


# GIT

סיכום קורס git



**החוברת מיועדת לתלמידות הקורס בלבד  
אין להעביר הלאה ללא רשות מפורשת!**

נחמי לרנר

[nechamy5834@gmail.com](mailto:nechamy5834@gmail.com)

## תוכן

2	מה זה GIT?
4	איך נשתמש ב git ב local?
4	התקנת git:
5	שמירת גרסאות ב git הלוקאלי:
5	פקודות נוספות:
6	github:
8	יצירת repository ב github:
11	איך נחבר בין git ל github?
11	מ github ללוקאלי (נפוץ יותר בעבודת צוות):
11	מלוקאל ל github (נפוץ יותר בעבודה על פרויקטים יחידניים):
12	פקודות נוספות:
13	הגדרת הרשאה בין המחשב ל github:
14	עבודה עם branch'ים:
14	פקודות רלוונטיות:
15	עוד קצת מעבר:
15	התאמה אישית של הגדרות git:
15	מיקום ההגדרות:
15	הגדרות alias:
15	הגדרות נוספות:
16	שיתוף פרויקט עם אנשים נוספים:
17	הגנה על branch הראשי:
19	מה זה pull request (PR)?
23	טיפול בקונפליקט:
24	נספח:
24	טיפול בפילות:
24	דגשים חשובים כלליים בנושא שגיאות:
24	שגיאות נפוצות:
27	קבצים בסיסיים ב git:
28	פקודות ביטול לתהליכים שנלמדו:
28	שינויים בהגדרות בסיסיות של repo:
30	Readme and Markdown Language
33	GIT Cheat Sheet

# מה זה GIT?

## הקדמה וקצת מעבר

git זו **תוכנה** שמותקנת על המחשב ומנהלת את גרסאות הקוד בצורה מקומית.

ניהול גרסאות זוהי יכולת חשובה מאוד במהלך פיתוח תוכנה, היא מאפשרת לנו:

- לנהל גרסאות שונות של הקוד - לדוגמה להוסיף תכונות חדשות וכדו' בגרסה נפרדת, בלי לפגוע בקוד המקור המקורי של הפרויקט. לאחר שהגרסה עם התכונה החדשה תהיה מוכנה ובשלה לשימוש, נוכל לאחד אותה עם הגרסה הראשית, ולקבל גרסה מושלמת ובשלה לשימוש. תוכנה חכמה לניהול גרסאות תוכל לנהל לנו את כל התהליך בקלות, ללא שנצטרך ידנית להעתיק קבצים וכדו'.
- שמירת היסטוריה של השינויים - יש אפשרות בכל רגע לחזור לגרסה קודמת של הקוד, אם מגלים במהלך העבודה ששינוי כלשהו הינו פגום, ניתן להסירו בקלות רבה. תוכנה לניהול גרסאות שומרת את כל ההיסטוריה השינויים עם שם לכל שינוי.

בחוברת הזו דבר ראשון נבין תהליך עבודה בgit - בעבודה מקומית,

ואח"כ נעבור לשילוב עם **github**, שזה **אתר**, שמאפשר לנו לעבוד עם גיט מרוחק - צורה זו מכונה **git remote**. האתר **github** הינו הגדול והמוכר ביותר, אולם ישנם אתרים נוספים מוכרים המשמשים לאותה מטרה, כמו **bitbucket**

עבודה ע"י **git remote** - כלומר **github** מאפשרת לנו לעבוד עם מספר אנשים בצוות על אותו הפרויקט בצורה מבוזרת ומסודרת, מאפשרת לשתף את גרסאות הקוד שלנו עם אחרים, כמו במעבר ממחשב למחשב, עם עובדים נוספים בצוות ועוד.

בצורה זו הקוד שמור ברשת, ויש לו גיבוי מסודר כל הזמן, וכל חבר בצוות יכול לעבוד על הקוד, ליצור לו גרסה משל עצמו, ולשמור את הכל בgit המרכזי. git ידע לנהל את הגרסאות בצורה חכמה, לאחד הכל, ומאפשר עבודת צוות נכונה וברורה.

- בחוברת הזו נשתמש בשם **github** כשם כולל עבור כל כלי **remote** הקיימים, הפקודות **git** תהיינה זהות גם בשימוש ב**bitbucket** או כלים אחרים.
- בחוברת זו נלמד להשתמש בגיט ע"י שורת הפקודה - נכיר את הפקודות המרכזיות לשימוש. אולם ברוב ה**IDEs** (=תוכנה לניהול הקוד, כמו **visual studio code**) יש תוסף שמאפשר להפעיל את כל הפקודות הכתובות כאן, בכפתורים מתאימים, כדאי שתכירי אותם גם כן.

### הנחיות חשובות לשימוש בשורת הפקודה

כאשר משתמשים בשורת הפקודה, יש לשים לב תמיד לתיקייה שבה עומדים, ולעמוד בתיקייה הנכונה. הניווט של התיקייה מופיע בתחילת השורה. ניתן לעבור בין תיקיות ע"י הפקודה **cd**. בכל הפקודות של גיט, חשוב לשים לב שנמצאים בתיקייה של הפרויקט המתאים. (בעת שימוש בתוכנת **visual studio code**, יש טרמינל - שורת פקודה בתוך העורך עצמו. ניתן לפתוח אותו ולהשתמש בו. הוא יפתח אוטומטית בתיקייה שפתוחה כבר בעורך. לכן חשוב מאוד קודם כל לפתוח פרויקט לעבודה, ואח"כ לפתוח את הטרמינל)

- הערה כללית, בחוברת זו אני משתמשת בסטנדרט להגדרת משתנים, ערכים שצריך להחליף בנתונים אמיתיים, על ידי הסימון **<param\_name>**. על מנת להפעיל את הפקודה בצורה הנכונה, יש להחליף את כל הטקסט, כולל ה**<>** בערך האמיתי המתאים.

## איך נשתמש ב **git** ב**local**?

התקנת **Git**:

בלינק הזה: <https://git-scm.com/>

נלחץ על הdownload שכאן



ונוריד מהלינק הזה

### Download for Windows

**Click here to download** the latest (2.43.0) 64-bit version of **Git for Windows**. This is the most recent **maintained build**. It was released **about 2 months ago**, on 2023-11-20.

נריץ את ההתקנה בלי שינויים.

## שמירת גרסאות ב-GIT הלוקאלי:

`git init` - יצירת מאגר קוד מקומי שיכיל את הגרסאות השונות של הקוד עבור הפרויקט הנוכחי.

`git add <.| folder/. | path/to/file>` - הוספת הקבצים הנבחרים לשלב `stagen`

`git commit -m '<message>'` - שמירת גרסה עם הודעה (message) במקרה ששכחת לעשות `-m` יפתח לך המסך של `vim`.

