

راهنمای استفاده از گیت و گیت لب

فهرست مطالب

۴	۱ GIT
۵	۱,۱ نصب گیت در ویندوز
۷	۱,۲ تعیین هویت در گیت
۹	۱,۳ شروع کار با گیت
۱۴	۱,۴ بررسی وضعیت فایل ها در گیت
۱۶	۱,۵ واگرد کردن در گیت
۱۸	۱,۶ شاخه ها (Branches)
۲۱	۲ GitLab
۲۲	۲,۱ شروع کار با GitLab
۲۵	۲,۲ ساخت SSH Key
۲۷	۲,۳ دستور clone
۲۸	۲,۴ git push
۳۰	۲,۵ git pull
۳۱	۲,۶ git remote
۳۳	۲,۷ merge request

۳۸ Fork ۲,۸

۴۰ fork ۲,۹

۴۳..... خصوصی بودن یا عمومی بودن یک پروژه. ۲,۱۰

۴۳..... اعضای یک پروژه. ۲,۱۱

۴۶..... تعریف گروه ۲,۱۲

۴۷ conflict حل ۲,۱۳

۱ GIT

گیت یک نوع سیستم کنترل ورژن (VCS) است که با آن می‌توانید تغییرات اعمال شده در فایل‌ها را ساده‌تر پیگیری کنید. یک مثال از عملکرد این ابزار این است که به کمک آن می‌توان تغییرات اعمال شده بر روی فایل‌ها را ثبت نمود تا همواره اطلاعات مربوط به خود تغییر، زمان اعمال آن و کاربری که آن تغییر را ایجاد کرده است را بازیابی نمود. این اعمال بخصوص برای هماهنگ کردن وظایف میان اشخاص مختلفی که روی یک پروژه کار می‌کنند، مفید است. همچنین می‌توان با ذخیره‌ی “Checkpoint” پیشرفتِ پروژه را در طی زمان بررسی کرد. می‌توان از گیت برای نوشتن یک مقاله استفاده کرد، یا صرفاً تغییرات اعمال شده را، مثلاً در یک فایل هنری یا طراحی، پیگیری کنید.

در این متن اصول اولیه‌ی استفاده از این ابزار معرفی می‌شود و سپس نحوه‌ی برقراری ارتباط بین گیت با GitLab و به کارگیری آنها برای مشارکت موثرتر در پروژه‌ها توضیح داده می‌شود.

۱.۱ نصب گیت در ویندوز

پیشنهاد می شود به عنوان نرم افزار ویرایشگر متن، Notepad++ را قبل از شروع فرآیند نصب، نصب کنید. این نرم افزار سپس در مراحل نصب به عنوان ویرایشگر گیت معرفی می شود. در صورت آشنایی با Vim امکان استفاده از آن نیز وجود دارد.

مراحل نصب گیت به ترتیب زیر می باشند.

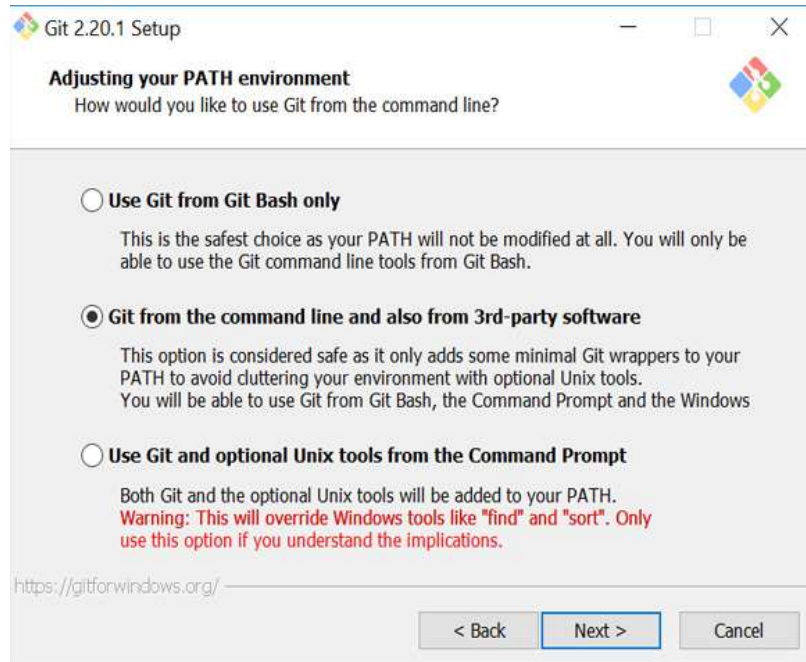
۱. در سایت <https://git-scm.com/downloads> روی مدل مربوط به ویندوز کلیک کنید تا فرآیند دانلود شروع شود.

۲. پس از اتمام دانلود فایل نصب، روی آن کلیک کرده تا فرآیند نصب آغاز شود.

۳. در پنجره ی Select Components تمام گزینه های تیک خورده را به حالت خود بگذارید و باقی گزینه ها را نیز انتخاب کنید.

۴. سپس، در Choosing the default editor، در صورتی که Notepad++ را نصب کرده اید، آن را به عنوان ویرایشگر/ editor خود انتخاب کنید.

۵. در پنجره ی Adjusting your PATH environment، پیشنهاد می شود که حالت پیش فرض Git from the command line and also from 3rd-party software را حفظ کنید. این گزینه امکان استفاده از گیت از طریق هر کدام از ویرایشگرهای Git Bash یا Windows Command Prompt را فراهم می کند.



۶. برای اتصال HTTPS حالت از همان حالت پیش فرض کتابخانه OpenSSL استفاده کنید.

۷. در پنجره‌ی Checkout Windows-configuring the line ending conversions, گزینه‌ی Windows-Checkout

style, commit Unix-style line endings انتخاب شود.

۸. در پنجره‌ی Configuring the terminal emulator to use with Git Bash, گزینه‌ی

Use MinTTY (the default terminal of MSYS2) انتخاب شود.

۹. در باقی مراحل و پنجره‌ها نیز گزینه‌های پیش فرض انتخاب شوند.

۱۰. روی آیکن install کلیک شود.

بعد از تکمیل فرآیند نصب می‌توان در منوی استارت ویندوز، از طریق یکی از گزینه‌های Git

Bash یا Git GUI، کار با گیت را آغاز نمود. همچنین می‌توان از Windows command line

نیز برای این منظور استفاده کرد (توصیه می‌شود از GUI استفاده نشود).

در صورت عدم آشنایی با دستورات قابل استفاده در Git Bash می‌توان از فایل آورده شده در

پیوست که دستورات پر کاربرد برای استفاده از این فضا را معرفی می‌کند، کمک گرفت. لازم به ذکر

است که با تایپ دستور git در فضای Git Bash نیز می توان خلاصه ای از دستورات پرکاربرد در گیت را مشاهده نمود.

در صورت عدم آشنایی با دستورات قابل استفاده در windows command line می توان از راهنمای آورده شده در صفحه های:

۱- <https://www.computerhope.com/issues/chusedos.htm>

۲- <https://www.computerhope.com/issues/chshell.htm>

استفاده نمود.

کلیه ی اطلاعات کاربردی برای کسب دانش گسترده تری از نحوه ی استفاده از گیت را همچنین می توان در سایت <https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup> یافت. در این سایت، با استفاده از انتخاب موضوع مورد بررسی در chapters می توان به اطلاعات کافی دست پیدا نمود. همچنین ترجمه ی فارسی کل متن نیز در این صفحه موجود است.

۱.۲ تعیین هویت در گیت

بعد از اتمام فرآیند نصب، اولین قدم تنظیم نام کاربری و آدرس ایمیل خود است. این اصل مهمی است چرا که گیت در هر commit از این اطلاعات استفاده می کند و به صورت غیرقابل تغییر درون commit هایی که ساخته می شوند، حک می شود:

```
$ git config --global user.name "John Doe"
```

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

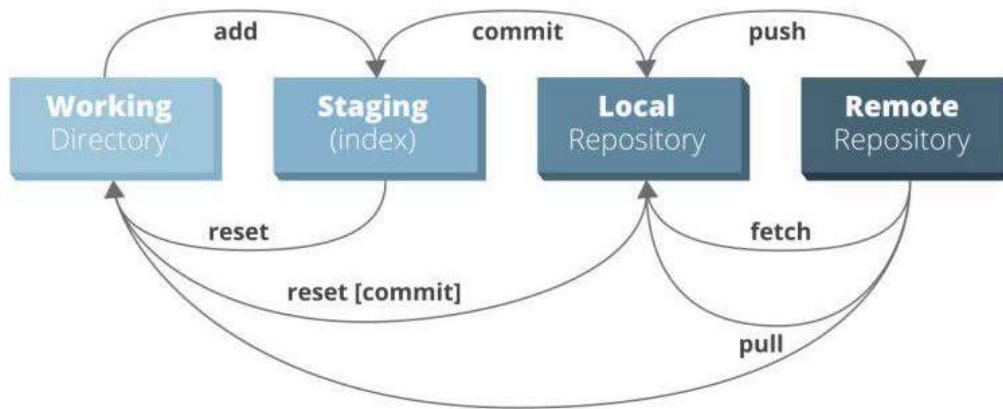
فقط لازم است که یکبار این کار را انجام شود (تنها در حالتی که آپشن --global را به دستور بدهید)، چرا که گیت همیشه از این اطلاعات برای هر کاری در آن سیستم استفاده خواهد کرد. اگر می خواهید این را با یک

نام یا ایمیل متفاوت برای پروژه‌ای خاص بازنویسی کنید، مادامی که در آن پروژه هستید می‌توانید بدون `--global` آنرا اجرا کنید. با اجرای دستور `git config --global --list` نیز می‌توانید لیستی از اطلاعات خود را در گیت فراخوانی کنید.

۱.۳ شروع کار با گیت

مدیریت فایل‌ها در گیت بر پایه‌ی چهار مفهوم `remote`، `local repository`، `staging`، `directory` و

`repository` امکان پذیر است. ارتباط این مفاهیم به به طور خلاصه و مفید در شکل زیر آورده شده است.



