

Git Temelleri

Versiyon Kontrol Sistemi

Version Control System bir döküman (yazılım projesi, ofis belgesi...) üzerinde yaptığımız değişiklikleri adım adım kaydeden ve isterseniz bunu internet üzerinde depoda (respository) saklamamızı ve yönetmemizi sağlayan bir sistemdir. Git, SVN, BitKeeper, Mercurial sürüm kontrol sistemlerine örnek olarak gösterilir.

Git

Git açık kaynak kodlu bir versiyon sürüm kontrol sistemidir

Neden git ?

Koyuyla belirttiklerime dikkat! :)

- Geliştirme süreçlerini izlemek
- **Projede yapılan değişikliklerin nerede ve ne zaman yapıldığı izlenmek**
- **Yanlış veya hatalı işlem yapıldığında daha önceki düzgün çalışılan versiyona dönmek istenebilir.**
- **Projelerde birden fazla kişi çalıştığını düşünürsek, gelişimin hızlanmasını sağlar.**
- Projede geliştirme yaparken, bulunduğumuz konuma nereden geldiğimizi anlamak için eski ve yeni kodumuz arasında karşılaştırma yapmamızı sağlar.
- Projede hatayla karşılaştığımız durumlarda eski kod kaydına dönmemizi sağlar.
- Açık kaynaklı projeler baz alınarak geliştirilecek yeni projelerde, süreci kolaylaştırmayı sağlar.

Versiyon Kontrol Sistem Tipleri

1. Local Versiyon Kontrol Sistemleri

En eski versiyon kontrol sistemi yaklaşımıdır. Çalıştığımız projemiz ve yaptığımız değişiklikler kullanıcı makinası üzerindeki veritabanında tutulur. Her yapılan commit bir versiyon olarak tutulur ve commit değerine hash ataması yapılarak her versiyon birbirinden ayırt edilmektedir. Ayrıca versiyon görüntüleme imkanını sağlar. Ancak bu sistemde sadece bir kullanıcı etkin bir şekilde çalışabilir.

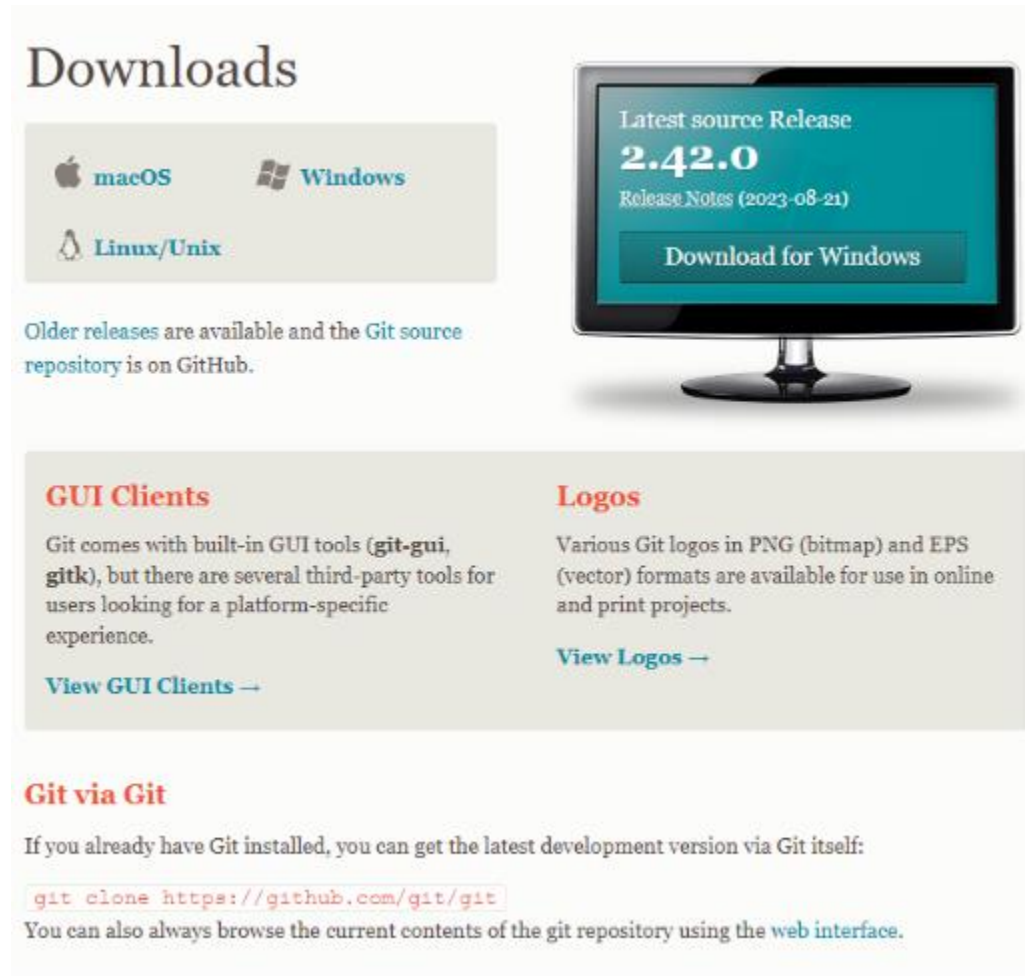
2. Merkezi Versiyon Kontrol Sistemleri

Birden fazla kişinin bir proje üzerinde etkin çalışması için ortaya atılmış versiyonlama sistemidir. CVS, SVN birer merkezi versiyon kontrol sistemleridir. Bu sistemde proje ortak bir respository'de tutulur ve birden fazla geliştirici aynı respository üzerinde checkout ve commit işlemlerini gerçekleştirmektedir. Bu yöntemde herkesin projeye katkı sağlamasının yanısıra bazı ciddi sorunları vardır. Tek merkezli sunucu 1 saatliğine arızalanması durumunda, kullanıcılar 1 saat boyunca çalışmalarını veya çalıştıkları projenin sürümlenmiş kopyalarını kaydetmeleri mümkün olmayacaktır.

3. Dağıtık Versiyon Kontrol Sistemleri

Git Kurulum İşlemleri

- <https://git-scm.com/downloads> adresinden indirmeniz gerekiyor.



Downloads

macOS Windows Linux/Unix

Older releases are available and the Git source repository is on GitHub.

Latest source Release
2.42.0
Release Notes (2023-08-21)
Download for Windows

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

[View GUI Clients →](#)

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

[View Logos →](#)

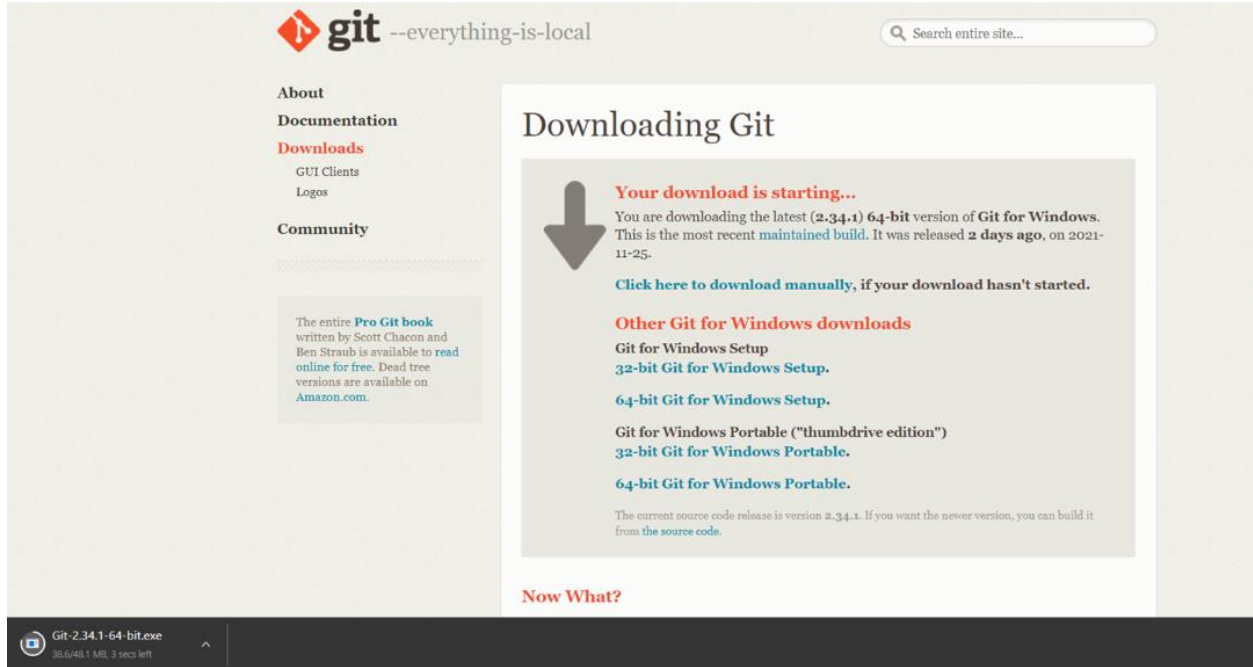
Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