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
# (use "git push" to publish your local commits)
#
# Changes to be committed:
#   new file:   src/data.js
#
.git/COMMIT_EDITMSG [unix] (08:54 10/12/2023)
"~/Documents/הארוה/git/testGit/.git/COMMIT_EDITMSG" [unix] 14L, 381B
```

זה הדרך לסגור אותו בצורה נכונה -

אם בשורה למעלה (איפה שנמצא הסמן בתמונה) כבר יש את `message` של ה `commit`, צריך רק לשמור ולצאת על ידי `:wq`.

אם יש צורך להוסיף את ה `message` (כמו בתמונה), צריך להיכנס למצב `insert` (שהיה כתוב `--INSERT--` בשורה התחתונה), בלחיצה על התו `I` או `A`. ולכתוב בשורה העליונה את ההודעה, לצאת ממצב `insert` על ידי המקש `esc`, ולצאת לגמרי מהמסך עם `:wq`.

### פקודות נוספות:

`git status` - בדיקת מצב השינויים ב `branch` הנוכחי.

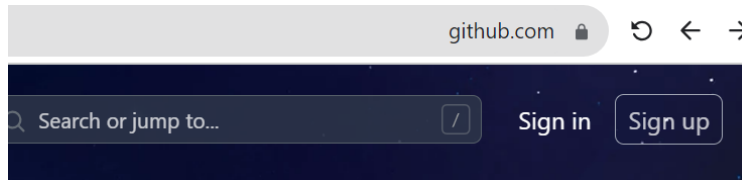
`*git diff` - השוואת השינויים בין `working dir` לבין `stagen` או `git local`, יש אפשרות להוסיף את שם הקובץ או התיקיה כדי לראות רק חלק מהשינויים.

`*git log` - צפיה ברשימת ה `commit`ים שבוצעה ב `branch`.

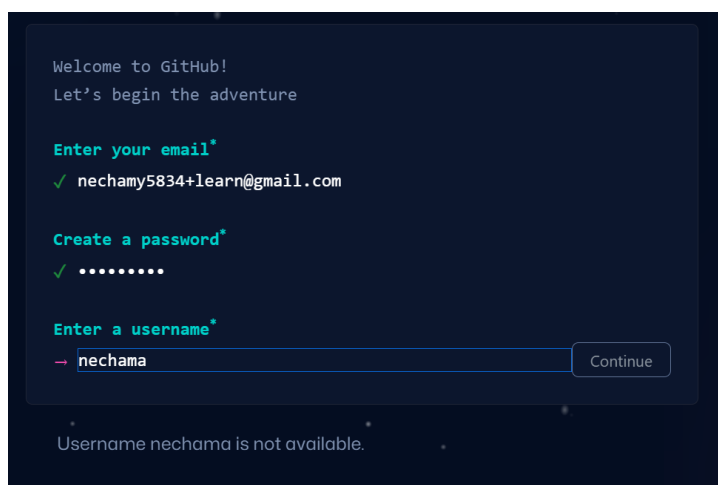
\* אפשרויות נוספות לפקודות האלו יש בקובץ `cheat sheet` המצורף (טבלה באנגלית).

את כל השלבים הנ"ל אפשר לעשות גם בלי חיבור לאינטרנט ובלי משתמש ב `github`, זו שמירה מקומית של גרסאות על המחשב, שמאפשרת גם לעבוד במקביל על כמה גרסאות, וכו'.

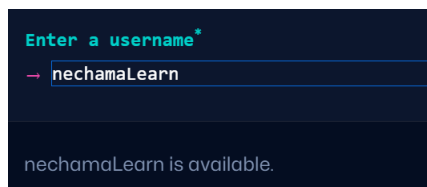
1. Go to github.com



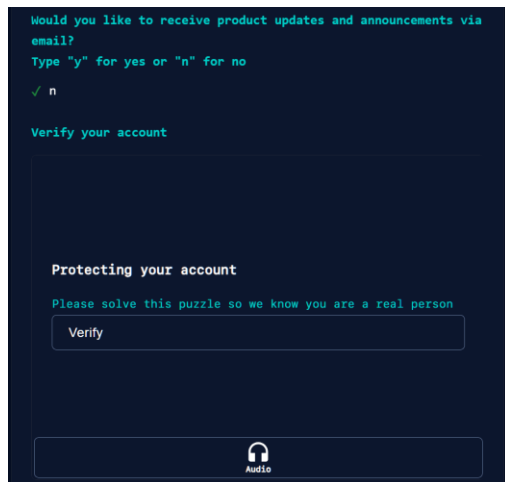
2. Choose sign up.
3. Insert your mail, choose password and username.



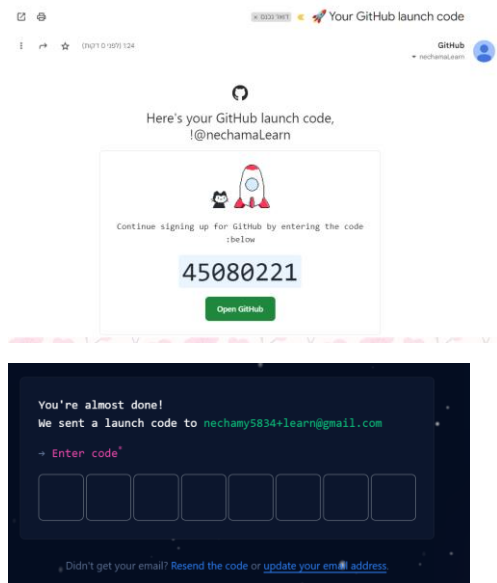
The username must be unique. Add last name or some else to get the message "is available".



4. You didn't want get announcements from GitHub, choose "n".
5. Press "verify" and complete the human test.



6. After create account you get mail with code, insert it to the website.

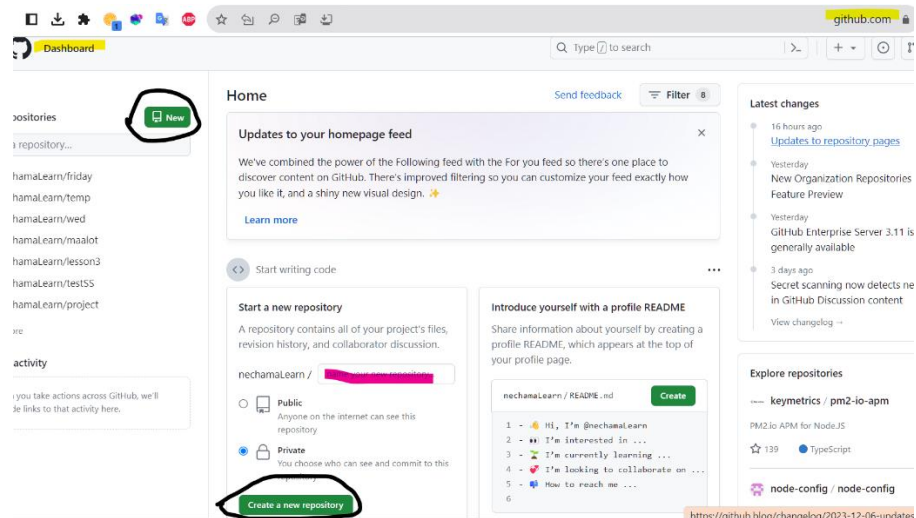


7. now, scroll down and chose "skip personalize" (gray in the center)



## יצירת REPOSITORY בGITHUB

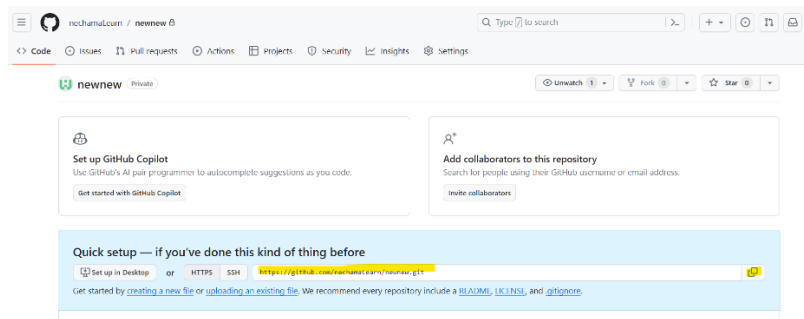
יצירת רפוזיטורי בgithub:



במסך הראשי של github יש לנו 2 כפתורים שמאפשרים לנו יצירת repo חדש: הכפתור הראשון במרכז המסך, בתוך טופס עם שם הrepo (מסומן בורוד) והגדרת הרשאות הצפייה שלו.

יש למלאות את השם ואח"כ ללחוץ על הכפתור הירוק.

יפתח לנו פרויקט ריק לגמרי:



אופציה נוספת היא הכפתור בצד שמאל של המסך, הוא יביא אותנו לעמוד חדש שבו נוכל להגדיר הגדרות נוספות על הrepo החדש שלנו:

## Create a new repository

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

Required fields are marked with an asterisk (\*).

Owner \* nechamaLearn / Repository name \*

Great repository names are short and memorable. Need inspiration? How about [super-duper-happiness](#) ?

Description (optional)

- ☒ **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:

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

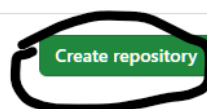
☒ **Add .gitignore**  
.gitignore template: None

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

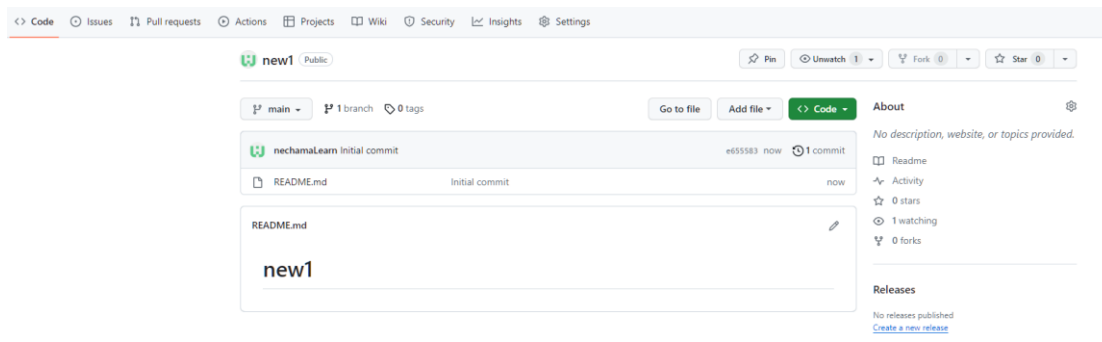
☒ **Choose a license**  
License: None

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

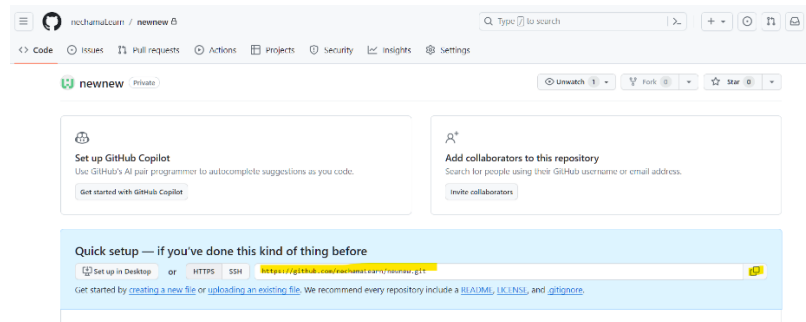
You are creating a public repository in your personal account.



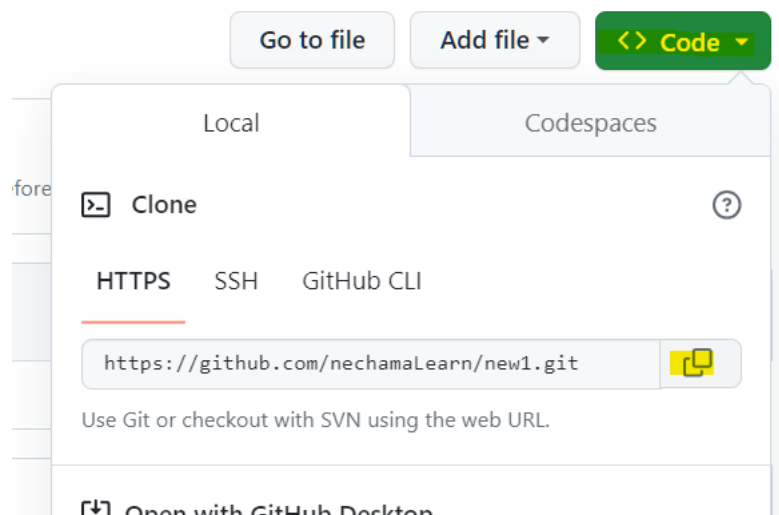
גם כאן נצטרך להכניס שם ולהגדיר הרשאות צפייה, אבל כאן נוכל להוסיף עוד 2 קבצים, `README` ו-`.gitignore`. ולהגדיר הרשאות. (פירוט על הקבצים בנספח)  
כאשר ניצור פרויקט שנבחר להוסיף בו את אחד הקבצים הוא כבר ייווצר כפרויקט משמעותי יותר, עם `branch` ראשוני והקובץ/קבצים שהוספנו.



לצורך עבודה עם repository החדש, נצטרך להעתיק את הלינק שלו.  
במאגר ריק:



במאגר עם קבצים זה יהיה תחת הכפתור `code`:



## איך נחבר בין GIT ל-GITHUB?

יש שתי דרכים לחיבור בין מאגר קוד (repository) מקומי לבין repository מרוחק (remote) בשתייהן נצטרך להשתמש בלינק של הרפוזיטורי החדש שיצרנו `<repo_link>` יוחלף בלינק שהעתקנו בשלב הקודם.

### github ללוקאלי (נפוץ יותר בעבודת צוות)

בתוכנת ה IDE שלנו ונוריד דרכה את הפרויקט החדש.

`git clone <repo_link>` - הורדת הפרויקט מהלינק בתיקיה חדשה אצלינו במחשב, עם כל השינויים שבוצעו בו עד היום.

