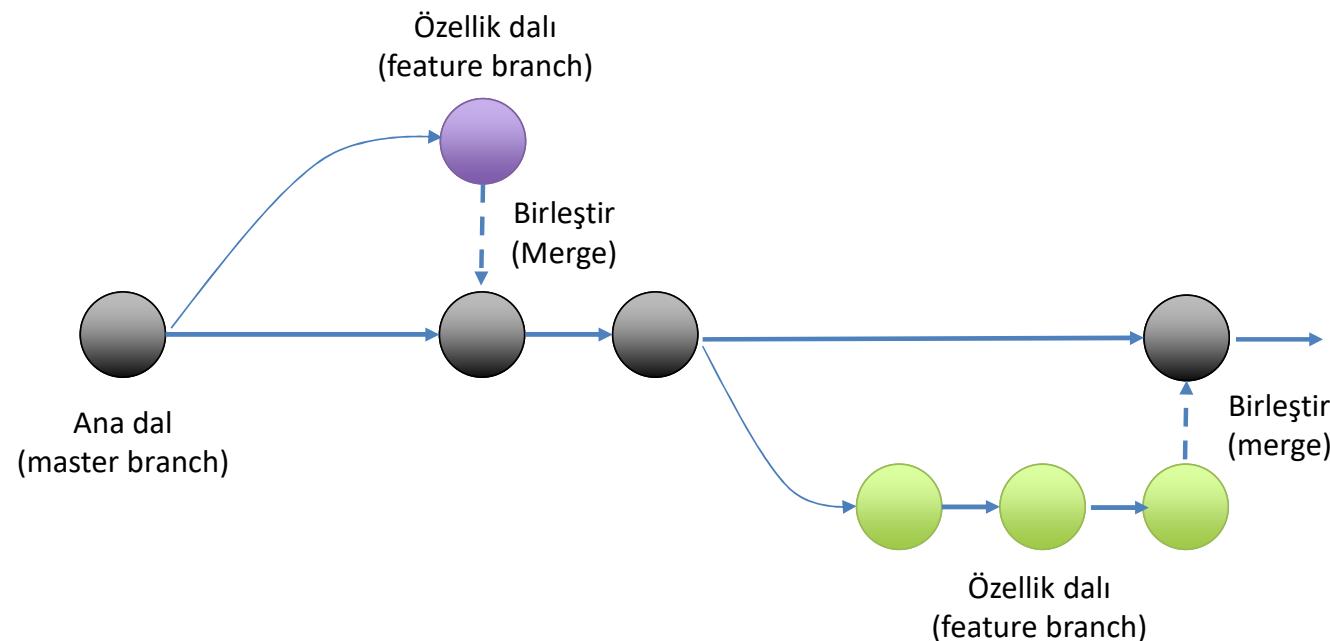


Git'te Dal (branch)

- Merkezi sürüm ve kaynak kodu yönetim sistemlerinin en büyük sorunu, aynı dosyada aynı alanlarda farklı geliştiricilerin benzer değişiklikler yapmasıdır. Bunların yönetilmesi zordur.
- Git'in en önemli avantajlarından biri, ana geliştirme dalından farklı bir dal üzerinde geliştirme yapılması imkanıdır.
- Dal (branch), proje dosyasında geliştiricilerin yapacağı değişiklıkların ayrı bir kaynak kodu kopyasında yapılmasını sağlayan bir olanaktır.
- Ana sürüm kopyalanarak oluşturulan dal üzerinde yapılan iyileştirmeler ana dalı etkilememekte ve ancak stabil bir sürüm elde edildiğinde ana dalla birleştirilmektedir.
- Bu şekilde proje kodunun yapılan değişikliklerle bozulması engellenmekte ve değişiklıkların özellik dallarında yapılması sağlanmaktadır.

Git'te Dallanma (branching)

- Özellik dalları (feature branch), kod havuzunda yapılacak değişiklikler için izole bir ortam sunar.
- Bu şekilde yapılan değişiklikler ana dalı etkilemez ve değişiklikler olgunlaşmadan ana dalla birleştirilmez.



Git Komutları

- Git programının çalıştırılmasında kullanılan çeşitli komutlar bulunmaktadır.
- Yeni bir kod deposu (repo) oluşturma
`$ git init`
- Kod deposunda değişiklik yapma
`$ git add`
`$ git commit`
`$ git status`
- Paralel geliştirme (dallara ayrılma)
`$ git branch`
`$ git merge`
`$ git rebase`
- Depoların senkron hale getirilmesi
`$ git push`
`$ git pull`
`$ git add origin`

Git kurulumu

- **git**'in bilgisayara kurulumu aşağıdaki komutla yapılabilir:
`sudo apt install git`
- Yapacağımız örnekler için GNU C Derleyicisinin de kurulması gereklidir.
`sudo apt install build-essential`
- Kurulum yapıldıktan sonra **git**'in kurulu olduğunu ve hangi sürümünün yüklediğini görmek için «**git --version**» komutu kullanılabilir.

proje dizin yapısı

- İlk önce proje için bir alt dizin oluşturmak gereklidir.
- **Bu proje dizininin içinde** eğer «`git init`» komutuyla yeni bir proje başlatıldığında, burada «`.git`» isimli gizli bir dizin oluşturulur.
- «`.git`» dizini içinde yapılandırma dosyaları ve dosyaların izlenmesi için oluşturulan dosyalar bulunmaktadır.
- Proje dizinine bir dosya eklendiğinde, `git`'in yeni dosya eklendiğinden haberi olmaz ve bu dosyanın `git`'e «`git add`» komutuyla bildirilmesi gereklidir. Bu durumda, `git` «`.git`» dizinindeki veri tabanına bu dosyayla ilgili bilgileri ekleyecektir.

.git dizini

- «**git init**» komutuyla yeni bir proje oluşturulduğunda, proje dizini altındaki «**git**» dizininde çeşitli dosyalar oluşturulmaktadır.
 - **COMMIT_EDITMSG**: Depoya gönderilen (commit) edilen bütün değişikliklerle ilgili veri içermektedir. Her «commit» bir satırda yer alır.
 - **HEAD** (metin dosyasıdır): Üzerinde çalıştığımız dal (branch) ile ilgili bilgiler içermektedir.
 - **config**: yapılandırma bilgileri bulunur. Depo sahibinin ismi, e-posta adresi gibi veriler de bulunur.
 - **description**: projeye ilgili bilgi yer alır.
 - **hooks**: projenin belirli safhalarında kullanılmak üzere yazılan betik programların bulunduğu dizindir.
 - **index**: sahneleme (staging) alanında bulunan ve «commit» edilmeyi bekleyen dosyaların takibiyle ilgili bilgileri tutar.
 - **info**: hangi dosyaların depoda yer almayıacağı bilgileri bulunan dizin.
 - **logs**: proje üzerinde yapılan işleri içerir.
 - **Objects**: Repo'ya teslim edilen dosyalardaki değişiklıkların hash'lenmiş değerlerini bulundurur. index dosyasındaki bilgilerle ilişkilidir.
 - **refs**: referans etiketlerini içeren dizindir.

Yerel git deposu oluşturma

- Ev dizininde şu komutu verelim:

```
git init ~/gps_parser
```

- Komut çalıştırıldığında şu tepkiyi verecektir:

```
Initialized empty Git repository in /home/adminpc/gps_parser/.git/
```

- «**.git**» dizinin içine bakıldığından şu dosya ve dizinler görülebilir:

```
$ ls -la .git
total 40
drwxrwxr-x 7 adminpc adminpc 4096 Mar 27 05:24 .
drwxrwxr-x 3 adminpc adminpc 4096 Mar 27 05:24 ..
drwxrwxr-x 2 adminpc adminpc 4096 Mar 27 05:24 branches
-rw-rw-r-- 1 adminpc adminpc    92 Mar 27 05:24 config
-rw-rw-r-- 1 adminpc adminpc   73 Mar 27 05:24 description
-rw-rw-r-- 1 adminpc adminpc   23 Mar 27 05:24 HEAD
drwxrwxr-x 2 adminpc adminpc 4096 Mar 27 05:24 hooks
drwxrwxr-x 2 adminpc adminpc 4096 Mar 27 05:24 info
drwxrwxr-x 4 adminpc adminpc 4096 Mar 27 05:24 objects
drwxrwxr-x 4 adminpc adminpc 4096 Mar 27 05:24 refs
```

Git yapılandırma

- «**git config**» komutu, projelerin yapılandırılması amacıyla kullanılmaktadır.
- «**git config --global**», bütün projelerde kullanılacak ortak bilgilerin girilmesini sağlar:

```
$ git config --global user.name "Hakan Senel"
$ git config --global user.email "hgsenel@gmail.com"
$ git config --global core.editor "/usr/bin/nano"
```
- Girilen ve varsayılan olarak git tarafından kullanılan bilgilerin görüntülenmesi için şu komut kullanılabilir.

```
$ git config --list
```

```
$ git config --global user.name "Hakan Senel"
$ git config --global user.email "hgsenel@gmail.com"
$ git config --global core.editor '/bin/nano'
$ git config --list
user.name=Hakan Senel
user.email=hgsenel@gmail.com
core.editor=/bin/nano
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
```

Git Yapılandırma

- Her git projesinin «**README .md**» adında, projeye ilgili bilgi sağlayan bir dosyaya sahip olması önerilir.
- Bu dosya, standart bir metin dosyası olarak oluşturulabilir. Ancak, bu dosya proje dizininde olsa bile «**git add**» komutuyla eklenmediği sürece proje dosyaları içinde yer almaz. Bu nedenle aşağıdaki komutla «**README .md**» depoya eklenmelidir:

```
$ git add README.md
```

- Bu komuttan sonra dosya artık **git** tarafından takip edilmeye başlanmıştır ve «**.git/index**» dosyasında dosyayla ilgili bilgi girilmiştir. «**cat .git/index**» komutuyla dosyanın içerisindeki **README .md** dosyasıyla ilgili bilgiler görülebilir.

Git «status» komutu

- «**git status**» komutu, proje dizini içinde çalıştırıldığında, projenin durumu ve eklenen dosyalarla ilgili bilgileri göstermektedir.
- Gösterilen bilgiler, «**staging**» yani sahneleme alanında bulunan bilgilerdir.

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md
```

Dizin ve dosya ekleme

- Proje için size verilen **gps.tar** dosyasını «**tar xvf gps.tar**» komutuyla git proje dizini içinde açın.
- «**util.h**», «**util.c**», «**gps.c**», «**gps.h**», «**config.h**», «**main.c**», «**setup.h**» isimli dosyalar, proje dizini altındaki **gps** dizininde yer alacaktır:
- Ancak «**git --status**» komutu verildiğinde, «**gps**» dizininin bulunduğu ama takip edilmediği belirtilecektir.

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   README.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    gps/
```



Kaynak Kod Dosyalarının Eklenmesi

- «gps» dizini içindeki dosyalar, henüz **git** projesine eklenmediği için aşağıdaki komutla projeye eklenebilir ve «**git status**» denilerek eklenen dosyalar görülebilir.

```
$ git add gps/*
$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file: README.md
  new file: gps/config.h
  new file: gps/gps.c
  new file: gps/gps.h
  new file: gps/main.c
  new file: gps/setup.h
  new file: gps/util.c
  new file: gps/util.h
```

«git status -s»

- Projede olmayan bir dosayı, **LICENSE** dosyasını «**touch LICENSE**» diyerek oluşturalım.
- «**git status**» komutu, **LICENSE** dosyasının henüz «**staging**» alanına eklenmediğini gösterecektir.
- «**git status -s**» komutu kısa formatta eklenen ve henüz eklenmemiş dosyaları göstermektedir. «**git add**» komutuyla **LICENSE** dosyası repo'ya eklenebilir.

```
$ git status -s
A  README.md
A  gps/config.h
A  gps/gps.c
A  gps/gps.h
A  gps/main.c
A  gps/setup.h
A  gps/util.c
A  gps/util.h
?? LICENSE
```



```
$ git add LICENSE
$ git status -s
A  README.md
A  gps/config.h
A  gps/gps.c
A  gps/gps.h
A  gps/main.c
A  gps/setup.h
A  gps/util.c
A  gps/util.h
A  LICENSE
```

Dosya Üzerinde Değişiklik

- Herhangi bir dosyanın içerisinde değişiklik yapılması durumunda, dosyanın zaman damgaları da değişmekte ve **git** bunları takip etmektedir.
- «**git status -s**» veya «**git status**» komutuyla, son kod tesliminden bu yana dosyada değişiklik yapıldığı görülebilir.

```
$ cat > LICENSE
License: GPS v3 GPLv3
$ git status -s
AM LICENSE
A README.md
A gps/config.h
A gps/gps.c
A gps/gps.h
A gps/main.c
A gps/setup.h
A gps/util.c
A gps/util.h
```