Working directory: فضای شخصی است که وقتی به گیت معرفی شود، می توان فایل های درون آن را

با استفاده از گیت مدیریت نمود. به این فضا معمولا دایرکتوری هم گفته می شود. معرفی یک دایرکتوری به گیت

به وسیله ی دستور `git init` امکان پذیر می شود. این دستور، از ابتدایی ترین دستوراتی است که برای شروع کار

با گیت مورد نیاز قرار می گیرد. برای مثال اگر بخواهیم دایرکتوری ای با نام `project` در یک مسیر دلخواه، مثلا

در `c:/users`، ایجاد کنیم، ابتدا باید `command window` یا `Git Bash` را باز کرده و وارد مسیر این فولدر

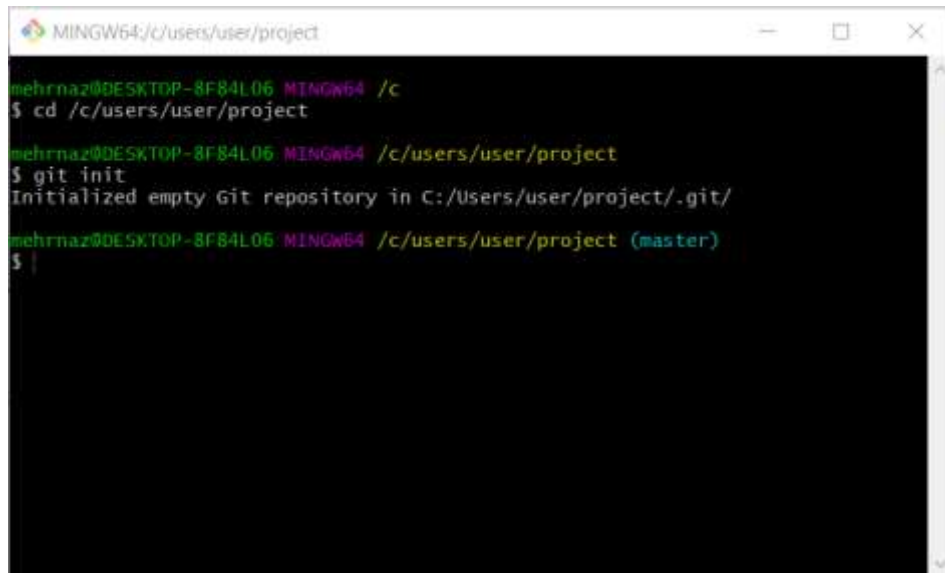
شویم. این کار با استفاده از دستور زیر امکان پذیر است:

```
cd /c/users/project
```

پس به این ترتیب، میانجی مورد استفاده، یعنی `git bash` یا `command window`، به این دایرکتوری

هدایت می شوند. حال برای معرفی این مسیر به گیت، باید دستور زیر را در آن اجرا کنیم:

```
git init
```



```
mingw64/c:/users/user/project
mehrnaz@DESKTOP-8F84L06 MINGW64 /c
$ cd /c:/users/user/project

mehrnaz@DESKTOP-8F84L06 MINGW64 /c:/users/user/project
$ git init
Initialized empty Git repository in C:/Users/user/project/.git/

mehrnaz@DESKTOP-8F84L06 MINGW64 /c:/users/user/project (master)
$
```

در این حالت گیت یک دایرکتوری به اسم **git**. تعریف می کند که اطلاعات اولیه ی لازم برای کنترل فولدر **project** در آن قرار می گیرند. از این به بعد گیت مسیر یا دایرکتوری معرفی شده را به عنوان شاخه ی **master** شناسایی کرده و می توان کار با گیت را بر روی آن ادامه داد. لازم به ذکر است که نام این شاخه قابل تغییر است که جلوتر و در بخش شاخه/**branches** به آن اشاره می شود و معمولاً با نام **main** هم شناخته می شود.

Staging: در دایرکتوری تعریف شده، در صورتی که یک فایل جدید اضافه شود یا تغییری روی یک فایل موجود در دایرکتوری اعمال شود، برای اینکه گیت از آن تغییر با خبر شود باید از دستور **git add filename** استفاده گردد. به این ترتیب گفته می شود که اصطلاحاً فایل موردنظر به حالت **stage** برده شده است که به این معنی است که گیت آماده ی ثبت آن است.

برای مثال اگر در فولدر **project** که پیشتر تولید شد یک فایل مانند **example1.txt** تعریف شود، گیت با اینکه این فایل در دایرکتوری تحت کنترل آن تولید شده است، هنوز آن را نمی شناسد. پس با دستور زیر را در **git bash** تایپ می کنیم:

```
git add example1.txt
```

```
MINGW64/c/users/user/project
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/project (master)
$ git add example1.txt
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/project (master)
$ |
```

این دستور یک آگاهی اولیه از حضور example1.txt را به گیت می دهد، هرچند هنوز گیت حضور فایل را به طور کامل به رسمیت نشناخته است.

در این حالت با استفاده از دستور git status می توان وضعیت فایل های موجود در دایرکتوری را مشاهده کرد که در این جا نشان می دهد که فایل example1.txt به عنوان فایل جدید شناسایی شده است و گیت منتظر با نمایش عبارت "Changes to be committed:" نشان می دهد که منتظر تایید نهایی یعنی اجرای دستور commit برای ثبت آن است. به شکل زیر توجه شود:

```
MINGW64/c/users/user/project
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/project (master)
$ git add example1.txt
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/project (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   example1.txt

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/project (master)
$ |
```

commit یکی دیگر از مفاهیم پایه در گیت است که به کمک آن، فایل هایی که در حالت **staging** باشند، یعنی پیشتر با استفاده از دستور **git add** به دایرکتوری شناخته شده باشند، به طور کامل در دایرکتوری ثبت می شوند. دستور **commit** به صورت زیر قابل اجرا است:

```
git commit -m 'your message'
```

بنابراین فایل از حالت **stage** خارج می شود و تغییرات در دایرکتوری ثبت می شوند. در قسمت **your message** نیز کاربر می تواند اطلاعات موردنظر درباره ی **commit** انجام شده را وارد نماید. برای **commit** کردن فایل **example1.txt** می توان از دستور زیر، به طور مثال، استفاده کرد:

```
git commit -m 'added example1.txt'
```

نوشتن یک پیام به این صورت به آسان شدن بازبینی های بعدی فایل ها کمک می کند. مخصوصا در صورتی که کاربران متعددی در حال تولید فایل ها و گسترش آنها در یک پروژه مشترک باشند، تمامی اطلاعات اضافه شده به این صورت یک تاریخچه ی کامل از گام های برداشته شده در هر مرحله از توسعه ی پروژه را به دست می دهند و به این ترتیب به **debug** کردن اشکالات در هر مرحله یا حل اختلالات ایجاد شده کمک می کند.

هر **commit** اطلاعات ایمیل و نام کاربری فردی که آن را اجرا کرده باشد را در خود دارد و این اطلاعات و همینطور زمان و تاریخ اعمال آن به همراه پیام نوشته شده برای آن **commit** را می توان با استفاده از دستور **git log** فراخوانی کرد.

در صورتی که کاربر پیامی برای **commit** وارد نکند و صرفا از دستور **git commit** استفاده کند، گیت ویرایشگری که قبلا به آن معرفی شده، مانند **notepad++**، را باز می کند و از کاربر می خواهد که پیام خود را در آن وارد کند.

تا اینجا با اصول اولیه و اساسی کار با گیت که شامل سه عمل `git init`، `git add` و `git commit` آشنا شدیم. این دستورات در تمامی میانجی های قایل استفاده برای گیت مانند `git bash`، `command window`، `Vim` و ترمینال در `Linux` و `MacOs` مشابه هستند و تنها نحوه ی دسترسی به فایل ها در هر کدام از این میانجی ها و یا دستورات کمکی ممکن است متفاوت باشند. استفاده از این دستورات به تولید یک مخزن و یا `repository` در گیت منجر می شوند که فضای پایه برای تعریف پروژه ها، توسعه ی آنها، انتقال و یا اشتراک آنها را فراهم می کند.

لازم به ذکر است که کمک گرفتن مداوم از دستور `git status` به کاربر این امکان را می دهد که وضعیت فایل ها را هر بار و بعد از هر `commit` یا `add` بررسی کند و از خطا یا گرفتن `error` جلوگیری شود.

دو نکته ی مهم درباره امکاناتی که گیت برای کار روی پروژه ها فراهم می کند، یکی امکان کار موازی روی یک پروژه به وسیله ی تعریف شاخه های مجزا و ادغام آن ها با شاخه ی اصلی پروژه است و دیگری قابلیت آن برای کار روی `remote repository` است. `Remote repository` یک نمونه از پروژه است که میزبان آن اینترنت یا شبکه ای غیر از سیستم هر فرد است. به این ترتیب می توان پروژه را با باقی کاربران اختصاصی آن پروژه یا کلیه ی کاربران یک سایت (مانند `GitHub` و `GitLab`) به اشتراک گذاشت. در ادامه ابتدا تعریف شاخه در گیت و نحوه ی ادغام آن با شاخه ی اصلی (`master/main`) توضیح داده می شود و سپس `GitLab` معرفی می شود.

۱.۴ بررسی وضعیت فایل ها در گیت

همانطور که پیشتر گفته شد، بررسی وضعیت فایل ها و مخزن تولید شده در گیت برای جلوگیری از خطا و یا حل اختلال ها بسیار مهم است. دو دستور پایه برای این کار `git status` و `git log` می باشند:

۱- `git status`

در هرکدام از مراحل معرفی شده در بالا، با استفاده از دستور `git status` می توان وضعیت دایرکتوری را بررسی نمود. این دستور به ما می گوید که چه فایل هایی در وضعیت `stage` هستند و چه فایل هایی هنوز `unstage` هستند. به این ترتیب می توان فایل ها را به گیت `add` کرد یا `commit` نمود.

۲- `git log`

با استفاده از دستور `git log` می توان تمامی اطلاعات مربوط به `commit` های اعمال شده در دایرکتوری را فراخوانی نمود. در این حالت، هر `commit` با کد اختصاصی یا `hash` مربوط به خود نمایش داده می شود و نام اپراتور، تاریخ `commit` شدن و پیام اختصاص داده شده به آن `commit` نیز در خط های بعدی آن آورده می شوند. در شکل زیر:

۱- کد نوشته شده به رنگ زرد `hash` مربوط به `commit` ها است.

۲- `HEAD` نوشته شده به رنگ فیروزه ای، در واقع نام دیگری برای آخرین `commit` اعمال شده است و

فلش کنار آن نشان می دهد که `commit` روی شاخه ی اصلی یعنی `main/master` بوده است.

```

$ git log
commit f167beb78dbe976a3eb838cc074be7e4b336cd7c (HEAD -> main, origin/main, origin/HEAD)
Author: Meh-hn <mehrnazhn@yahoo.com>
Date:   Sun Jul 25 20:06:33 2021 +0430

    deleted the html file

commit f75a49f977738267c8de43e828017713b3923309
Author: Meh-hn <mehrnazhn@yahoo.com>
Date:   Sun Jul 25 20:05:23 2021 +0430

    addin an html file

commit 312966b55b7eb12fd2a5063c1f48458af1b51120
Merge: d8b1871 b01d398
Author: Meh-hn <mehrnazhn@yahoo.com>
Date:   Sun Jul 25 19:54:51 2021 +0430

    Merge branch 'main' of github.com:Meh-hn/demo-repo

```

با کپی کردن hash مربوط به هر commit و تایپ آن در مقابل دستور git show می توان اطلاعات مربوط به آن commit خاص را مشاهده کرد. این اطلاعات نشان می دهد که دقیقا چه تغییری در آن commit انجام شدند، برای مثال چه خط هایی به کد اضافه شده اند یا چه دستوراتی از آن پاک شده اند. بنابراین برای debug کردن یک کد این اطلاعات می توانند بسیار مفید باشند.