```
PS C:\Users\Administrator\Documents\הארוה\git> git clone https://github.com/nechamaLearn/new1.git
Cloning into 'new1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

במקרה של הצלחה נראה done 100%.

אם הורדנו פרויקט ריק לגמרי (המסך התכלת) אז נקבל את ההערה הזו:

```
PS C:\Users\Administrator\Documents\הארוה\git> git clone https://github.com/nechamaLearn/new1.git
Cloning into 'nn'...
warning: You appear to have cloned an empty repository.
PS C:\Users\Administrator\Documents\הארוה\git>
```

התיקיה נוצרה, החיבור לgithub נוצר, אבל אין כלום מעבר לזה, גם לא default branch(main).

`cd <repo_name>` - מעבר לתיקיה החדשה שנוצרה (יש המון באגים שקורים ששוכחים את השלב הזה!)

עכשיו את עובדת על הפרויקט מקומית.

### מלוקאל לgithub (נפוץ יותר בעבודה על פרויקטים יחידניים)

נפתח פרויקט אותו נרצה להעלות לgithub.

נחבר אותו לgit ע"פ השלבים של עבודה לוקאלית (init, add, commit)

ניצור repo ריק לגמרי בgithub כנ"ל (המסך התכלת).

ונחבר ע"י הפקודה `remote add`:

`git remote add origin <repo_link>` - origin זה כינוי סטנדרטי שאנחנו נותנות ללינק אצלינו, מעכשיו כל פעם שנכתוב origin בפרויקט זה יביא אותנו ללינק של הרפו בgithub.

## פקודות נוספות:

**git push** - דחיפת השינויים לgithub.

**git pull** - משיכת שינויים מgithub - אלו שקרו בbranch עליו אני עומדת בלבד, הקוד שלי מתעדכן ומשתנה בפועל.

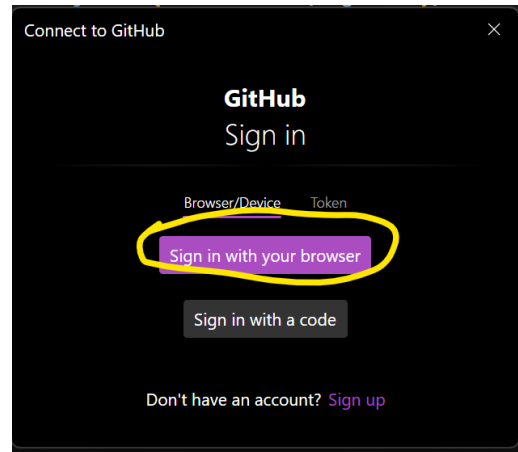
**git fetch** - משיכת שינויים מgithub - רק אלו שלא קשורים לbranch שלי, כולל commit בbranchים האחרים, וbranchים חדשים שעוד לא מוגדרים אצלי לוקאלית. שימי לב, שהפקודה לא משנה בפועל את הקוד שאני עובדת עליו, וגם לא את הbranchים האחרים שמשכתי את השינויים עבורם, ולכן אם ארצה לממש את השינויים האלו אשתמש בorigin/<branch> ולא רק <branch> מלבד הפקודה checkout ששולפת לנו את הbranch וממילא מושכת את כל השינויים בפועל.

**\*\* בשלבים אלו עלולות להיות נפילות שונות, הסתכלי בנספח.**

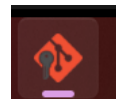
## הגדרת הרשאה בין המחשב לgithub:

בפעם הראשונה שבה נחבר בין github למחשב בפקודות שדורשות אימות של יוזר כמו `clone` של `private` או `push`, נצטרך לאמת את שם המשתמש בgithub בעזרת תוכנת `git credentials manager(GCM)`. התוכנה הזו אמורה להיות מותקנת אוטומטית בעת התקנת `git`, אבל היא לא תתמך ב**vsCode** בגרסה ישנה, צריך לוודא מראש שגירסת ה**IDE** לא רחוקה מטווח של חצי שנה אחורה.

ככה זה נראה:

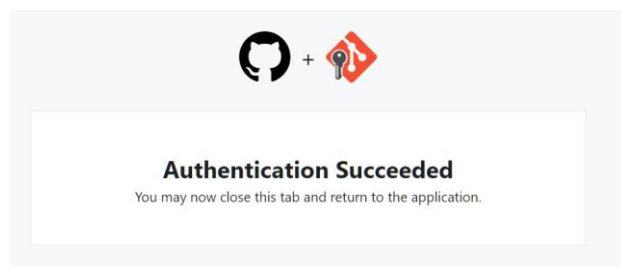


לפעמים זה לא יקפוץ מעל המסכים הפתוחים, ונוכל למצוא את התוכנה הפתוחה בשורת המשימות, עם האייקון הזה:



נבחר ב"sign in with your browser" ויפתח לנו חלון בדפדפן עם לינק לאימות המשתמש בgithub.