M: Modified

A: Added

Dosya Üzerinde Değişiklik

- Dosyada yapılan ama henüz «sahneleme» (staging) alanına eklenmeyen değişiklik, «**git add LICENSE**» denilerek, «**commit**» edilebilmesi için sahneleme yani «**staging**» bölümüne eklenmelidir.

```
$ git status
On branch master

No commits yet

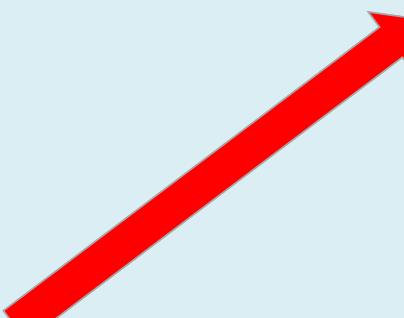
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:  LICENSE
  new file:  README.md
  new file:  gps/config.h
  new file:  gps/gps.c
  new file:  gps/gps.h
  new file:  gps/main.c
  new file:  gps/setup.h
  new file:  gps/util.c
  new file:  gps/util.h

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
  modified:   LICENSE
```

```
$ git add LICENSE
$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:  LICENSE
  new file:  README.md
  new file:  gps/config.h
  new file:  gps/gps.c
  new file:  gps/gps.h
  new file:  gps/main.c
  new file:  gps/setup.h
  new file:  gps/util.c
  new file:  gps/util.h
```



«verbose status»

- «**staging**» alanında bulunan dosyalarda ne değişiklikler yapıldığı «**git status -v**» komutuyla ayrıntılı şekilde görülebilir.
- Yapılan değişiklikler, oluşan farklılıklar, eklenen satırlar ve çıkarılan bölümler hepsi belirli bir formatta gösterilmektedir.

```
$ git status -v
On branch master

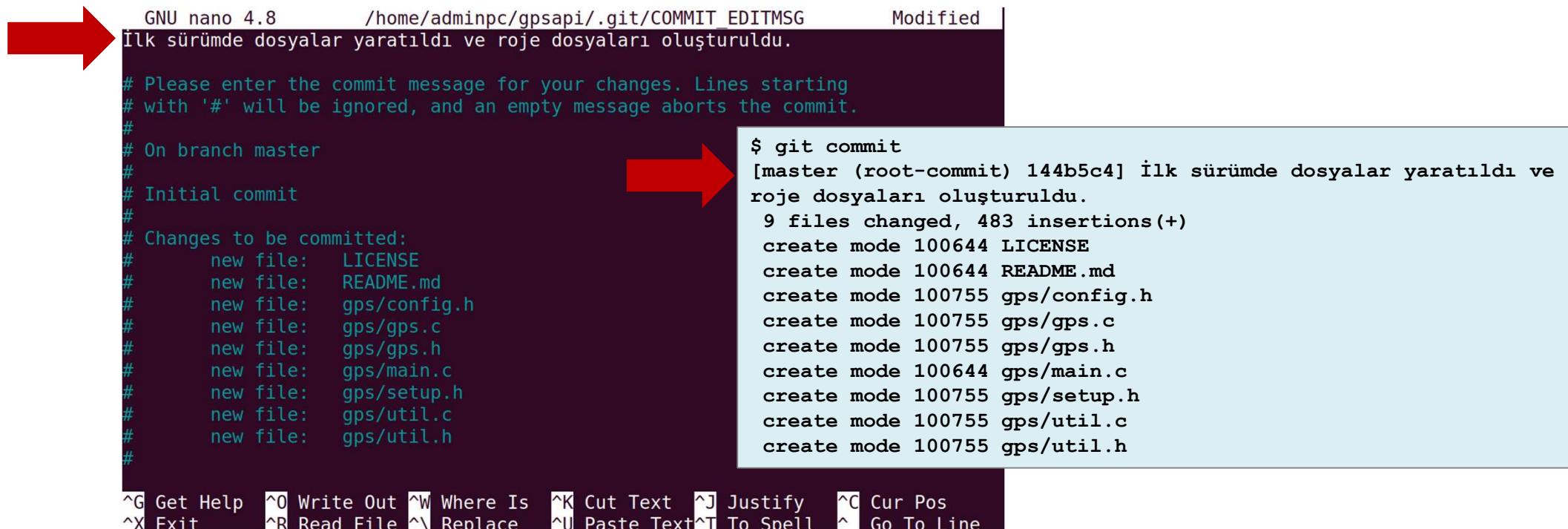
No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file:  LICENSE
  new file:  README.md
  new file:  gps/config.h
  new file:  gps/gps.c
  new file:  gps/gps.h
  new file:  gps/main.c
  new file:  gps/setup.h
  new file:  gps/util.c
  new file:  gps/util.h

diff --git a/LICENSE b/LICENSE
new file mode 100644
index 000000..82019c8
--- /dev/null
+++ b/LICENSE
@@ -0,0 +1 @@
+License: GPS v3 GPLv3
diff --git a/README.md b/README.md
```

Kod Teslimi «commit»

- «işleme alma» yani «commit» projelerdeki dosyalarda değişiklik yapılmasıından sonra ve bu değişikliklerin kod havuzuna teslim edilmesi sürecine verilen isimdir.
- «git commit» komutu verildiğinde, bir editör penceresi açılır ve «teslim» süreciyle ilgili bilgi girişi istenir.



The screenshot shows a terminal window with two main sections. On the left, the terminal is in nano editor mode, displaying a commit message template. On the right, the terminal has run a git commit command, showing the resulting commit log.

```

GNU nano 4.8          /home/adminpc/gpsapi/.git/COMMIT_EDITMSG      Modified
→ İlk sürümde dosyalar yaratıldı ve roje dosyaları oluşturuldu.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:  LICENSE
#   new file:  README.md
#   new file:  gps/config.h
#   new file:  gps/gps.c
#   new file:  gps/gps.h
#   new file:  gps/main.c
#   new file:  gps/setup.h
#   new file:  gps/util.c
#   new file:  gps/util.h
#
$ git commit
[master (root-commit) 144b5c4] İlk sürümde dosyalar yaratıldı ve
roje dosyaları oluşturuldu.
9 files changed, 483 insertions(+)
create mode 100644 LICENSE
create mode 100644 README.md
create mode 100755 gps/config.h
create mode 100755 gps/gps.c
create mode 100755 gps/gps.h
create mode 100644 gps/main.c
create mode 100755 gps/setup.h
create mode 100755 gps/util.c
create mode 100755 gps/util.h

```

→ Get Help → Write Out → Where Is → Cut Text → Justify → Cur Pos
 → Exit → Read File → Replace → Paste Text → To Spell → Go To Line

Kod Teslimi «commit»

- «teslim» yani «commit» sürecinden sonra «`.git/COMMIT_EDITMSG`» dosyasına «commit» süreciyle ilgili bilgiler yazılır.
- Her «commit»te dosyaya yeni bilgiler eklenir.
- «`git log`» komutu da alınan logları gösterir.

```
$ cat .git/COMMIT_EDITMSG
ilk sürümde dosyalar yaratıldı ve proje dosyaları oluşturuldu.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:  LICENSE
#   new file:  README.md
#   new file:  gps/config.h
#   new file:  gps/gps.c
#   new file:  gps/gps.h
#   new file:  gps/main.c
#   new file:  gps/setup.h
#   new file:  gps/util.c
#   new file:  gps/util.h
#
```

```
$ git log
commit 144b5c41356d22487602f780df5b763ba031c027 (HEAD -> master)
Author: Hakan Senel <hgsenel@gmail.com>
Date:   Mon Mar 27 05:52:09 2023 -0700

ilk sürümde dosyalar yaratıldı ve proje dosyaları oluşturuldu.
```

Kod Teslimi «commit»

- «teslim» yani «commit» sürecinden sonra «git log» komutuyla oluşan hash değeri görülebilir.
- Dosyalardaki değişikliklerle ilgili bilgiler (hash değerinin ilk iki harfi «.git/objects/» dizinin altındaki dizinde görülebilir (.git/objects/96)

```
$ git log  
commit 144b5c41356d22487602f780df5b763ba031c027 (HEAD -> master)  
Author: Hakan Senel <hgsenel@gmail.com>  
Date:   Mon Mar 27 05:52:09 2023 -0700  
  
    İlk sürümde dosyalar yaratıldı ve projeler dosyaları oluşturuldu.
```

```
$ ls .git/objects/14/  
4b5c41356d22487602f780df5b763ba031c027
```

Metin dosyası değildir

Kod Teslimi (işleme al) «commit»

- «**git commit**» komutu, yapılan değişikliklerin sahneleme (staging) alanına aktarılmasını sağlar. Teslim yapılmazsa, kodda yapılan değişiklikler uzaktaki depoya aktarılmaz.



Yeni Dosya Ekleme

- İlk kod teslimimizi (commit) gerçekleştirdikten sonra, **gps_data.tar** dosyasını proje dizini içinde «**tar xvf gps_data.tar**» diyerek açacağız ve dosyaları «**git add**» ile ekleyeceğiz.

```
$ tar xvf ../gps_data.tar
gps_data/
gps_data/gps_data.txt
$ ls
gps  gps_data  LICENSE  README.md
$ git add gps_data/*
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   gps_data/gps_data.txt
```

«Hızlı» Kod Teslimi

- «teslim» süreci «**git commit**» komutuyla yapılmakta ve «teslim» işleminin amacının girilmesi için bir editör penceresi açılmaktadır.
- Bu süreci hızlı geçmek ve mesajı editör olmadan vermek için «**git commit -m "açıklama"**» komutuyla açıklama girilebilir.
- İstenirse, git'ten **gps_data.txt** dosyasının takip edilmemesi «**git rm --cache dosyaismi**» komutuyla sağlanabilir. Dosya, proje dizininde duracaktır. Yeniden «**git commit**» yapılmalıdır.

```
$ git commit -m "test için gps verisi de eklendi"
[master aa9c7c8] test için gps verisi de eklendi
 1 file changed, 408 insertions(+)
  create mode 100755 gps_data/gps_data.txt
$ git rm --cached gps_data/gps_data.txt
rm 'gps_data/gps_data.txt'
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:   gps_data/gps_data.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    gps_data/
```

Takip edilmeyecek dosyaların girişi

- Proje dizini içinde bulunan bazı dosyaların (örneğin derleme sırasında oluşacak object ve çalıştırılabilir dosyaların bulunduğu **build** dizini) git ile depolanmaması ve takip edilmemesi iyi olur.
- Bunların git tarafından takip edilmemesini sağlamak için, her satırında takip edilmeyecek dosya isimlerini bulunduran «**.gitignore**» dosyası proje dizininde oluşturulmalı ve «**git add**» ile projeye eklenmeli ve ardından «**git commit**» denilerek teslim edilmelidir.
- **.gitignore** içinde «wildcard» kullanarak dosya isimleri tanımlanabilir.

```
$ nano .gitignore
$ git add .gitignore
$ git commit -m "gitignore dosyası eklendi"
[master 335abb5] gitignore dosyası eklendi
 2 files changed, 3 insertions(+), 408 deletions(-)
 create mode 100644 .gitignore
 delete mode 100755 gps_data/gps_data.txt
```

.gitignore

```
build/*
gps_data/*
```

«build» dizininde derlemesi

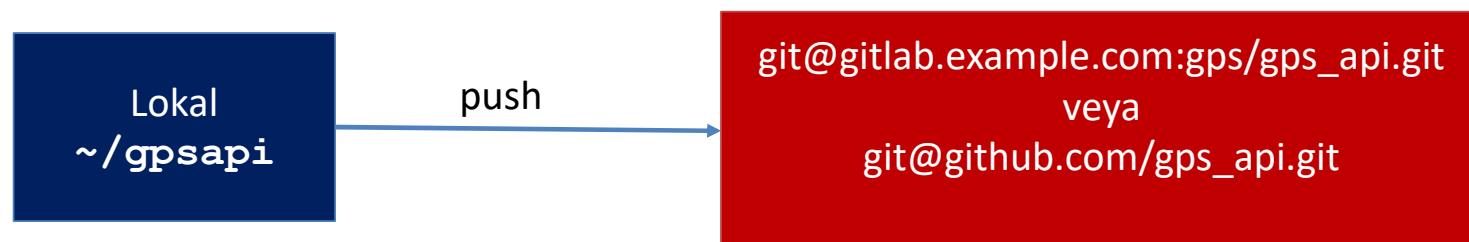
- Projemizin EXE dosyasını «~/gpsapi/build» dizininde oluşturmak için, ilk önce «**mkdir -p build**» komutuyla, **build** dizini oluşturmamız gerekiyor.
- Derlemek için aşağıdaki komut kullanılabilir:
**gcc -o build/gps_parser gps/main.c gps/gps.c
gps/util.c**
- «**build/gps_parser**» denilerek EXE dosyası çalıştırılabilir.

```
$ mkdir -p build
$ gcc -o build/gps_parser gps/main.c gps/gps.c
$ build/gps_parser
hello World!
$ git status
On branch master
nothing to commit, working tree clean
```



«git remote add origin»

- Lokalde «**git init**» ile oluşturduğunuz repoyu, Github'daki reponuza veya Gitlab'daki projenize aktarmak istiyorsanız, «**git remote add origin git@github.com:kullanıcı/repoadı.git**» komutuyla Gitlab veya Github'daki git adresinin verilmesi gereklidir.
- «**ssh-keygen**» programıyla, SSH anahtarlarının oluşturulması ve Github.com'da Settings bölümüne bu anahtarın eklenmesi gereklidir.
- Bu komutu, Gitlab'la ilgili sunumda ayrıntısıyla anlatacağız.



git'te etiket (tag) kavramı

- Git, etiketler sayesinde, yapılan bir kod tesliminin (commit) isimlendirilmesini sağlar.
- Git'te iki adet etiket bulunur:
 - Açıklamalı etiket (annotated tag)
 - Hafif etiket (lightweight tag)
- Açıklamalı **tag** tercih edilen türdür. «**-a**» açıklamalı **tag** olduğunu söylerken, «**-m**» opsiyonu etikete açıklama ekler.
`git tag -a v0.9 -m "ilk surum"`
- «**git tag**» komutu projedeki etiketleri listelememizi sağlar.