۳- git diff

با استفاده از این دستور می توان دو ورژن از یک فایل را با هم مقایسه نمود. برای مثال: دستور git diff HEAD آخرین commit انجام شده را با وضعیت فعلی یک فایل که تغییری در آن ایجاد شده باشد، modified file، مقایسه می کند. همانطور که قبلا گفته شد، منظور از HEAD در گیت آخرین commit اعمال شده در آن repository است.

دستور git diff –staged آخرین وضعیت stage را با فایل staged فعلی مقایسه می کند.

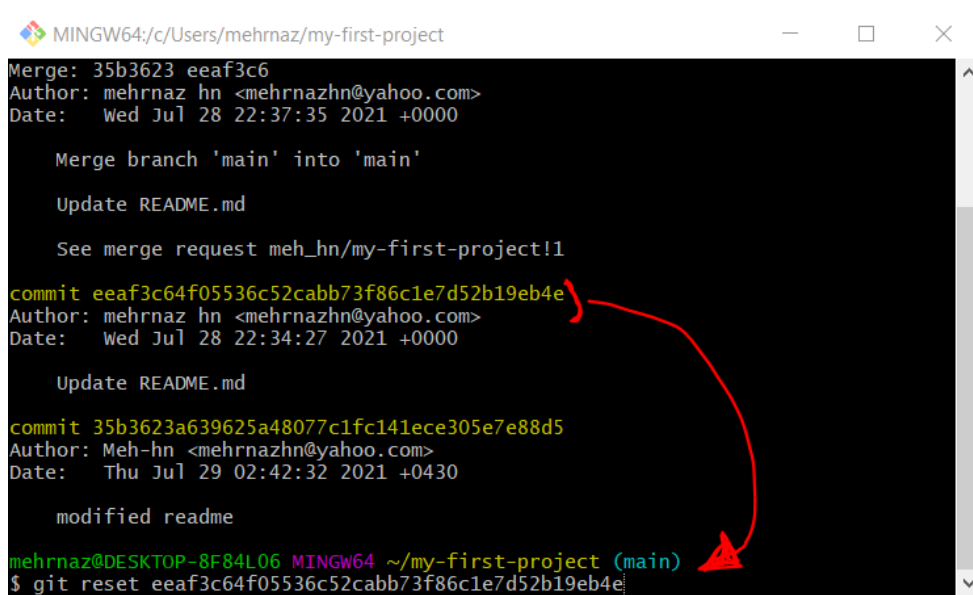
۱.۵ واگرد کردن در گیت

وقتی بخواهیم درباره ی ثبت تغییرات انجام شده روی یک فایل تجدید نظر کنیم یا به طور کلی یکسری از تغییرات را حذف کنیم، می توانیم از موارد زیر استفاده کنیم:

git reset-۱

برای بیرون آوردن یک فایل از حالت stage می توان از دستور `git reset filename` یا `git reset` استفاده نمود. که با این کار فایل به صورت `unstaged` شناخته می شود. این کمک می کند که اگر از `commit` کردن یک تغییر منصرف شدیم یا قبل از `commit` باید تغییرات دیگری اعمال کنیم، بتوانیم یک قدم به عقب برگردیم یا به عبارتی تغییر را `undo` کنیم.

برای برگرداندن یک `commit` هم می توان از دستور `git reset HEAD~1` استفاده کرد که گیت را به `commit` قبلی بر می گرداند. برای برگشتن به یک `commit` خاص نیز می توان از کپی کردن `hash` آن و سپس دستور `git reset Hashcode` استفاده کرد. به این ترتیب تغییراتی که بعد از این `commit` خاص ایجاد شده بودند به حالت `unstaged` می روند، هرچند که حذف نمی شوند.



```
MINGW64/c:/Users/mehrnaz/my-first-project
Merge: 35b3623 eeaf3c6
Author: mehnaz hn <mehrnazhn@yahoo.com>
Date: Wed Jul 28 22:37:35 2021 +0000

Merge branch 'main' into 'main'

Update README.md

See merge request meh_hn/my-first-project!1

commit eeaf3c64f05536c52cabb73f86c1e7d52b19eb4e
Author: mehnaz hn <mehrnazhn@yahoo.com>
Date: Wed Jul 28 22:34:27 2021 +0000

Update README.md

commit 35b3623a639625a48077c1fc141ece305e7e88d5
Author: Meh-hn <mehrnazhn@yahoo.com>
Date: Thu Jul 29 02:42:32 2021 +0430

modified readme

mehrnaz@DESKTOP-8F84L06 MINGW64 ~/my-first-project (main)
$ git reset eeaf3c64f05536c52cabb73f86c1e7d52b19eb4e
```


برای حذف کامل تغییرات بعد از یک commit خاص نیز می توان از `git reset --hard HashCode` استفاده کرد. بنابراین، تمامی تغییرات بعد از آن commit حذف می شوند.

git checkout -- filename -2

اگر فایلی خراب شد و لازم باشد به حالت قبلی برگردانده شود، باید از این دستور استفاده کنیم. این دستور در واقع آخرین تغییرات اعمال شده روی فایل `filename` را از بین می برد و فایل را به وضعیتش در آخرین commit آن بر می گرداند. ولی ابتدا باید با یکی از روش های توضیح داده شده در بالا فایل به حالت `unstaged` دربیاید و سپس از این دستور استفاده شود.

git rm -۳

با استفاده از دستور `git rm filename` می توان فایل `filename` را از دایرکتوری گیت و کل سیستم حذف

نمود.

۱.۶ شاخه ها (Branches)

اگر مراحل مختلف یک پروژه را به صورت سلسله مراتبی و پشت سر هم تصور کنیم، در صورتی که بخواهیم یک مرحله مجزا را به بخشی از مسیر پروژه اضافه کنیم یا از کاربران مختلف برای همکاری در آن استفاده کنیم، باید از مفهوم شاخه ها استفاده کنیم. با استفاده از مفهوم شاخه ها در گیت می توان در مسیر خطی پروژه، که معمولا با نام `master` یا شاخه ی اصلی شناخته می شود، یک شاخه ی مجزا تعریف نمود و به صورت موازی با مراحل اصلی پروژه آن را به انجام رساند و در نهایت نتایج آن را با یکی از مراحل اصلی ادغام یا `merge` کرد. برای هر کدام از این شاخه های اضافه شده می توان نام های اختصاصی تعریف نمود. روش تعریف شاخه ها در داخل گیت به ترتیب زیر می باشد:

۱- ابتدا با استفاده از دستور `git branch` چک می شود که روی کدام شاخه قرار داریم. برای مثال اگر گیت در آن لحظه روی شاخه ی اصلی `master/main` باشد، گیت نام شاخه ی اصلی را به صورت `master *` یا `*` `main` نشان می دهد. با استفاده از همین دستور و فراخوانی آن به شکل زیر می توان نام شاخه ی اصلی را تغییر داد. مثلا از نام `master` به `main`:

`git branch -M main`

نکته: این دستور در حسن کار با `GitLab` و `GitHub` بسیار کاربردی است چون معمولا شاخه ی اصلی در این فضاها با نام `main` تعریف می شود.

۲- سپس با استفاده از همان دستور ولی این بار به صورت `git branch new-branch` می توان یک شاخه با نام `new-branch` (یا هر نام دلخواه دیگری) به شاخه ی `main` اضافه نمود. (حالا اگر دوباره دستور `git branch` را تایپ کنیم، شاخه هایی که تا حالا تولید شده اند، یعنی `main` و `new-branch` را نشان می دهد و همینطور مشخص می کند که در حال حاضر روی کدام شاخه در حال اعمال تغییر هستیم.)

جایجا شدن بین این شاخه ها با استفاده از دستور `git checkout branchName` امکان پذیر است. برای مثال با استفاده از دستور زیر می توان از شاخه ی `main` به شاخه ی `new-branch` جایجا شد:

`git checkout new-branch`

بعد از اعمال این دستور، با تایپ نمودن `git branch` مشاهده می شود که شاخه ی `new-branch` فعال است و از اینجا به بعد تمام دستورها روی این شاخه اعمال می شوند بدون اینکه تغییری روی شاخه ی اصلی یعنی `main` اعمال شود. به عبارت دیگر، در حین کار روی هر شاخه، فقط دایرکتوری فایل های مربوط به آن شاخه را به ما نشان می دهد و تا زمانی که عمل ادغام یا `merge` انجام نشود، نمی توان هیچ یک از تغییرات اعمالش ده یا فایل های اضافه شده در شاخه ی `new-branch` را در شاخه ی اصلی دید. این نکته برای اطلاعات شاخه ی اصلی نیز درست است.

نکته ی دیگر این است که تمامی امکاناتی که قبلا روی شاخه ی اصلی قابل استفاده بودند مانند `commit`، `add`، `log` و `status` و باقی دستورات قابل استفاده در گیت، روی این شاخه نیز به طور مجزا قابل استفاده هستند.

برای `merge` کردن شاخه ها نیز باید به ترتیب زیر عمل شود:

- ۱- ابتدا باید با استفاده از دستور `git checkout main` بازگشت به شاخه ی اصلی یا `main` صورت گیرد.
- ۲- بعد از وارد شدن به شاخه ی `main` با تایپ دستور `git merge new-branch` می توان شاخه ی `new-branch` را با شاخه ی `main` ادغام نمود تا به این ترتیب تمام تغییرات اعمال شده در شاخه ی `new-branch` به شاخه ی اصلی انتقال داده شود، فایل ها اضافه شوند، یا تغییرات روی فایل های موجود در شاخه ی اصلی به تک تک فایل ها اعمال شود.

نکته ۱: بهتر است که در حین کار با شاخه ها جوری `commit` ها را اعمال کنیم که برای هر عمل فقط یک `commit` داشته باشیم. این کار به منظم کردن تاریخچه کمک می کند و `undo` کردن را در صورتی که بدانیم اشکالی در یک مرحله وجود داشته را تسهیل می نماید.

نکته ۲: باید مواظب باشیم تا حد امکان فایل هایی که قبل از تولید شاخه های فرعی روی شاخه ی اصلی موجود بودند، تغییر ندهیم. در غیر این صورت ممکن است برای عمل ادغام با خطا رو به رو شویم به خصوص وقتی که چند نفر همزمان روی یک پروژه کار می کنند ممکن است چنین عملی به ایجاد `conflict` یا درگیری بیانجامد که جلوتر درباره ی آن توضیح داده می شود.

بعد از اتمام کار با `new-branch` می توان آن را با استفاده از دستور `git branch -d new-branch` حذف کرد. این کار به مرتب شدن محیط گیت کمک می کند و در صورتی که بیش از یک نفر در حال کار روی یک پروژه باشند، احتمال خطا کاهش می یابد. هر چند در بعضی از پروژه ها ممکن است شاخه هایی با اهمیت برابر با شاخه ی اصلی تعریف شوند که در این موارد معمولاً شاخه در طول کار روی پروژه حفظ می شود.