נוכל להעתיק את הלינק שנפתח לחלון סמוך לgithub שפתוח כבר, כדי לחסוך בהכנסת שם משתמש וסיסמה, לפעמים נצטרך ללחוץ על כפתור אישור מפורש, ולפעמים האישור יתבצע אוטומטית ברגע שיזוהה המשתמש.



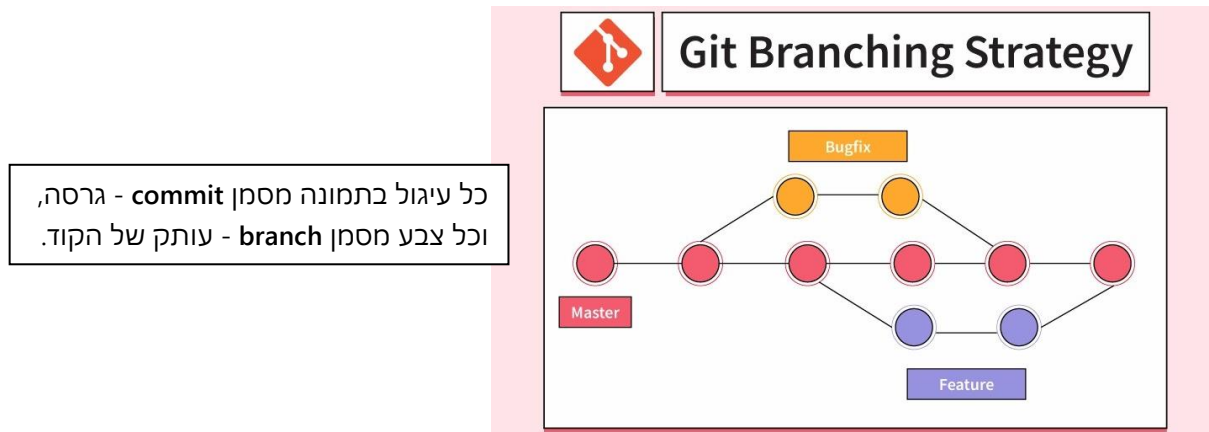
(לא תמיד המסך הזה יוצג, יש ל**GCM** באג בזה)

בשלב הבא נצטרך לחזור ל**vscode**, יהיו מקרים שזה יחזור אוטומטית, ויהיו מקרים שנצטרך לרענן את העמוד או לאשר `alert` או ללחוץ על לינק, כדי לחזור ל**vscode**, ושם נראה שהתהליך ממשיך ומסתיים ב**done 100%** או בכל סיום רצוי אחר.

(לפעמים יהיה כבר חיבור של משתמש אחר לgithub ונצטרך למחוק את האישור ולחזור על התהליך הנ"ל מחדש, פירוט נוסף בנספח)

## עבודה עם BRANCHים:

git יש לנו אפשרות לעבוד על כמה עותקים של הקוד במקביל, אם זה בקוד משותף הסיבה תהיה עותק עבור כל מפתח, וגם בעבודה לבד לפעמים ארצה ליצור עותק נוסף עבור תוספת קטנה לפרויקט שלא אמורה להיות תלויה בשינוי משמעותי וגדול שאני עושה במקביל(כמו פתרון באג במקביל לפיתוח של כמה ימים), לאפשרות הזו קוראים **branch**, ענף מהעץ של הפרויקט, שיוצא מנקודת גרסה מסוימת (**commit**) בד"כ מהענף הראשי, עליו נבצע שינויים כמו התהליך הרגיל, בלי תלות בשינויים נוספים שיקרו במקביל בעותקים אחרים.



כל עיגול בתמונה מסמן **commit** - גרסה,  
וכל צבע מסמן **branch** - עותק של הקוד.

מתוך: <https://www.scaler.com>

אנחנו לפעמים גם נשתמש בזה כדי להגדיר סביבות עבודה שונות, סביבת פיתוח - משותפת בין כל המתכנתים, סביבת בדיקות, וסביבת **production** - יצור.

## פקודות רלוונטיות:

**git branch** - יחזיר לנו על איזה **branch** אנחנו עובדים עכשיו.

**git branch <branch\_name>** - ייצור לי **branch** חדש, בלי לעבור אליו (פחות שימושי).

**git checkout <branch\_name>** - מעבר מ**branch** ל**branch**.

**git checkout -b <branch\_name>** - יצירת **branch** חדש ומעבר אליו.

**git merge <branch\_name>** - איחוד בין שני **branch**ים, כך שה**branch** שאני עומדת עליו יקבל את כל השינויים שהתרחשו ב**branch** השני.

לדוג' אם אני עומדת על ה**main branch** ואני רוצה למשוך אליו את השינויים מ**bug1** אכתוב -

**git merge bug1**

בדרך כלל כאשר נרצה לאחד שינויים אל ה**branch** הראשי, נגן עליו ולא נבצע את הפקודה **merge**, אלא נעטוף אותה ב**pull request**. (פירוט בהמשך)

## עוד קצת מעבר:

### התאמה אישית של הגדרות **git**:

**git** יש מאגר של הגדרות אישיות שאפשר לעדכן לפי הצורך.  
מבנה הפקודה לשינוי היא:

```
git config <key> <value> <options_like_--global>
```

הדבר הראשון שנצטרך להגדיר ב**git** זה שם משתמש ומייל, הפרטים האלו ישמרו עבור כל **commit**.

```
git config user.name <your_name>
```

```
git config user.email <your_email>
```

### מיקום ההגדרות:

כדי להגדיר את ההגדרות על כל הפרויקטים במחשב נוסף **--global** ואם נרצה זה יהיה רק על הפרויקט הנוכחי נסיף **--local**.  
יש לנו קובץ נוסף של הגדרות **system** שבו אפשר רק לצפות ולא לכתוב אותו, ניתן לדרוס את ההגדרות שלו בשלבים של **global** או **local** או להתקין מחדש את **git** וע"י שינוי הבחירה בתהליך לשנות את ההגדרות האלו.  
לדוג:

```
git config --global -l
```

– יציג ברשימה את כל ההגדרות שנמצאות ברמת **global** על כל הפרויקטים במחשב

### הגדרות **alias**:

יש לנו אפשרות ליצור קיצורים לפקודות ארוכות של **git**, לדוג' הפקודה:  
**log -5 --oneline** שמדפיסה לי 5 **commit**ים אחרונים שבוצעו בשורה אחת, בגלל שעבורי היא שימושית מאוד אני רוצה לקצר אותה למילה אחת.

```
git config --global alias.l5o "log -5 --oneline"
```

בפעם הבאה שאכתוב **git l5o** יודפסו לי 5 **commit**ים האחרונים בשורה אחת.

### הגדרות נוספות:

הגדרה חשובה לדוג' היא הגדרת יצירת **branch** חדש ב**github** ב**push** של **branch** שלא מוכר ב**github**. (פירוט בנספח)

עוד הגדרות שהגיוני שאצטרך להגדיר, אלו ההגדרות של תעודת האבטחה של החסימה:  
אם תיפול לי שגיאת **SSL** אדע שככל הנראה מדובר בבעיה של נטפרי/אתרוג ואתקין תעודת אבטחה מתאימה בהגדרות של **git**. (פירוט בנספח)

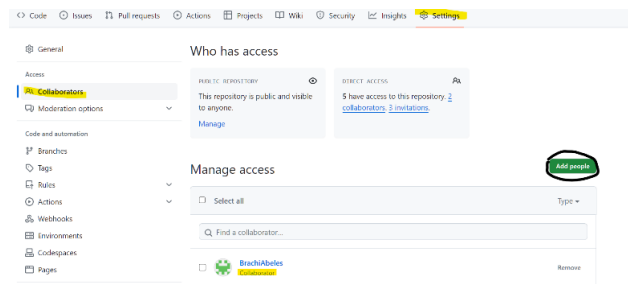


## שיתוף פרויקט עם אנשים נוספים:

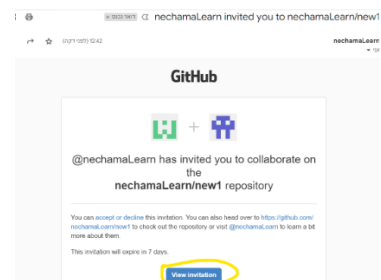
בכל פרויקט אנחנו יכולים לאפשר לאנשים נוספים לתרום קוד. (בד"כ במקרים כאלו נקפיד להגן על main, ע"י PR וחובת אישור)

זה משמעותי מאוד כשאנו רוצים לאפשר לחברה לעבוד איתנו על פרויקט גמר, וע"י שיתוף בgithub נוכל לעבוד במקביל בלי העתקות של קוד, ובלי לשבת ביחד על אותה רשת בסמינר.