<https://devconnected.com/how-to-create-git-tags/>

git'te etiket (tag) kavramı

- «**git show v0.9**» v0.9 etiketiyle ilgili yapılanları gösterir.

```
$ git tag -a v0.9 -m "ilk RC"
$ git tag
v0.9
$ git show v0.9
tag v0.9
Tagger: Hakan Senel <hgsenel@gmail.com>
Date:   Mon Mar 27 07:05:40 2023 -0700

ilk RC

commit 335abb5957aa19e8a845721cd1952a413ba44a84 (HEAD -> master, tag: v0.9)
Author: Hakan Senel <hgsenel@gmail.com>
Date:   Mon Mar 27 06:57:38 2023 -0700

    gitignore dosyası eklendi

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..b77343a
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,3 @@
+build/*
+gps_data/*
+
diff --git a/gps_data/gps_data.txt b/gps_data/gps_data.txt
deleted file mode 100755
```

git'te etiket (tag) kavramı

- Açıklamalı etiket, git veri tabanında tam nesneler olarak depolanır ve ekstra metaveriye sahiptir: isim, e-posta, zaman/tarih, «hash» vb.
- Hafif etiketler (lightweight tags) «**-a**» opsiyonu olmadan «**git tag etiketisim**» komutuyla oluşturulurlar.
- Hafif etiketler geçici olarak ve hatırlatma amacıyla oluşturulur ve kullanılır.
- «**git tag -d etiketismi**» komutuyla silinebilirler.

git'te dal (branch) kavramı

- Şu ana kadar gördüğümüz işlemlerin tamamı ana dal (master branch) üzerinde yapılan işlemlerdi.
- Teslim (commit) işlemleriyle, ana daldaki kaynak kodları güncellenmektedir.
 - **Soru:** Ana dal üzerinde yapılan değişikliklerin geri alınması istendiğinde ve bir önce commit'teki duruma geri dönmek gerekiğinde ne olacak?
 - **Cevap:** O zaman, ana dalın bir kopyasını çıkarıp ayrı bir dal haline getirelim ve bunun üzerinde geliştirmeleri yapalım. Daldaki kod olgunlaşlığında, ana dalla birleştirelim.
 - **Neden:** Kodun kirlenme ve yapılanların kaybedilme riski var. Sonuçlanmayacak denemeleri ana dal üzerinde yapmayağım.

Dal (branch) oluşturma

- «**git branch**» komutu ile yeni bir dal oluşturulabilir.
- «**git status**» komutuyla görüleceği üzere yeni dal oluşturulmasına rağmen, commit'ler «master» dalda alınmaktadır.
- Yeni dal üzerinde kod teslimi yapabilmek için «**git checkout yenidalismi**» komutunun verilmesi gereklidir.

```
$ git branch yenidal
$ git status
On branch master
nothing to commit, working tree clean
$ git checkout yenidal
Switched to branch 'yenidal'
$ git status
On branch yenidal
nothing to commit, working tree clean
```

Yeni dalda kod değişikliği

- «gps/main.c» dosyası üzerinde değişiklik yapalım.

```
GNU nano 4.8                      gps/main.c                      Modified
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("GPS parser test program\n");
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Tex ^T To Spell

Yeni dalda teslim (commit)

- «**yenidal**» isimli dalda teslim yaparak «**gps/main.c**» dosyasındaki değişikliğin repoya eklenmesini sağlayacağız.
- Aşağıdaki komutla değişiklikleri yeni dala aktaracağız.
`git commit -a -m "main programı degisti"`

```
$ nano gps/main.c
$ git commit -a -m "main programı degisti"
[yenidal 5bdb5b1] main programı degisti
 1 file changed, 1 insertion(+), 1 deletion(-)
```

«master» dala dönüş

- «yenidal»dan «master» dala «git checkout master» denilerek dönülebilir.
- «yenidal»da yapılan `gps/main.c`'de yapılan değişiklik «master» dalda görünmeyecektir.
- «`.git/HEAD`» hangi dalın aktif olduğunu belirtir.

```
$ git checkout master
Switched to branch 'master'
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("Hello World!\n");
}
$ cat .git/HEAD
ref: refs/heads/master
```

«yenidal» dalına geçiş

- «**master**» dan «**yenidal**»a «**git checkout yenidal**» komutuyla dönüldüğünde, **gps/main.c**'de yapılan değişiklik görülebilir.
- «**.git/HEAD**» hangi dalın aktif olduğunu belirtir.

```
$ git checkout yenidal
Switched to branch 'yenidal'
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("GPS parser test program\n");
}
$ cat .git/HEAD
ref: refs/heads/yenidal
```

Dalların birleştirilmesi (merge)

- Özellik dalında yapılan değişikliklerin kod olgunlaşımından bir süre sonra «**master**» dalıyla birleştirilmesi gereklidir.
- Her dalda son yapılan teslimin (commit) tutulduğu dizin «**.git/refs/heads**» dizinidir. Dizin altında yer alan dosyaların içinde son yapılan teslimlerin **hash** değerleri bulunmaktadır.

```
$ cat .git/refs/heads/master
335abb5957aa19e8a845721cd1952a413ba44a84
$ cat .git/refs/heads/yenidal
5bdb5b1c4fd606043c66f46a22cd69125594b7b5
```

Dalların birleştirilmesi (merge)

 «**yenidal**» dalını «**master**» dalıyla birleştireceksek, öncelikli olarak «**master**» dalında olmaya dikkat edeceğiz.

- Eklenecek dalda olduğumuzu anlamak için «**git status**» komutu verilerek, «**master**» dalında olduğumuz teyit edilebilir. Yoksa «**git checkout master**» diyerek olmamız gereken dala geçmeliyiz.

```
$ git status
On branch yenidal
nothing to commit, working tree clean
$ git checkout master
Switched to branch 'master'
```

Dalların birleştirilmesi (merge)

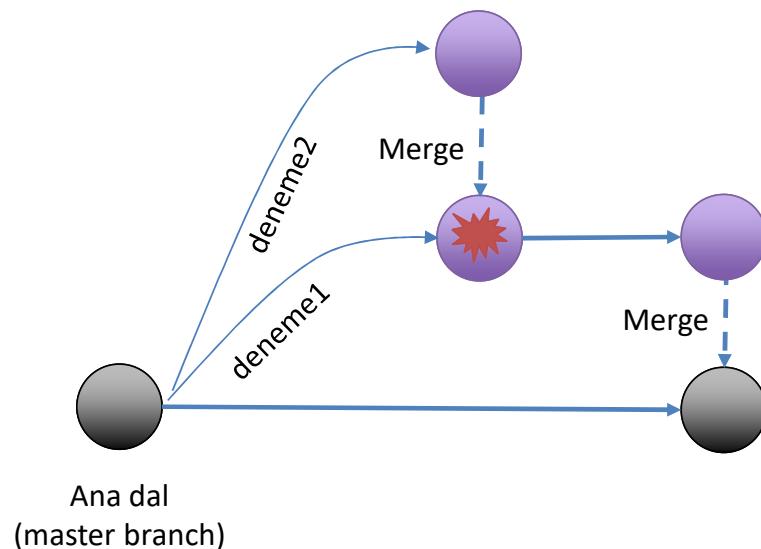
- «**yenidal**» dalını «**master**» dalıyla birleştireceksek, «**master**» dalındayken, «**git merge yenidal**» diyerek, **yenidal** dalını **master** dalıyla birleştireceğiz.

```
$ git merge yenidal
Updating 335abb5..5bdb5b1
Fast-forward
  gps/main.c | 2 ++
  1 file changed, 1 insertion(+), 1 deletion(-)
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("GPS parser test program\n");
}
```

Örnek: İki dalın birleştirilmesinde çakışma

- «**deneme1**» ve «**deneme2**» isimli iki dal oluşturacağız ve her iki dalda da **gps/main.c** dosyasını değiştirerek çakışma olmasını sağlayacağız ve sonra çakışmayı engelleyeceğiz.



Örnek: İki dalın birleştirilmesinde çakışma

- İki yeni dal oluşturulursa (mesela «**deneme1**» ve «**deneme2**») ve her iki dalda da aynı kaynak kod dosyasının aynı satırında değişiklikler yapılrsa, bu iki dalın birleştirilmesinde «birleştirmede çakışma» hatası verecek ve git bu işlemi yapamayacaktır.

```
$ git branch deneme1
$ git branch deneme2
$ git checkout deneme1
Switched to branch 'deneme1'
$ nano gps/main.c ←
$ git commit -a -m "deneme1"
[deneme1 b6d4b82] deneme1
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git checkout deneme2
Switched to branch 'deneme2'
$ nano gps/main.c ←
$ git commit -a -m "deneme2"
[deneme2 54294df] deneme2
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git checkout deneme1
Switched to branch 'deneme1'
$ git merge deneme2
Auto-merging gps/main.c
CONFLICT (content): Merge conflict in gps/main.c
Automatic merge failed; fix conflicts and then commit the result.
```

gps/main.c dosyasında
`printf("deneme1");`
yazılacak.

gps/main.c dosyasında
`printf("deneme2");`
yazılacak.



Örnek: İki dalın birleştirilmesinde çakışma

- Çakışma nedeniyle iki dalın birleştirilememesi sorununun çözümü iki şekilde gerçekleştirilebilir:
 - Birleşme iptal edilebilir (`git merge --abort` komutuyla)
 - Çatışan dosyalara bakılarak, nasıl birleştirilebileceği konusunda bir çözüm bulunabilir. Çakışma olan dosyaya bakılabilir ve düzeltmeler yapılabilir.

```
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
<<<<<< HEAD
    printf("deneme1");
=====
    printf("deneme2");
>>>>> deneme2
}
```

Birleştirilecek ana dal
(deneme1)

Ana dala katılacak dal

Örnek: İki dalın birleştirilmesinde çakışma

- «**nano gps/main.c**» ile girilerek, dosya düzeltilebilir.

```
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
<<<<< HEAD
    printf("deneme1");
=====
    printf("deneme2");
>>>>> deneme2
}
$ nano gps/main.c
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("deneme (duzeltildi)");
}
```

Çatışmanın belirtildiği hali

Düzeltilmiş hali

Birleştirme sırasında çakışma

- Çatışmanın sonlanması için «**git add gps/main.c**» yazılmalı ve «**git commit -m "çatışma halledildi"**» ile dosyanın son hali teslim edilmelidir.
- Daha sonra «**master**» dalla **deneme1** birleştirilebilir.

```
$ git add gps/main.c
$ git commit -m "çatışma halledildi"
[deneme1 1f9317a] çatışma halledildi
$ git checkout master
Switched to branch 'master'
$ git merge master
Already up to date.
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("Hello World!\n");
}
```

Dalların silinmesi

- «**deneme1**» ve «**deneme2**» dallarını birleştirdik, çatışmayı çözdük ve «**master**» dalıyla birleştirdik.
- Artık «**deneme1**» ve «**deneme2**» dallarına ihtiyaç duymuyoruz. «**merge**» edilmese de silmek için «**-D**» opsiyonunu kullanıyoruz. Birleştirilmesi aşağıdaki komutlar kullanılabilir.

```
git branch -D deneme1
```

```
git branch -D deneme2
```

```
$ git branch -D deneme1
Deleted branch deneme1 (was 1f9317a) .
$ git branch -D deneme2
Deleted branch deneme2 (was 54294df) .
```

rebase (merge alternatifi)

- Birleştirme (**merge**) işleminde, birleştirilecek iki dal da eşit ağırlıktadır ve bu nedenle önceki örnekte çakışma yaşanmıştır.
- **SORU:** bir dalı daha baskın yapsak ve çakışma durumunda baskın olan daldaki değişiklige öncelik versek?
- **CEVAP:** Bir dalda herhangi bir dosyadaki değişikliğin diğerinin üzerine yapıldığı birleştirme işlemi, «**git rebase**» komutuyla yapılabilmektedir.
- «**rebase**» komutu, loglarda daha temiz çıktı üretmektedir.
- «**rebase**» veya «**merge**» komutlarının hangisinin tercih edileceği yazılım şirketindeki iş kurallarına bağlıdır.

«teslim»in geri alınması

- Bazı durumlarda, hata çıktığında veya beklemedik bir arıza oluştuğunda, son kod tesliminin (commit) geri alınması gerekebilir.
- «git» kullanımının sağladığı önemli avantajlardan biri, değişikliklerin geri alınması ve kod geliştirme işinde belirli bir ana geri dönüşe izin vermesidir.
- Bunu gösterebilmek için «gps/main.c» üzerinde bir değişiklik yapalım ve bir satır ekleyelim ve «git commit -a» yapalım.