۲ GitLab

گسترش فضای قابل استفاده در گیت از دایرکتوری سیستم در حال استفاده به یک فضای جداگانه و یا شبکه، در واقع همان ویژگی distributed بودن گیت است. به این ترتیب که برای مثال، کاربر می تواند یک کد یا فایل را از حافظه در سیستم خود (فضای local) در شبکه بارگذاری (push) کند یا از شبکه استخراج کرده و به سیستم خود وارد (pull) نماید. GitHub و GitLab و یا یک گیت اختصاصی در یک سازمان مواردی هستند که می توان در آنها از ویژگی distributed بودن گیت بهره گرفت.

در سایت گیت لب افراد می توانند پروژه های گوناگون را در حساب کاربری خود یا حساب کاربری شرکت یا سازمانی که با آن کار می کنند، بارگذاری کنند. به این ترتیب همه ی کاربران گیت لب می توانند از این پروژه ها استفاده کنند. پروژه ها در گیت لب در هر حساب کاربری در project/repository مخصوص به خود ذخیره می شوند و دسترسی به آنها برای عموم آزاد است به جز در مواردی که حساب کاربری به صورت خصوصی (private) تعریف شود یا یک پروژه یا گروه به صورت خصوصی تعریف شوند. در این موارد دسترسی به آن پروژه ی خاص تنها به وسیله ی دریافت اجازه از حساب اصلی پروژه فراهم می شود.

در گیت لب مفاهیمی که در گیت معرفی شدند، مانند commit، branch، master یا log نیز قابل استفاده هستند. علاوه بر این مفاهیمی مانند clone، push، pull و remote repository نیز در این فضا قابل استفاده هستند. این مفاهیم هم امکاناتی را برای اعمال تغییر روی پروژه ها در داخل گیت لب و در repository مربوطه را فراهم می کنند و هم امکان کار روی پروژه ها توسط کاربران مختلف در سیستم های مختلف را فراهم می کنند. در صفحه ی زیر می توانید فیلم آموزشی مفیدی برای سهولت در فرآیند یادگیری را مشاهده کنید:

https://www.youtube.com/watch?v=Jt4Z1vwtXT0&list=PLhW3qG5bs-L8YSnCiyQ-jD8XfHC2W1NL_&index=1

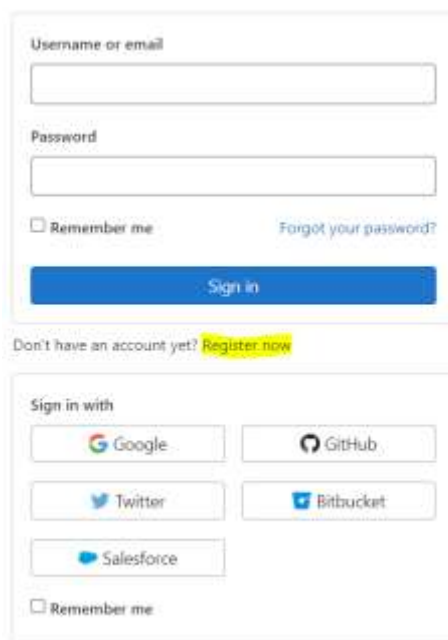
در ادامه نیز مراحل مختلف استفاده از این وب سایت برای بهبود مشارکت در پروژه ها آموزش اینکه چگونه می توان به یک پروژه دسترسی پیدا کرد، توضیح داده می شود.

۲.۱ شروع کار با GitLab

برای استفاده از گیت لب ابتدا لازم است در سایت ثبت نام کنید. البته در صورتی که حساب کاربری نداشته باشید نیز باز امکان گشتن در میان پروژه های عمومی و دانلود کردن آنها یا مطالعه ی پروژه ها وجود دارد. برای شروع کار با گیت لب باید مراحل زیر را انجام دهید:

۱- به gitlab.com وارد شوید.

۲- روی [log in](#) در گوشه ی سمت چپ کلیک کنید و سپس لینک [register now](#) را در صفحه ی باز شده انتخاب کنید.

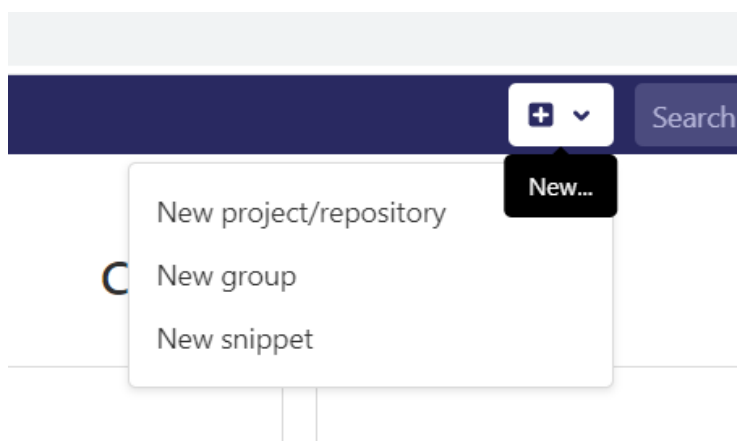


The image shows the GitLab login and registration page. It features a 'Sign in' section with fields for 'Username or email' and 'Password', a 'Remember me' checkbox, and a 'Forgot your password?' link. Below this is a 'Sign in with' section with buttons for Google, GitHub, Twitter, Bitbucket, and Salesforce. A 'Don't have an account yet? Register now' link is also present, with 'Register now' highlighted in yellow.

۳- بعد از وارد کردن نام و رمز عبور، روی گزینه ی [register](#) کلیک کنید.

وقتی فرآیند ثبت نام تکمیل شد یک ایمیل تاییدیه از گیت لب به شما ارسال می شود که با استفاده از آن می توانید به حساب کاربری خود وارد شوید. بعد از این هربار برای شدن دوباره به سایت گیت لب مراجعه شود و از گزینه ی **log in** استفاده کنید. (معمولا برای وارد شدن به این سایت نیز به فیلترشکن احتیاج است).

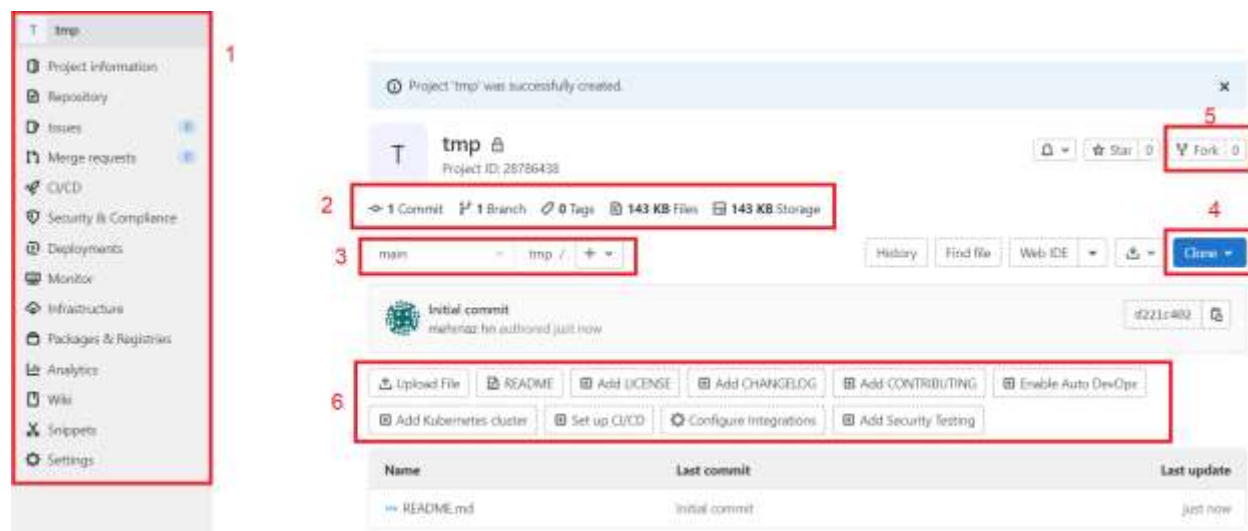
با وارد شدن به حساب کاربری خود در **GitLab** می توانید با استفاده از گزینه ی **new project** یک پروژه ی جدید تولید کنید.



با انتخاب این گزینه و سپس انتخاب **blank project**، صفحه ی زیر باز می شود که در آن باید نام و ویژگی های پروژه ی موردنظر خود را وارد کنید و گزینه **create project** را وارد کنید.

A screenshot of the 'Create blank project' form in GitLab. The form is titled 'New project' and 'Create blank project'. It contains several input fields and checkboxes. The 'Project name' field has the value 'tmp'. The 'Project URL' field has the value 'https://gitlab.com/'. The 'Project slug' field has the value 'tmp'. There is a checkbox for 'Initialize repository with a README' which is checked. At the bottom, there are two buttons: 'Create project' and 'Cancel'.

پیشنهاد می شود که برای یادگیری بهتر شما نیز با این فایل آموزش گام های فوق را تکرار کنید و یک پروژه با نام دلخواه تعریف کنید. در اینجا پروژه ای که برای مثال آورده شده است tmp نامگذاری شده است. به این ترتیب پروژه ی جدید تولید می شود و همانطور که در شکل زیر دیده می شود. در این تصویر امکاناتی که در گیت لب قرار داده شده است با شماره مشخص شده اند.



با توجه به این شکل، خلاصه ای از عملکرد بخش های مختلفی که به کار روی یک پروژه در گیت لب کمک می کنند، در اینجا لیست شده است:

۱- در سمت چپ امکاناتی مانند project information, repository, issues, merge request و repository قرار داده شده اند. با انتخاب هر کدام از این گزینه ها می توان لیستی از اطلاعات یا ویژگی های قابل استفاده در گیت لب را فراخوانی کرد. همچنین در قسمت setting می توان اطلاعات اختصاصی مربوط به پروژه مانند نام آن را تغییر داد یا پروژه را حذف نمود.

۲- در این قسمت نواری از اطلاعات قرار دارد که تعداد commitها، شاخه ها، و حجم فایل ها را نشان می دهند.

۳- در بخش ۳ می توان شاخه ی تحت کنترل را تعویض کرد، که چون در این پروژه تا اینجا هیچ شاخه ی فرعی تولید نشده است، این فضا فقط نام شاخه ی main را نشان می دهد.

۴- Clone یکی از امکاناتی است که در گیت لب و یا گیت هاب تعریف شده است تا بتوان به کمک گیت، پروژه ها را در سیستم شخصی دانلود نمود.

۵- Fork یکی از امکانات کاربردی برای توسعه ی یک پروژه است. عملکرد آن مانند شاخه ها می باشد با این تفاوت که کل پروژه را به کمک آن می توان به عنوان یک پروژه ی مستقل در یک repository مجزا کپی نمود و روی آن به طور اختصاصی کار کرد. کاربر می تواند تغییرات ایجاد شده روی پروژه ی کپی شده به وسیله ی این روش را دوباره با مسیر اصلی ادغام کند که جلوتر توضیح داده می شود.

۶- در این نوار مشاهده می شود که امکانات مختلفی برای آپلود کردن یک فایل یا تولید آن قرار داده شده است. لازم به ذکر است که در صورت دسترسی داشتن به یک پروژه می توان فایل ها را هم از طریق گیت و هم از طریق این نوار در گیت لب آپلود نمود.

قبل از شروع معرفی هر کدام از این بخش ها ابتدا لازم است برای خود یک ssh key تولید کنید تا با استفاده از آن بتوانید هویت خود را در سرورهای مختلف گیت اثبات کنید.

۲.۲ ساخت SSH Key

برای اثبات اصالت کاربر به گیت لب و یا گیت هاب باید از یک SSH key استفاده شود. این کلید به دو صورت خصوصی و عمومی وجود دارد و در فولد ssh در دایرکتوری اصلی گیت که معمولا مسیر C:\User\systemName است، با نام های پیش فرض id_rsa و id_rsa.pub یافت می شود. کلید خصوصی است که متعلق به خود کاربر است و نباید برای کارهای عمومی از آن استفاده شود. id_rsa.pub کلید عمومی است که از آن در سرورهای گیت استفاده می شود. در صورتی که این کلیدها در دایرکتوری نام برده شده یافت نشدند، کاربر باید آنها را بسازد. نحوه ی ساخت SSH key به ترتیب زیر است:

- ۱- با استفاده از دستور `cd` به دایرکتوری `ssh` وارد شوید. (در اینجا ابتدا با استفاده از دستور `ls` چک کنید که آیا فایلی با نام مشابه به `id_rsa` و `id_rsa.pub` در این مسیر وجود دارد یا خیر. اگر نبود گام های زیر را ادامه دهید و گرنه به مورد ۶ مراجعه کنید).
- ۲- ابتدا باید دستور `ssh-keygen -t rsa -b 4056 -C "owners email address"` در گیت وارد شود. در این دستور به جای عبارت `"owners email address"` باید ایمیل کاربر که در گیت لب یا گیت هاب استفاده شده است، وارد شود.
- ۳- در پاسخ به `enter file in which to save the key`، نام مورد نظر برای فایل کلید را وارد کنید. در صورت خالی گذاشتن این بخش گیت کلید ها را با نام پیشفرض `id_rsa` ذخیره می کند.
- ۴- در پاسخ به `enter passphrase` کد رمز خود را وارد کنید.
- ۵- بعد از بازنویسی کد رمز، کلیدها در فولدر `ssh` تولید می شوند.
- ۶- با استفاده از دستور `cat id_rsa.pub` متن کلید عمومی را در ترمینال فراخوانی کنید و کل متن را انتخاب کرده و کپی کنید.
- ۷- به حساب کاربری خود در گیت لب رفته و در گوشه ی بالا سمت راست صفحه روی گزینه ی `preferences` کلیک کنید. سپس گزینه ی `SSH Key` را انتخاب کرده و کد کپی شده را در فضای خالی موجود زیر گزینه ی `key` پیست کنید.
- ۸- با تایید این کد، `SSK key` فعال می شود.
- ۹- به گیت در سیستم خود بروید و دستور `$(ssh-agent -s)` را وارد کنید.
- ۱۰- در انتها نیز دستور `ssh-add ~/.ssh/id_rsa` را وارد کنید. در اینجا گیت `passphrase` یا همان رمز عبور را می خواهد که باید همان رمز عبوری که در ابتدا وارد کردید را در اینجا مجددا وارد کنید.

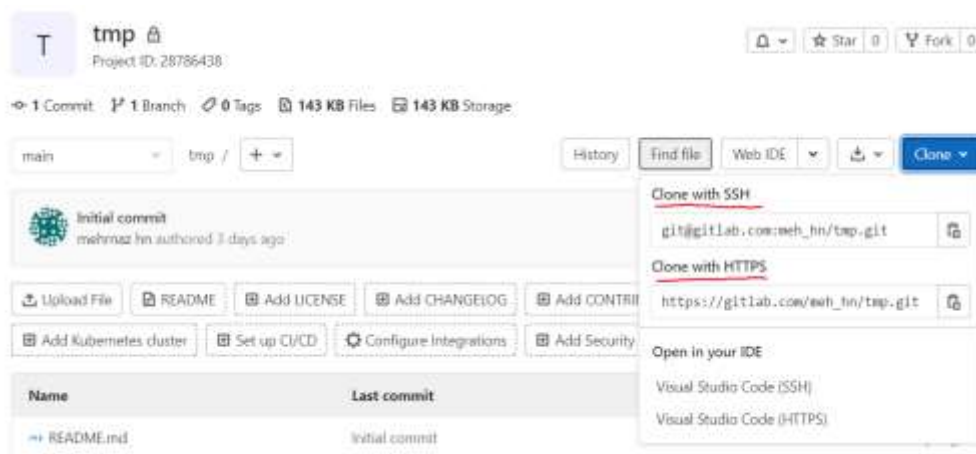
با تکمیل این مراحل شما موفق به تعریف ssh key در گیت و گیت لب می شوید. همچنین از همین کلید تولید شده برای گیت هاب نیز می توان استفاده کرد و نیازی به تعریف کلیدهای جداگانه برای هر کدام از این سایت ها نیست. این کلید به خصوص در گیت لب برای کار با remote ها اهمیت دارد.

۲.۳ دستور clone

همانطور که گفته شد clone کردن یک پروژه مانند دانلود کردن فایل های است. با استفاده از دستور git clone url می توان یک دایرکتوری جدید در گیت تولید کرد که در این دایرکتوری فایل های مربوط به پروژه ای که در GitHub در آدرس url ذخیره شده است دانلود می شود یا اصطلاحاً clone می گردد. برای این کار ابتدا لازم است با استفاده از دستور cd گیت را به مسیری که می خواهیم پروژه ی جدید در آن دانلود شود، هدایت کنیم. سپس باید دستور زیر را در گیت وارد کنیم:

git clone url

این url یک مسیر با پسوند git. است و برای کپی کردن آن کافی است که پروژه ی مورد نظرم را مثلا tmp را در گیت لب باز کنیم و روی گزینه ی clone که در تصویر زیر نمایش داده شده است کلیک کنیم. این url را می توان به صورت HTTPS یا SSH کپی کرد.



با این کار یک فولدر با نام tmp و تمامی زیرفولدرهای آن تولید می شود.

```
mehrnaz@DESKTOP-8F84L06 MINGW64 /c
$ cd /c/users/user

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user
$ git clone git@gitlab.com:meh_hn/tmp.git
Cloning into 'tmp'...
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
client_global_hostkeys_private_confirm: server gave bad signature for RSA key 0
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user
$ |
```

۲.۴ git push

با اعمال هر تغییر روی اجزای پروژه ی clone شده لازم است که ابتدا گام های مربوط به add کردن (بردن تغییرات به حالت stage) و commit کردن به ترتیب انجام شوند. سپس برای انتقال فایل های commit شده به origin در سایت، لازم است ابتدا دستور زیر اجرا شود:

git push origin main

سپس نام کاربری و رمزعبور خود را وارد کنید تا بتوانید تغییرات را اعمال کنید.

در تصویر زیر نحوه ی اضافه کردن یک فایل متلب به نام lineDetection.m و به ترتیب add و commit

کردن آن نشلن داده شده است. در این مثال با استفاده از دستور push این فایل به repository مربوط به پروژه ی tmp در گیت لب اضافه می شود.

```

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user
$ cd tmp

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ touch lineDetection.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ git add lineDetection.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   lineDetection.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ git commit -m "added matlab lineDetection script"
[main f6e0758] added matlab lineDetection script
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 lineDetection.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ git push origin main
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
client_global_hostkeys_private_confirm: server gave bad signature for RSA key 0
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 296 bytes | 148.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To gitlab.com:meh_hn/tmp.git
 d221c40..f6e0758  main -> main

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)

```

با وارد کردن نام کاربری و رمز عبور تغییرات اعمال می شود و اطلاعات مربوط به این فعالیت ها در گیت لب

در تب commits نمایش داده می شود.

The screenshot shows the GitLab web interface for a project named 'tmp' (Project ID: 28786438). The interface includes a header with project details, a sidebar with navigation options, and a main content area showing the commit history. The commit history table lists two commits: 'Initial commit' and 'added matlab lineDetection script'.

Name	Last commit	Last update
README.md	Initial commit	3 days ago
lineDetection.m	added matlab lineDetection script	3 minutes ago

۲.۵ git pull

پروژه هایی که توسط گیت لب مدیریت می شوند، لازم است مدام در گیت شخصی هر کاربر به روز رسانی (update) شوند. به عبارت دیگر هر تغییری که روی یک پروژه در گیت لب اعمال می شود، مثلاً کاربران دیگر یک فایل یا یک شاخه به پروژه اضافه می کنند یا اینکه خطوط کد در یک فایل خاص را تغییر می دهند، لازم است روی نسخه ای از پروژه که در سیستم شخصی کاربر موجود است، نیز اعمال شوند. برای این نوع به روز رسانی باید از دستور pull استفاده کنیم:

`git pull origin main`

برای مثال به پروژه ی خود در گیت لب بروید و روی فایل README.md کلیک کنید. گزینه ی edit را انتخاب کنید و یک خط به خطوط README.md اضافه کنید. با تایید را تغییر بدهید. به پایین صفحه بروید و روی گزینه ی commit changes کلیک کنید تا تغییرات در گیت لب ثبت شود. حالا با برگشتن به پروژه می توانید تغییر ایجاد شده را در فایل README.md مشاهده کنید.

به Git Bash مراجعه کنید و دستور pull را به صورت زیر اجرا کنید:

```
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ git pull origin main
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
client_global_hostkeys_private_confirm: server gave bad signature for RSA key 0
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 361 bytes | 2.00 KiB/s, done.
From gitlab.com:meh_hn/tmp
 * branch          main      -> FETCH_HEAD
   f6e0758..71d74be main      -> origin/main
Updating f6e0758..71d74be
Fast-forward
 README.md | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ |
```

حال اگر فایل README.md را باز کنید، مشاهده می کنید که خطی که در گیت لب به آن اضافه کردید

در فولدر clone شده ی شما نیز تغییر کرده است.