נכנס ל `setting->collaborators`:



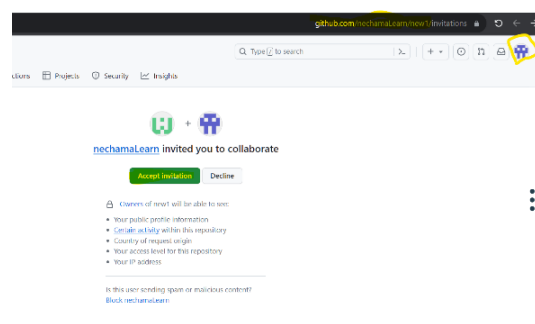
נבחר ב `add people` ונוסיף את שם המשתמש בgithub של מי שאנו רוצים שיצטרף אלינו. הוא יקבל התראה על הזימון ב2 מקומות: במייל:



או בתוך האתר של github ליד האיקון של היוזר, יש איקון של דואר נכנס ותהיה שם נקודה כחולה שמסמנת שיש הודעה שלא נקראה.



ואז המוזמן יוכל לאשר את ההזמנה ולהתחיל לתרום קוד.



## הגנה על ה-BRANCH הראשי:

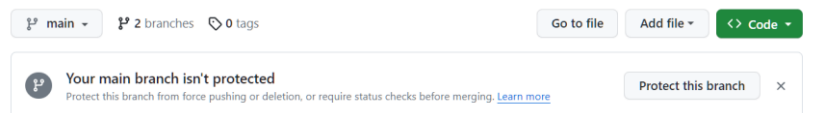
הענף המרכזי ממנו מתחילה התוכנית (default branch), נקרא בד"כ main או master, יכולים להיות מקומות עבודה שיקראו לו אחרת.

מכיון שהוא הראשי פעמים רבות נרצה להגן עליו, במקומות עבודה הוא פעמים רבות עולה אוטומטית לסביבת יצור עם כל שינוי, וגם בפרויקט ששתיים עובדות יחד, יהיה הרבה יותר מורכב לטפל בקונפליקטים בצורה נכונה אם הכל יהיה בbranch אחד.

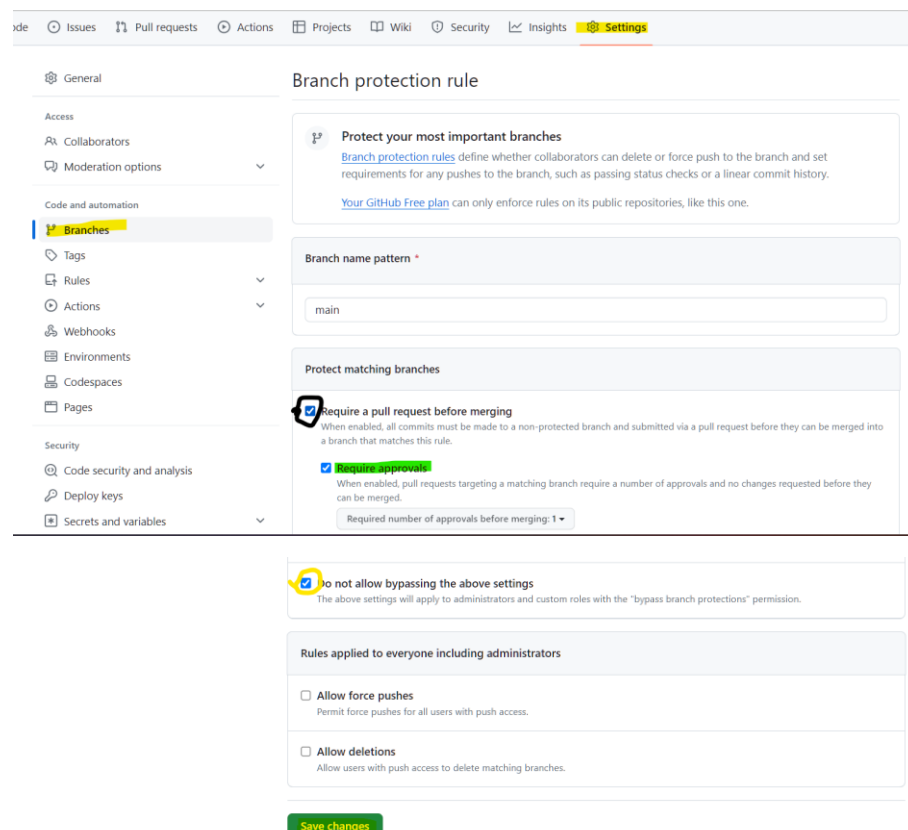
דבר ראשון שחשוב לזכור כשאנחנו רוצים להגן על הbranch הראשי, זה הימנעות מוחלטת מכתובת קוד ישירות בgithub, כי commit בgithub יעבור את השלבים בלחיצת כפתור, גם add גם commit וגם push (פעמים רבות לmain), אז המלצה חמה שלי, כל קוד חדש ייכתב לוקאלית, יעבור תהליך של בדיקת קוד והגנה מינימלית לפני שהוא יעלה לmain ויתפרסם.

יש לנו גם אפשרות להגדיר הגנה על branch'ים, ובdefault branch בד"כ github יציעו לנו את זה מצידם, ברגע שהם יקלטו שיש לנו יותר מbranch אחד בפרויקט, ההגדרה הזו תמנע לבצע push או merge ישירות לmain בgithub ללא תהליך של pr.

כך זה נראה כשgit מציעים לנו:



וככה זה נראה בהגדרות (setting->branches) בפועל:



אנחנו נבחר בrequire pull request ואם נעבוד עם מישהו נוסף נוסיף גם את ההגנה של require approve (בד"כ זה יבחר אוטומטית יחד, ובמקרה של עבודה עצמאית נצטרך להסיר את הבחירה בcheckbox).

כאשר ננסה לדחוף עדכון לbranch המוגן נקבל את השגיאה הזו:

```
PS C:\Users\Administrator\Documents\הארה\git\new1> git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 326 bytes | 326.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote: error: GH006: Protected branch update failed for refs/heads/main.
remote: error: Changes must be made through a pull request.
To https://github.com/nechamaLearn/new1.git
! [remote rejected] main -> main (protected branch hook declined)
error: failed to push some refs to 'https://github.com/nechamaLearn/new1.git'
```

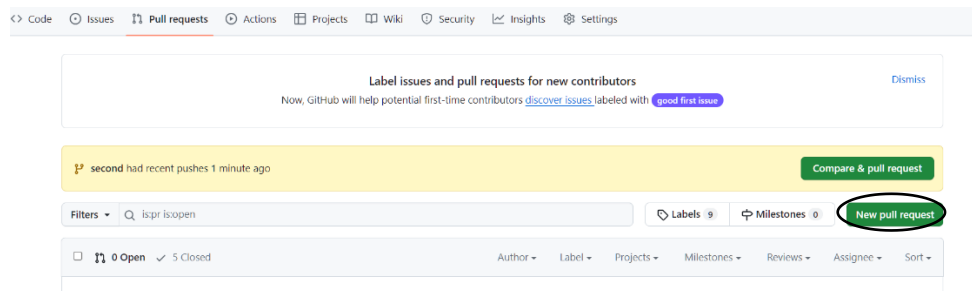
ונצטרך לצאת לbranch אחר לוקאלי, ואותו לדחוף ואז לעשות merge ע"י pr בgithub.