```
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    // yorum satiri eklendi
    printf("hello World!\n");
}
```

```
$ nano gps/main.c
$ git commit -a -m "yorum satiri eklendi"
[master bde04cc] yorum satiri eklendi
 1 file changed, 1 insertion(+)
```

«commit»in geri alınması

- Son yaptığımız kod teslimini (commit) geri almak için «git revert HEAD» komutunu kullanmalıyız.

```
$ git revert HEAD
[master 7631595] Revert "yorum satiri eklendi"
 1 file changed, 1 deletion(-)
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("Hello World!\n");
}
$ git log --oneline
7631595 (HEAD -> master) Revert "yorum satiri eklendi"
bde04cc yorum satiri eklendi
57e3e48 git dosyaları repoya eklendi
```



«commit»in geri alınması

- «git revert» komutu, sadece son yapılan kod teslimini (commit) geri almak için kullanılmaz.
- Kod geliştirme sürecinin belirli bir anına dönüşümümüzü sağlayabilir.
- «git revert HEAD~4» komutu, son teslimden itibaren 4 teslim öncesine götürür.

Dönülebilecek
Teslimler
(commit)

```
adminpc@ubuntu:~/gps_parser$ git log --oneline
7143837 (HEAD -> master) Revert "yorum eklendi"
e44d671 yorum eklendi
5e44907 catisma halledildi
5a3d490 deneme2
97e6e8f deneme1
769c8e1 (yenidal, show) main programi degisti
07a754f (tag: v0.9) ignore dosyasi olusturuldu
2e3b304 veri dosyasi silindi
220c1d6 test icin gps verisi eklendi
960b20a ilk surumde dosyalar yaratildi ve proje dosyaları hazırlandı.
```

«git diff» komutu

- «git diff» komutu, teslimler arasında nelerin değiştiğini gösterir ve bu nedenle oldukça yararlı bir komuttur.
- Örneğin, «gps/main.c» dosyasında bir değişiklik yapalım ve kod dosyasına «#include <math.h>» satırını ekleyelim.
- «git diff» komutu yapılan bütün değişiklikleri özel bir formatta gösterecektir.

```
$ nano gps/main.c
$ cat gps/main.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "config.h"
#include "setup.h"
#include "gps.h"

void main(void)
{
    printf("Hello World!\n");
}
$ git diff
diff --git a/gps/main.c b/gps/main.c
index 61b830d..08d0a0e 100644
--- a/gps/main.c
+++ b/gps/main.c
@@ -1,5 +1,6 @@
 #include <stdio.h>
 #include <stdlib.h>
+#+include <math.h>
 #include "config.h"
 #include "setup.h"
 #include "gps.h"
```

«git diff» komutu

- «git diff» komutuna verilecek iki teslim «hash» değeriyle, iki teslim arasındaki farkları gösterebilir.
- «hash» değerleri «git log --oneline» komutuyla görülebilir.

```
$ git commit -a -m "math.h eklendi"
[master d481b0c] math.h eklendi
 1 file changed, 1 insertion(+)
$ git log --oneline
d481b0c (HEAD -> master) math.h eklendi
7631595 Revert "yorum satiri eklendi"
bde04cc yorum satiri eklendi
57e3e48 git dosyaları repoya eklendi
$ git diff 57e3e48 d481b0c
diff --git a/gps/main.c b/gps/main.c
index 61b830d..08d0a0e 100644
--- a/gps/main.c
+++ b/gps/main.c
@@ -1,5 +1,6 @@
 #include <stdio.h>
 #include <stdlib.h>
+#include <math.h>
 #include "config.h"
 #include "setup.h"
 #include "gps.h"
```

Çöp Temizleme

- Özellikle büyük yazılım geliştirme projelerinde, çok sayıda dal kullanımı ve sık yapılan dosya değişiklikleri, **git** veritabanını gereksiz yere büyütебilir.
- **git** veritabanı içinde kullanılmayan ve hiçbir yerde referansı olmayan nesnelerin temizlenmesine «çöp toplama» veya «çöp temizleme» yanı «**garbage collection**» adı verilir.
- Çöp temizleme için «**git gc**» komutu kullanmaktadır. Bu komut, kullanılmayacak kadar eski nesneleri silmekte ve bazı içerikleri sıkıştırarak diskte yer açmaktadır.
- «**git gc --prune**» komutu, istenen bir zamandan öncesini (varsayılan olarak 2 hafta sürede) silmektedir.

```
$ git gc --prune
Enumerating objects: 34, done.
Counting objects: 100% (34/34), done.
Delta compression using up to 8 threads
Compressing objects: 100% (31/31), done.
Writing objects: 100% (34/34), done.
Total 34 (delta 18), reused 0 (delta 0)
```

Çöp Temizleme

- Çöp temizliğine ihtiyaç olup olmadığını görmek için «**git gc --auto**» komutu kullanılabilir. Eğer herhangi bir çıktı dönmediyse, yapılacak herhangi bir temizlik bulunmadığı anlamına gelir.
- «**--prune**» opsiyonuyla ne kadar süre öncesine kadar silme işlemi yapılacağı belirtilebilir. Örneğin 60 gün öncesine kadar silinecekse «**git config gc.pruneexpire "60 days"**» komutuyla gerçekleştirilebilmektedir. Böylelikle, **git** otomatik olarak 60 gün öncesinden daha eski işlemleri otomatik olarak silecektir.

```
$ git gc --auto
$ git config gs.pruneexpire "60 days"
```

«git log» komutu

- Log gösterme komutu, şu ana kadar kullandığımız kadarıyla, projede hangi aşamalarda ne yapıldığını gösterdiği için faydalı bir komuttur.
- Ancak bu komutun farklı opsiyonları bulunmaktadır.
- Örneğin, «**git log --graph**» komutu, log bilgilerini daha açıklayıcı şekilde gösterir.
- Bütün logları değil de belirli bir tarih/zamandan sonrakileri göstermesi (örneğin 10 gün öncesinden itibaren) istenirse, «**git log --since "10 days ago"**» komutu kullanılabilir.
- Kod teslimlerindeki açıklamalarda arama yaparak, belirli bir kelime ve kelime grubunun geçtiği teslimleri göstermek için «**git log -S kelime**» komutu kullanılabilir.
- İstatistiksel bilgiler almak için «**git log --stat**» veya «**git log --shortstats**» komutları kullanılabilir.

«git log --pretty=format»

- Log kayıtlarının formatlı ve isteğe bağlı verilerle oluşturulması istendiğinde «git log --pretty=format» komutu kullanılabilir.

```
adminpc@ubuntu:~/gps_parser$ git log --pretty=format:"%h %an %s"
7143837 Hakan Senel Revert "yorum eklendi"
e44d671 Hakan Senel yorum eklendi
5e44907 Hakan Senel catisma halledildi
5a3d490 Hakan Senel deneme2
97e6e8f Hakan Senel deneme1
769c8e1 Hakan Senel main programi degisti
07a754f Hakan Senel ignore dosyasi olusturuldu
2e3b304 Hakan Senel veri dosyasi silindi
220c1d6 Hakan Senel test icin gps verisi eklendi
960b20a Hakan Senel ilk surumde dosyalar yaratildi ve proje doosyalar hazirlandi.
```

%h short hash
%an author name
%s açıklama

Lokal depoyu lokale klonlamak

- Kod geliştirme sırasında bir kod deposunun klonlanması ve değişik bir isimle üzerinde çalışılması gerekebilir.
- Var olan bir git deposunu diğer bir depoya klonlamak için kullanılacak komut:
`$ git clone «lokal repo» «yeni repo»`
- Yeni kod deposunun dizinine gidilerek işlem yapılabilir.

Herhangi bir git repo
dizini içinde
olmamalıyız.



```
$ git clone gpsapi gpsapi_deneme
Cloning into 'gpsapi_deneme'...
done.
$ ls gpsapi_deneme
gps LICENSE README.md
$ cd gpsapi_deneme
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Uzaktaki depoyu klonlamak

- Github, bir çok açık kaynak kodlu projeye ev sahipliği yapan bir web sitesidir ve **git** sistemini desteklemektedir.
- Github'daki bir depoyu klonlamak için, kod deposunun **http** adresini bulmak gereklidir.
- Örneğin aşağıdaki komutla, git komutunun çalıştırıldığı dizinde, **FakeSMTP** diye bir dizin oluşturulmakta ve uzaktaki depo buraya klonlanmaktadır.

```
$ git clone https://github.com/Nilhcem/FakeSMTP.git
```

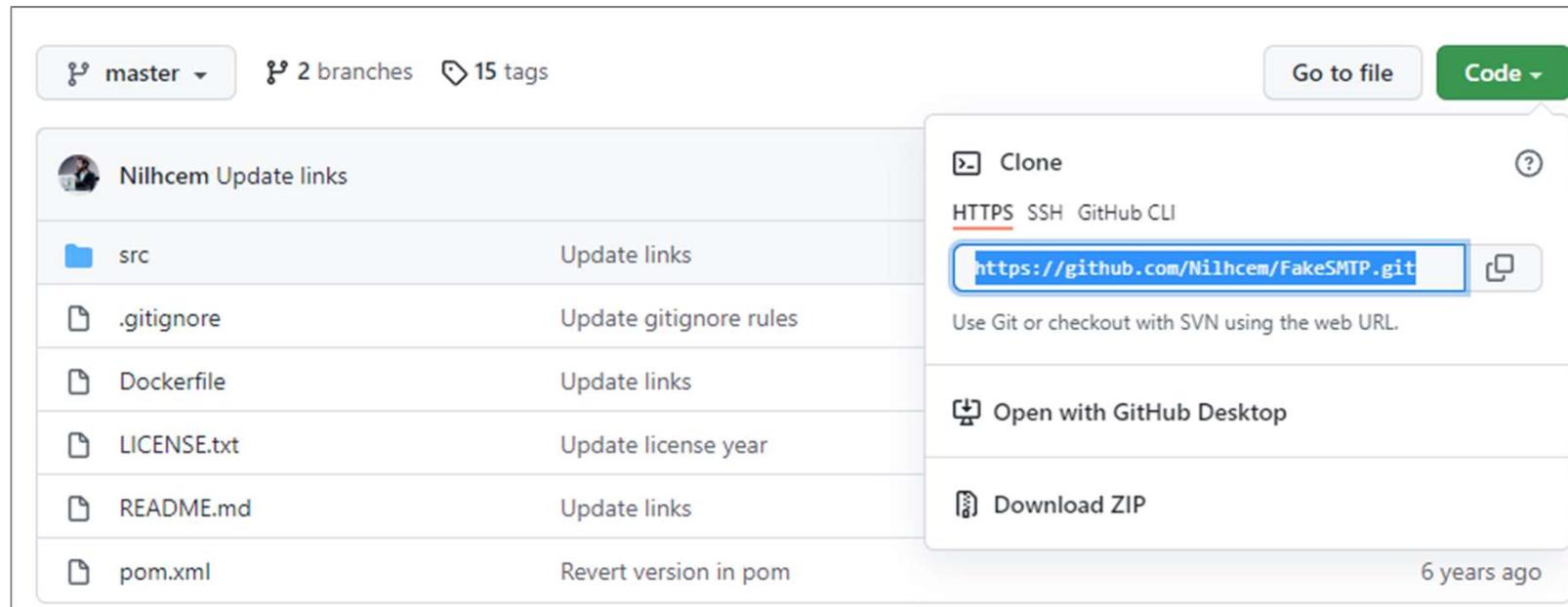
```
$ git clone https://github.com/Nilhcem/FakeSMTP.git
Cloning into 'FakeSMTP'...
remote: Enumerating objects: 5971, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 5971 (delta 4), reused 14 (delta 4), pack-reused 5945
Receiving objects: 100% (5971/5971), 9.60 MiB | 5.13 MiB/s, done.
Resolving deltas: 100% (3896/3896), done.

$ cd FakeSMTP/
$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Uzak depo URL adresi

- Github'da uzak depo URL adresi, «Code» butonuna basılarak bulunabilir.



Çatallanma «forking»

- Yazılım geliştirme projelerinde bazı durumlarda olan projenin klonlanması ve farklı kullanıcılar tarafından geliştirme yapılması istenebilir.
- Örneğin, Raspberry Pi için yapılan Linux çalışmasının, klonlanarak, bu klonun Raspberry Pi v4'ye uyarlanması için kullanılması durumunda bir çatallanma sürecine ihtiyaç duyulabilir.
- Örneğin, bir video IC'si için geliştirilen bir sürücü projesinin, başka bir sürüm video IC'si için klon üzerinde geliştirme yapılması
- Örneğin, geliştirme yapılmayan ama geliştirme gereken bir projenin, başkaları tarafından sürdürülmesinin istenmesi ve bu nedenle çatallanma yapılması
- Çatallanan proje aslında orijinal projenin bütün veritabanlarını ve nesnelerini devralmaktadır.

Git Cheat Sheet

- Aşağıdaki adressteki pdf'in çıktısını alarak, yanınızda bulundurmalısınız:
 - <https://about.gitlab.com/images/press/git-cheat-sheet.pdf>

Git Cheat Sheet



01 Git configuration

```
$ git config --global user.name "Your Name"
```

Set the name that will be attached to your commits and tags.