۲.۶ git remote

برای کار روی یک پروژه از طریق سرورهای مختلف گیت، مانند گیت لب، لازم است یک ارتباط بین گیت و آن سرور برقرار شود تا آن سرور از حضور آن پروژه آگاه شود. برای این کار لازم است ابتدا یک repository خالی با نام مشابه آن پروژه در گیت لب تعریف شود. با کپی کردن url این فضای جدید در دستور زیر می توان اتصال remote بین دو repository، یکی در گیت لب و یکی در گیت، را ایجاد نمود:

```
git remote add origin url
```

به این ترتیب می توان از دستور push برای ارسال اطلاعات به گیت لب استفاده کرد. ولی قبل از استفاده از دستور push، باید نام شاخه ی master (در صورتی که قبلا تغییر داده نشده است) را به main با استفاده از دستور زیر تغییر داد:

```
Git branch –M main
```

سپس می توان از دستور push به صورت زیر استفاده کرد:

```
git push origin main
```

برای آشنایی بیشتر با این قسمت یک فولدر جدید به نام tmp2 در سیستم خود ایجاد کنید و با استفاده از git init آن را به گیت معرفی کنید. با استفاده از دستور touch README.md یک فایل README.md در آن ایجاد کنید و از git add README.md و git commit -m “added the first file” استفاده کنید تا فایل جدید در گیت ثبت شود. سپس به گیت لب بروید و روی گزینه ی new project کلیک کنید. گزینه ی create blank project را انتخاب کنید و تیک مربوط به initialize repository with a README را بردارید. روی گزینه ی create project کلیک کنید. حال شما یک پروژه ی جدید در گیت لب خود تولید کرده اید. url این پروژه را کپی کنید.

برای استفاده از دستور remote باید مجدداً به Git Bash باز گردید و دستور `git remote add origin`

url را وارد کنید. سپس نام شاخه اصلی را به `main` تغییر بدهید و در نهایت با استفاده از دستور `push` فایل

ها را به گیت لب `push` کنید:

```
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp (main)
$ cd ../tmp2

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2
$ git init
Initialized empty Git repository in C:/Users/user/tmp2/.git/

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (master)
$ touch README.md

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (master)
$ git status
On branch master

No commits yet

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

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

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (master)
$ git add README.md

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (master)
$ git commit -m "added the first file"
[master (root-commit) 3cd2529] added the first file
 1 file changed, 2 insertions(+)
 create mode 100644 README.md

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (master)
$ git remote add origin git@gitlab.com:meh_hn/tmp2.git

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (master)
$ git branch -M main

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git push origin main
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
client_global_hostkeys_private_confirm: server gave bad signature for RSA key 0
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 265 bytes | 265.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To gitlab.com:meh_hn/tmp2.git
 * [new branch]      main -> main

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ |
```


merge request ۲.۷

merge request مربوط به زمانی است که بخواهیم یک شاخه را با شاخه ی اصلی در پروژه در گیت لب ادغام کنیم. برای ادغام شاخه در گیت لب ابتدا باید از دستور push اینبار با استفاده از نام شاخه ی فرعی، در اینجا new-branch، استفاده کنیم:

```
git push origin new-branch
```

برای آشنایی بیشتر با این قسمت، ابتدا در Git Bash و در دایرکتوری پروژه ی tmp2 دستور git branch new-branch را وارد کنید. به این ترتیب شاخه ی new-branch تولید می شود. با استفاده از git checkout new-branch به این شاخه وارد شوید و یک فایل جدید تولید کنید، مثلا new.m. این فایل را با استفاده از git add new.m و سپس "added a file to new-branch" –m git commit را اجرا کنید. حال دستور git push origin new-branch را مانند شکل زیر اجرا کنید:

```

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git branch new-branch

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git checkout new-branch
Switched to branch 'new-branch'

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ touch new.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ git status
On branch new-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        new.m

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

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ git add new.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ git commit -m "added a file to new-branch"
[new-branch f8f2968] added a file to new-branch
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 new.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ git push origin new-branch
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
Client_global_hostkeys_private_confirm: server gave bad signature for RSA key 0
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 277 bytes | 277.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for new-branch, visit:
remote: https://gitlab.com/meh_hn/tmp2/-/merge_requests/new?merge_request%5Bsource_branch%5D=new-branch
remote:
To gitlab.com:meh_hn/tmp2.git
 * [new branch]      new-branch -> new-branch

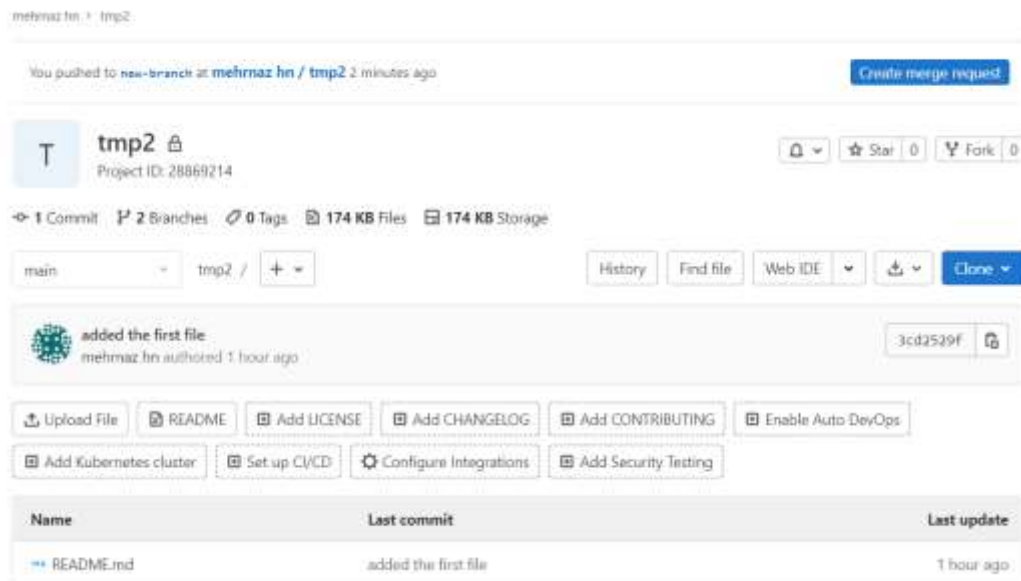
mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ |

```

در اینجا گیت از شما می خواهد که به آدرس گیت لب پروژه ی tmp2 بروید و در آنجا یک merge request تولید کنید.

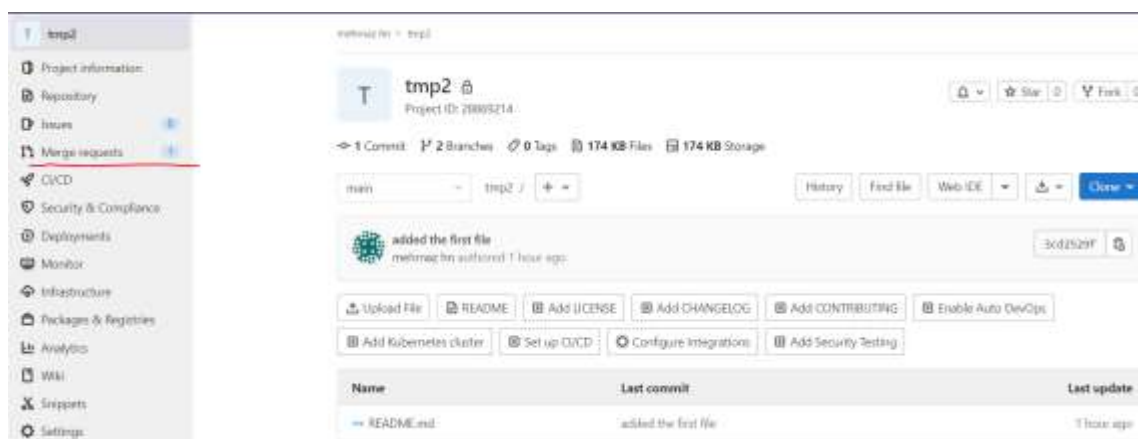
وقتی به این آدرس مراجعه شود، مشاهده می شود که هنوز فایل new.m به repository اضافه نشده

است ولی در بالای صفحه آیکنی به نام create merge request تولید شده است، مانند شکل زیر:



روی این آیکن کلیک کنید. صفحه ای که باز می شود برای شما امکاناتی مانند فضایی برای نوشتن description را فراهم کرده است. شما می توانید توضیحات لازم را درباره ی تغییراتی که در این شاخه ایجاد کردید را در این قسمت وارد کنید و در نهایت روی گزینه ی create merge request کلیک کنید. به این ترتیب در خواست ادغام به صاحب اصلی پروژه، که در اینجا خود شما هستید، ارسال می شود.

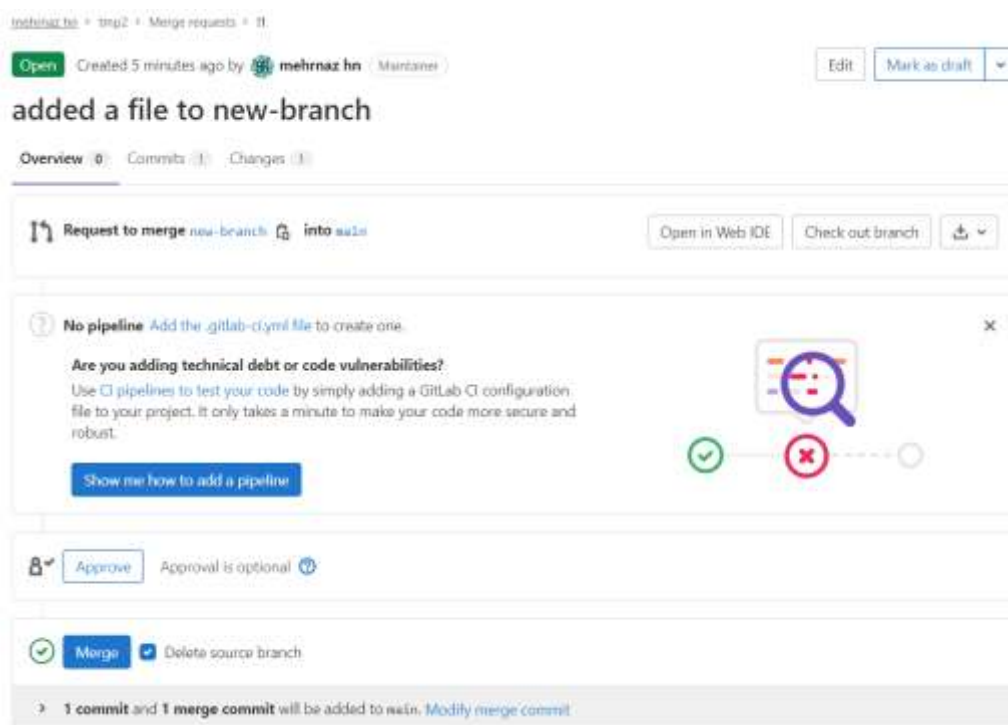
حال به صفحه ی اصلی پروژه ی tmp2 بروید و از بین گزینه های سمت چپ صفحه روی بخش merge request کلیک کنید. در تصویر زیر این گزینه با خط قرمز مشخص شده است.



عدد یک در مقابل گزینه ی merge request نشان می دهد که یک درخواست ادغام جدید برای این پروژه ایجاد شده است و با کلیک روی آن، صفحه ی زیر باز می شود که در آن لیستی از درخواست های ادغام موجود است:

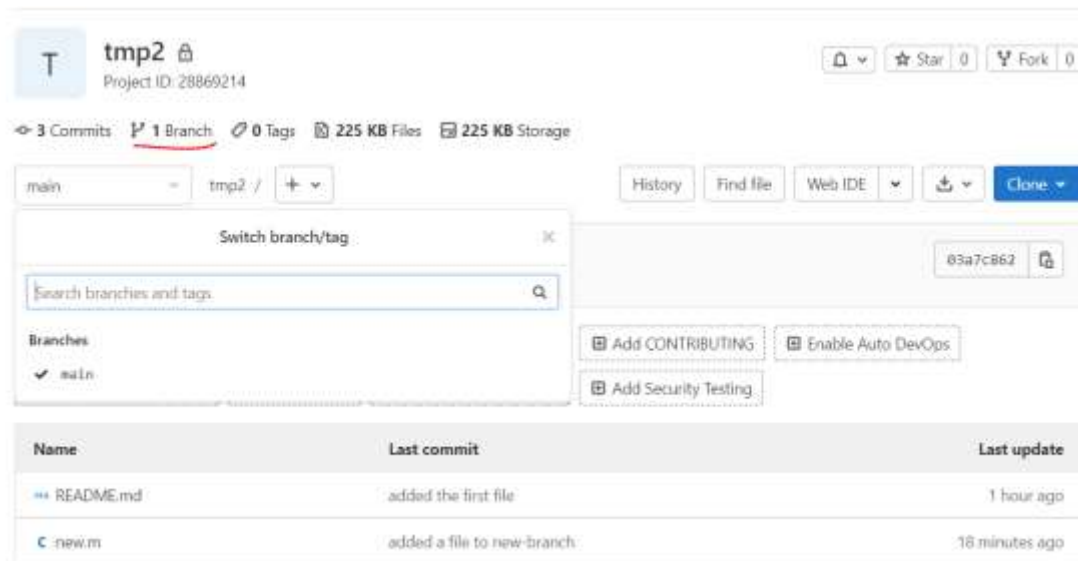


روی added a file to new-branch کلیک کنید و در صفحه ی باز شده روی گزینه ی merge کلیک کنید.



به این ترتیب عمل ادغام تکمیل می شود. در این ادغام چون تیک delete source branch فعال بود، خود به خود، گیت لب شاخه ی new-branch را حذف می کند. بنابراین با برگشت به پروژه ی tmp2 در

گیت لب، مشاهده می شود که فایل new.m اضافه شده است و تنها شاخه ی موجود در پروژه شاخه ی اصلی یا main می باشد.



حال برای اینکه تغییرات merge روی شاخه ی اصلی در پروژه ی tmp2 شما نیز اعمال شود، باید ابتدا با استفاده از git checkout main به شاخه ی اصلی بروید. در این شاخه دستور git pull origin main را اجرا کنید. مشاهده می شود که فایل new.m به شاخه ی اصلی اضافه شده است. در نهایت با استفاده از دستور git branch -d new-branch این شاخه را حذف کنید.

```

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (new-branch)
$ git checkout main
Switched to branch 'main'

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git pull origin main
Enter passphrase for key '/c/Users/mehrnaz/.ssh/id_rsa':
client_global_hostkeys_private_confirm: server gave bad signature for RSA key 0
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 247 bytes | 1024 bytes/s, done.
From gitlab.com:meh_hn/tmp2
 * branch          main          -> FETCH_HEAD
   3cd2529..03a7c86 main          -> origin/main
Updating 3cd2529..03a7c86
Fast-forward
 new.m | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 new.m

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git branch
* main
  new-branch

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git branch -d new-branch
Deleted branch new-branch (was f8f2968).

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$ git branch
* main

mehrnaz@DESKTOP-8F84L06 MINGW64 /c/users/user/tmp2 (main)
$

```

حال اگر دوباره git branch را اجرا کنید، مشاهده می کنید که فقط شاخه ی main موجود است.

Fork ۲.۸

این گزینه به معنی تولید یک کپی کامل از repository یک پروژه در یک حساب کاربری مجزا از حساب اصلی پروژه است. تمام پروژه های عمومی در گیت لب امکان استفاده از گزینه ی fork را به کاربران می دهند و برای پروژه های خصوصی نیز در صورت دادن دسترسی، اعضا می توانند پروژه را fork کنند.

مهمترین کاربرد fork در این است که کاربر دیگر نیازی به اعمال تغییر روی شاخه ی اصلی پروژه ندارد و می تواند به صورت موازی با پروژه ی اصلی تغییراتی را که می خواهد روی نمونه ی fork شده ی خود اعمال کند و در نهایت مانند یک شاخه ی فرعی و با استفاده از merge request فرستادن تغییرات خود را با شاخه ی اصلی پروژه ادغام کند.

برای fork کردن یک پروژه کافیست وارد صفحه ی آن پروژه بشوید و روی گزینه ی fork کلیک کنید. به این ترتیب صفحه ای به صورت زیر باز می شود که باید در آن در بخش project url نام گروه یا برای مثال حساب کاربری خود را وارد کنید:

Fork project

A fork is a copy of a project.

Forking a repository allows you to make changes without affecting the original project.

Project name

Project URL

Project slug

به این ترتیب یک کپی از پروژه در حساب کاربری شما به اشتراک گذاشته می شود. اطلاعات این پروژه و اینکه مرجع اولیه ی آن کجا بوده است همواره در حساب کاربری شما موجود می باشد ولی دسترسی شما کاملاً مربوط به حساب خودتان است و امکان هر عملی برای شما روی کپی fork شده ی شما فراهم خواهد بود. این اعمال شامل استفاده از دستوراتی مانند clone, pull, push و branch نیز می شود.

برای ادغام پروژه ی fork شده با پروژه ی اصلی لازم است که یک merge request تولید کنید. برای این کار نیز باید در سمت چپ صفحه روی گزینه ی merge request کلیک کنید، و یک new merge request ایجاد کنید:

My Awesome Group > My second project > Merge requests > New

New merge request

Source branch

Update README.md

mehnaz hn authored 10 minutes ago

62563fd7



Target branch

Initial commit

mehnaz hn authored 14 minutes ago

c8f558da



Compare branches and continue

در Source branch باید گروه، شاخه، یا repository شما انتخاب شود و در target branch نیز باید مسیری که می خواهید پروژه ی شما با آن ادغام شود، مسیر اصلی پروژه، انتخاب شود. با انتخاب گزینه ی compare branches and continue درخواست ادغام ارسال می گردد.

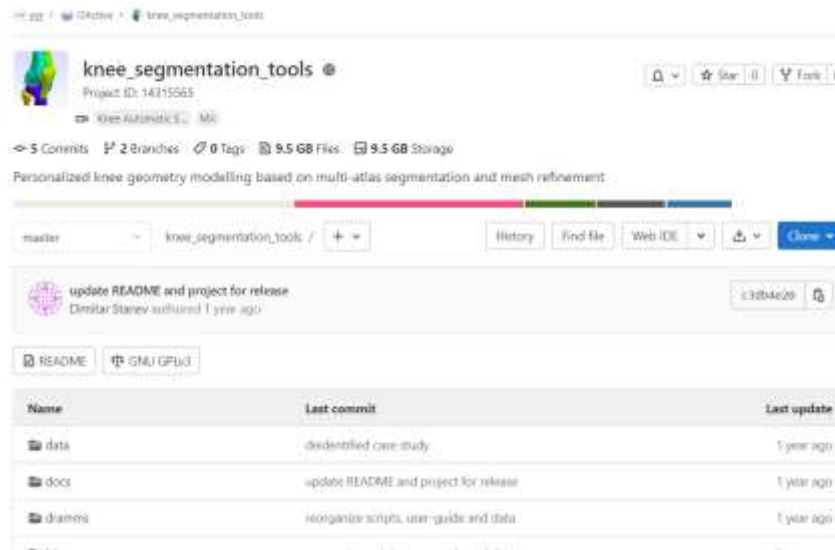
به این ترتیب برای حسابی که به عنوان target انتخاب کردید، درخواست ادغام تولید می شود و کاربر آن می تواند بعد از مقایسه آن را تایید کند:



fork ۲.۹

Fork کردن مانند تولید یک کپی از یک پروژه و به صورت یک شاخه ی کاملاً مجزا از آن پروژه می باشد.

برای fork کردن پروژه ها نیز ابتدا لازم است به repository پروژه ی مورد نظر خود بروید، برای مثال با استفاده از گزینه ی explore projects یک پروژه که مد نظر شماست را انتخاب کنید، برای مثال:



با انتخاب گزینه ی fork می توانید یک کپی از این پروژه را در مسیر مورد نظر خود ایجاد کنید. برای انتخاب مسیر نیز باید در بخش project url نام گروه یا برای مثال حساب کاربری خود را وارد کنید:

Fork project

A fork is a copy of a project.

Forking a repository allows you to make changes without affecting the original project.

Project name

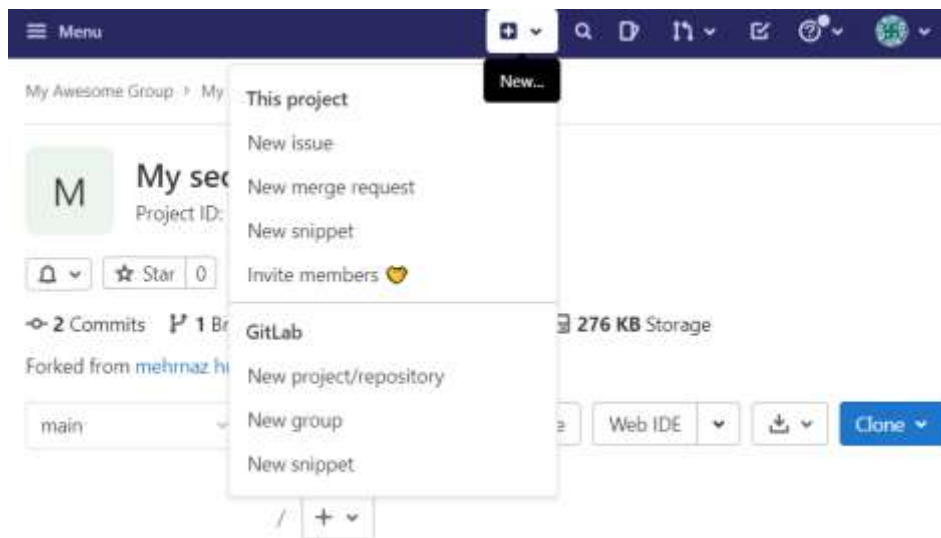
Project URL

 Select a namespace

Project slug

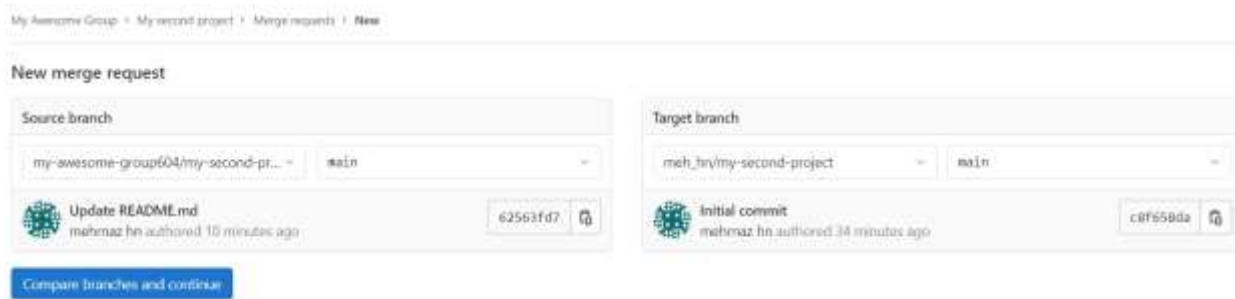
. برای مثال در زیر بعد از ادیت کردن فایل README.md در پروژه، بروی گزینه ی + در بالای صفحه

کلیک کنید و سپس گزینه ی new merge request را انتخاب کنید:



به این ترتیب صفحه ی زیر باز می شود که در آن باید مسیری که می خواهید پروژه ی شما با آن ادغام شود،

مشخص شود:



در Source branch باید گروه، شاخه، یا repository شما انتخاب شود و در target branch نیز باید

مسیری که می خواهید پروژه ی شما با آن مرج شود انتخاب شود. با انتخاب گزینه ی compare branches

and continue درخواست برای شما، اگر پروژه ی مرجع یکی از پروژه ها یا یکی از شاخه های پروژه ی شما

باشد، یا برای repository مرجع ارسال می شود. در صفحه ی باز شده کاربر می تواند فایل های تغییر داده شده

را با هم مقایسه کند، درباره ی آنها نظر بدهد و در صورت تایید آن ها را با مسیر اصلی پروژه ی خود ادغام کند.

که در نهایت صفحه ی زیر تولید می شود:



۲.۱۰ خصوصی بودن یا عمومی بودن یک پروژه

پروژه و حتی حساب کاربری را می توان در گیت لب به صورت `public` یا `private` تعریف کرد. در صفحات `public` یا عمومی، امکان دسترسی به پروژه ها برای همه ی کاربران وجود دارد و هر فردی به راحتی می تواند از امکاناتی مانند `clone`، `import` و `fork` استفاده کند و به عبارتی در یک پروژه مشارکت کند. به این ترتیب هر کاری که روی پروژه ی عمومی انجام بدهید، در نهایت می توانید با استفاده از دستوراتی که پیشتر توضیح داده شد، مانند `push`، استفاده کنید و تغییرات را ادغام کنید.

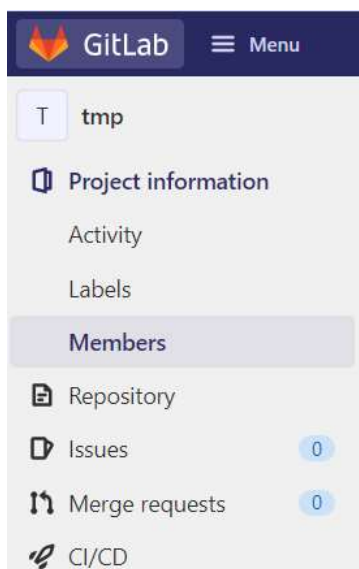
در پروژه های خصوصی یا `private` ولی در صورتی امکان دسترسی به `repository` وجود خواهد داشت که کاربر به عنوان یک عضو در آن پروژه یا گروه معرفی شود.

۲.۱۱ اعضای یک پروژه


وقتی که یک فرد یا حساب کاربری به عنوان عضو یا `member` یک پروژه انتخاب می شود، براساس نقشی که برای او تعیین شده باشد، می تواند به آن پروژه دسترسی داشته باشد. چهار نقش برای اعضا قابل تعریف هستند که عبارتند از `guest`، `reporter`، `developer` و `maintainer`. به جز نقش `maintainer` باقی نقش ها با محدودیت هایی برای دسترسی به فایل ها رو به رو هستند. نقش `maintainer` نیز عموماً همان حساب اصلی

که پروژه در آن تعریف شده باشد یا owner است. SSH Key به Owner خصوصی و عمومی حساب کاربری دسترسی دارد و نام کاربری او و ایمیلی که در Git Bash یا ترمینال تعریف کرده است، مشابه با نام کاربری حساب گیت لب است. هرچند که maintainer می تواند کسی غیر از owner نیز باشد ولی owner تنها یک نفر است.

برای تعیین یک عضو در پروژه، owner/maintainer باید ابتدا پروژه را باز کند. سپس روی گزینه ی project information کلیک کند و گزینه ی member را انتخاب کند.



در صفحه ی باز شده ابتدا نام کاربری فرد مورد نظر و سپس نوع عضویت او را انتخاب کند. با کلیک روی گزینه ی invite یک پیام به ایمیل عضو جدید فرستاده می شود و در حساب کاربری او در گیت لب نیز پروژه ای که در آن عضو شده است به اشتراک گذاشته می شود. به این ترتیب فرد می تواند براساس نوع عضویت به پروژه دسترسی داشته باشد.

در صورتی که بخواهید عضویت فردی را خاتمه بدهید فقط کافیست که در صفحه ی `project members` روی گزینه ی  کلیک کنید. همچنین در صورتی که تاریخ انقضا برای عضویت تعیین کنید، اعضا به صورت خود به خود حذف می شوند.

امکاناتی که هر نقش برای عضو یک پروژه تعیین می کند در ادامه آورده شده است:

۱- Guest

فقط امکان دسترسی به اطلاعات کلی فایل ها را دارد و مشاهده ی کدها و سایر اطلاعات به جز فایل `wiki` را ندارد.

۲- Reporter

به همه ی فایل ها دسترسی دارد. امکان `clone` کردن پروژه، `pull` و `fork` را دارد ولی نمی تواند فایلی را تغییر دهد مگر اینکه از قبل پروژه را `fork` کرده باشد. در این صورت بعد از اعمال تغییرات می تواند با فرستادن `merge request` از `owner/maintainer` بخواهد تا تغییراتی را که اعمال کرده است با شاخه ی اصلی ادغام کنند. کاربری که به عنوان `reporter` معرفی شده باشد امکان `push` کردن فایل را نیز به جز روی کپی `fork` شده ی خود نخواهد داشت.

۳- Developer

به همه ی فایل ها دسترسی دارد و امکان `clone` و `pull` و `fork` نیز برای او مهیا است. همچنین `developer` می تواند فایل ها را ویرایش کند یا فایلی اضافه یا کم کند، ولی قبل از اینکه تغییری را روی شاخه ی اصلی نهایی کند باید `merge request` بفرستد تا `owner/maintainer` آن را تایید کند. برای این منظور وقتی تغییرات مستقیماً روی فایل ها در سایت گیت لب اعمال شود، سایت قبل از `commit` گزینه ی `merge request` را در اختیار کاربر می گذارد تا درخواست ادغام فرستاده شود.

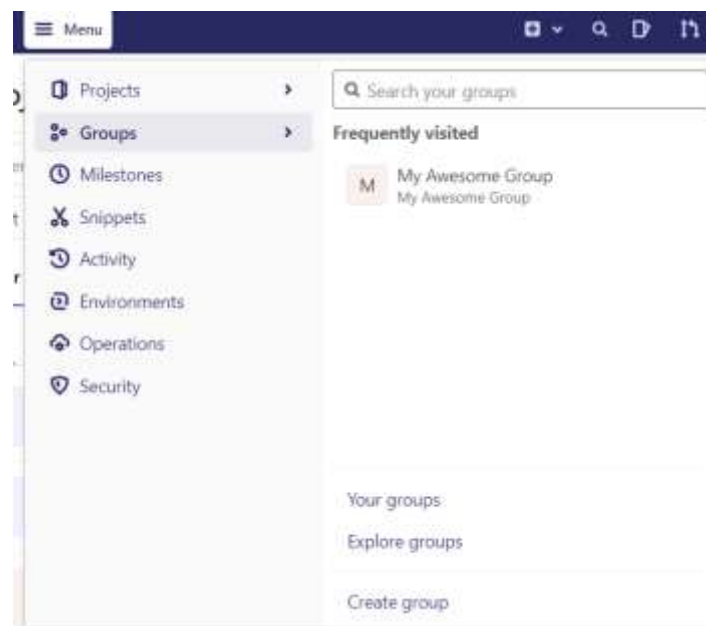
اگر کاربر بخواهد از طریق گیت تغییری را اعمال کند باید ابتدا یک شاخه ی جدید تولید کند و تغییرات را روی شاخه اعمال کند و با استفاده از دستور `git push origin new-branch`، تغییرات را `push` کند. در این حالت گیت از کاربر می خواهد تا به حساب کاربری خود برود و یک `merge request` تولید کند. در نهایت با تایید این درخواست تغییراتی که اعمال کرده توسط `maintainer/owner` ادغام خواهد شد.

۴- Maintainer

به کلیه ی فایل ها دسترسی دارد و می تواند عضو نیز تعریف کند.

۲.۱۲ تعریف گروه

برای تعریف باید به `menu` در بالای صفحه سمت چپ مراجعه کنید و در این قسمت روی گزینه ی `groups` و سپس `create group` کلیک کنید. با وارد کردن اطلاعات مربوط به نام گروه و نوع گروه و سپس دعوت اعضا، با استفاده از ایمیلشان، می توانید گروه خود را تولید کنید.



همانطور که مشاهده می شود در این قسمت می توانید با انتخاب **your groups** باقی گروه هایی که در آن عضو هستید را مشاهده نمایید.

ویژگی اصلی گروه در این است که اعضای گروه می توانند به طور همزمان به تمامی پروژه هایی که در آن گروه تعریف شده اند، دسترسی داشته باشند. همچنین، اعضای گروه می توانند تمامی **issues** و **merge requests** را مشاهده کنند و به ارزیابی ها دسترسی داشته باشند. با استفاده از چنین ساختاری می توانید با تمامی اعضای گروه به طور همزمان ارتباط برقرار کنید و پروژه ها را مدیریت کنید.

۲.۱۳ حل conflict

Conflict معمولاً وقتی پیش می آید که دو نفر یک خط در یک کد را عوض کنند و همزمان تغییرات خود را **commit** کنند. در این حالت گیت نمی تواند هر دو تغییر را روی فایل اصلی اعمال کند و یکی از دو نفر باید ابتدا تغییرات اعمال شده توسط فرد دیگر را به سیستم خود **pull** کند و درباره ی اولویت تغییرات تصمیم بگیرد، مشکل حل شده را **add** و سپس **commit** را اجرا کند و در نهایت کد نهایی را به گیت لب انتقال داده یا **push** کند.

برای جلوگیری از رخداد چنین شرایطی لازم است تا بلافاصله بعد از هر تغییری که روی **remote** اعمال شد، عمل **pull** انجام شود تا کاربران دیگر بتوانند از آن استفاده کنند قبل از اینکه خودشان تغییری ایجاد کنند.