## מה זה (PR)PULL REQUEST?

על מנת לבצע merge מוגן, הגדירו בgithub תהליך של בקשה לדחפת השינויים לbranchים המוגנים.  
ברגע שנבצע push לשינויים בbranch שלנו, github יזהה את השינויים ויציע לנו לפתוח בקשת PR:



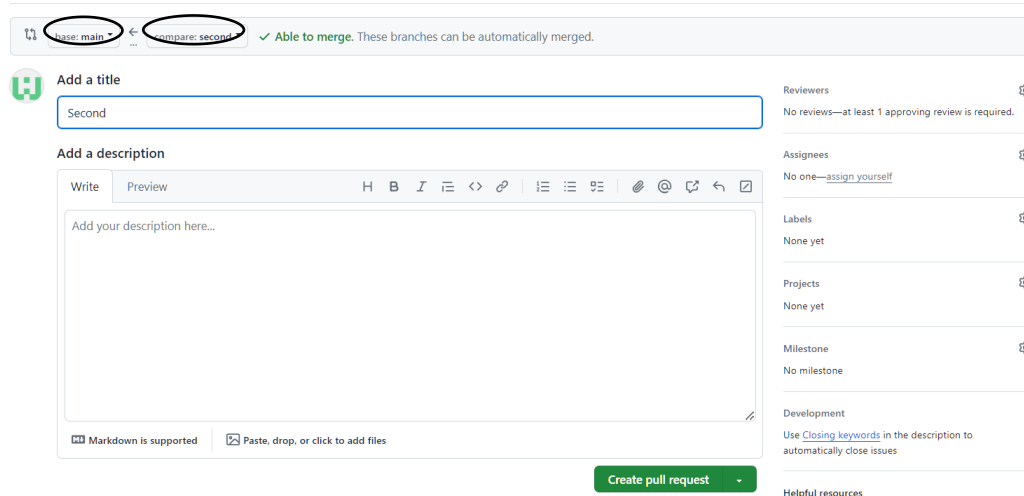
גם אם ההצעה לא תוצג במסך הראשי, תמיד נוכל לפתוח PR, נכנס לטאב של pull requests ונבחר new pull request



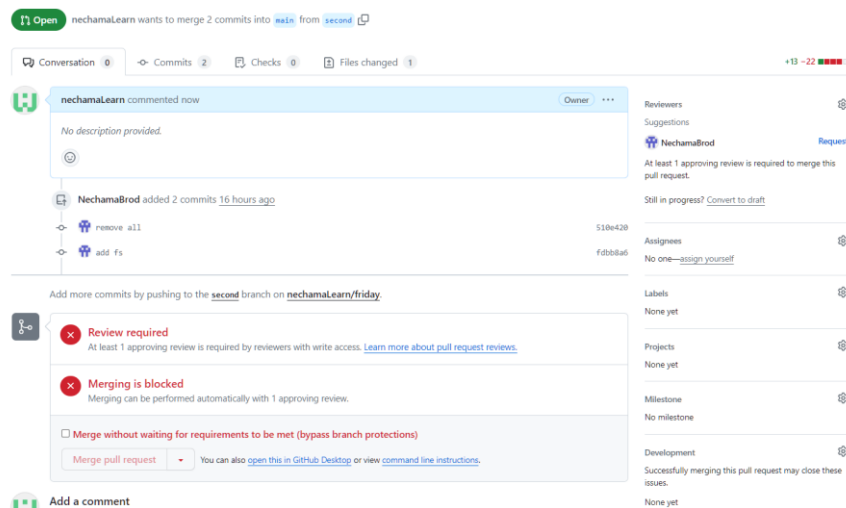
ואז יפתח לנו המסך הזה:

### Open a pull request

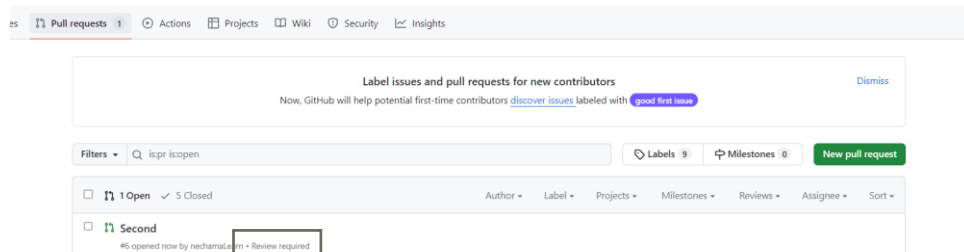
Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).



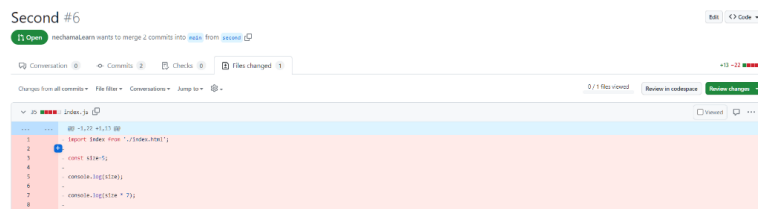
כאן נוכל לבחור מי branch שמביא את השינויים (compare) ומי מקבל אותם אליו (base), בתמונה זה שינויים שמגיעים מsecond לmain.  
בתחתית המסך נוכל לראות גם את השינויים שעתידיים להתבצע, וניצור את הPR.  
בשלב הזה נוכל כבר לזהות קונפליקטים, כאשר במקום הV הירוק למעלה יהיה X אדום, זה לא ימנע מאיתנו ליצור את הPR ונטפל בזה אחרי שהPR יפתח.



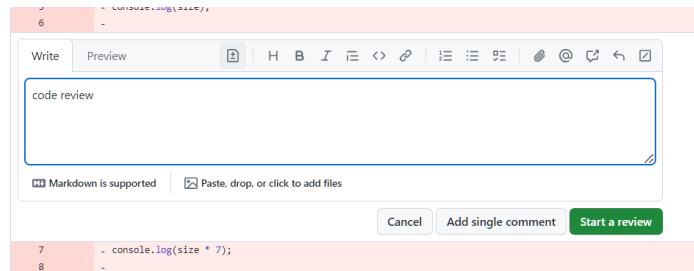
במסך הזה נוכל לראות את פרטי PR - התכתבות והערות על השינוי בטאב **Conversations**. רשימת הcommits שנכנסים בmerge ב**Commits**. ובטאב **File changed** יהיו כל הקבצים עם השינויים בפועל. נוכל גם לראות שבגלל שאנחנו דורשים גם אישור של אדם נוסף שיקרא את הקוד, הכפתור **Merge pull request**, לא מאפשר. לפעמים הוא יהיה לא מאפשר בגלל קונפליקט, בו נטפל ונעלה commit עם הפתרון. לתהליך אישור הPR על ידי מפתח אחר קוראים **code review**. נוכל לראות את הPR שממתין לאישור במסך **pull requests** הראשי:



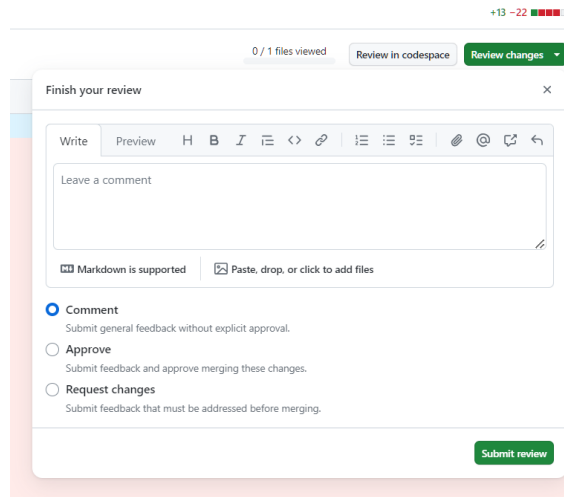
נבחר בPR הרצוי, נפתח אותו, ונעבור לטאב **File changed** שם נראה את השינויים שבוצעו.



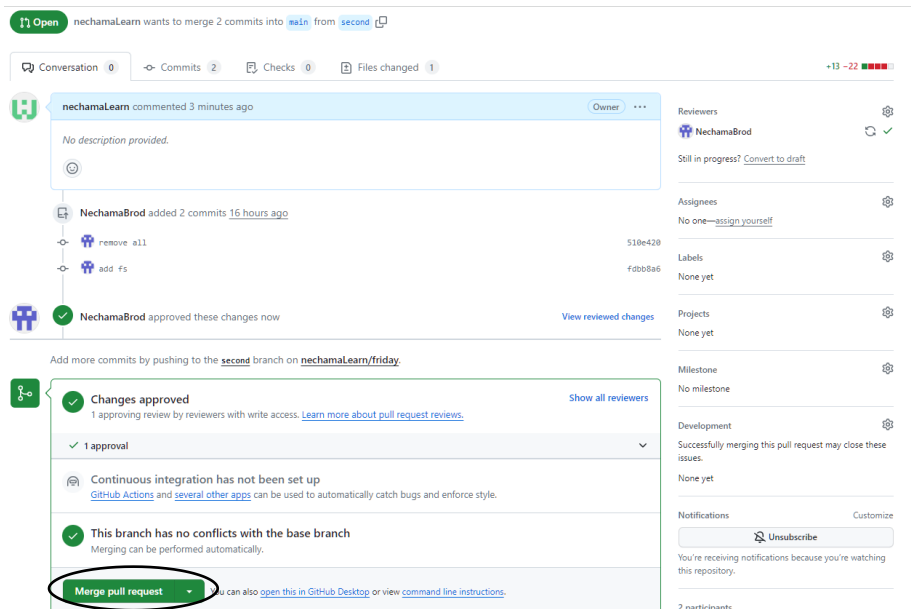
נוכל לדרוש שינויים ולכתוב הערות למפתח בלחיצה על **plus** הכחול.