```
$ git config --global user.email "you@example.com"
```

Set the e-mail address that will be attached to your commits and tags.

```
$ git config --global color.ui auto
```

Enable some colorization of Git output.

02 Starting A Project

```
$ git init [project name]
```

Create a new local repository. If [project name] is provided, Git will create a new directory name [project name] and will initialize a repository inside it. If [project name] is not provided, then a new repository is initialized in the current directory.

```
$ git clone [project url]
```

Downloads a project with the entire history from the remote repository.

03 Day-To-Day Work

```
$ git status
```

Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.

```
$ git add [file]
```

Add a file to the **staging** area. Use in place of the full file path to add all changed files from the **current directory** down into the **directory tree**.

```
$ git diff [file]
```

Show changes between **working directory** and **staging area**.

```
$ git diff --staged [file]
```

Shows any changes between the **staging area** and the **repository**.

```
$ git checkout -- [file]
```

Discard changes in **working directory**. This operation is **unrecoverable**.

```
$ git reset [file]
```

Revert your **repository** to a previous known working state.

```
$ git commit
```

Create a new **commit** from changes added to the **staging area**. The **commit** must have a message!

<https://about.gitlab.com/images/press/git-cheat-sheet.pdf>

```
$ git rm [file]
```

Remove file from **working directory** and **staging area**.

```
$ git stash
```

Put current changes in your **working directory** into **stash** for later use.

```
$ git stash pop
```

Apply stored **stash** content into **working directory**, and clear **stash**.

```
$ git stash drop
```

Delete a specific **stash** from all your previous **stashes**.

04 Git branching model

```
$ git branch [-a]
```

List all local branches in repository. With **-a**: show all branches (with remote).

```
$ git branch [branch_name]
```

Create new branch, referencing the current **HEAD**.

```
$ git checkout [-b][branch_name]
```

Switch **working directory** to the specified branch. With **-b**: Git will create the specified branch if it does not exist.

```
$ git merge [from name]
```

Join specified **[from name]** branch into your current branch (the one you are on currently).

```
$ git branch -d [name]
```

Remove selected branch, if it is already merged into any other.
-D instead of **-d** forces deletion.

05 Review your work

```
$ git log [-n count]
```

List commit history of current branch. **-n count** limits list to last **n** commits.

```
$ git log --oneline --graph --decorate
```

An overview with reference labels and history graph. One commit per line.

```
$ git log ref..
```

List commits that are present on the current branch and not merged into **ref**. A **ref** can be a branch name or a tag name.

```
$ git log ..ref
```

List commit that are present on **ref** and not merged into current branch.

```
$ git reflog
```

List operations (e.g. checkouts or commits) made on local repository.

06 Tagging known commits

```
$ git tag
```

List all tags.

```
$ git tag [name] [commit sha]
```

Create a tag reference named **name** for current commit. Add **commit sha** to tag a specific commit instead of current one.

```
$ git tag -a [name] [commit sha]
```

Create a tag object named **name** for current commit.

```
$ git tag -d [name]
```

Remove a tag from local repository.

07 Reverting changes

```
$ git reset [--hard] [target reference]
```

Switches the current branch to the **target reference**, leaving a difference as an uncommitted change. When **--hard** is used, all changes are discarded.

```
$ git revert [commit sha]
```

Create a new commit, reverting changes from the specified commit. It generates an **inversion** of changes.

08 Synchronizing repositories

```
$ git fetch [remote]
```

Fetch changes from the **remote**, but not update tracking branches.

```
$ git fetch --prune [remote]
```

Delete remote Refs that were removed from the **remote** repository.

```
$ git pull [remote]
```

Fetch changes from the **remote** and merge current branch with its upstream.

```
$ git push [--tags] [remote]
```

Push local changes to the **remote**. Use **--tags** to push tags.

```
$ git push -u [remote] [branch]
```

Push local branch to **remote** repository. Set its copy as an upstream.

Commit an object

Branch a reference to a commit; can have a **tracked upstream**

Tag a reference (standard) or an object (annotated)

Head a place where your **working directory** is now

A Git installation

For GNU/Linux distributions, Git should be available in the standard system repository. For example, in Debian/Ubuntu please type in the **terminal**:

```
$ sudo apt-get install git
```

If you need to install Git from source, you can get it from git-scm.com/downloads.

An excellent Git course can be found in the great **Pro Git** book by Scott Chacon and Ben Straub. The book is available online for free at git-scm.com/book.

B Ignoring Files

```
$ cat .gitignore
/logs/*
!logs/.gitkeep
/tmp
*.swp
```

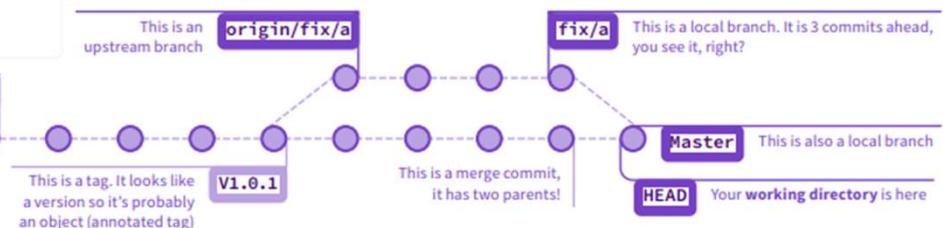
Verify the **.gitignore** file exists in your project and ignore certain type of files, such as all files in **logs** directory (excluding the **.gitkeep** file), whole **tmp** directory and all files ***.swp**. File ignoring will work for the directory (and children directories) where **.gitignore** file is placed.

C Ignoring Files

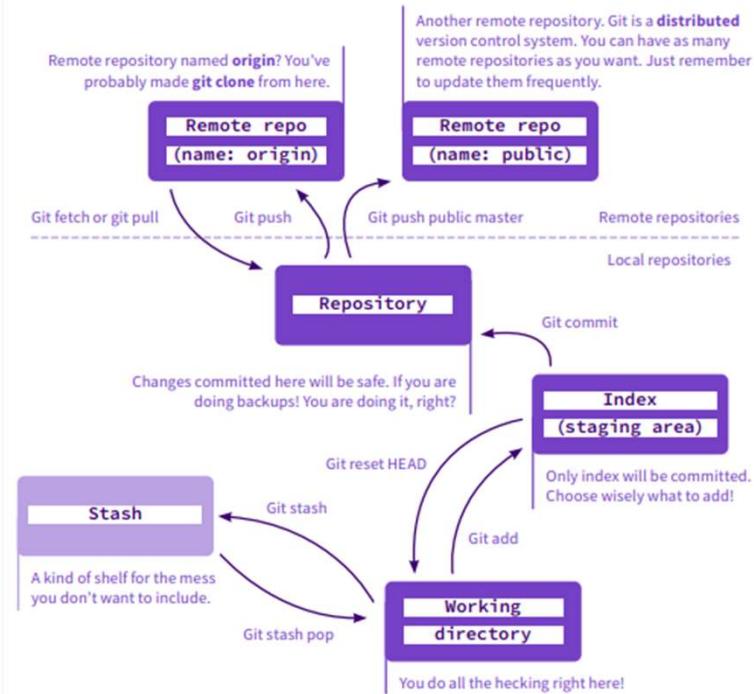
This is a tag. It looks like a developer's note so it's probably a reference, not an object.

working-version

This is an initial commit, it has no parents



D The zoo of working areas



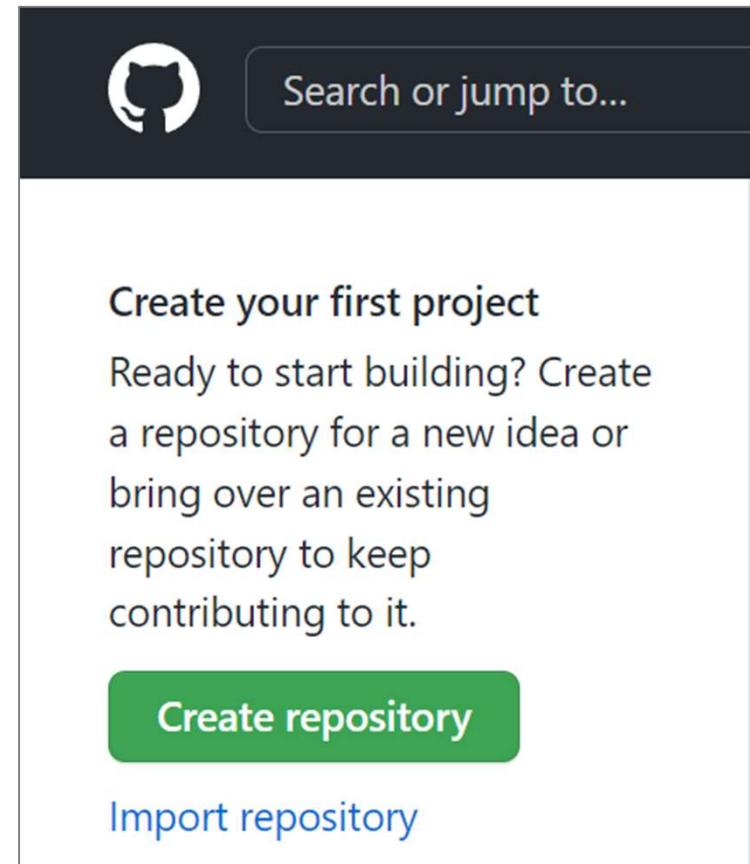
GITHUB

Github nedir?

- Sürüm kontrol sistemi olarak git'i kullanan ve yazılım geliştirme projelerinin depolanması, değişikliklerin izlenmesi ve üzerlerinde ortak çalışılmasını sağlamak üzere tasarlanmış web temelli kod depolama sistemidir.
- 2008'de «Tom Preston-Werner», «Chris Wanstrath», «P. J. Hyett» ve «Scott Chacon» tarafından geliştirilmiştir.

Github'da «Repo» oluşturma

- İlk projenizi oluşturmak için Github'da **«Create Repository»** kısmına tıklamak gereklidir.



Github'da «Repo» oluşturma

- Proje isminizi vermelii ve lisansınızı belirlemelisiniz.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * Repository name *

 hgsenel / gps_parser 

Great repository names are short and memorable. Need inspiration? How about [friendly-octo-potato?](#)

Description (optional)

GPS parsing library

 Public
Anyone on the internet can see this repository. You choose who can commit.

 Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: [GNU General Public ...](#) 

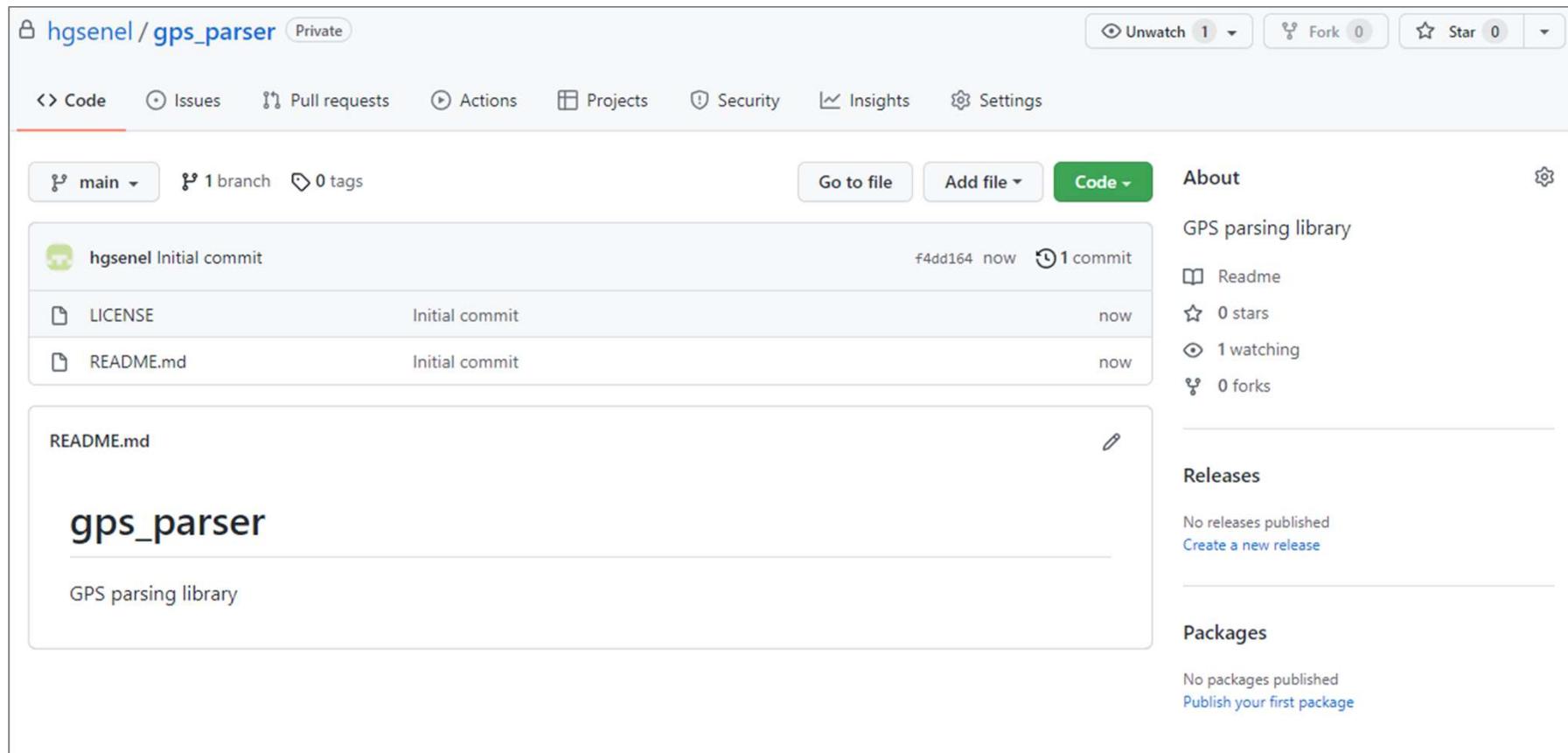
This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a private repository in your personal account.

Create repository

Github'da «Repo» oluşturma

- Proje oluşturuldu.



hgsenel / gps_parser Private

Code Issues Pull requests Actions Projects Security Insights Settings

main 1 branch 0 tags Go to file Add file Code About

GPS parsing library

Readme 0 stars 1 watching 0 forks

Initial commit f4dd164 now 1 commit

LICENSE Initial commit now

README.md Initial commit now

README.md

gps_parser

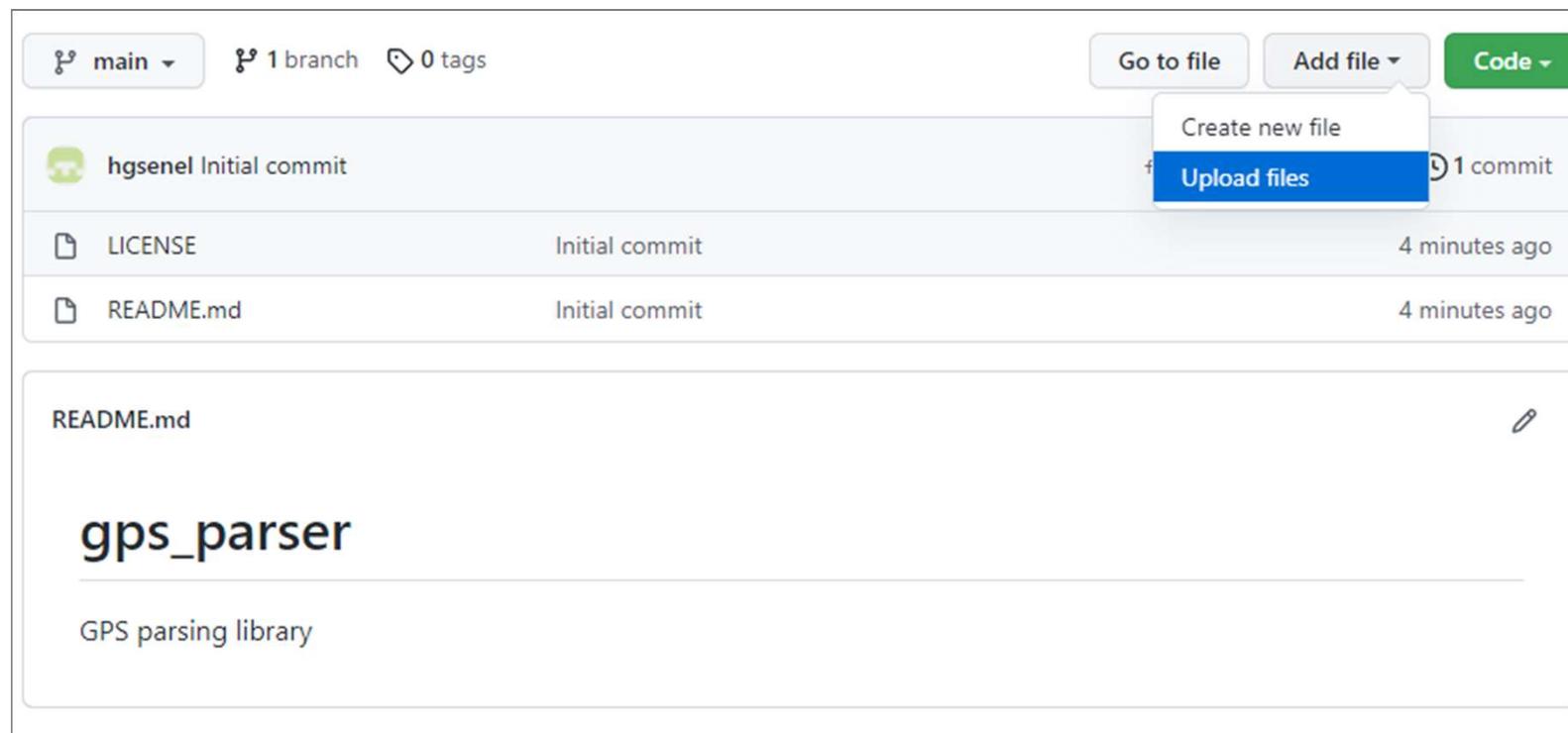
GPS parsing library

No releases published Create a new release

No packages published Publish your first package

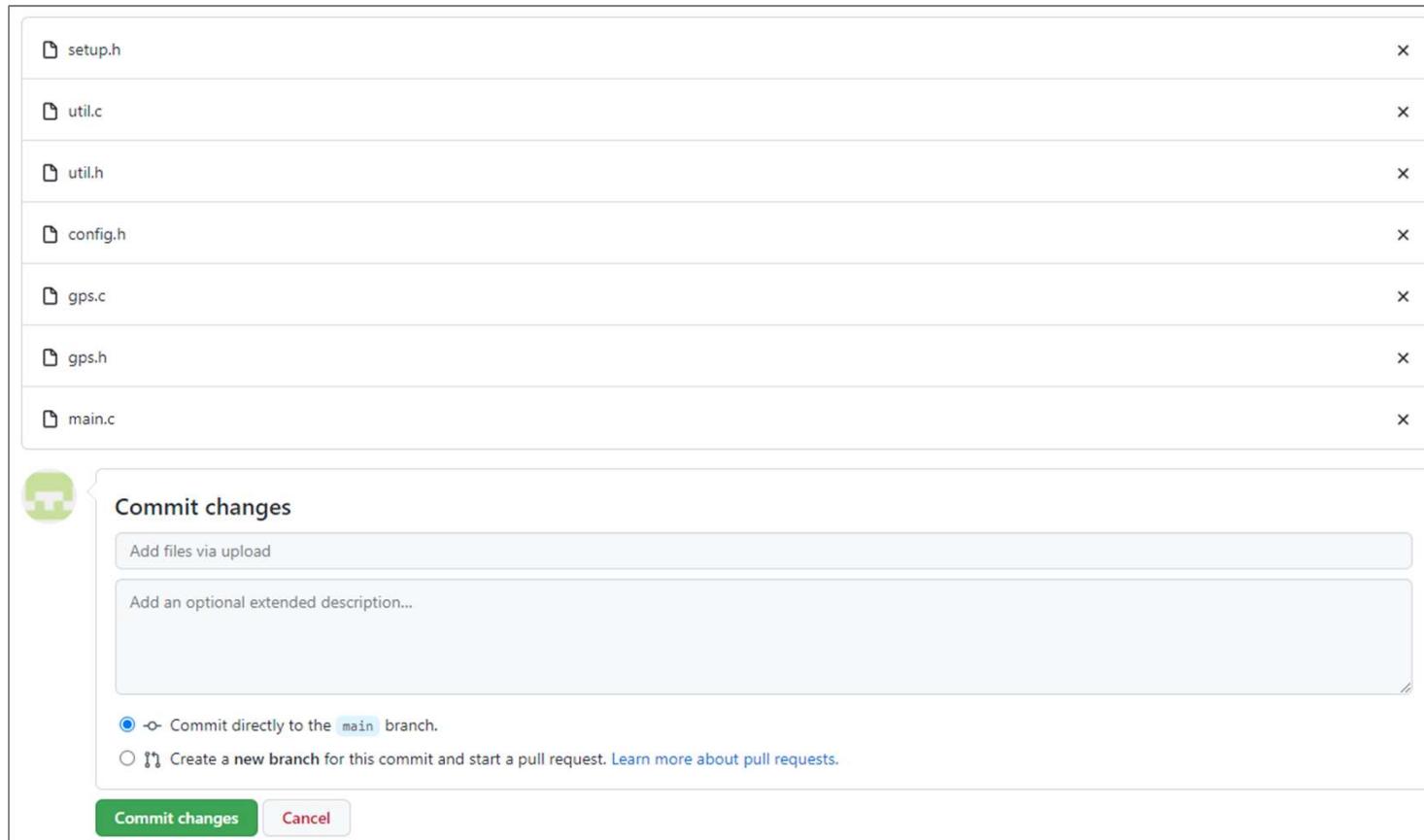
Github'da «Repo» oluşturma

- Dosya ekleme



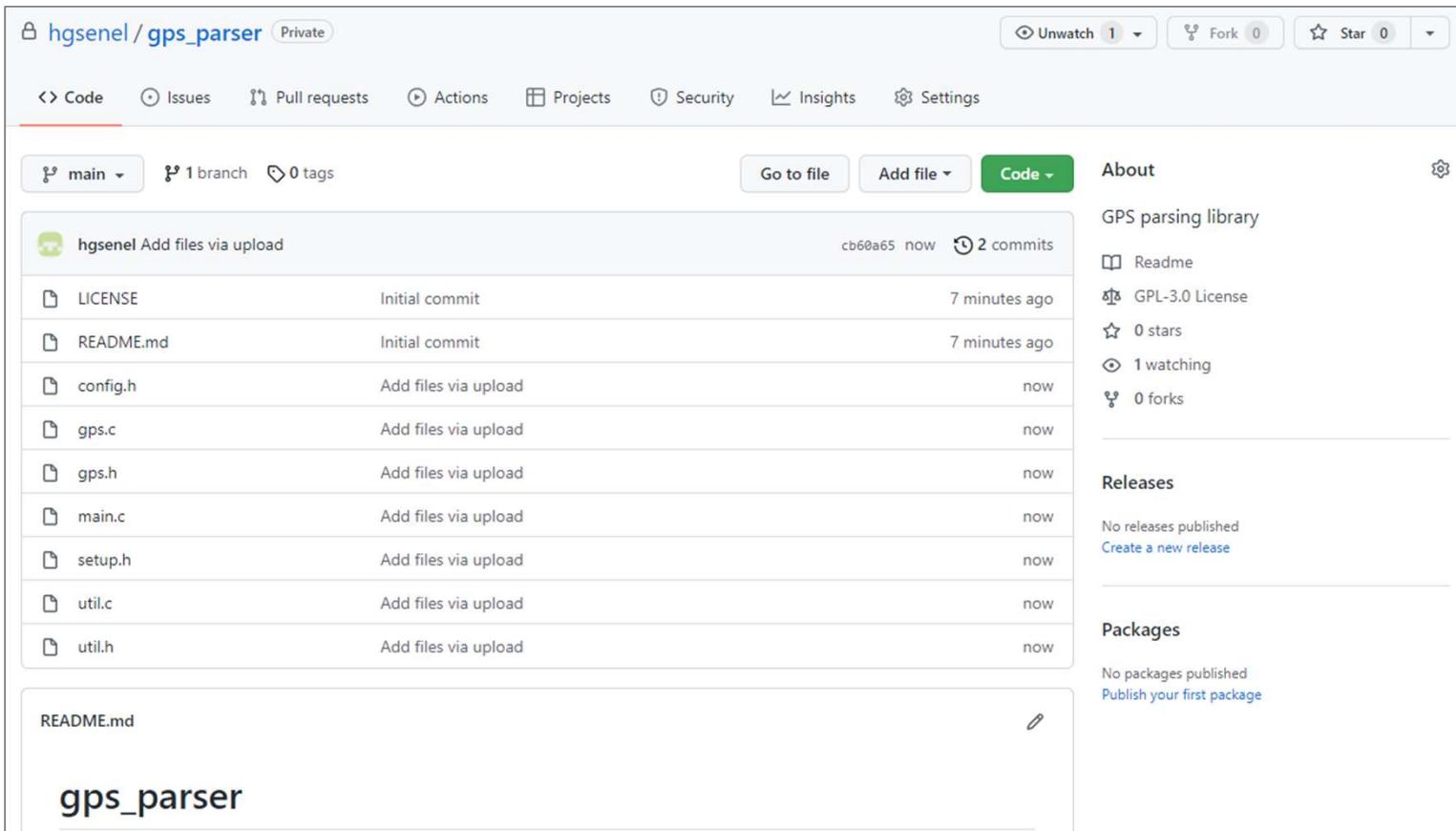
Github'da «Repo» oluşturma

- Dosya ekleme



Github'da «Repo» oluşturma

- Dosya ekleme



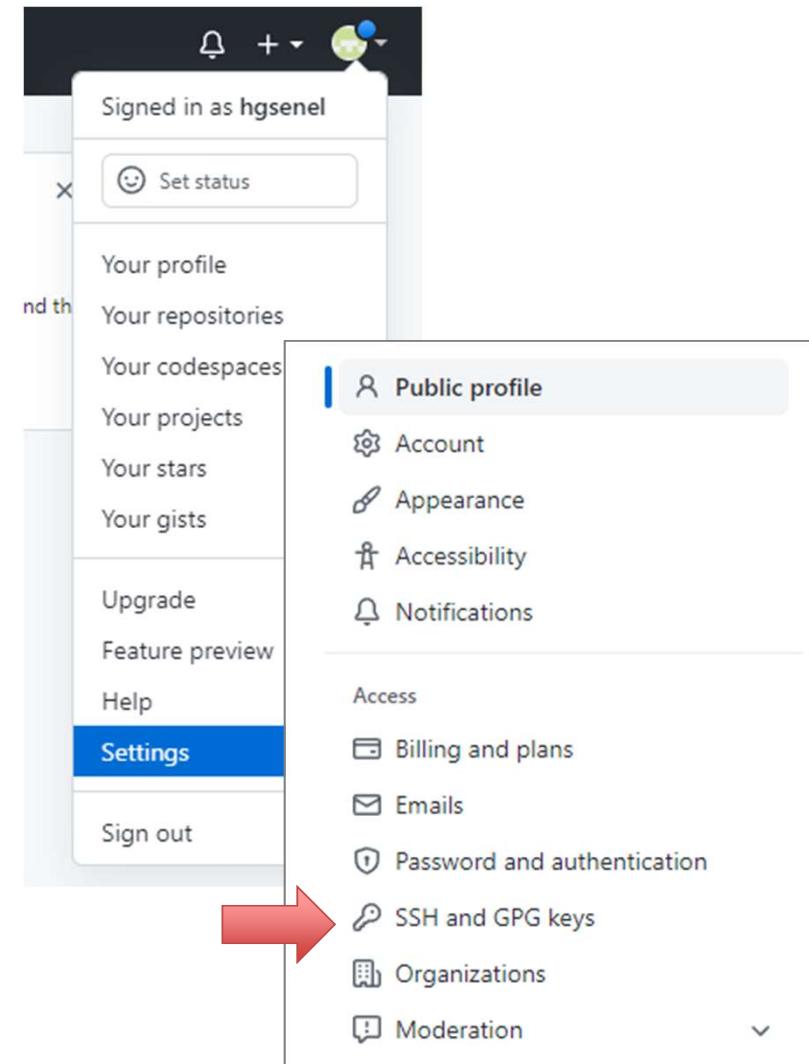
The screenshot shows a GitHub repository page for 'hgsenel/gps_parser'. The repository is private. The main interface displays a list of files uploaded via drag-and-drop:

File	Description	Time
LICENSE	Initial commit	7 minutes ago
README.md	Initial commit	7 minutes ago
config.h	Add files via upload	now
gps.c	Add files via upload	now
gps.h	Add files via upload	now
main.c	Add files via upload	now
setup.h	Add files via upload	now
util.c	Add files via upload	now
util.h	Add files via upload	now

On the right side, there's an 'About' section with details like 'GPS parsing library', 'Readme', 'GPL-3.0 License', and '0 stars'. Below it is a 'Releases' section stating 'No releases published' and a 'Packages' section with the same message.

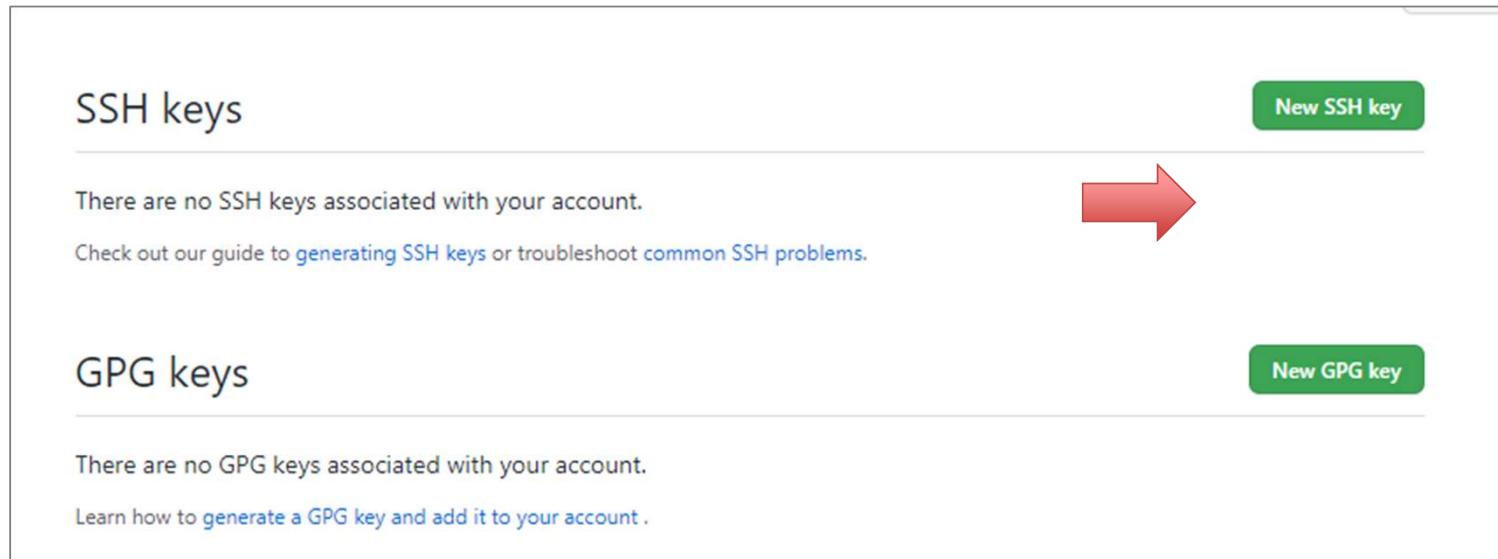
Github hesabına SSH ile erişim

- Github'da oluşturduğunuz bir kod projesine erişimi daha güvenli hale getirmek için SSH anahtarları kullanılabilir.
- Lokal bilgisayarınızda, «**ssh-keygen**» komutuyla gizli «**.ssh/id_rsa**» ve açık «**.ssh/id_rsa.pub**» anahtarlarının oluşturulması gerekmektedir.
- Açık anahtar kopyalayarak, **Github.com**'daki profilinize gitmelisiniz ve «**Settings**»e tıklamalısınız.



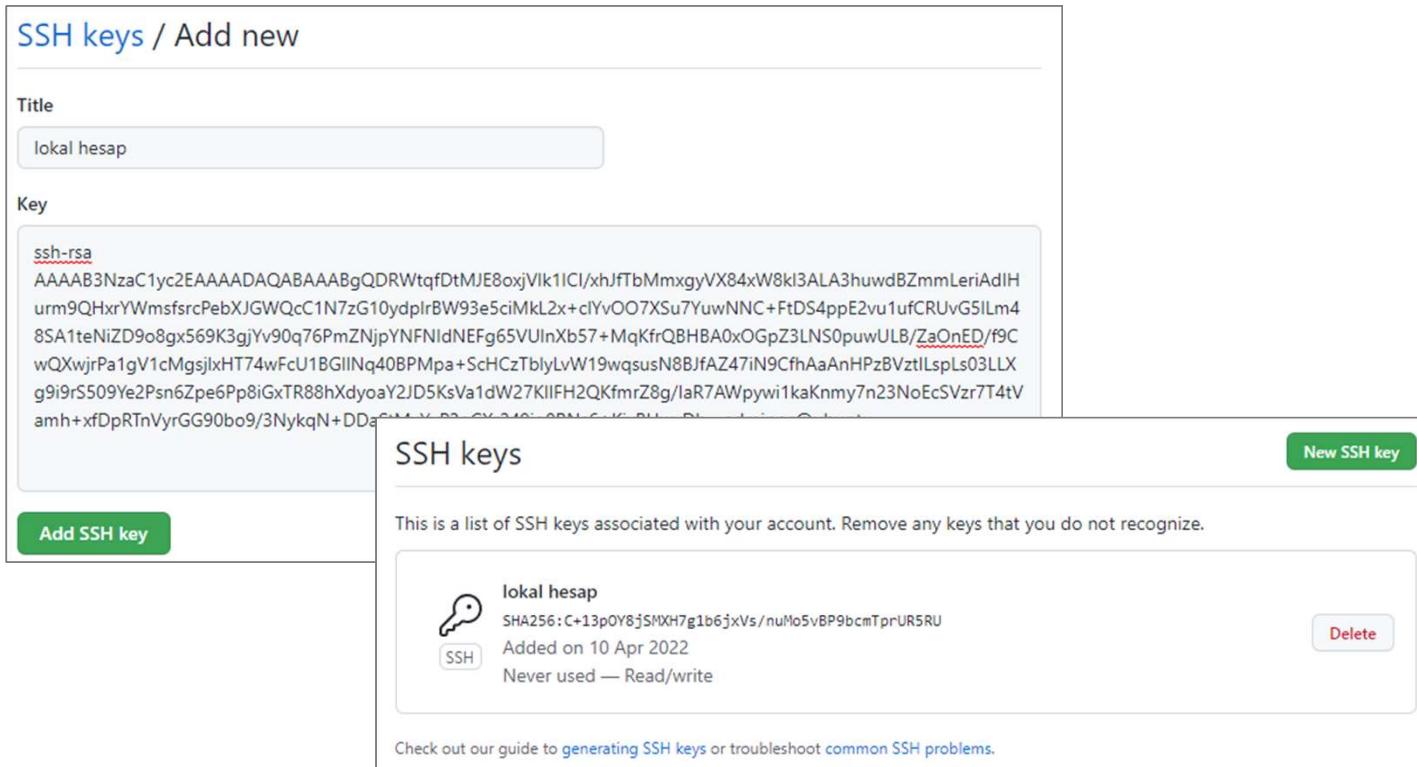
Github hesabına SSH ile erişim

- «SSH Keys» bölümüne giderek, «New SSH key» butonuna tıklayın.



Github hesabına SSH ile erişim

- «SSH Keys» «Add New» bölümünde açık anahtarınızı giriniz.



The screenshot shows two parts of the GitHub SSH keys interface:

- SSH keys / Add new**: A form for adding a new SSH key. It has a "Title" field containing "lokal hesap" and a "Key" field containing a long ssh-rsa public key. Below the form is a green "Add SSH key" button.
- SSH keys**: A list of existing SSH keys associated with the account. It shows one key named "lokal hesap" with the following details:
 - SHA256: C+13p0Y8jSMXH7g1b6jxVs/nuMo5vBP9bcmTprUR5RU
 - Added on 10 Apr 2022
 - Never used — Read/writeA red "Delete" button is located next to the key details.

At the bottom of the list, there is a link to "Check out our guide to generating SSH keys or troubleshoot common SSH problems."

Github hesabına SSH ile erişim

- Lokal Linux bilgisayarlarınızdan aşağıdaki komutla, Github'da hesabınızda bulunan kod deposunu klonlayabilirsiniz.

```
git clone git@github.com:hesapisim/projeadi.git
```

```
adminpc@ubuntu:~$ git clone git@github.com:hgsenel/gps_trial.git
Cloning into 'gps_trial'...
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeI0ttrVc98/R1BUFWu3/LivKaUFQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,140.82.121.4' (ECDSA) to the list of known hosts.
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 13 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (13/13), 17.51 KiB | 4.38 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

Uzak Depoyu Takip Etme

- Github'daki proje kod deposundan «**gps_trial**» isimli bir proje klonladık ve «git» komutunu çalıştığımız dizinde «**gps_trial**» isimli dizine proje dosyalarını indirdi.
- Dizinin içinde, «**git remote -v**» komutu verildiğinde, uzaktaki kod deposunun URL adresini verir.
- «**git remote show origin**» ise depoya ilgili ayrıntılı bilgiler sunar.

```
adminpc@ubuntu:~/gps_trial$ git remote -v
origin  git@github.com:hgsenel/gps_trial.git (fetch)
origin  git@github.com:hgsenel/gps_trial.git (push)
adminpc@ubuntu:~/gps_trial$ git remote show origin
* remote origin
  Fetch URL: git@github.com:hgsenel/gps_trial.git
  Push URL: git@github.com:hgsenel/gps_trial.git
  HEAD branch: main
  Remote branch:
    main tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```

«git fetch»

- Uzak depoya yüklenen yeni bir dosyayı çekmek için «git fetch» komutu kullanılır. Eğer, depoya (başka biri tarafından yüklenen) bir dosya varsa bunu indirir.

```
adminpc@ubuntu:~/gps_trial$ git fetch
Warning: Permanently added the ECDSA host key for IP address '140.82.121.3' to the l
ist of known hosts.
```

- Proje üzerinde sizin dışınızda birileri tarafından da geliştiriliyorsa, **fetch** komutu kesinlikle gereklidir.
- Eğer projeye yaptığınız değişiklikleri yüklemek (**push**) istiyorsanız, öncesinde «**fetch**» yapmanız çakışmaları önleyecektir.
- «**git pull**» komutu, uzak depodaki değişiklikleri indirir ve lokal deponuzla birleştirmeye çalışır. Uzak depoda eğer değişiklik yoksa, lokal deponuzun uzaktakiyle aynı olduğunu ve **merge** yapmanıza gerek olmadığını gösterir.

«git fetch» ve «git pull» farkı

- Her iki komut çoğu zaman birbiriyle karıştırılmaktadır.
- Her ikisi de uzaktaki içeriği indirmek için kullanılır.
- «git fetch» komutunun daha güvenilir olduğu düşünülür zira uzaktaki içeriği indirir ama lokal deponun durumunu değiştirmez.
- Alternatif olarak «git pull» uzaktaki içeriği indirir ve lokal deponun durumuyla eşitlemeye çalışır. Bu nedenle güvenilir bir komut olmadığı düşünülür.

«git push»

- Lokalde yaptığınız değişikliklerin «**master**» dalına doğrudan yüklemek riskli olabilir ve kodda bozulmalara yol açabilir.
- Bu nedenle ayrı bir dal (branch) oluşturarak üzerinde değişiklikler yapılması ve sonrasında bu dalın yüklenmesi iyi bir pratiktir.

```
cd ~/gps_trial  
git branch deneme3  
git checkout deneme3  
echo "deneme işlem" >> README.md  
git commit -a -m "deneme işlem"  
git push -u origin deneme3
```

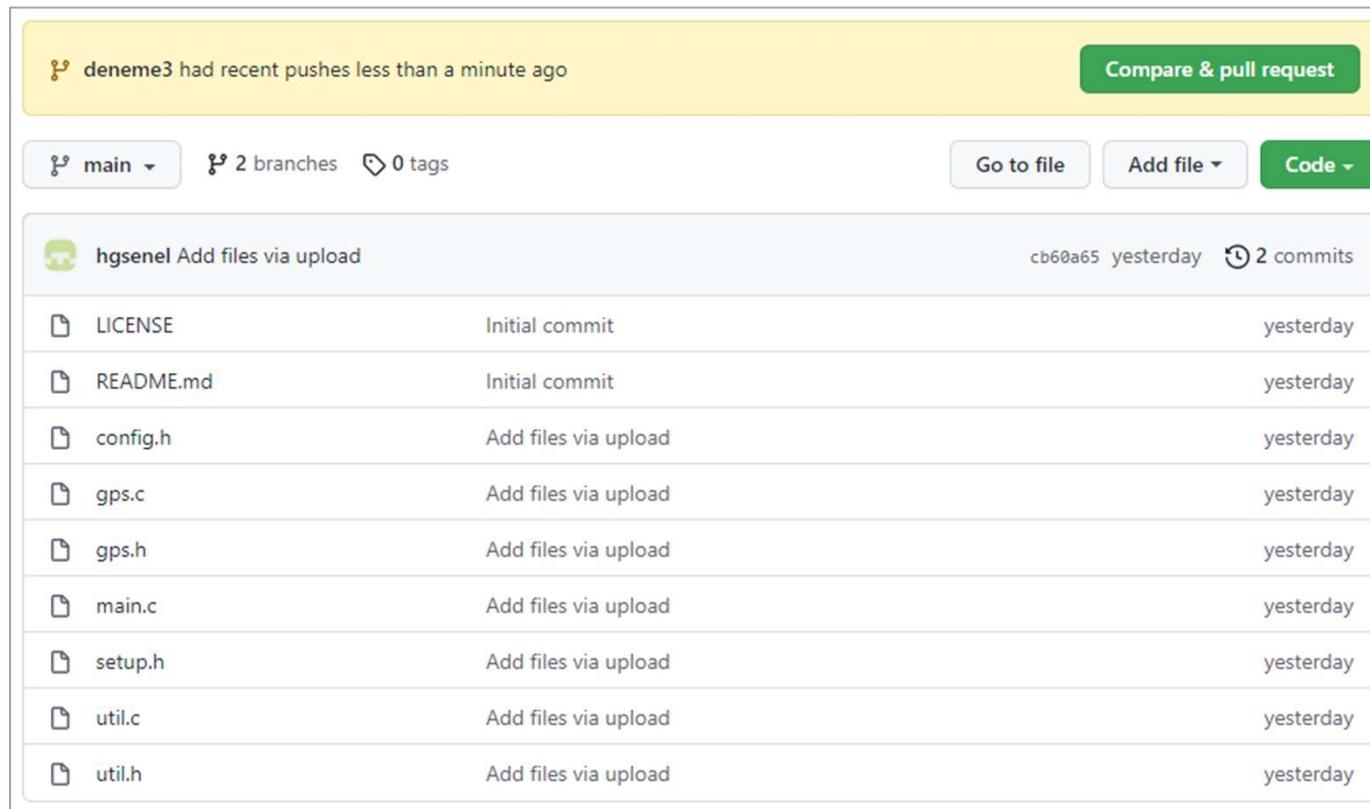
«push»

- «push» komutu, `deneme3` dalındaki değişiklikleri uzak depoya yüklenmiştir.

```
adminpc@ubuntu:~$ git clone git@github.com:hgsenel/gps_trial.git
Cloning into 'gps_trial'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 13 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (13/13), 17.51 KiB | 3.50 MiB/s, done.
Resolving deltas: 100% (1/1), done.
adminpc@ubuntu:~$ cd ~/gps_trial
adminpc@ubuntu:~/gps_trial$ git branch deneme3
adminpc@ubuntu:~/gps_trial$ git checkout deneme3
Switched to branch 'deneme3'
adminpc@ubuntu:~/gps_trial$ echo "deneme islem" >> README.md
adminpc@ubuntu:~/gps_trial$ git commit -a -m "deneme islem proses"
[deneme3 1492ef6] deneme islem proses
 1 file changed, 1 insertion(+)
adminpc@ubuntu:~/gps_trial$ git push -u origin deneme3
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 304 bytes | 76.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'deneme3' on GitHub by visiting:
remote:     https://github.com/hgsenel/gps_trial/pull/new/deneme3
remote:
To github.com:hgsenel/gps_trial.git
 * [new branch]      deneme3 -> deneme3
Branch 'deneme3' set up to track remote branch 'deneme3' from 'origin'.
```

«push»

- Github'da **deneme3** dalının yükleniği görülebilir.
«Compare & pull request» tıklanmalıdır.



deneme3 had recent pushes less than a minute ago

Compare & pull request

main ▾ 2 branches 0 tags

Go to file Add file ▾ Code ▾

File	Commit Message	Date
LICENSE	Initial commit	yesterday
README.md	Initial commit	yesterday
config.h	Add files via upload	yesterday
gps.c	Add files via upload	yesterday
gps.h	Add files via upload	yesterday
main.c	Add files via upload	yesterday
setup.h	Add files via upload	yesterday
util.c	Add files via upload	yesterday
util.h	Add files via upload	yesterday

«pull request»

«pull request»

«Merge» işlemi, kod deposunun sahibi tarafından yapılabilmektedir.

deneme islem proses #1

[Open](#) hgsenel wants to merge 1 commit into `main` from `deneme3`

Conversation 0 Commits 1 Checks 0 Files changed 1

hgsenel commented now
README.md dosyasına bir satır ekledi.
 deneme islem proses 1492ef6

Add more commits by pushing to the `deneme3` branch on [hgsenel/gps_trial](#).

Continuous integration has not been set up
GitHub Actions and [several other apps](#) can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

[Merge pull request](#) You can also open this in GitHub Desktop or view command line instructions.

Yapılan değişikliklerin birleştirilmesi

deneme islem proses #1

Merged hgsenel merged 1 commit into main from deneme3 now

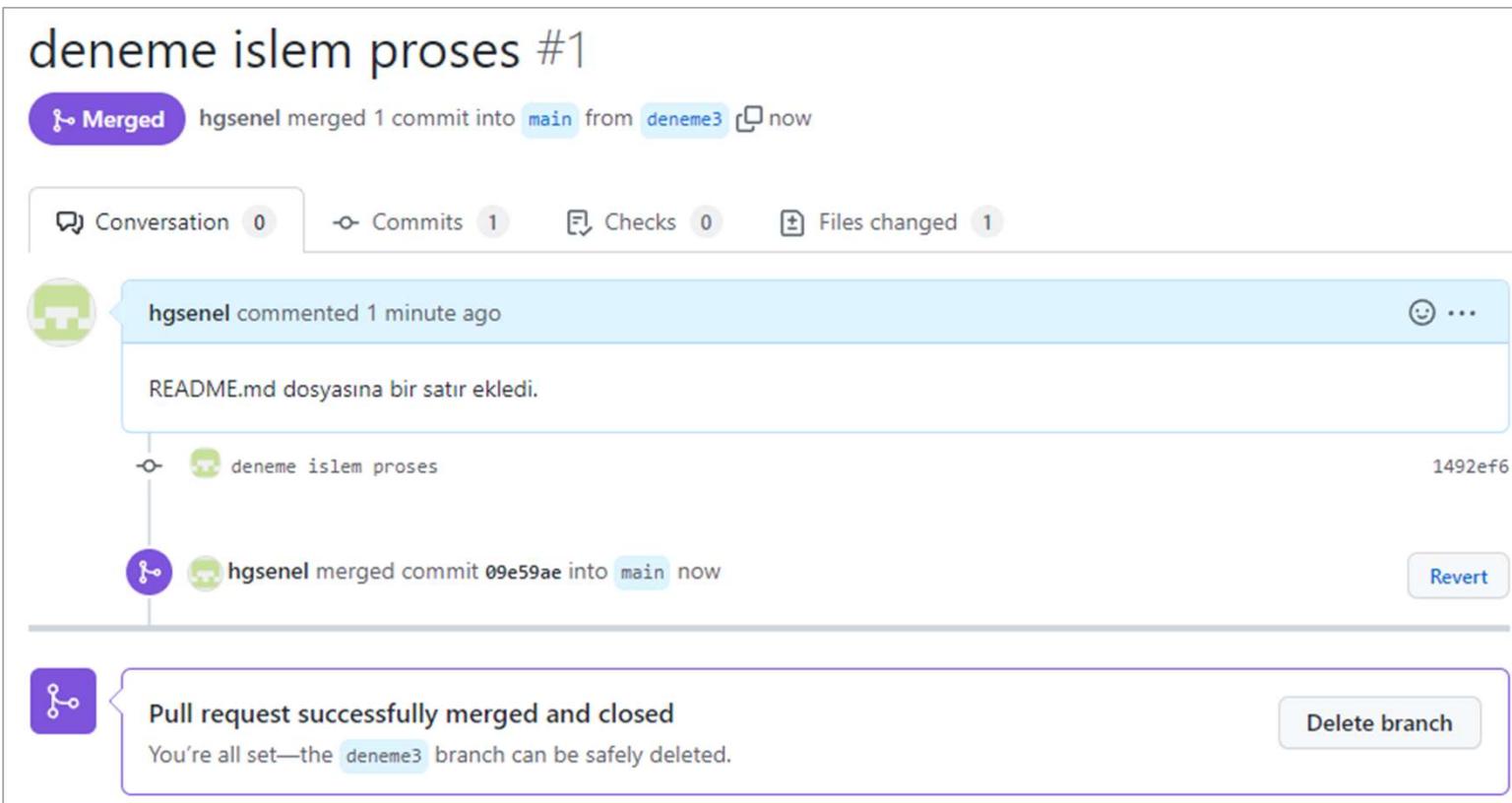
Conversation 0 Commits 1 Checks 0 Files changed 1

hgsenel commented 1 minute ago
README.md dosyasına bir satır ekledi.

deneme islem proses 1492ef6

hgsenel merged commit 09e59ae into main now Revert

Pull request successfully merged and closed
You're all set—the deneme3 branch can be safely deleted. Delete branch



Yapılan değişikliklerin birleştirilmesi

The screenshot shows a GitHub repository interface. At the top, there are navigation buttons: 'main' (selected), '2 branches', '0 tags', 'Go to file', 'Add file', and a green 'Code' button. Below this is a list of commits from a user named 'hgsenel'. The first commit is a 'Merge pull request #1 from hgsenel/deneme3' made 2 minutes ago with 4 commits. The subsequent commits show the addition of various files via upload, all made yesterday. A red arrow points to the commit for 'README.md'. Below this, a detailed view of the 'README.md' file is shown, containing the text 'gps_parser' and 'GPS parsing library deneme islem'. A red arrow points to the word 'gps_parser'.

File	Commit Message	Time
LICENSE	Initial commit	yesterday
README.md	deneme islem proses	11 minutes ago
config.h	Add files via upload	yesterday
gps.c	Add files via upload	yesterday
gps.h	Add files via upload	yesterday
main.c	Add files via upload	yesterday
setup.h	Add files via upload	yesterday
util.c	Add files via upload	yesterday
util.h	Add files via upload	yesterday

README.md

gps_parser

GPS parsing library deneme islem