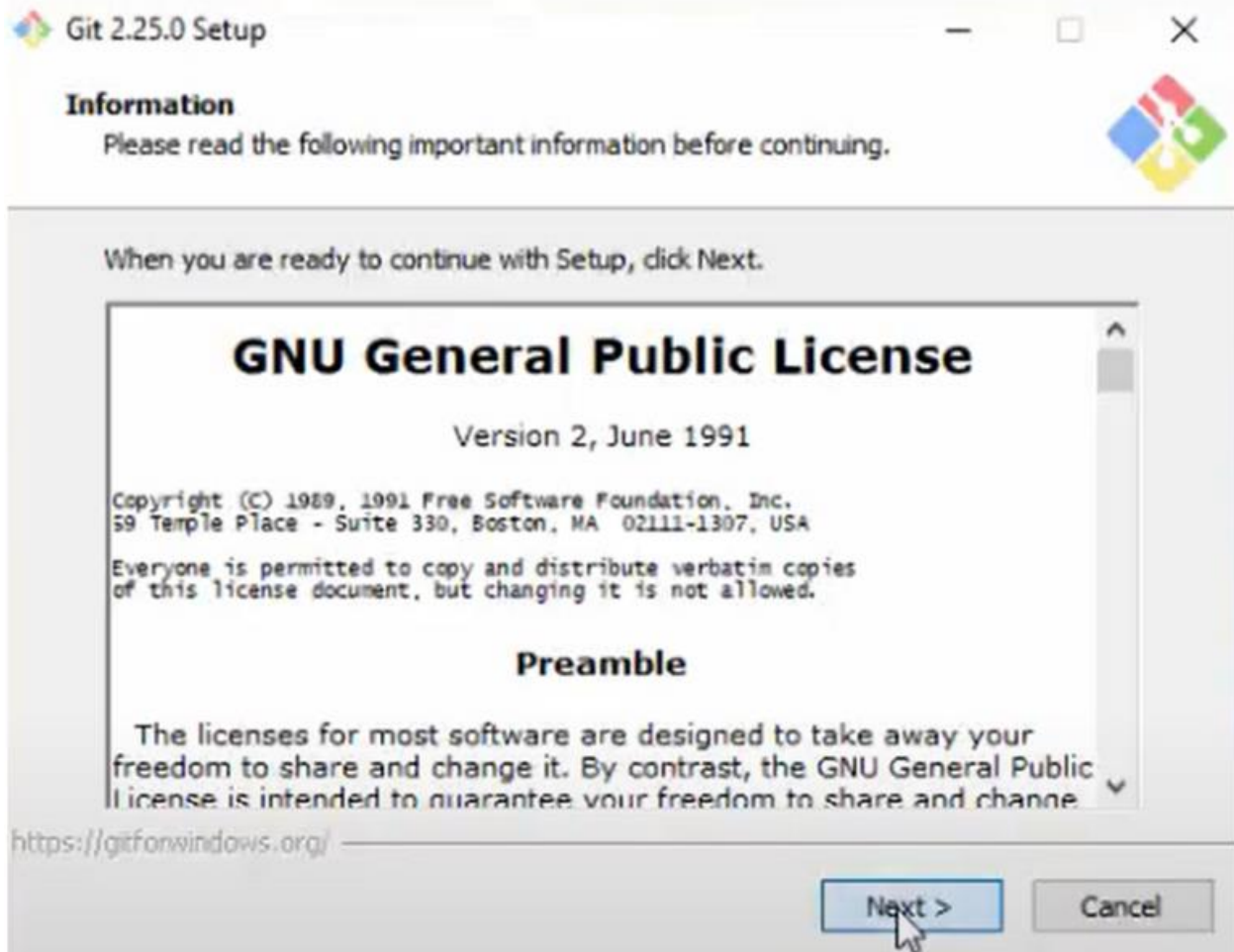
```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the [web interface](#).

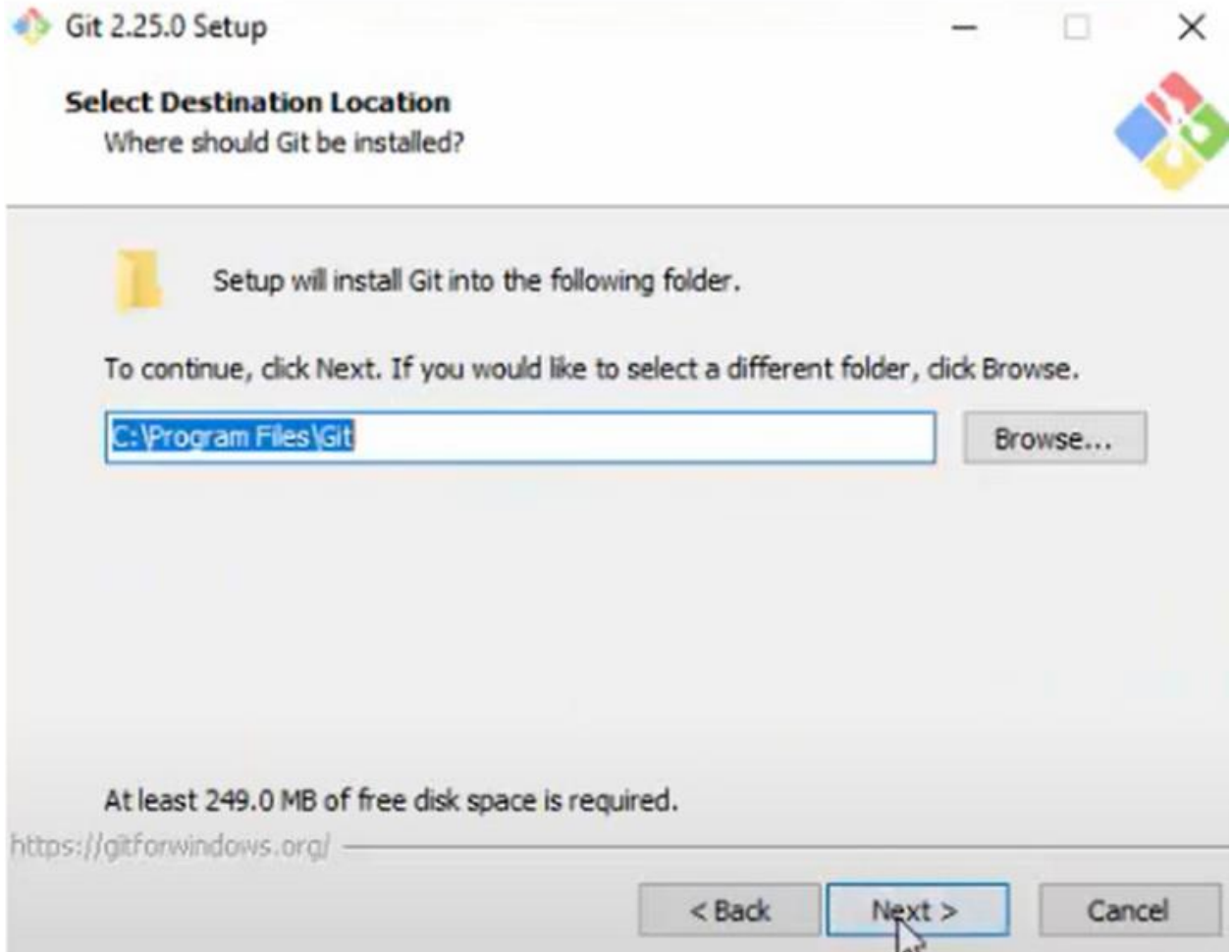
- Kullanıcı hesap denetimi ayarlarınız açık ise karşınıza bir pencere gelecektir. Bu aşama da işlemin tarafımızdan gerçekleştirildiğini onaylamamız gerekiyor. "Evet" seçeneğini seçerek işleme izin vermeniz gerekir.



- Kurulumun ilk aşamasında bizi "GNU General Public License" lisansı karşılamaktadır. "Next" butonuna tıklayarak bir sonra ki aşamaya geçiyoruz.



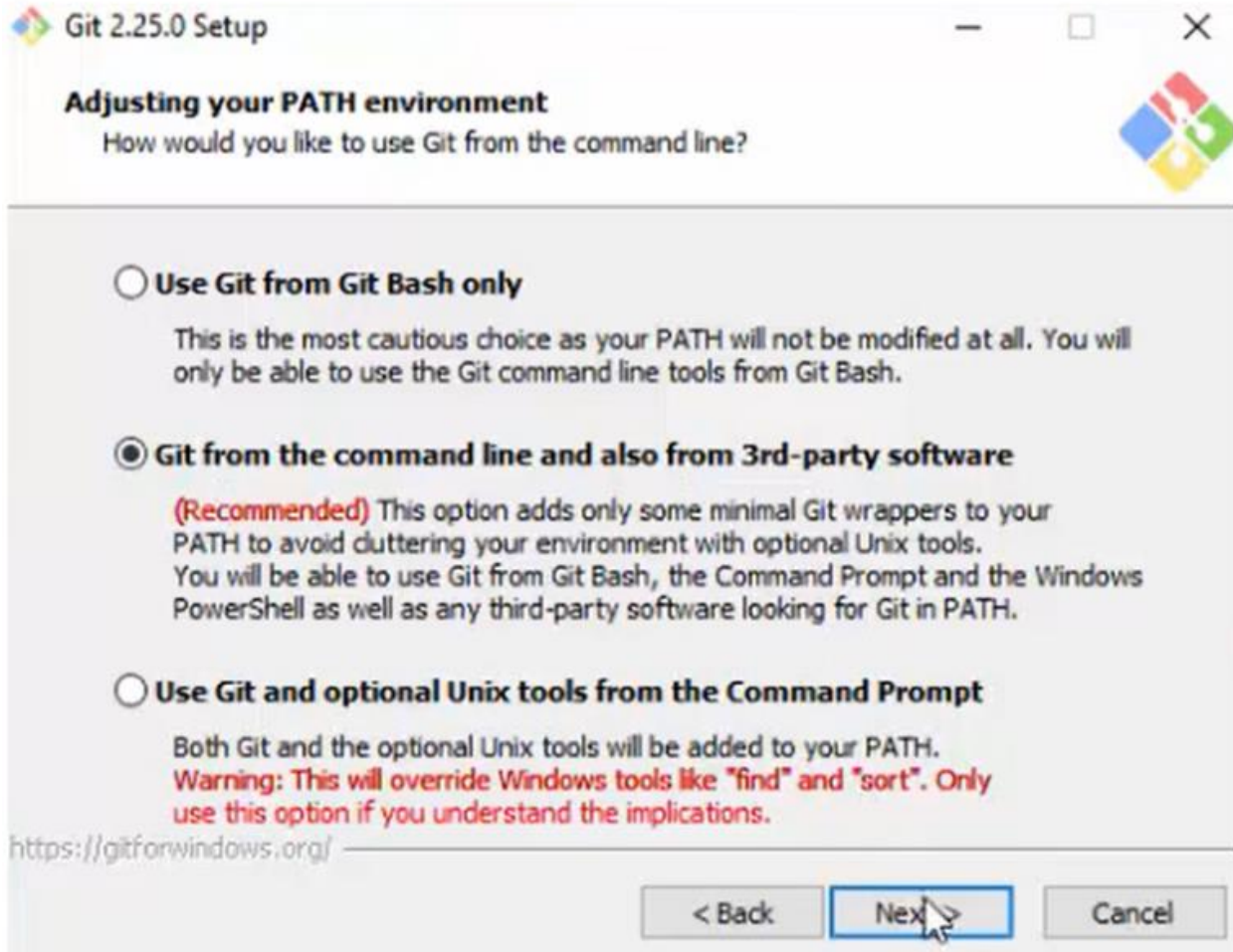
- Bu aşamada Git'in kurulacağı dizini belirtmemiz gerekiyor. Eğer farklı bir dizin seçmeyecek seniz, "Next" butonu ile bir sonra ki aşamaya geçiyoruz.



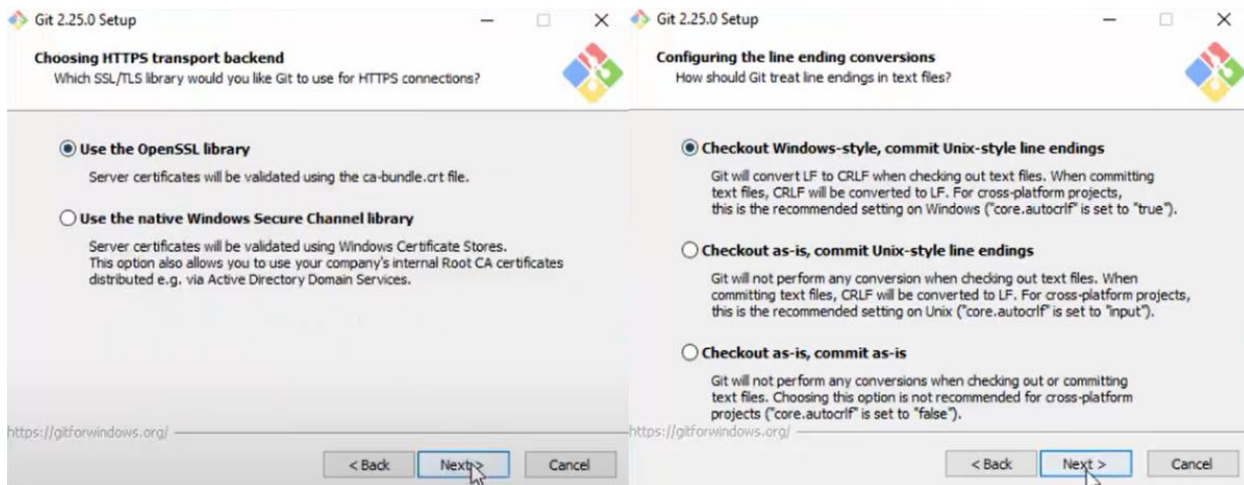
- Bu aşamada kurmak istediğiniz ekstra bileşenler varsa bu aşamada seçiyoruz. Bu aşamayı da "Next" butonu ile geçiyoruz.

![[image]](<https://github.com/KardelRuveyda/sektor-kampuste-sanayi-bakanligi/assets/33912144/19122356-48a4-4e73-8187-b4f2bd2392bd>)

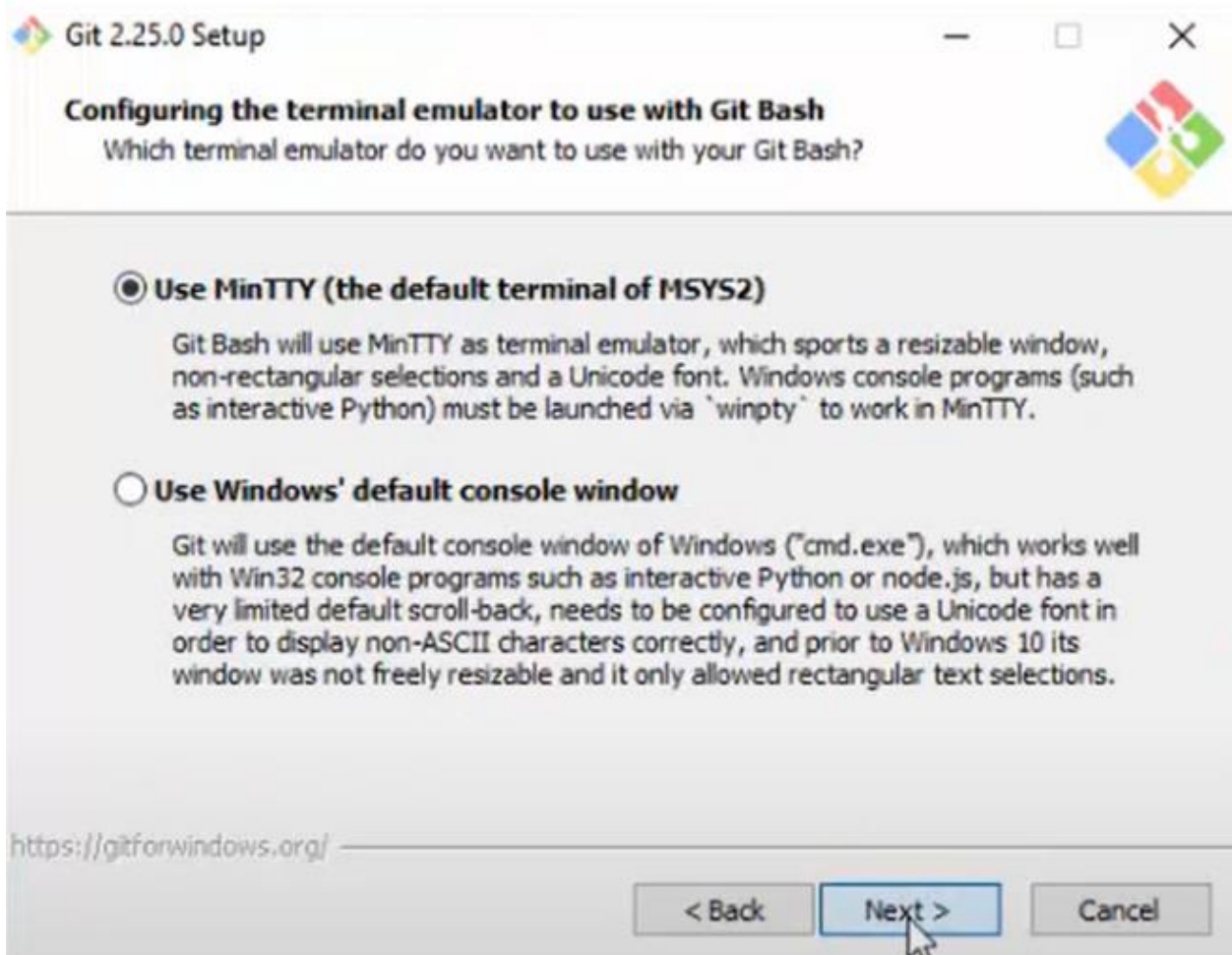
- Varsayılan olarak kullandığımız bir editör varsa, burada belirtmemiz gerekiyor. Ben varsayılan olarak Visual Studio Code kullandığımdan "Use Visual Studio Code as Git's default editor" seçeneğini seçtim. Eğer bu seçimi yaptıysanız, sonra ki aşamaya geçelim.



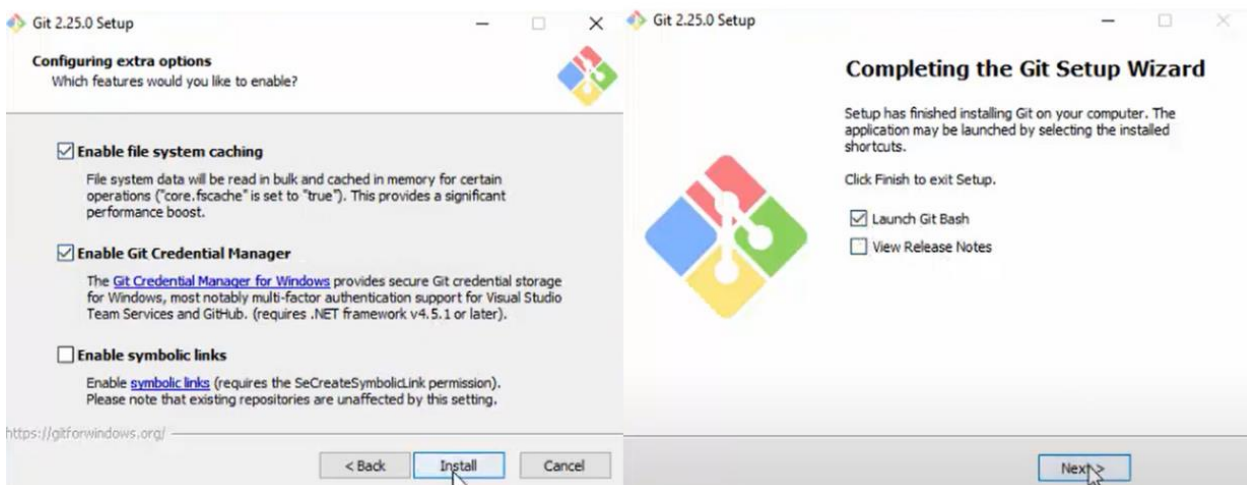
- Bu aşamada ise bağlantı sağlayacağımız yöntemi seçmemiz gerekiyor. commit ederken siz Windows, arkadaşınız Linux kullanıyorsa önemli!



- Bu aşamada Git Bash'ın kullanacağı terminali belirtiyoruz. İsterseniz MinTTY gibi çok fonksiyonlu bir terminal kullanabilir veya klasik "Komut istemcisini" kullanabilirsiniz.



- Veee, mutlu son :)



- Git Bash, Git'i kullanırken Windows işletim sistemi üzerinde bir komut satırı arabirimi sağlayan bir uygulamadır. Windows kullanıcılarına Git komutlarını çalıştırmak, depolarını yönetmek ve Unix benzeri komutları kullanmak için bir çözüm sunar. Git Bash aynı zamanda Git'in özelliklerini kullanmanıza ve Linux veya macOS terminali benzeri bir deneyim elde etmenize olanak tanır.

Git Bash, aşağıdaki önemli özelliklere sahiptir:

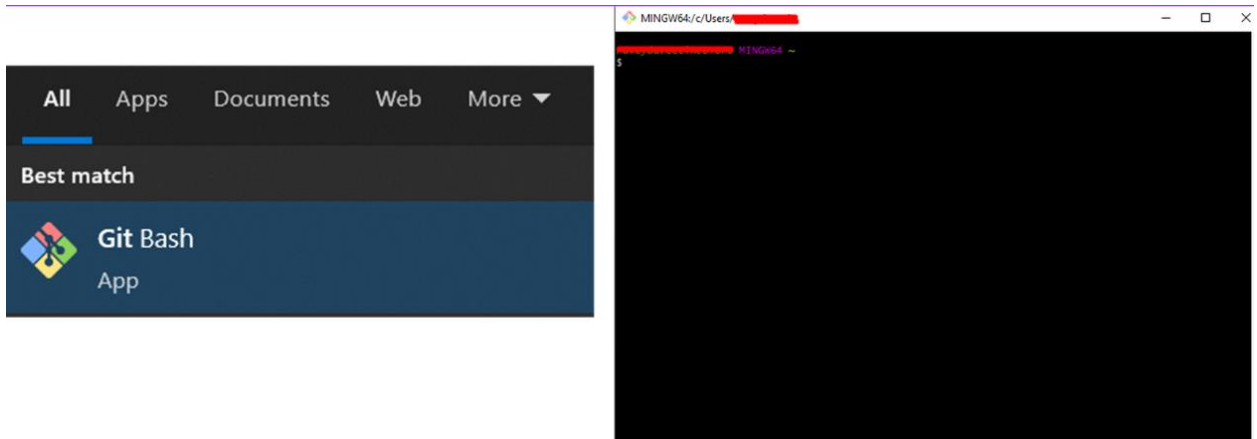
Git Komutları: Git Bash, Git komutlarını Windows'ta kullanmanıza izin verir. git init, git clone, git commit, git pull, git push gibi Git komutlarını burada kullanabilirsiniz.

Unix Benzeri Ortam: Git Bash, Unix ve Linux terminalini taklit eden bir komut satırı deneyimi sunar. Bu, Unix benzeri komutları kullanmayı kolaylaştırır.

Shell Desteği: Git Bash, Bash shell üzerine inşa edilmiştir ve birçok Unix komut dosyası (shell script) ve araçlarıyla uyumludur. Bu, özelleştirme ve otomasyon için kullanışlıdır.

Daha Fazla Araç: Git Bash ayrıca temel Git dışındaki araçları da içerebilir. Örneğin, SSH anahtarları oluşturmak ve yönetmek için kullanabileceğiniz ssh-keygen gibi araçlar mevcuttur.

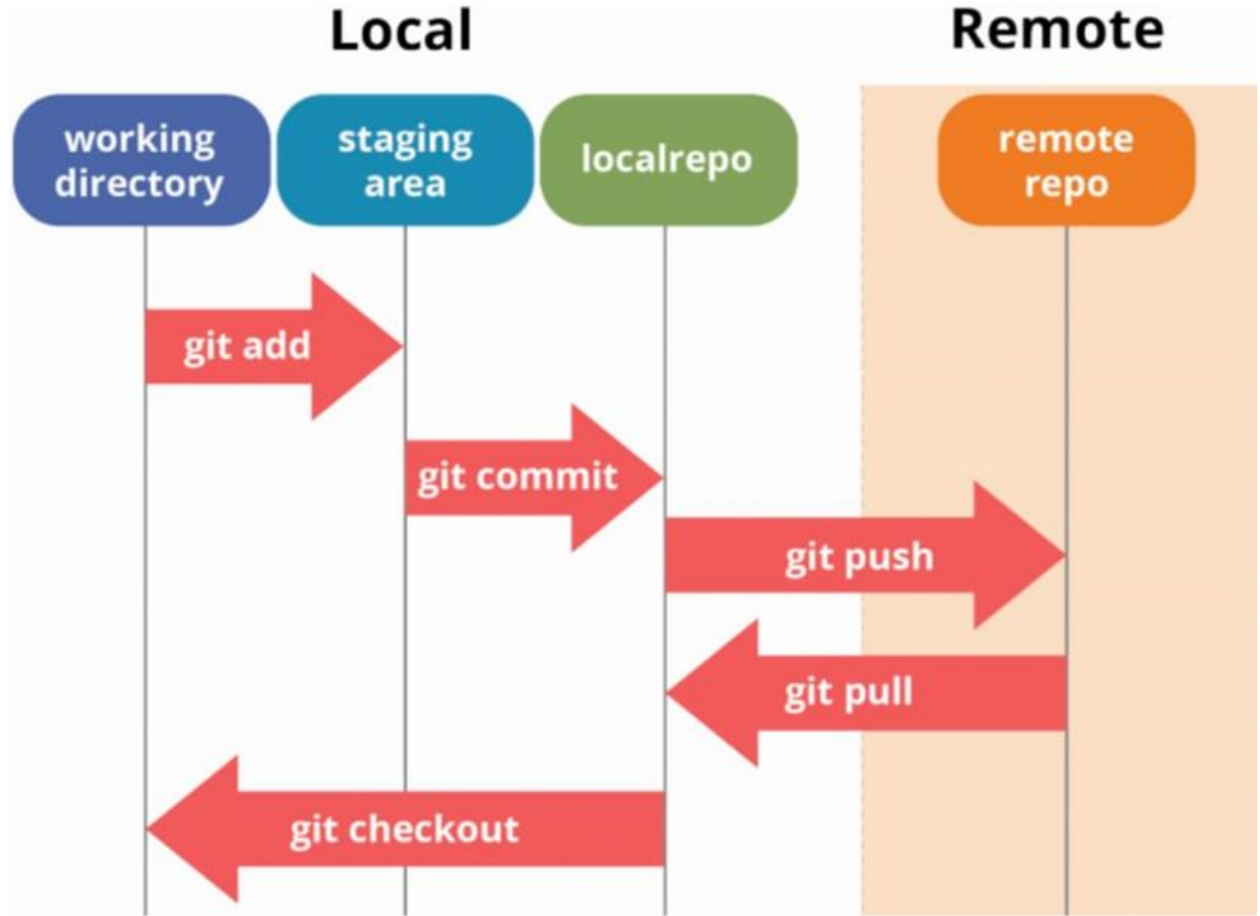
Git Bash, Windows kullanıcılarına Git'i kullanma ve geliştirme projelerini yönetme konusunda güçlü bir araç sunar. Ayrıca Git'in komut satırı arayüzünü tercih edenler için ideal bir seçenektir. Git Bash, Git'i Windows ortamında daha etkili bir şekilde kullanmanıza yardımcı olur.



Git Workflow

Genel Git iş akışı şu şekilde çalışır: Çalışmaya başladığınızda, değişiklikleri Working Directory'de yaparsınız. Değişiklikleri Staging Area'ya eklersiniz, bu değişiklikleri bir sonraki commit'e dahil etmek için hazırlama adımdır. Staging Area'daki değişiklikleri bir commit ile Local Repository'ye kaydedersiniz. Değişiklikler Local Repository'ye kaydedildikten sonra,

projeyi diğerk geliřtiricilerle paylařmak veya uzak depoya yklemek iin Remote Repository'e push yapabilirsiniz. Diğerk geliřtiricilerin yaptığı deėiřiklikleri almak ve projeyi gncellemek iin Remote Repository'den Local Repository'ye pull yapabilirsiniz. Bu genel Git iř akıřı, projelerin ynetimini ve iřbirliėini kolaylařtırmak iin kullanılır. Her bir adımı nasıl yapılacaėını ve Git komutlarını kullanarak bu iř akıřını nasıl uygulayacaėınızı ėrenmek, Git'i etkili bir řekilde kullanmanıza yardımcı olur.



Repository Kavramı

zerinde alıřılan projeyle ilgili tm dosyaları deėiřiklikleri, dallandırmalar bu repo zerinde bulundurulur. Projemizin zaman makinesi olarak dřnlebilir. İleri bir tarihe gidilebilir. Geri bir tarihe de gidilebilir. Repository Kavramı zetle bu anlama gelir.



Working Directory/Staging Area/Local Repository/Remote Repository

Git'in genel çalışma akışı, dört ana bölümden oluşur ve bu bölümler arasındaki geçişler, Git iş akışının temelini oluşturur.

Working Directory (Çalışma Dizini)

Bu, projenizin fiziksel dosyalarının saklandığı ve üzerinde çalıştığınız yerdir. Yaptığınız değişiklikler bu alanda bulunur.

Staging Area (Hazırlık Alanı)

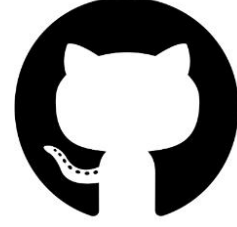
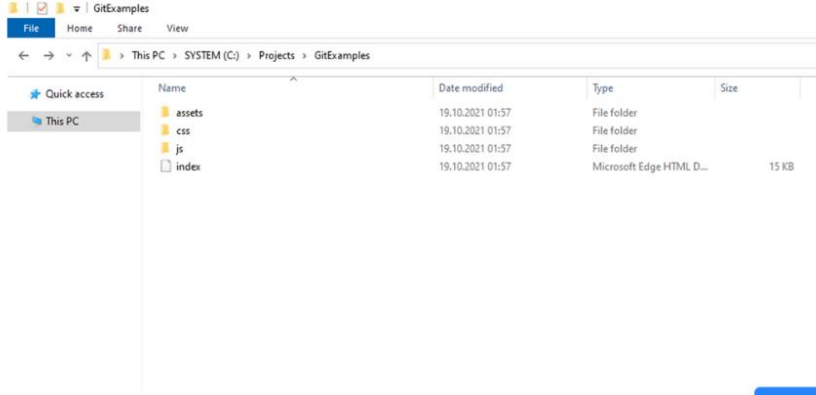
Değişikliklerinizi Working Directory'den Staging Area'ya eklersiniz. Staging Area, bir sonraki commit (sürüm) için hangi değişikliklerin dahil edileceğini belirlemenizi sağlar. `git add` komutu ile değişiklikleri Staging Area'ya eklersiniz.

Local Repository (Yerel Depo)

Staging Area'daki değişiklikleri bir commit (sürüm) olarak kaydedebilirsiniz. Local Repository, tüm projenin geçmiş sürümlerini ve tarihçesini içerir. Değişiklikleri burada sakladıktan sonra geri alabilir ve geçmiş sürümlere dönebilirsiniz.

Remote Repository (Uzak Depo)

Projenin merkezi depo (repository) olarak düşünülür. Diğer geliştiricilerle işbirliği yapmak için kullanılır. Projeyi paylaşmak ve senkronize etmek için kullanılır. Uzak Depo, GitHub, GitLab, Bitbucket gibi platformlarda veya başka bir sunucuda bulunabilir.



Commit Kavramı ve Commit Mesajı Önemi

Git için yapılan değişiklikleri kaydettiğimiz bir alandır. Yani yaptığımız değişiklikleri kaydediyoruz ve gönderiyoruz olarak düşünebiliriz.

Özgün ve Açıklayıcı Olun

Commit mesajınız neyi değiştirdiğinizi ve neden değiştirdiğinizi açık bir şekilde ifade etmelidir.

Özgün Başlık Kullanın

Commit başlığınız kısa, özgün ve açıklayıcı olmalıdır. Başlık, bu değişikliği açıklayan bir anahtar kelime veya ifade içermelidir.

Ayrıntıları Alt Satıra Ekleyin

Commit başlığından sonra, ayrıntıları açıklamak için ikinci bir satır ekleyin. Ayrıntılar, neyi değiştirdiğinizi ve nedenini daha fazla açıklamalıdır.

İş İlgili Referansları Ekleyin:

İş akışınızı izlemek ve değişikliklerin nedenini anlamak için ilgili **sorun numarası** veya **talep numarası** gibi referansları eklemek iyi bir uygulamadır.

Zaman Etiketini Ekleme

Commit mesajınıza tarih veya saat eklemek yerine Git'in bu bilgiyi zaten sakladığını unutmayın.

İmla ve Dilbilgisine Dikkat Edin

Mesajınızın imla, dilbilgisi ve yazım kurallarına uygun olduğundan emin olun.



```
% git add .  
% git commit -m '1'
```

Remember, well-written commit messages **improve collaboration**, make it easier to understand changes, and help maintain a clean and organized commit history. Adopting these best practices will contribute to **better code** management and collaboration within your team.



GIT COMMIT-M
"T:JIRA-123|ISPRINT2|
(V1.2)FIXED
BUG PREVENTING
USER SIGNING IN"

GIT COMMIT
-M
"FIXES"

Faydalı Git Komutları: git config

- Kullanıcı adı email işlemleri kaydedilir.
- Config'den de kullanıcı bilgileri görünmüş olur.
- git config, Git'in yapılandırma ayarlarını belirlemenizi ve yapılandırmalarınızı görüntülemenizi sağlayan bir Git komutudur.
- Git'i daha iyi kişiselleştirmek, kullanıcı bilgilerini tanımlamak, proje özgü ayarları yapmak ve diğer Git ayarlarını kontrol etmek için kullanılır. git config komutu ile aşağıdaki türde yapılandırmaları belirleyebilirsiniz

```
git config--list
git config -- global user.name = 'Kardel'
git config -- global user.emalil = 'ruveydakardelcetin@gmail.com'
```

```
MINGW64 /c/Projects/GitExamples
$ git config --list
diff.astextplain.textconv=astextplain
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
credential.helper=manager-core
pull.rebase=false
credential.https://[redacted]
init.defaultbranch=master
user.name=Ruveyda Cetin
user.email=[redacted]
core.editor="C:/Program Files (x86)/GitExtensions/GitExtensions.exe" fileeditor
```

Faydalı Git Komutları: git init

- Local Repository oluşturmak için çalışma klasörünün içerisine git init denildiğinde boş bir git repository oluşturulur.
- .git adında bir klasör oluşturulur. Eğer bu klasör görülmediyse Hidden Items kısmı kapalıdır.
- .git dosyası, bir Git deposunun (repository) kalbidir ve Git'in projeyi yönetmek, versiyon kontrolü yapmak ve geçmiş değişiklikleri izlemek için kullanıldığı bir dizi dosya ve alt klasörden oluşur. Bu dosyalar ve klasörler, projenin tam tarihçesini, dallarını, commit'lerini ve diğer Git verilerini içerir.

.git dosyası şunları içerir:

objects: Bu klasör, Git'in içindeki nesnelerin (bloklar ve ağaçlar) saklandığı yerdir. Bu nesneler, commit'lerin, ağaçların ve dosyaların sürüm bilgilerini içerir.

refs: Bu klasör, daların (branches), etiketlerin (tags) ve diğer başvuruların depolandığı yerdir. Özellikle, refs/heads klasörü, projedeki tüm dalları ve onların son commit'leri içerir.

HEAD: Bu dosya, şu an hangi dalın aktif olduğunu gösterir. Yani projenin şu anki konumunu belirler.














config: Bu dosya, Git yapılandırma ayarlarını içerir. Kullanıcı bilgileri, uzak depo (remote repository) adresleri ve diğer yapılandırma seçenekleri burada saklanır.

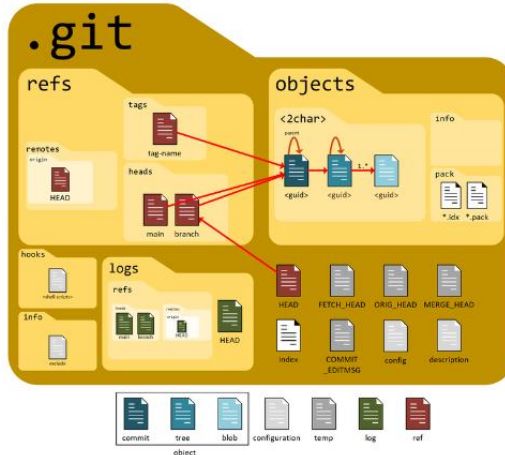
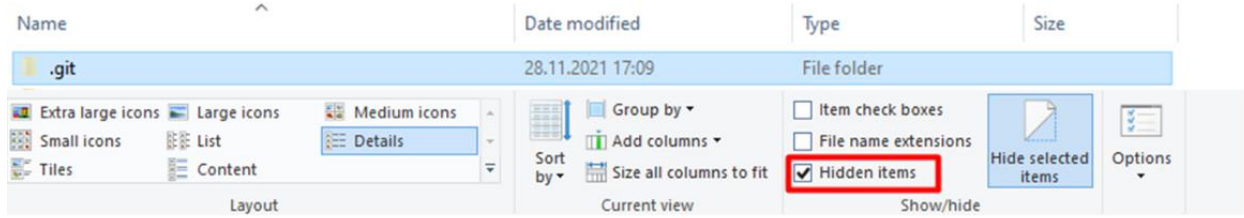
index: Bu dosya, Staging Area veya Hazırlık Alanı olarak bilinen alandaki değişikliklerin bir listesini içerir. Yani, bir sonraki commit'te hangi dosyaların dahil edileceğini belirler.

logs: Bu klasör, Git'in tarihçe kayıtlarını (log) tuttuğu yerdir. Özellikle, refs/heads altındaki dalların değişiklikleri burada kaydedilir.

.git dosyası, projenin tam tarihçesini ve Git'in yönetimini sağlamak için kritik bir rol oynar. Genellikle bu dosyayı elle düzenlemeniz gerekmez; çünkü Git, bu dosyaları otomatik olarak yönetir. Ancak, bu dosyaları incelemek ve anlamak, Git'i daha derinlemesine anlamanıza yardımcı olabilir.

Örnek bir .git klasörü

 hooks	17.10.2023 12:53	File folder
 info	17.10.2023 12:53	File folder
 logs	17.10.2023 12:54	File folder
 objects	21.11.2023 19:04	File folder
 refs	17.10.2023 12:54	File folder
 COMMIT_EDITMSG	21.11.2023 19:04	File
 config	16.11.2023 11:14	File
 description	17.10.2023 12:53	File
 FETCH_HEAD	21.11.2023 19:04	File
 HEAD	16.11.2023 11:14	File
 index	21.11.2023 19:04	File
 ms-persist	23.11.2023 10:27	Microsoft Edge HT...
 ORIG_HEAD	21.11.2023 19:04	File



Faydalı Git Komutları: git status

git status, Git komutlarından biridir ve mevcut çalışma dizinindeki değişiklikleri ve Git deposunun (repository) durumunu incelemenizi sağlar. Bu komut, Git ile çalışırken proje ve dosyalarınızın hangi aşamalarda olduğunu anlamanıza yardımcı olur. git status komutu aşağıdaki bilgileri sağlar:

- **Değiştirilen Dosyalar (Modified Files):** Eğer çalışma dizinindeki dosyalarda değişiklikler varsa, bu dosyalar git status çıktısında listelenir. Bu değişiklikler henüz stajlanmış (Staging Area'ya eklenmemiş) durumdadır.
- **Staging Area'da Bulunan Dosyalar (Staged Files):** Eğer bazı dosyalar Staging Area'ya eklenmişse, yani bir sonraki commit'te dahil edilmeyi bekliyorlarsa, bu dosyalar da git status çıktısında listelenir.
- **Yeni Eklenen Dosyalar (New Files):** Eğer yeni oluşturulmuş dosyalar varsa ve henüz Staging Area'ya eklenmemişlerse, bu dosyalar git status çıktısında "untracked files" (izlenmeyen dosyalar) olarak gösterilir.
- **Silinen Dosyalar (Deleted Files):** Eğer bir dosya silinmişse ve bu silme işlemi henüz commit edilmemişse, bu dosya da git status çıktısında görünür.
- **Hangi Dalın Üzerinde Çalıştığınız (On branch):** Hangi dalın (branch) üzerinde olduğunuz git status çıktısında belirtilir. Bu, projenin hangi sürümünü düzenlediğinizi gösterir.

- **Geçmiş ve Gelecekteki İşaretçiler (Your branch is ahead of 'origin/master' by 3 commits):** Eğer uzak bir depo (remote repository) ile senkronize değilseniz ve örneğin yerel depo daha fazla commit içeriyorsa, bu bilgi git status çıktısında görünür.

git status komutunu düzenli olarak kullanmak, projenizdeki değişiklikleri ve durumu takip etmenize yardımcı olur. Bu, hangi dosyaların Staging Area'ya eklenmesi gerektiğini belirlemenize ve hangi dosyaların commit'e dahil edilmesi gerektiğini anlamanıza yardımcı olur.

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    assets/
    css/
    index.html
    js/

nothing added to commit but untracked files present (use "git add" to track)
```

Faydalı Git Komutları: git add .

git add . komutu, Git ile çalışırken çalışma dizininde yapılan tüm değişiklikleri Staging Area (Hazırlık Alanı) olarak adlandırılan alana eklemek için kullanılır. Staging Area, bir sonraki commit (sürüm) için hangi değişikliklerin dahil edileceğini belirlediğiniz yerdir. Bu komut, değişiklikleri tüm dosyaları ve alt klasörleri içeren bir şekilde Staging Area'ya ekler.

- **Tüm Değişiklikleri Ekleme:** git add . komutunu çalıştırdığınızda, çalışma dizinindeki tüm değişiklikler (yeni dosyalar, değiştirilen dosyalar ve silinen dosyalar) Staging Area'ya eklenir. Bu, mevcut çalışma dizininde yapılan tüm değişikliklerin bir sonraki commit'e dahil edilmesini sağlar.
- **Hızlı ve Kapsamlı Ekleme:** Bu komut, tüm değişiklikleri toplu bir şekilde Staging Area'ya eklemenizi sağlar. Bu, birden fazla dosya üzerinde çalıştığınızda ve tüm değişiklikleri tek bir commit'te kaydetmek istediğinizde oldukça kullanışlıdır. -**
Commit Öncesi Kontrol: **git status komutunu kullanarak hangi dosyaların Staging Area'da olduğunu ve hangilerinin henüz eklenmediğini kontrol edebilirsiniz. Bu, bir commit yapmadan önce son kontrolü yapmanıza yardımcı olur.

Not: git add . komutu, tüm değişiklikleri ekler, ancak dikkatli kullanılmalıdır çünkü yanlışlıkla eklenmemesi gereken dosyaları da dahil edebilir. Değişiklikleri dikkatlice gözden geçirip sadece ilgili dosyaları Staging Area'ya eklemek, daha kontrollü ve temiz bir iş akışı sağlayabilir.

```
MINGW64 /c/Projects/GitExamples (master)
$ git add .
warning: LF will be replaced by CRLF in css/styles.css.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in js/scripts.js.
The file will have its original line endings in your working directory
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   assets/img/favicon.ico
    new file:   assets/img/profile.jpg
    new file:   css/styles.css
    new file:   index.html
    new file:   js/scripts.js
```

Faydalı Git Komutları: git commit

git commit, Git'in temel komutlarından biridir ve yerel depoya (local repository) yapılan değişiklikleri kaydetmek veya bir sürümü oluşturmak için kullanılır. Commit işlemi, çalışma dizinindeki (working directory) değişiklikleri Staging Area'da hazırlanmış (staged) dosyalar haline getirir ve bu değişiklikleri yerel depoya kaydeder.

Değişiklikleri Kaydetme: git commit, çalışma dizininde yapılan değişiklikleri yerel depoya kalıcı olarak kaydeder. Bu, bir iş veya özellik üzerinde çalışmanızı tamamladığınızda bu değişiklikleri projenin tarihçesine dahil etmenizi sağlar.

Commit Mesajı Ekleme: Her commit işlemi için açıklayıcı bir commit mesajı eklemek önemlidir. Commit mesajı, bu değişikliğin neden yapıldığını ve neyi ifade ettiğini anlamak için önemlidir.

Projeyi Versiyonlamak: Commitler, projenizin versiyonlarını oluşturur. Her commit, projenin bir önceki sürümünden ne kadar değiştiğini ve neyin eklenip çıkarıldığını gösterir.

```
MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "Başlangıç dosyaları gönderildi."
[master (root-commit) 6c06d91] Başlangıç dosyaları gönderildi.
5 files changed, 11694 insertions(+)
create mode 100644 assets/img/favicon.ico
create mode 100644 assets/img/profile.jpg
create mode 100644 css/styles.css
create mode 100644 index.html
create mode 100644 js/scripts.js

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
nothing to commit, working tree clean
```

- Yukarıdaki örnekte, -m bayrağı ile birlikte commit mesajını doğrudan komut satırından ekledik. Commit mesajını daha detaylı bir şekilde eklemek için ise git commit komutunu çalıştırdığınızda bir metin düzenleyici açılır ve burada commit mesajınızı yazabilirsiniz.
- Commit işlemi, Git'in temel prensiplerinden biridir ve projelerin geçmişini, işbirliği yapmayı ve hata ayıklamayı yönetmek için hayati bir rol oynar. Commitler, projenizin tarihini belgelemek için kullanılır ve gerektiğinde geçmiş sürümlere geri dönmenize olanak tanır. -m, git commit komutuyla birlikte kullanılan bir bayrak (flag) veya seçenektir. -m, commit mesajını komut satırında doğrudan girmenizi sağlar. Yani, -m bayrağı, commit mesajını bir metin dizesi olarak girme işlevi sağlar.

Faydalı Git Komutları: clear

Git bash'de clear işlemi yapar ve tüm komut sistemi temizlenir.

```
MINGW64 /c/Projects/GitExamples (master)
$ clear|
```



Faydalı Git Komutları: Projede Değişiklik Yapalım!

- Projede index.html sayfasında değişiklik gerçekleştirdik Çalıştığımız dizinde hangi alanda değişiklik yaptığımızı git status ile görebiliriz. Git status görüldüğü zaman dosyanın düzenlendiği bilgisi verilir.

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
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:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

- Öncelikle tek bir dosya olduğu için geçiş bölgesine gönderirken git add index.html diyerek dosya dizininden geçiş bölgesine gönderilir.

```
MINGW64 /c/Projects/GitExamples (master)
$ git add index.html
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html
```


```
MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "Bir takım değişiklikler"
[master 4cffd4e] Bir takım değişiklikler
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
nothing to commit, working tree clean
```

- Yeni bir dosya ekleyelim.
- Varolan dosyada değişiklik yapalım.
- Touch yeni bir dosya eklenmesini sağlar.
- Dosyan düzenlendi. Ancak untracked diye belirlemiş, çünkü bu dosya git'e henüz bildirilmedi. Local repo bilmediği dosya için izlenmemiş untrackked der. Staging areaya göndermek için git add -A dersek yaptığımız tüm değişiklikleri staging area'ya gönderir.

 MINGW64:/c/Projects/GitExamples

```
 MINGW64 /c/Projects/GitExamples (master)
$ touch text.txt|
```

```
 MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
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:   index.html

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

no changes added to commit (use "git add" and/or "git commit -a")
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git add -A
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html
        new file:   text.txt
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "text.txt oluşturuldu. Index.html düzenlendi."
[master 45bdad1] text.txt oluşturuldu. Index.html düzenlendi.
2 files changed, 1 insertion(+), 1 deletion(-)
create mode 100644 text.txt
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Faydalı Git Komutları: Bir dosyanın ismini değiştirelim!

- Ls -al dersiniz dosyaların isimlerini görmüş olursunuz.
- Mv text.txt readme.txt
- Önceki dosya ismini silinmiş olarak algılar
- Yeni dosya ismini ise untracked olarak algılanır.
- Git add . Veya git add -A diyebilirsiniz.

```
MINGW64 /c/Projects/GitExamples (master)
$ ls -al
total 28
drwxr-xr-x 1 ruveyda.cetin 1049089  0 Nov 28 20:22 ./
drwxr-xr-x 1 ruveyda.cetin 1049089  0 Nov 28 15:57 ../
drwxr-xr-x 1 ruveyda.cetin 1049089  0 Nov 28 21:24 .git/
drwxr-xr-x 1 ruveyda.cetin 1049089  0 Oct 19 01:57 assets/
drwxr-xr-x 1 ruveyda.cetin 1049089  0 Nov 28 20:20 css/
-rw-r--r-- 1 ruveyda.cetin 1049089 14606 Nov 28 20:22 index.html
drwxr-xr-x 1 ruveyda.cetin 1049089  0 Oct 19 01:57 js/
-rw-r--r-- 1 ruveyda.cetin 1049089  0 Nov 28 20:22 text.txt
```

```
MINGW64 /c/Projects/GitExamples (master)
$ mv text.txt readme.txt
```

Faydalı Git Komutları: Geçiş bölgesine gönderilen değişiklikler arası geçiş

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    text.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git add -A
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "İsim değişikliği"
[master 7941c42] İsim değişikliği
1 file changed, 0 insertions(+), 0 deletions(-)
rename text.txt => readme.txt (100%)
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
nothing to commit, working tree clean
```

- index.html'de değişiklik yapalım. Sonrasında bu yaptığımız değişikliği geçiş bölgesine aldıktan sonra geçiş bölgesinden çıkaralım.
- Geçiş bölgesinden çıkardıktan sonra çalışma düzenine düşer. Oradan da çıkaralım!

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git restore --staged index.html
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
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:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git restore index.html
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Faydalı Git Komutları: git log

git log, Git deposundaki commit tarihçesini incelemek için kullanılan bir komuttur. Bu komut, projenizin geçmiş commit'lerini, kim tarafından yapıldığını, ne zaman yapıldığını, hangi commit mesajlarına sahip olduğunu ve commit kimliklerini (SHA-1 hash) görüntüler. git log, projenin evrimini izlemek, hata ayıklama yapmak, işbirliği yapmak ve projenin geçmiş sürümleri hakkında bilgi edinmek için çok önemli bir araçtır.

git log komutunun bazı yaygın kullanımları şunlardır:

- git log: Tüm commitleri tarih sırasına göre gösterir.
- git log -n: En son n commit'i gösterir. Örneğin, git log -3, en son 3 commit'i gösterir.
- git log --author=: Belirli bir yazarın commit'lerini filtreler.
- git log : Belirli bir dalın commit tarihçesini görüntüler.

- `git log --oneline`: Her commit için yalnızca kısa bir kimlik ve commit mesajı gösterir. `git log` komutu, projenizin geçmişini anlamak, hata ayıklama yapmak ve işbirliği yapmak için oldukça önemli bir araçtır. Tüm commitleri tarih sırasına göre gösterir.

```

MINGW64 /c/Projects/GitExamples (master)
$ git log -p -2
commit 7941c428bd8a8d586e4bb208c68a8bc548422999 (HEAD -> master)
Author: Ruveyda Cetin <ruveyda.cetin@d-teknoloji.com.tr>
Date:   Sun Nov 28 21:43:39 2021 +0300

    İsim değişikliği

diff --git a/text.txt b/readme.txt
similarity index 100%
rename from text.txt
rename to readme.txt

commit 45bdad1c8c2a0b5069324256af10ebba0456b546
Author: Ruveyda Cetin <ruveyda.cetin@d-teknoloji.com.tr>
Date:   Sun Nov 28 21:22:53 2021 +0300

    text.txt oluşturuldu. Index.html düzenlendi.

diff --git a/index.html b/index.html
index f87229b..9f2805a 100644
--- a/index.html
+++ b/index.html
@@ -60,7 +60,7 @@
     <!-- Experience-->
     <section class="resume-section" id="experience">
       <div class="resume-section-content">

```

```

MINGW64 /c/Projects/GitExamples (master)
$ git log --since=10minutes

```

Faydalı Git Komutları: ZAMAN GERİYE AKSIN!

Commitler içerisinde Birtakım değişiklikler commit'ine geri dönelim. Bunu yapabilmek için `git checkout logId` dememiz yeterlidir. Bu işlem yapıldığında Head kısmı o commit'e dönmüş olur. Zaman makinesinde geriye gelmiş oldu. Geriye gidebiliyorsak, ileriye de gidebiliriz. Tamamıyla geri gelmek değil de versiyonlar arasında geçiş yapılabilir.

`git checkout`, Git'in esnek bir komutu olup, proje çalışma dizinini değiştirmek ve farklı dallar arasında geçiş yapmak için kullanılır. Bu komut, işbirliği ve projenin farklı sürümleri arasında geçiş yapma yeteneği sağlar. `git checkout` komutu aşağıdaki temel amaçlar için kullanılır:

Dallar Arasında Geçiş Yapma:

`git checkout` komutu, farklı projelerin dalları arasında geçiş yapmanıza olanak tanır. Örneğin, bir geliştirme dalından ana dala geçiş yapabilirsiniz.

Belirli Bir Commit'e Geri Dönme:

git checkout komutu, belirli bir commit'in durumuna geri dönmeyi sağlar. Bu, projenizin geçmiş sürümlerini incelemek veya bir hatayı düzeltmek için kullanışlıdır. Dosyaları veya Dalları İşaretleme (tagging):

git checkout komutu, etiketleri (tags) veya dal isimlerini kullanarak belirli bir projenin sürümünü işaretleme için kullanılabilir.

```
MINGW64 /c/Projects/GitExamples (master)
$ git log --oneline
7941c42 (HEAD -> master) İsim değişikliği
45bdad1 text.txt oluşturuldu. Index.html düzenlendi.
4cffd4e Bir takım değişiklikler
6c06d91 Başlangıç dosyaları gönderildi.

MINGW64 /c/Projects/GitExamples (master)
$ git checkout 4cffd4e
Note: switching to '4cffd4e'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false
HEAD is now at 4cffd4e Bir takım değişiklikler

MINGW64 /c/Projects/GitExamples ((4cffd4e...))
$ git log --oneline
4cffd4e (HEAD) Bir takım değişiklikler
6c06d91 Başlangıç dosyaları gönderildi.
```

```
MINGW64 /c/Projects/GitExamples ((4cffd4e...))
$ git checkout master
Previous HEAD position was 4cffd4e Bir takım değişiklikler
Switched to branch 'master'

MINGW64 /c/Projects/GitExamples (master)
$ git log --oneline
7941c42 (HEAD -> master) İsim değişikliği
45bdad1 text.txt oluşturuldu. Index.html düzenlendi.
4cffd4e Bir takım değişiklikler
6c06d91 Başlangıç dosyaları gönderildi.
```



Faydalı Git Komutları: Gidip de dönmek istemezsek?

İki yol vardır. Revert ve Reset

git revert, Git'te bir önceki commit'in değişikliklerini geri almak için kullanılan bir komuttur. Bu komut, mevcut bir dal üzerindeki son commit'i geri alır ve bu değişiklikleri yeni bir commit ile kaydeder. git revert işlemi, geçmiş commit'lerin değiştirilmesi yerine, yeni bir commit ekleyerek geçmiş bir hatanın düzeltilmesine veya istenmeyen bir değişikliğin geri alınmasına olanak tanır.

git revert komutunu kullanmanın temel nedenleri şunlar olabilir:

- **Hatalı bir commit'i geri almak:** Eğer son commit'inizde bir hata yaptıysanız, bu hatayı düzeltmek için git revert kullanabilirsiniz. Yeni bir commit ile hata düzeltilir ve geçmiş sürüm korunur.
- **Bir değişikliği geri almak:** Eğer bir önceki commit'de eklediğiniz bir değişikliği geri almak isterseniz, git revert bu değişikliği geri almanıza yardımcı olur.

- **İşbirliği durumlarında kullanım:** Ekip içindeki diğer geliştiricilerin çalışmalarını bozmamak için git revert, daha önce paylaşılmış olan bir commit'i düzeltebilmeniz için kullanışlıdır. Bu, geçmişteki bir hata veya eksiklik durumunda kullanışlıdır.

git revert komutu, geçmiş commit'leri korurken hataları düzeltmek veya değişiklikleri geri almak için kullanışlı bir araçtır. Bu sayede geçmiş sürümleri bozmamanız ve işbirliği içinde çalışmanız mümkün olur.

```
MINGW64 /c/Projects/GitExamples (master)
$ git revert 45bdad1
hint: Waiting for your editor to close the file... |
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git revert 45bdad1
[master 961e09e] Revert "text.txt oluşturuldu. Index.html düzenlendi."
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git log --oneline
961e09e (HEAD -> master) Revert "text.txt oluşturuldu. Index.html düzenlendi."
7941c42 İsim değişikliği
45bdad1 text.txt oluşturuldu. Index.html düzenlendi.
4cffd4e Bir takım değişiklikler
6c06d91 Başlangıç dosyaları gönderildi.
```

```
C:/Projects/GitExamples/git/COMMIT_EDITMSG
1 Revert "text.txt oluşturuldu. Index.html düzenlendi."
2
3 This reverts commit 45bdad1c2a0b5069324256af10ebba0450b546.
4
5 # Please enter the commit message for your changes. Lines starting
6 # with '#' will be ignored, and an empty message aborts the commit.
7 #
8 # On branch master
9 # Changes to be committed:
10 #   modified:   index.html
11 #
12 |
```



Faydalı Git Komutları: Sil Baştan Başlamak Gerek Bazen!

git reset, Git'te bir önceki commit'leri veya projenizin durumunu değiştirmek için kullanılan bir komuttur. Bu komut, projenin tarihçesini değiştirir ve belirli commit'leri geri alabilir veya birleştirebilir. git reset komutu, farklı modlarıyla gelir ve hangi modun kullanıldığına bağlı olarak farklı işlevlere sahiptir.

- **Soft Reset (git reset --soft):** Bu mod, belirli bir commit'i geri alır, ancak yapılan değişiklikleri çalışma dizini (working directory) ve Staging Area'da (Hazırlık Alanı) korur. Yani, geri alınan commit sonrası değişiklikleri tekrar inceleme ve yeni bir commit oluşturma fırsatı sunar.
- **Mixed Reset (git reset --mixed):** Bu mod, belirli bir commit'i geri alır ve Staging Area'daki değişiklikleri siler, ancak çalışma dizinindeki değişiklikleri korur. Bu, commit'e gitmeyi unuttuğunuz veya hatalı bir commit'i düzeltmeniz gerektiğinde kullanışlıdır.

- **Hard Reset (git reset --hard):** Bu mod, belirli bir commit'i geri alır ve bu commit sonrası yapılan tüm değişiklikleri tamamen siler. Bu, bir commit'i veya bir dizi commit'i geri almak ve değişiklikleri tamamen kaldırmak istediğinizde kullanılır.

git reset, özellikle projenizin geçmişini yönetirken veya hatalı commit'leri düzeltirken kullanışlıdır. Ancak dikkatli kullanılmalıdır çünkü projedeki değişiklikleri geri alabilir ve bu değişiklikler kaybolabilir. Bu nedenle, bu komutu kullanmadan önce dikkatlice düşünmelisiniz.

TEHLİKELİDİR. GERİ DÖNÜŞ YOKTUR

```
MINGW64 /c/Projects/GitExamples (master)
$ git log
commit 961e09e94c221fbec1616291ab93d597f86d6545 (HEAD -> master)
Author: Ruveyda Cetin <ruveyda.cetin@td-teknoji.com.tr>
Date: Sun Nov 28 22:48:23 2021 +0300

    Revert "text.txt oluşturuldu. Index.html düzenlendi."

    This reverts commit 45bdad1c8c2a0b5069324256af10ebba0456b546.

commit 7941c428bd8a8d586e4bb208c68a8bc548422999
Author: Ruveyda Cetin <ruveyda.cetin@td-teknoji.com.tr>
Date: Sun Nov 28 21:43:39 2021 +0300

    Isim değişikliği

commit 45bdad1c8c2a0b5069324256af10ebba0456b546
Author: Ruveyda Cetin <ruveyda.cetin@td-teknoji.com.tr>
Date: Sun Nov 28 21:22:53 2021 +0300

    text.txt oluşturuldu. Index.html düzenlendi.

commit 4cfff44efcb6e2a0d575eda5ffbb57b983daf4e6a
Author: Ruveyda Cetin <ruveyda.cetin@td-teknoji.com.tr>
Date: Sun Nov 28 19:11:42 2021 +0300

    Bir takım değişiklikler

commit 6c06d91bc6537a4b9fc4d177e82a66c48824a520
Author: Ruveyda Cetin <ruveyda.cetin@td-teknoji.com.tr>
Date: Sun Nov 28 17:27:38 2021 +0300

    Başlangıç dosyaları gönderildi.
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git reset --soft 6c06d91bc6537a4b9fc4d177e82a66c48824a520
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html
        new file:   readme.txt
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git log --oneline
6c06d91 (HEAD -> master) Başlangıç dosyaları gönderildi.
```

```

MINGW64 /c/Projects/GitExamples (master)
$ git reset --mix 6c06d91bc6537a4b9fc4d177e82a66c48824a520
Unstaged changes after reset:
M    index.html

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
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:   index.html

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

no changes added to commit (use "git add" and/or "git commit -a")

MINGW64 /c/Projects/GitExamples (master)
$ git log --oneline
6c06d91 (HEAD -> master) Başlangıç dosyaları gönderildi.

```

```

MINGW64 /c/Projects/GitExamples (master)
$ git reset --hard 6c06d91bc6537a4b9fc4d177e82a66c48824a520
HEAD is now at 6c06d91 Başlangıç dosyaları gönderildi.

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        readme.txt

nothing added to commit but untracked files present (use "git add" to track)

```

Faydalı Git Komutları: Git ignore

- Repository üzerinden takip edilmesini istenmeyen dosyalar bu dosyada tutulur.
- .gitignore dosyası tutulmalıdır.
- .gitignore dosyası en başta oluşturulmalıdır. Sonradan oluşturulduğunda hemen algılanmayabilir. O nedenle cacheleri temizlemek gerekir.

.gitignore dosyasının kullanımı şunlara hizmet eder:

- **Üretilen Dosyaları veya Derleme Çıktılarını İzlememek:** Projenizin derleme çıktıları, geçici dosyalar veya projenin çalışması sırasında üretilen dosyalar genellikle .gitignore dosyasına eklenir. Bu, bu tür dosyaların Git tarihçesine dahil edilmemesini sağlar.

- **Dosyaları veya Gizli Bilgileri Korumak:** API anahtarları, şifreler veya başka hassas bilgiler, .gitignore ile korunabilir. Bu sayede bu tür bilgilerin Git tarihçesinde saklanması önüne geçilir.
- **Geçici Dosyaları İzlememek:** Geçici dosyalar ve veritabanı yedeklemeleri, Git tarihçesine eklenmemesi gereken dosyalardır. .gitignore, bu tür dosyaları izlemekten korumanıza yardımcı olur.

.gitignore dosyası, projenin daha temiz ve daha yönetilebilir olmasına yardımcı olur. Ayrıca, hassas bilgilerin Git tarihçesine eklenmesini önler ve gereksiz dosyaları paylaşarak projeyi daha hızlı hale getirir. Bu nedenle, Git depolarını oluştururken ve paylaşırken .gitignore dosyası oluşturmak önemlidir.

```
MINGW64 /c/Projects/GitExamples (master)
$ touch .gitignore

MINGW64 /c/Projects/GitExamples (master)
$ touch log.txt

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        log.txt

nothing added to commit but untracked files present (use "git add" to track)

MINGW64 /c/Projects/GitExamples (master)
$ git add .

MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "Git ignore ve log.txt"
[master 26e4641] Git ignore ve log.txt
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 .gitignore
create mode 100644 log.txt

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
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:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git add .
g
MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "Git ignore güncellendi. "
[master 03ced57] Git ignore güncellendi.
1 file changed, 1 insertion(+)

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
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:   log.txt
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git rm -r --cached
fatal: No pathspec was given. which files should I remove?

ruveyda.cetin@DT870 MINGW64 /c/Projects/GitExamples (master)
$ git rm -r --cached .
rm '.gitignore'
rm 'assets/img/favicon.ico'
rm 'assets/img/profile.jpg'
rm 'css/styles.css'
rm 'index.html'
rm 'js/scripts.js'
rm 'log.txt'

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:   .gitignore
        deleted:   assets/img/favicon.ico
        deleted:   assets/img/profile.jpg
        deleted:   css/styles.css
        deleted:   index.html
        deleted:   js/scripts.js
        deleted:   log.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        assets/
        css/
        index.html
        js/
```



```
MINGW64 /c/Projects/GitExamples (master)
$ git add .
warning: LF will be replaced by CRLF in css/styles.css.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in js/scripts.js.
The file will have its original line endings in your working directory
```

```
MINGW64 /c/Projects/GitExamples (master)
$ git add .
warning: LF will be replaced by CRLF in css/styles.css.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in js/scripts.js.
The file will have its original line endings in your working directory

MINGW64 /c/Projects/GitExamples (master)
$ git commit -m "Git ignore cache çözüldü. "
[master 156cd0a] Git ignore cache çözüldü.
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 log.txt

MINGW64 /c/Projects/GitExamples (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Faydalı Git Komutları: Git Branching

Üzerinde çalışılan projenin farklı geliştirmeler için farklı dallara ayrılmasını sağlar. Örneğin bir projede html ile ilgili değişikliklerden bir kişi sorumlu olsun, css ile ilgili gelişmelerden bir kişi sorumlu olsun. Bu kişilerin çalışmalarının engelenmemesi için branch mantığından yararlanılır. Ayrı ayrı branchlerde çalışarak sonrasında hepsi master'a merge işlemini gerçekleştirebilirler. Master'ın kendisini de bir dal olarak düşünebilirsiniz.


```
MINGW64 /c/Projects/GitExamples (css)
$ git branch -a
* css
  html
  master

MINGW64 /c/Projects/GitExamples (css)
$ git status
On branch css
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:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

MINGW64 /c/Projects/GitExamples (css)
$ git add index.html

MINGW64 /c/Projects/GitExamples (css)
$ git status
On branch css
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html

MINGW64 /c/Projects/GitExamples (css)
$ git commit -m "index değişiklik"
[css b85f797] index değişiklik
1 file changed, 1 insertion(+), 1 deletion(-)

MINGW64 /c/Projects/GitExamples (css)
$ git status
On branch css
nothing to commit, working tree clean
```

```
MINGW64 /c/Projects/GitExamples (css)
$ git branch -a
* css
  html
  master

MINGW64 /c/Projects/GitExamples (css)
$ git status
On branch css
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:   index.html

no changes added to commit (use "git add" and/or "git commit -a")

MINGW64 /c/Projects/GitExamples (css)
$ git add index.html

MINGW64 /c/Projects/GitExamples (css)
$ git status
On branch css
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.html

MINGW64 /c/Projects/GitExamples (css)
$ git commit -m "index değışiklik"
[css b85f797] index değışiklik
1 file changed, 1 insertion(+), 1 deletion(-)

MINGW64 /c/Projects/GitExamples (css)
$ git status
On branch css
nothing to commit, working tree clean
```

Faydalı Git Komutları: git clone

git clone komutu, bir uzak depoyu kopyalayarak yerel bir kopyasını oluşturmanıza yardımcı olan bir Git komutudur. Bu komut, ornek-repo adlı uzak depoyu GitHub'dan kopyalayarak, mevcut dizinde bir ornek-repo dizini oluşturur ve bu depoyu bu dizine klonlar.



```
git clone https://github.com/kullanici_adi/ornek-repo.git
```

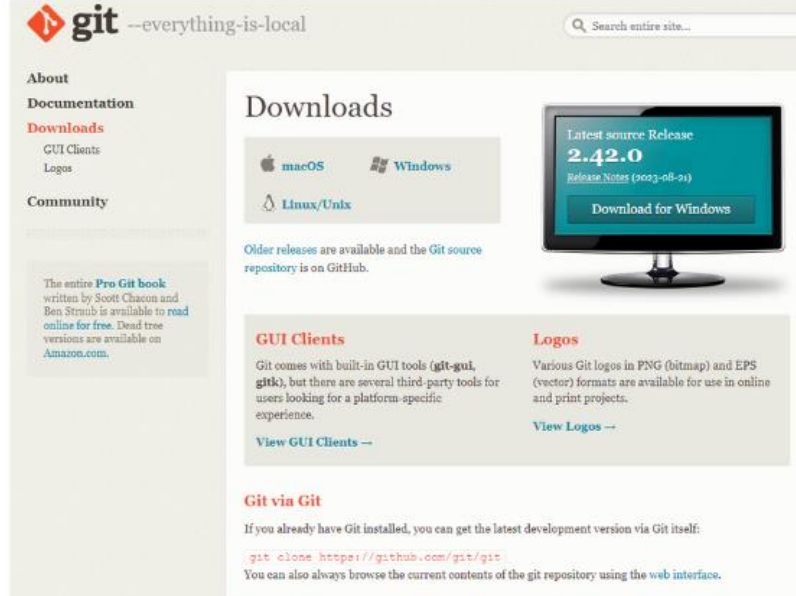
Faydalı Git Komutları: Projeyi Github'a atalım!

1 GitHub Hesabı Oluşturun



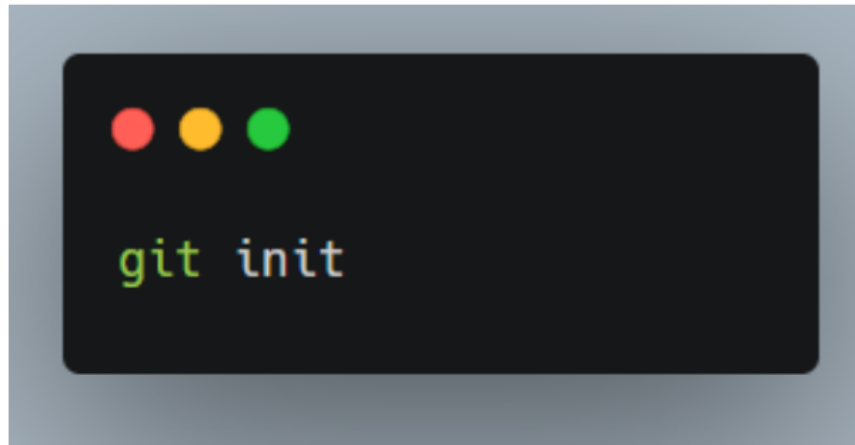
2

Git'i Kurun



3

Projenizi Git ile İzlemeye Alın



4

Commit Yapın

```
git commit -m "Proje başlatıldı"
```

5

GitHub Repository Oluşturun

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 *

KardellRueveyda / testproject ✓

Great repository names are short and memorable. Need inspiration? How about [fuzzy-couscous](#)?

Description (optional)

Test

☒ 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.](#)

Creating repository...

6

GitHub Deposunu Uzak Sunucu Olarak Ekleyin



```
git remote add origin <GitHub_depo_URL>
```

7

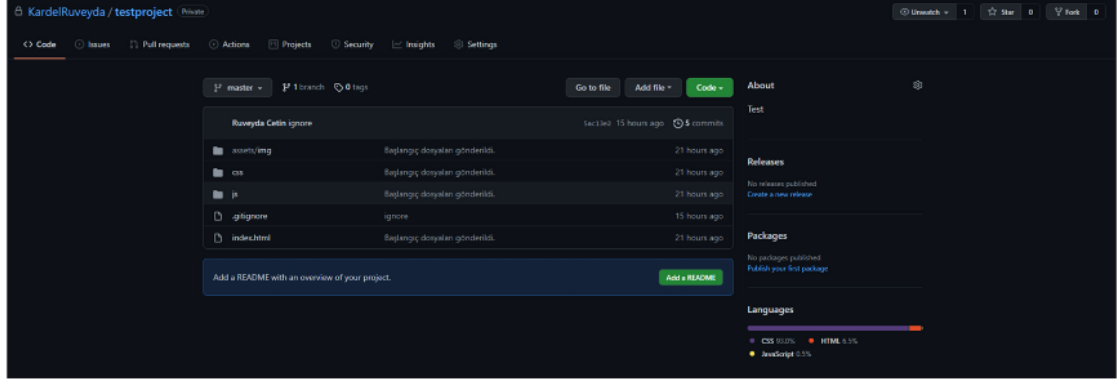
Projeyi GitHub'a Yükleyin



```
git push -u origin master
```


8

Github'a gelmiş mi kontrol et :)



Faydalı olabilecek bir Cheat-Sheet!

Git Commands Cheat Sheet

Basic

Use `git help [command]`

master default devel branch
origin default upstream branch
HEAD current branch
HEAD^ parent of HEAD
HEAD~4 great-great grandparent of HEAD
Foo..bar from branch foo to branch bar

Create

From existing files

`git init`
`git add`
From existing repository
`git clone ~/old ~/new`
`git clone git://....`
`git clone ssh://....`

Publish

`git commit [-m]`
(-m add changed files automatically)
`git format-patch origin`
(create set of diffs)
`git push remote`
(push to origin or remote)
`git tag foo`
(mark current version)

Update

`git fetch` (from def upstream)
`git fetch remote`
`git pull` (= remote & merge)
`git am -3 patch .mbox`
`git apply patch.diff`

Command Flow



View

`git status`
`git diff [oldid newid]`
`git log [-p] [file | dir]`
`git blame file`
`git show id {meta data+ diff}`
`git show id:file`
`git branch.` (show list, += current)
`git tag -l` (shows list)

Revert

`git reset -hard` (NO UNDO)
(RESET to last commit)
`git revert branch`
`git commit -a --amend`
(replaces prev commit)
`git checkout id file`

Branch

`git checkout branch`
(switch working dir to branch)
`git merge branch`
(merge into current)
`git branch branch`
(branch current)
`git checkout -b new other`
(branch new from other and switch to it)

Conflicts

`git diff [--base]`
`git diff --ours`
`git diff --theirs`
`git log --merge`
`gitk --merge`

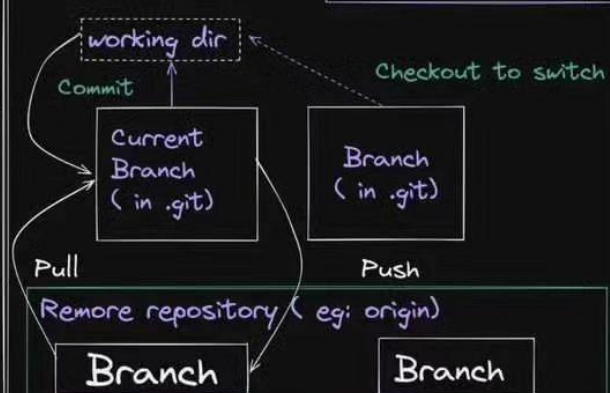
Useful Tools

`git archive`
Create release tarball
`git bisect`
Binary search for defects
`git cherry-pick`
Take single commit from elsewhere
`git fsck`
Check tree
`git gc`
Compress metadata (performance)
`git rebase`
(Forward-port local changes to remote branch)
`git remote add URL`
(Register a new remote repository for this tree)
`git stash`
(Temporarily set aside changes)
`git tag` (there's more to it)
`gitk`
(TK GUI for GIT)

Tracking Files

`git add files`
`git mv old`
`git rm files`
`git rm --cached files`
(stop tracking but keep files in working dir)

Local Repository



Structure Overview