ובסוף סקירת הקוד, נבחר ב"review changes" ונאשר או נדרוש שינויים מהמפתח, אם נדרוש שינויים נצטרך לעבור שוב על הקוד ולאשר אותו לפני שהמפתח יוכל לעשות merge.



אחרי האישור המפתח יראה את הכפתור "Merge pull request" כשהוא מאופשר.



נוכל להוסיף הערה לפני סגירת ה PR ולאשר את merge.

Add more commits by pushing to the `second` branch on [nechamaLearn/muday](#).

Merge pull request #6 from nechamaLearn/second

Second

This commit will be authored by 150091782+nechamaLearn@users.noreply.github.com

Confirm merge

Cancel

Add a comment

ה PR יסומן כmerged.

## Second #6

Merged nechamaLearn merged 2 commits into `main` from `second` now

## טיפול בקונפליקט

### הפתרון תמיד יהיה לוקאלי ולא ישירות בgithub!!!

לפעמים כאשר אנו עובדים על כמה **branch**ים במקביל ייווצרו קונפליקטים, שינויים שבוצעו בשני **branch**ים ומפריעים אחד לשני, כאשר **git** לא יודע להחליט לבד מי השינוי המועדף. פעמים רבות נגיע לקונפליקט כזה כחלק מתהליך ה**pull request** (**pr**) ולכן התהליך המתואר כאן יכלול גם את העדכון מול **github**.

(אם זה יקרה לוקאלית נדלג על 2 השלבים הראשונים)

**git fetch** - יבוא כל השינויים שקיימים ב**github** ולא נמצאים לוקאלית מלבד השינויים ב**branch** שעליו אני עובדת עכשיו

**git merge origin/main** - מיזוג השינויים מ**main** ל**branch** הנוכחי

טיפול בבעיות של הקונפליקט, בד"כ יהיה הנחיות של ה**IDE**

**git add** - העלאת כל הקבצים שטיפלת בקונפליקטים בהם

**git commit** - ללא הודעה, יש הודעה ברירת מחדל, יפתח חלון **vim** כדי לצאת ממנו צריך **wq**:

**git push** - נשלח את העדכון ל**github**, העדכון יכנס לאותו **pr** שכבר מחכה על ה**branch** הזה.

בד"כ בשלב הזה ה**pr** יתרענן ויתאפשר לי לעשות לו **merge**



## נספח

### טיפול בנפילות:

#### דגשים חשובים כלליים בנושא שגיאות

חשוב מאוד לשים לב אם מתרחשת שגיאה, **לקרוא את התשובה שחוזרת אחרי כל פקודה**, ולוודא שאין בה fatal או error שעצר את הרצת הפקודה.

השגיאות של git בד"כ מפורטות ומסבירות בצורה ממש טובה, דבר ראשון תקראי את השגיאה ותנסו להבין אותה, פעמים רבות git מציעים גם פתרונות מעולים, ואין צורך לחפש בגוגל או בchat AI.

לאחר פתרון השגיאה, ברוב המקרים יש צורך להריץ מחדש את הפקודה שנפלה, המשיך לפקודות הבאות יגרום לתהליך שגוי וחסר, והרבה שגיאות נגררות...

#### שגיאות נפוצות:

- **GIT לא מותקן במחשב**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + v [ ] [ ] ...
PS C:\Users\Administrator\Documents\הארוה\git\testGit> git init
git : The term 'git' is not recognized as the name of a cmdlet, function, script file, or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct and
try again.
At line:1 char:1
+ git init
+ ~~~
+ CategoryInfo          : ObjectNotFound: (git:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException
```

יש להתקין git ואח"כ לסגור ולפתוח מחדש את IDE.

- **אין שם משתמש ומייל**

כל גרסה (commit) שנשמרת בgit חייבת להכיל את פרטי המבצע שלה, והיא תכשל אם לא.

```
PS C:\Users\Administrator\Documents\הארוה\git\new1> git commit -m 'add index.html'
Author identity unknown

*** Please tell me who you are.

Run

git config --global user.email "you@example.com"
git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'Administrator@MININT-4GVBK7H.(none)')
```

יש להריץ את הפקודות המוצעות תחת המילה Run, לא לשכוח להחליף את מה שיש בתוך ה"" בערכים הנכונים עבורך.

- **אין תעודת אבטחה**

יקרה בדרך כלל למי שיש חסימה מסוג נטפרי או אתרוג על המחשב

```
PS C:\Users\Administrator\Documents\הארוה\git> git clone https://github.com/nechamaLearn/new1.git
Cloning into 'new1'...
fatal: unable to access 'https://github.com/nechamaLearn/new1.git/': SSL certificate problem: unable to
get local issuer certificate
```

הוראות מדויקות יש לנטפרי [כאן](#) (שימי לב שלא תמיד הלינק שנטפרי מביאים כdefault  
זו מה שמתאים למחשב שלי, תבדקי אותו לפני ההדבקה בטרמינל)  
אפשר להיעזר בהוראות גם בשביל חברות חסימה אחרות, אבל כמובן שאז יהיה צורך  
להוריד את תעודת האבטחה הנכונה.

- **אין הרשאה לכתיבה בREPO בGITHUB**

לפעמים יהיה לנו אימות ישן של github במחשב, וזה ימנע מאיתנו לבצע פעולות  
שדורשות הרשאות בgithub

כמו push לrepo שאין למשתמש הרשום במחשב הרשאת כתיבה אליו

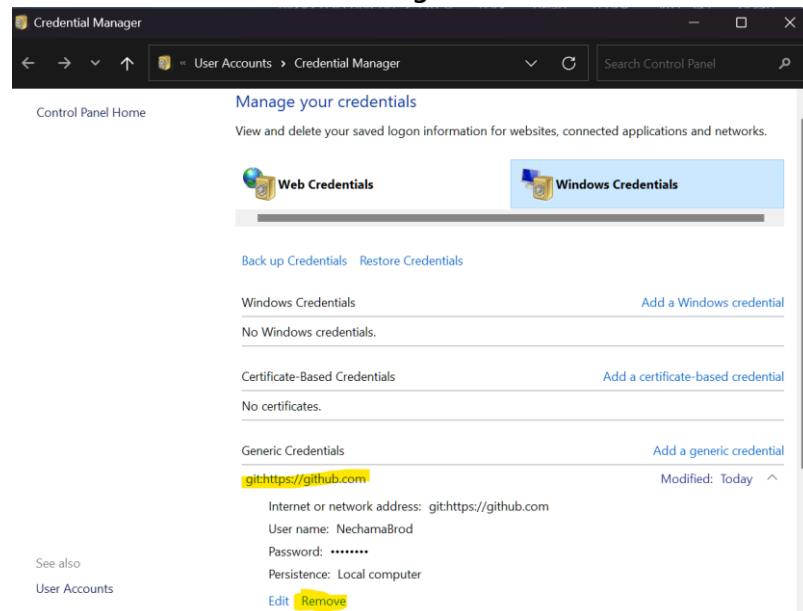
```
PS C:\Users\Administrator\Documents\הארוה\git\friday> git push
remote: Permission to nechamaLearn/friday.git denied to NechamaBrod.
fatal: unable to access 'https://github.com/nechamaLearn/friday.git/': The requested URL returned error: 403
```

או clone לrepo שאין למשתמש הרשום במחשב הרשאת צפייה(private)

```
PS C:\Users\Administrator\Documents\הארוה\react> git clone https://github.com/NechamaBrod/bn.git
Cloning into 'bn'...
remote: Repository not found.
fatal: repository 'https://github.com/NechamaBrod/bn.git/' not found
```

הפתרון יהיה למחוק את המשתמש הרשום ממנהל האישורים במחשב, ולעבור תהליך  
אימות חוזר

נכנס ל"מנהל האישורים" - "אישורי windows"  
ושם נמצא את האישור של github ונסיר אותו.



לפעמים יהיה כמה אישורים שיכללו את המילה 'github', צריך להסיר את כולם.  
אח"כ נריץ שוב את הפקודה המבוקשת ונעבור תהליך אימות מול המשתמש הנכון.

## • אין BRANCH תואם בGITHUB

```
PS C:\Users\Administrator\Documents\הארה\git\friday> git push
fatal: The current branch new has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin new

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

יהיו מקרים שבהם ננסה להעלות branch שעדיין לא קיים בgithub, כמו לדוג' branch חדש שיצרנו לוקאלית או push ראשון לbranch master אחרי remote. ואז תיזרק שגיאה שהוא לא יודע לאן לחבר את הbranch הזה, ישנם שני פתרונות לזה (שניהם מובאים בתוך השגיאה של git):

אומרות לgithub ליצור branch חדש בשם ששלחנו לו, וליצור קישור בינו לבין הbranch שעליו אנחנו עומדות.

הרצה חד פעמית של הגדרה של git, שתגדיר יצירת branch חדש אוטומטית בgithub בשם זהה לbranch המקומי בכל פעם שננסה לעשות פוש לbranch חדש. אחרי הרצת הפקודה יש צורך להריץ מחדש git push.

## קבצים בסיסיים ב-GIT:

### .gitignore

ישנם קבצים שאנחנו מעדיפים שלא לעקוב אחרי השינויים שבהם בgit כמו קבצי לוג שמתעדכנים כל הזמן, או קבצי node\_modules בשפות שעובדות עם npm. או שאנחנו מעדיפים להתעלם מהם בgithub כמו קבצי הגדרות שמותאמים עבור כל מתכנת.

ע"מ למנוע את המעקב אחריהם נוסיף אותם לקובץ .gitignore, כל קובץ או תיקיה בשורה נפרדת, שימי ♡! קבצים שכבר הועלו לgit פעם אחת ימשיכו להיות במעקב, עד שנמחק אותם ידנית מgit וgithub.

הקובץ יראה כך:

```
.gitignore
1  # התעלמות מהתיקיה וכל הקבצים שבתוכה
2  logs
3
4  # התעלמות מכל הקבצים שמסתיימים בסיומת
5  *.log
6
```

### readme.md

קובץ שכתוב בשפת markdown מוצג בעמוד הראשי של הפרויקט, כהסבר על הפרויקט ומדריך שימוש בו.

לדוג' <https://github.com/remy/nodemon>

מצורף קובץ באנגלית עם פירוט נוסף על הקובץ, ו2 דפים עם דוגמאות לכתיבה בשפת markdown, ניתן להתנסות בתצוגה של הכתיבה הזו, גם בלינק באתר שמצורף בקובץ וגם בgithub.

## פקודות ביטול לתהליכים שנלמדו:

ישנן פקודות שמטרתן לבטל שלבים שבוצעו מול git, הן לא נלמדו לעומק במהלך הקורס, אבל חשוב שתכירי את הפקודה כדי לדעת לחפש אחריה במקרה הצורך.

`git restore --staged <file | folder/>` - הסרת הקובץ עם השינויים משלב הstagen.

שימי! ללא flag של `--staged` השינויים ימחקו מהworking directory.

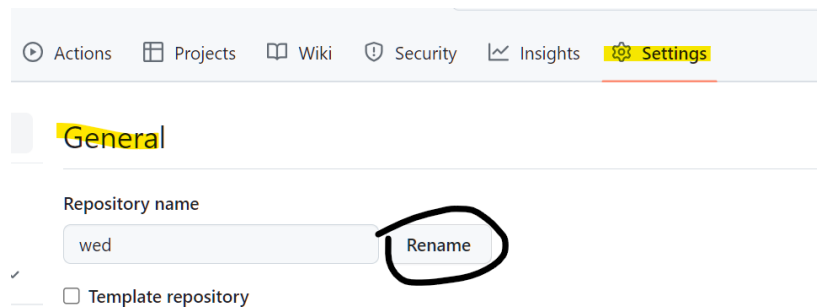
`git revert <commit_uid>` - ביטול השינויים שבוצעו בcommit מסוים, שימי! אם תשמרי את הcommit\_uid תוכלי לגשת אליו אח"כ ולראות מה היו העדכונים שבו.

ע"מ לבטל שינויים שכבר עלו לgithub נשתמש בrevert לוקאלי ואח"כ push שיעדכן את github במחיקת השינויים.

## שינויים בהגדרות בסיסיות של REPO:

בעמוד ההגדרות הראשי בgithub נוכל לבצע משמעותיים על repo לדוג':

שינוי שם הrepo



שינוי הרשאות צפייה (public, private) ומחיקת repo

שינויים אלו נחשבים למסוכנים ולכן הם נמצאים בתחתית העמוד ממש, תחת הקטגוריה danger zone

## Danger Zone

### Change repository visibility

This repository is currently public.

[Change visibility](#)

### Disable branch protection rules

Disable branch protection rules enforcement and APIs

[Disable branch protection rules](#)

### Transfer ownership

Transfer this repository to another user or to an organization where you have the ability to create repositories.

[Transfer](#)

### Archive this repository

Mark this repository as archived and read-only.

[Archive this repository](#)

### Delete this repository

Once you delete a repository, there is no going back. Please be certain.

[Delete this repository](#)

# README AND MARKDOWN LANGUAGE

## Readme

1. Provides an overview of your project: A README file provides a brief overview of your project, including its purpose, features, and any other relevant information. This can help other developers understand what your project does and whether it's relevant to their needs.
2. Helps with installation and setup: A README file can include instructions for installing and setting up your project, including any dependencies that need to be installed. This can make it easier for other developers to get started with your project.
3. Provides usage instructions: A README file can include instructions for using your project, including any command-line arguments or other configuration options. This can help other developers understand how to use your project and get the most out of it.
4. Includes examples: A README file can include examples of how to use your project, including code snippets or screenshots. This can help other developers understand how your project works and how they can use it in their own projects.
5. Helps with collaboration: A README file can include information about how other developers can contribute to your project, including guidelines for submitting pull requests and reporting issues. This can help encourage collaboration and make it easier for other developers to contribute to your project.

Overall, a README file is an important part of any project, as it provides a way to communicate important information about your project to other developers. By taking the time to write a good README file, you can make it easier for other developers to understand and use your project, and encourage collaboration and contributions.

## Rules to good readme file

Here are some general rules to follow when creating a good README file:

- Use clear and concise language: Your README file should be easy to read and understand. Use simple language and avoid technical jargon whenever possible.
- Include a license: Specify the license under which your project is released. This is important for other developers who may want to use or contribute to your project.
- Include a contribution guide: Explain how other developers can contribute to your project, including guidelines for submitting pull requests and reporting issues.
- Include a code of conduct: Specify the code of conduct that contributors are expected to follow when participating in your project.
- Update your README file regularly: Keep your README file up to date as your project evolves, and make sure to include information about any major changes or updates.

By following these rules, you can create a README file that is informative, easy to read, and helpful to other developers who may want to use or contribute to your project.

## Markdown syntax:

website to playground - <https://dillinger.io/>

Note that not all Markdown syntax is supported by all platforms and tools.

### #Headings

#H1

##H2

###H3

####H4

#####H5

#####H6

### #Emphasis

*\*Italic\**

**\*\*Bold\*\***

***\*\*\*Bold and Italic\*\*\****

~~~~Strikethrough~~~~

### #Lists

*\*Item 1*

*\*Item 2*

*\*Item 3*

### ##Ordered

.1Item 1

.2Item 2

.3Item 3

### #Links

[Link text](URL)

[Link with title](URL "title")



## #Images

![Alt text](image URL)

![Alt text with title](image URL "title")

## #Code

`Inline code`

...

Code block

...

## #Blockquotes

<Quote

## #Horizontal Rule

---

## #Tables

|Column 1 | Column 2|

| --- | --- |

|Row 1, Column 1 | Row 1, Column 2|

|Row 2, Column 1 | Row 2, Column 2|

## #Task Lists

-[x] Task 1

[ ] -Task 2

[ ] -Task 3

## #Mentioning People and Teams

@username

@teamname

## #Emoji

:smile: