סדנת GIT



**לפני שמתחילים**

**מטרות הסדנה**

* הסדנה נועדה להכיר לנו את כלי בקרת התצורה

**מבנה הסדנה**

* הכרת הכלי ומושגים בסיסיים
* קונפיגורציה
* יצירת פרויקט
* Committing
* Branches
* Merge Request
* GitLab
* [תרגול gitlab מעשי](https://learngitbranching.js.org/) – לאחר הקריאה

**לפני שמתחילים וודאו שעשיתם את הדברים הבאים:**

* יש לכם משתמש ב - GitLab
  + אם אין – זה הזמן ליצור <https://gitlab.com/users/sign_up>
* Git מותקן לכם במחשב
  + אם לא – זה הזמן להתקין <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
  + אחרי שהתקנתם, תריצו את הפקודה git –version כדי לוודא שההתקנה עברה ותצפו לתוצאה הבאה:



* Git מקונפג לכם במחשב – אם לא, יש להריץ את הפקודות הבאות ב - CMD ולשנות את הפרטים בהתאם לשם משתמש שיצרתם

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

git config --global user.email you@example.com

אז עכשיו כשהכל סגור – בואו נצא לדרך!

**מה זה מערכת בקרת תצורה?**

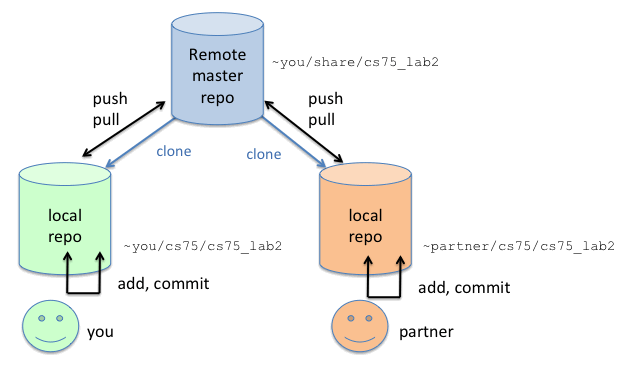
מערכת שעוזרת לנו לנהל שינויים של מסמכים, תוכנות ואתרי רשת גדולים ועוד סוגי מידע רבים אחרים.

**במה היא עוזרת לנו?**

* יכולת להגדיר גרסאות מערכת
* יכולת לחזור לגרסה קודמת
* יכולת לעבוד בצוות במקביל על קבצים משותפים
* יכולת לעקוב אחר שינויים קוד ולבצע סקרי קוד (CR - Code Review)

**מה זה GIT?**

כלי בקרת תצורה שעובד עם תיקייה (Repository) שנמצאת אצלנו על המחשב (Local) ומתקשרת לאותה התיקייה במקום חיצוני (Remote).



**מה זה GITLAB?**

מערכת המבוססת WEB הבנויה מעל GIT, ומאפשרת יכולות ניהול קבצים וגרסאות, ניהול מעקב אחר משימות, הרצת בדיקות ועוד.

GitLab מאפשרת תמיכה בתהליך פיתוח מלא ובצורה חדשנית ואטרקטיבית.

והנה הכתובת - <https://gitlab.com/>

**GIT**

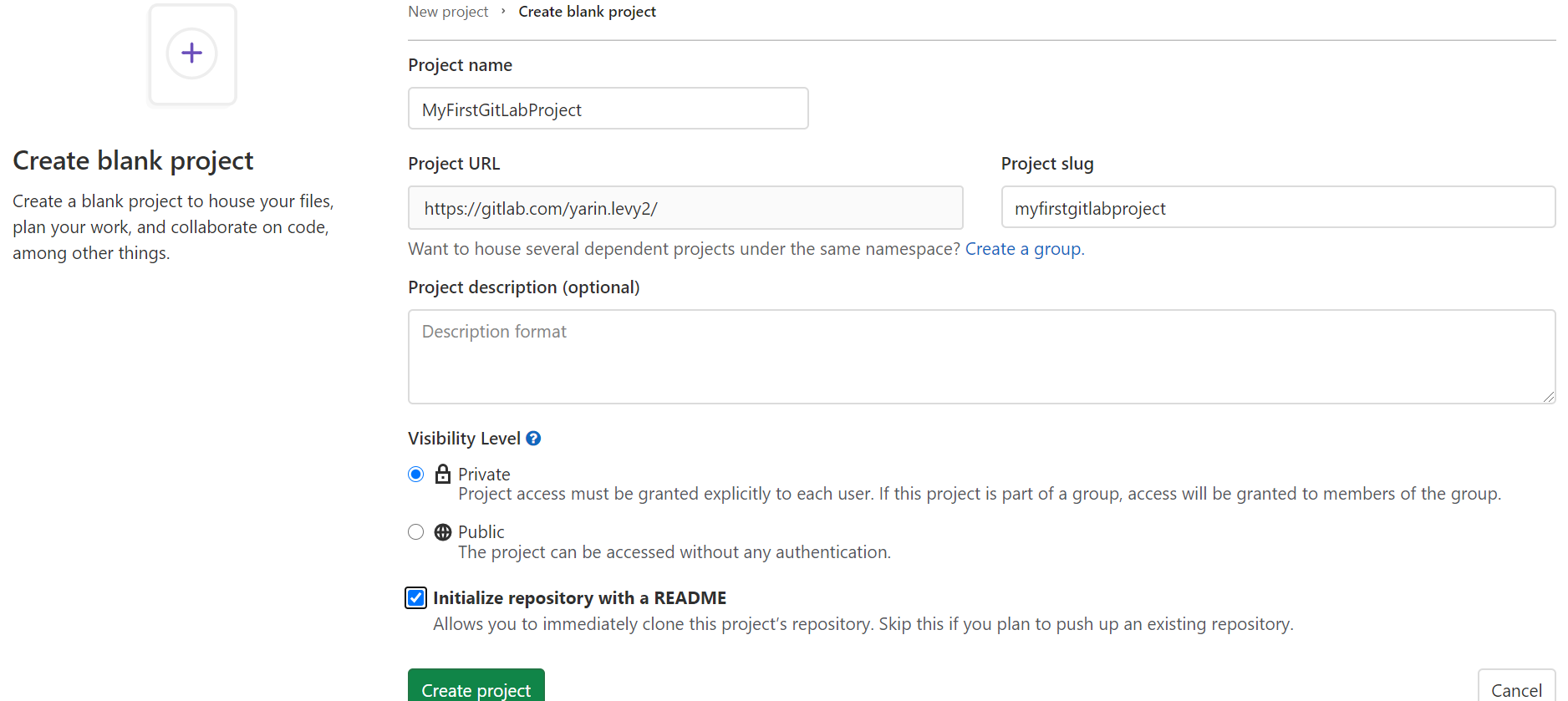


**#0 – פתיחת פרויקט**

כדי שנוכל להתחיל ולעבוד ולתרגל, ניכנס לgitlab ונפתח שם פרויקט חדש, לדוגמה בשם myFirstGitLabProject

<https://gitlab.com/projects/new#blank_project>

נמלא את שם הקובץ ונלחץ create project

****

מה שיצרנו עכשיו זה פרויקט ב-remote-repository. מה זה אומר?

יצרנו פרויקט שמוחזק במקום אחסון מרוחק, כלומר על שרת של GITLAB וכעת נגיש מכל מקום וגם ישמש כמקום לשים פה את התוצר שעובד.

**#1 - שכפול הפרויקט לעבודה מקומית  
 >> git clone**

כאשר נרצה להתחיל לעבוד בפרויקט הGitLab שלנו נצטרך לייבא אותו אלינו מקומית, כלומר לעבוד ב**local-repository**.



איך? פשוט מאוד ->

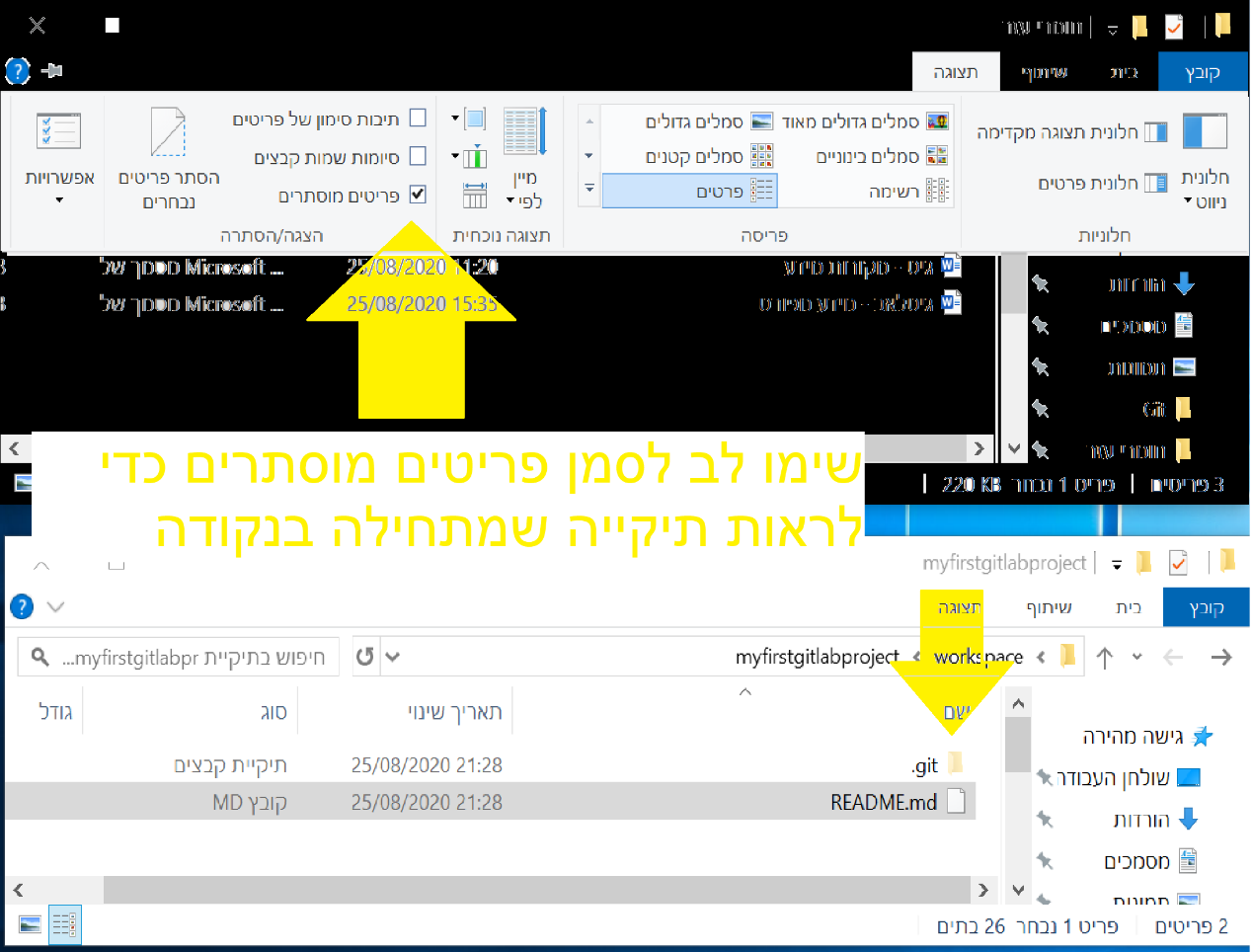
* חזרו לדף הראשי ב-GitLab וכנסו לפרויקט שלכם
* העתיקו את ה-URL של הפרויקט – בחרו בClone with **HTTP** (אפשר גם דרך אחרת, SSH, מוזמנים לחקור)



* פתחו במיקום במחשב שלכם תיקייה ריקה, בה אתם רוצים לעבוד
* פתחו Git Bash ונווטו לתיקייה מהסעיף הקודם.
* איך מנווטים? עם הפקודה >> cd "path"
  + במקום path יש לרשום את הנתיב של התיקייה
* הריצו את הפקודה >> git clone עם הURL שהעתקתם

שימו לב שנוצרה לכם תיקיה בשם הפרויקט, ובתוכה תיקייה בשם **.git** !

(!) אתם לא תראו אותה כיוון שהיא תיקייה מוסתרת, אז איך כן נראה אותה? כפי שמתואר בתמונה הבאה:



התיקייה הזו היא ה-**local repository** שלכם (שם נשמרים ה-**commit**ים, נבין ממש עוד שניה מה זה אומר...)

**#2 – עדכון השינויים מlocal לremote**

כאשר אנו עובדים מקומית, השינויים נשמרים אצלנו, ואנחנו צריכים לעדכן אותם בRemote repository.

כדי שנוכל לבצע את העדכון בשרת החיצוני, נצטרך לעבור כמה שלבים.

השלב הראשון זה להעביר את השינויים ממצב unstaged למצב staged.

מה זה אומר?

נבחין בין שני מצבים בהם יכולים להיות מצויים הקבצים עליהם אנו עובדים:

**UNSTAGED** – קבצים שעבדנו עליהם (הקוד ששינינו) ויש הבדל בין מה שיש לי ב -LOCAL לעומת מה שנמצא ב – REMOTE.

**STAGED** – קבצים שמוכנים להעלאה (כלומר ביצענו להם ADD, עליו מיד נלמד)

איך אני יודע באיזה מצב הקבצים שלי?

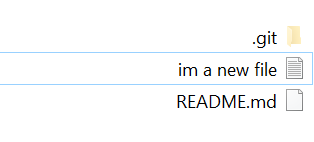
באמצעות הפקודה >>git status

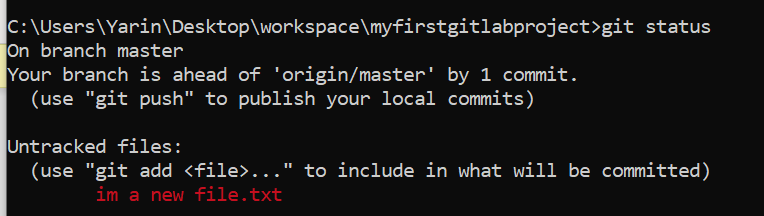
בואו נסביר.

**Status  
>> git status**

בעזרת הפקודה, ניתן לראות אילו קבצים הם STAGED ואילו UNSTAGED וכך להבין מה ייכנס ל – COMMIT עליו עכשיו נלמד**.**

בואו ננסה ונראה את מה שהפקודה נותנת.  
תיצרו בתיקייה שאליה עשיתם clone קובץ חדש.

  
אחרי שיצרתם, תעשו **git status** ותראו מה תראו בcmd.

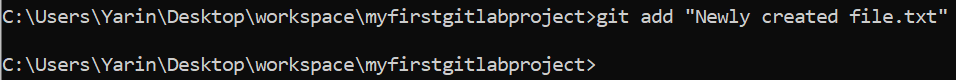


מה אנחנו רואים כאן? שינוי אצלנו מקומי שאנחנו רוצים להעלות לremote repo.

כמו שאמרנו, צריך להעביר את הקבצים ממצב unstaged למצב staged, אז איך עושים את זה?

**ADD**  
**>> GIT ADD**

לאחר שנבצע שינויים בקבצי הפרויקט, בין אם מדובר בלערוך קובץ קיים או להוסיף קובץ חדש, למשל נוסיף קובץ טקסט שקוראים לו Newly created file ונכתוב בו כמה אנחנו אוהבים את בה"ד 1, בה"ד 7 והצבא בכלל, נוכל לעשות להם **add** ובכך להעביר אותם למצב **staged**.



עכשיו נהוג לבצע *git status*  שוב – כדי לראות מה מצב הקבצים כעת.

*תזכורת* – רק קבצים במצב Staged (בירוק) יתווספו ל-**commit** הקרוב.

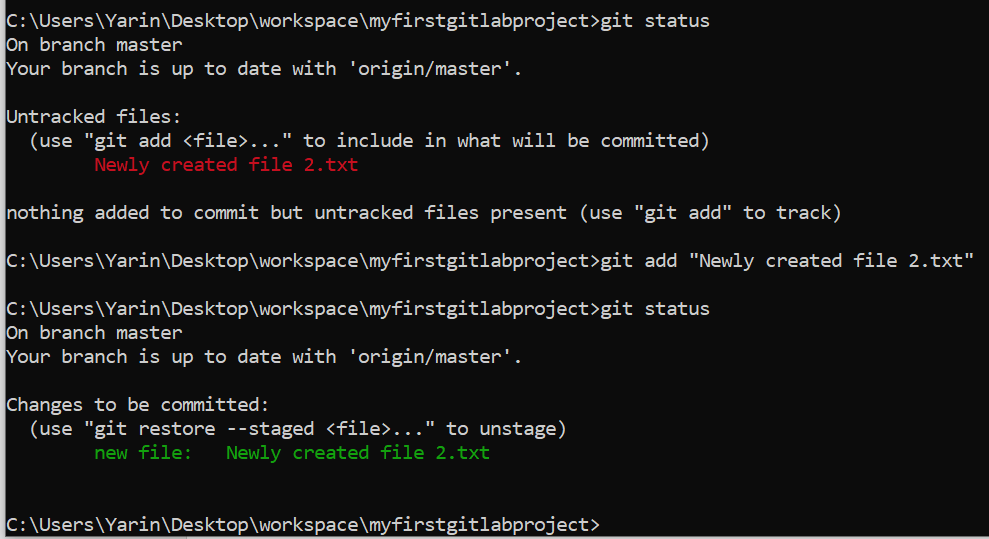
git add *filename1 filename2* …

***טיפ: אפשר לכתוב את האות הראשונה של שם הקובץ וללחוץ TAB.***

***נ.ב:*** במקום לבחור קבצים, ניתן להעביר את **כולם** למצב staged בעזרת הדגל "A"

git add –A

דוגמא לבדיקה באמצעות STATUS מה קורה לקבצים כאשר משתמשים ב – ADD:



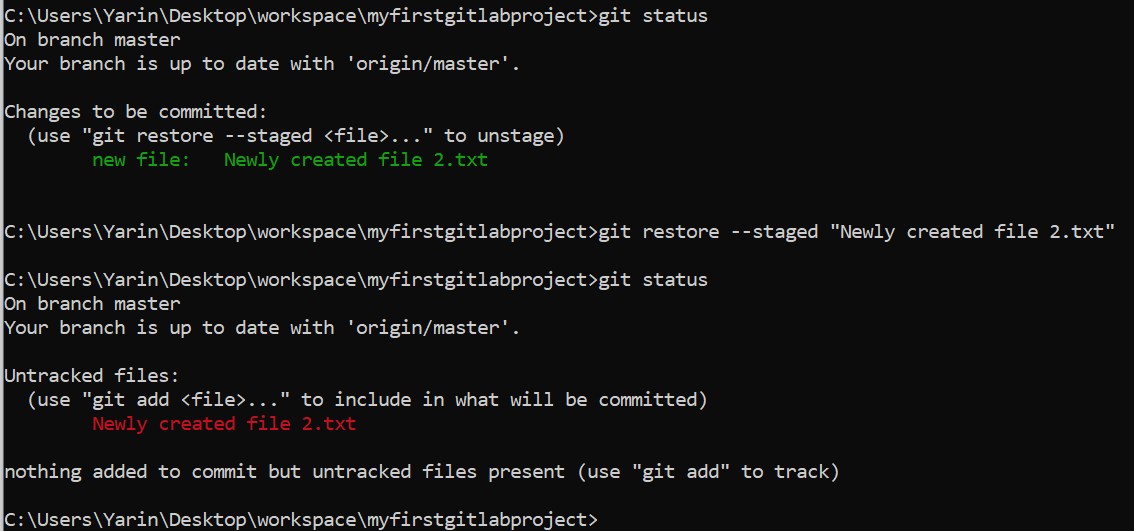
ואם עשיתי add בטעות לקבצים שלא רציתי להעביר למצב staged. יש דרך להציל את עצמנו. באמצעות...

#### Restore

**>> git restore –staged "file\_name"**

הפעולה ההופכית ל – ADD. נשתמש בה לעיתים חריגות כאשר עשינו ADD על קובץ שלא רצינו להוסיף לקומיט. (בקיצור יצאנו פיתה והתחרטנו. או פשוט היינו עצלנים אך יעילים ועשינו git add -A כדי להוסיף את כל הקבצים ואז רצינו להסיר קובץ אחד במקום להקליד את כל מה שאנחנו כן צריכים.)

דוגמא:



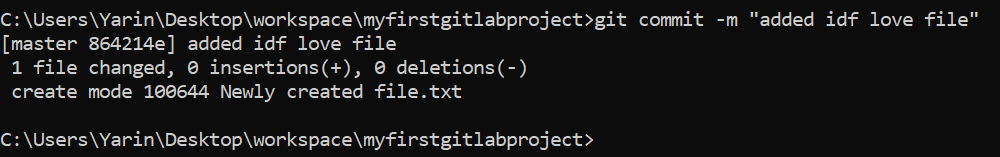
#### Commit >> git commit -m "my\_commit"

לאחר שעשינו **add** לקבצים שעליהם ביצענו את השינויים, נרצה לבצע Commit ובכך ליצור "Checkpoint" לקוד שלי.

**לעולם לא (NEVER EVER GEVER)** אעשה commitלגרסה שלא מתקמפלת, **ותמיד** אצרף ל- **commit**הודעה אינדיקטיבית שמסבירה בקצרה את השינויים בעזת הדגל "m". (את ההודעה **תמיד** נכתוב בפורמט שנלמד, ונמצא בהמשך הסדנה)

git commit –m "*indicative-message*"

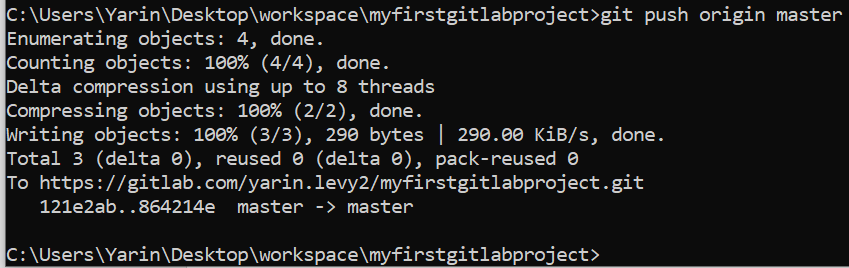
לדוגמא:

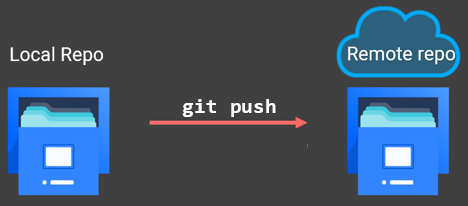


לאחר שעשינו **commit** אחד לפחות, אפשר להעלות את הקוד שלנו לשרת הGitLab- בעזרת הפקודה **push**.

*Git push origin <branch\_name>*

דוגמא להעלאה מוצלחת:



******

עכשיו אתם יכולים לראות את ה- **commit**גם ב-GitLab!

שימו לב! במקרה הזה, לצורך קלות הדוגמא, לא יצרנו branch מקומי חדש ועבדנו ישירות מול ה – master. בעבודה רגילה נעדיף ליצור branch חדש ואת זה נלמד ממש בקרוב.

**#3 – לקיחת שינויים מהremote לlocal**

#### Pull >> git pull [origin] [dest]

פעולת Pull היא למעשה הפעולה ההפוכה של הפעולה Push, בגלל שאתם עדיין לא עובדים כמה אנשים על אותו הפרויקט כנראה שאין לכם מה למשוך מהשרת. בכל מקרה הפעולה נראית ככה:

git pull

אפשר למשוך גם קוד מ - branch אחר, כנראה שאנחנו לא נעשה את זה כי לא נסתמך בקוד שלנו על קוד שעובד על issue אחר. בכל אופן זה יראה ככה:

Branch\_name>> git pull origin

**אמרנו שבצורת עבודה אידיאלית ניצור BRANCH מקומי שייחודי לנו, ממנו נעלה את כל הקומיטים שלנו. מהתיקיה נהוג לפתוח פרויקט חדש (למשל פרויקט JAVA חדש) ולהתחיל לעבוד עליו. מכאן אתם נדרשים ליצור בראנץ' משלכם, לבצע שינויים בפרויקט ולבצע פעולות בהתאם על פי ההוראות:**

**Branch  
>> git branch [branch\_name]**

נרצה לעבוד על קו פיתוח נקי וחדש עבור הפיתוח שלי, נעשה את זה בעזרת הפעולה branch שיוצרת לנו ענף חדש.

Branch\_name>> git branch

שימו לב שזה רק יוצר את הבראנץ' ולא עובר לעבוד עליו.

אם נרצה לראות את כל הבראנצ'ים שקיימים אצלי לוקלית, נבצע את הפעולה:

git branch

זה יוצג לנו כך:



נוכל לראות שהבראנץ' שעליו אני עומד כרגע זוהר בירוק, הוא זה שכרגע אנחנו עובדים עליו וכל פקודות ה – git שנכתוב יתבצעו עליו.

**הערה חשובה:**

הבראנץ' מאסטר יחיל רק דברים שנבדקו ועובדים בוודאות!  
כדי שמספר אנשים יוכלו לעבוד כל אחד על ה – issue שלו, נהוג לפתוח בראנץ' אב ב-REMOTE (בדרך כלל קוראים לו develop) ואליו כל אחד יכול לבצע PUSH מהבראנץ' הלוקאלי שלו.

**הערה חשובה 2.0:**

אין קשר בין השמות של הבראנצ'ים ב - LOCAL לבין הבראנצ'ים ב – REMOTE. זו העדפה שלנו לקרוא להם בשמות דומים או זהים, בדומה למה שאנחנו עושים עם שמות של ארגומנטים ופרמטרים.

אם נרצה להציג את כל הבראנצ'ים שנמצאים גם בremote, נעשה:

git branch -r

**Checkout  
>> git checkout**

כדי לעבור לבראנץ' אחר, נשתמש בפעולה checkout.

Branch\_name>> git checkout

אם נרצה ליצור בראנץ' ולעבור אליו בפקודה אחת, נעשה:

git checkout –b <Branch\_name>

**Merge  
>> git merge**

כאשר סיימתי לעבוד על הבראנץ' שלי ואני ארצה להעביר את התוכן למקום ממנו יצאתי (בראנץ' אב או המאסטר) נרצה לבצע פעולה של מיזוג. נעבור למקום **אליו** נרצה למזג ונבצע:

git merge <Branch\_name>

וודאו שאתם לא יוצרים בטעות קונפליקטים.   
שימו לב שתמיד נעשה merge בעזרתmerge request .

**Fetch  
>> git fetch**

נניח ועכשיו אין אצלי את כל הבראנצ'ים מקומית ואני עדיין רוצה לגשת לבראנץ' או לעשות בו שימוש (נניח ויצרתי בראנץ' דרך הgitlab ולא ידנית) נרצה ליידע את הlocal repo שלנו על אותם הבראנצ'ים.

לשם כך יש פעולה בשם fetch שהיא מיידעת את הlocal repo על בראנצ'ים שקיימים בremote repo.

git fetch

**ביצוע commitים וpushים בהתאם**

מעתה נבצע commitים בהתאם לסטנדרטים! 

#### מבנה ה-commit

הודעה של Commit תראה לנו ככה:



##### **וקצת יותר בפירוט...**

1. **מבנה מסודר** – הודעה צריכה להיות ברורה ומובנת, שיהיה ברור אילו קבצים עיקריים שונו ומה מהות השינוי. שים לב שההודעות צריכות להיות במבנה המוגדר (כמו בדוגמא למעלה)

commitType(file): commit message

1. את הcommit-ים כותבים **באנגלית ברורה** – לא בעברית, לא מילים "מאונגזלות" ובלי הערות מיותרות. NACHON? NACHON!  
   את ה-commit נכתוב תמיד בהווה ולא בעבר! מדוע? כי הcommit- הוא בסופו של דבר "תמונת מצב" - זה משהו קיים.
2. **קישור הcommit לissue שמתאים לו בגיט** – פשוט כותבים '#' ולאחר מכן את מספר ה – ISSUE בגיטלאב. בהמשך נלמד על עבודה עם issues – נכון לעכשיו אתם לא נדרשים לבצע קישור זה.

##### **למה זה דווקא המבנה שנעשה ואפילו יותר פירוט...**

commitType – מסביר לנו את הסוג של ה- commit שלנו, מעשה מה בגדול העלנו? העלו משהו חדש לגמרי למערכת שלנו? העלנו תיקון באג? הסוגים שיש לנו הם:

feat – פיצ'ר, משהו חדש שלא היה קודם לכן בקוד שלנו.

fix – תיקון, העלנו תיקון למשהו בקוד שלנו.

Test – בדיקה, העלנו משהו קטן שבדקנו בקוד.

Doc – מסמך שלא קשור לקוד שלנו.

Chore – מטלה כלשהי שהייתה לנו לעשות, שלא קשורה בהכרח לקוד.

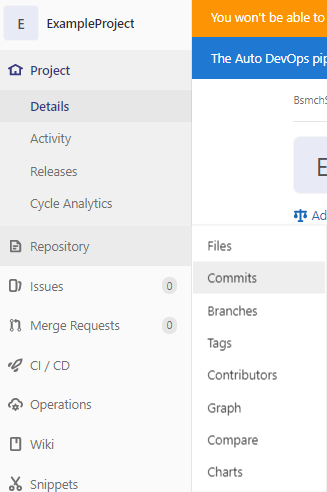
File – הקובץ שעליו עבדנו (אפשר גם את הנושא הכללי עליו עבדנו) למשל בדוגמא עבדנו על הmain.

**שלבי ביצוע commit:**

1. העלאת קבצים למצב staged ממצב unstaged על ידי **git add**
2. ביצוע commit על ידי הפקודה **git commit -m “commit message”** 
   1. שימו לב לביצוע commitים על ידי הסטנדרטים שיש בקובץ **עבודה נכונה עם commit**.
3. ביצוע **git push origin <branchName>** כדי לעדכן את הcommitים בremote repository (וכך אוכל לראות את זה ב - gitlab כמו שמפורט בהמשך).

* אתם יכולים לראות את כל ה – commitים שלכם כאן דרך ה - gitlab

Repository -> Commits ושם יש תצוגה של כל הcommitים שעשיתם (אתם תראו אותם רק אחרי שתעשו git push).



**Merge Request**

חלוקת אנשי צוות ל-merge request.

זאת אומרת, לכל איש צוות יצוות מישהו אחר מהצוות שיהיה אחראי להסתכל על כל בקשת מיזוג שלו ולאשר אותה ולתת הערות בהתאם. זה לא צריך להיות אותו אחד שעושה תמיד את אישור הmerge לחבר הצוות. זה צריך להיות אקראי, אבל קריטי שכל merge request יאושר ע"י חבר צוות אחר.

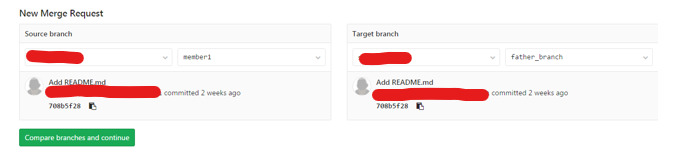
הכוונה היא שבכל פעם שמישהו סיים לכתוב קוד והוא מעוניין להעלות אותו ל-git הוא מחויב לעבור סקר קוד על ידי חבר מהצוות (שמבוצע באמצעות Merge Request). במידה ויש הערות – כותבים אותן בתוך ההערות ב-MR ולא מאשרים את ה-MR עד לתיקונן.

**איך עושים את זה?**

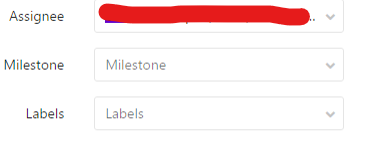
קודם כל צריך לעשות merge request מהענף של השולח לענף שאותו אנחנו רוצים לעדכן







אחרי זה צריך למלא את הפרטים על הmerge לשים את חבר הצוות השני שמאשר את הmerge בתור assignee

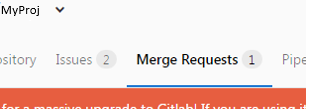


יש ללחוץ לבסוף על submit merge request

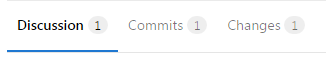
בשלב הזה מאשר הבקשה צריך לאשר אותה ולהעיר הערות על הקוד.

**איך מעירים על הקוד??**

צריך להיכנס לtab של merge requests



נכנסים לאותו merge request שעליך לאשר ומסתכלים למטה – יש שלושה Tabs –

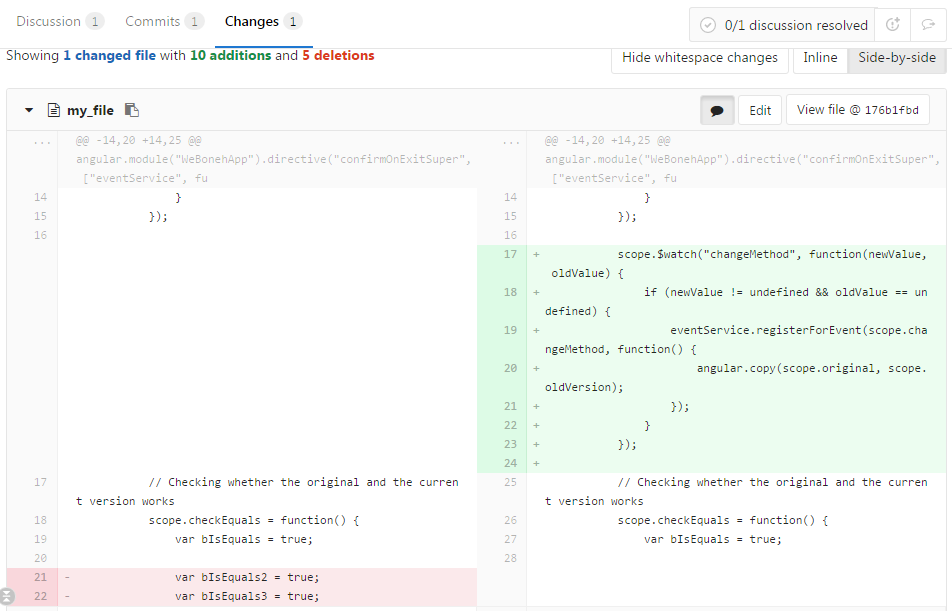


ב-discussions – יש את ההערות שנכתבו על השינויים שנעשו

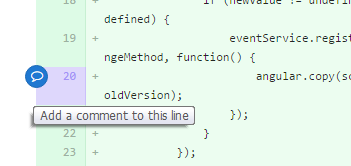
ב-commits – הcommit ים שעליהם נעשה ה-merge request

ב-changes – הדברים ששונו בcommit- ים

לדוגמה דף השינויים נראה כך -



כדי לכתוב ההערות על הקוד, יש לשים את העכבר על הצד השמאלי של השורה:



כשתלחצו על ההערה הכחולה יפתח לכם חלון שבו תוכלו לכתוב את ההערה וכשתלחצו על comment היא תופיע גם בdiscussions. בשלב זה לא תוכלו לעשות merge עד שלא נפתרו כל הבעיות.



לאחר ששני חברי הצוות דיברו על הבעיות בקוד והן תוקנו במידה וצריך, ניתן להגיד שההערה טופלה, ואתה מסמן אותה בתור resolved.



ברגע שתעשו את זה עבור כל ההערות שנכתבו אז תוכלו להשלים את ה-merge. תלחצו על הכפתור הירוק כדי לבצע את ה-merge

כשאתם מעלים לmaster- אתם גם תעשו את זה בצורה הזו עם merge request, ואתם תהיו assignee

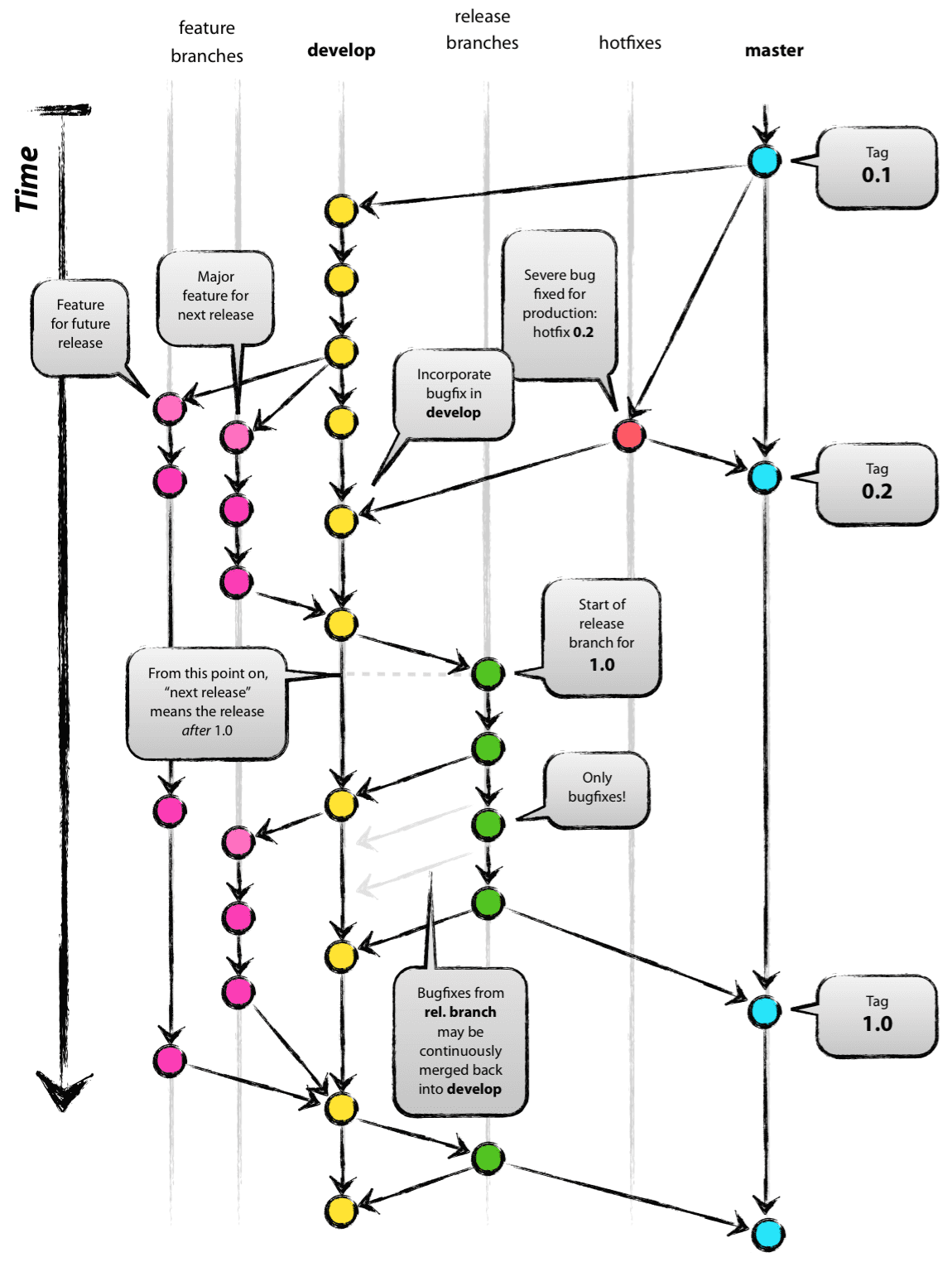
ותאשרו את הmerge.



**GitLab – מה למדנו**

* יצירת משימות
  + Label
  + Milestone
  + Status
  + Estimate
  + Tags
* Branch – לוודא שמבינים את התמונה Best Practice.
  + בראנצ'ים לפיצ'רים/issues
  + מאסטר – לא נוגעים בו – תמיד יהיו בו קוד עובד
  + Develop

**Best Practice**



**מקורות מידע נוספים**

* תרגול GIT - <https://learngitbranching.js.org/>
* פקודות GIT -

<https://www.datacamp.com/community/tutorials/git-push-pull?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=332602034349&utm_targetid=aud-392016246653:dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=1007965&gclid=Cj0KCQjw4f35BRDBARIsAPePBHyV1C2u8DLHHQX3CF33GqcqwuCxzY5MsPeleYOez2tTdh3a1RniYKAaAsfYEALw_wcB>

* מישהו אמר ביטחון מידע? חוץ מיוזר וסיסמא, דרך נוספת להזדהות מול גיט היא באמצעות מפתח ייחודי - כך נוכל לעבוד עם גיט באמצעות מפתח מוצפן SSH (Secure Shell):

<https://docs.gitlab.com/ee/university/training/user_training.html#configure-ssh-key>

* WIP – Next Page

**GIT WIP - WORK IN PROGRESS**

עד היום בפרויקטים שלנו, עבדנו על issue מסוים ולאחר שסיימנו לעבוד עליו, פתחנו Merge request וחיכינו לאישור. היום נלמד פיצ'ר של Gitlab שמשנה לנו את צורת העבודה הזאת, שבו משתמשים ה-מ-ו-ן ביחידות!  הפיצ'ר הזה נקרא WIP – Work In Progress, והוא מאפשר לפתוח Branch **וגם** Merge request בתחילת העבודה, עוד לפני שעשינו אפילו commit אחד.

*מה?! לפתוח Merge request לפני שהתחלנו לעבוד בכלל!?*

כן כן, זה אולי יהיה מוזר בהתחלה כי זה שונה מהדרך שאליה אנחנו רגילים, אבל בהמשך אתם תבינו את ההיגיון מאחורי זה, ואתם לא תאמינו כמה זה יזרז את תהליך הפיתוח בצוות.

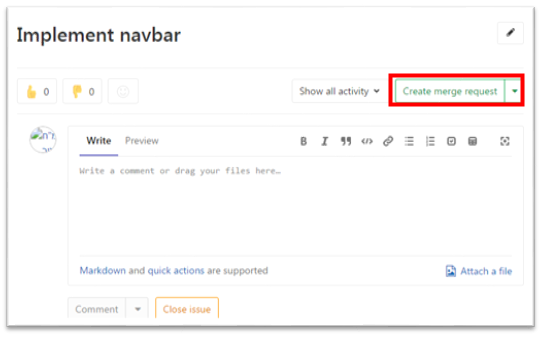
למעשה כשתתרגלו לעבוד ככה, אתם תראו ש:

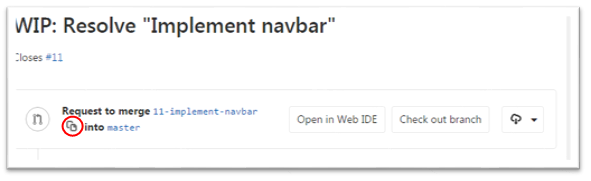
1. הRemote repository -  הרבה יותר מסודר (כל בראנץ' וכל MR מקושר לissue מסוים)
2. הבראנצ'ים שלכם הרבה יותר ברורים ואחידים (שמות שניתנים אוטומטית לפי issue)
3. אתם מתעסקים **פחות** בהתנהלות מול gitlab ו**יותר** בפיתוח
4. לראשי הצוות תהיה בקרה יותר טובה על ההתקדמות של כל issue ושל כל מתכנת (ניתן לקבל תמונת מצב לפי Merge request פתוחים)

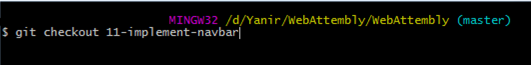
שימו לב שהשיטה הזאת מתאימה עבור צוותים שעובדים branch-per-issue, ז"א שלכל אישיו נפתח branch משלו.

**יצירת (Merge Request)MR מתוך issue מסוים**

ניכנס לissue שנרצה לעבוד עליו. נלחץ על כפתור ה Create merge request(או על החץ כדי לבחור branch אחר) :

****

****כעת יצרנו MR וגם Branch שמקושר לissue  הזה. שימו לב לתגית הWIP בכותרת שמסמנת שהבראנץ' שאנחנו הולכים למזג נמצא בתהליך עבודה(Work In Progress . נלחץ על הכפתור המסומן בתמונה כדי להעתיק את שם ה branch שנוצר :

****כעת נוכל להדביק את השם ל- git bash בעזרת shift + insert:   
ולהתחיל לעבוד על הissue שלקחנו.

(מומלץ להוסיף לתיאור בגיט task list כדי לעקוב אחרי ההתקדמות של העבודה, , קראו על זה בגוגל – GitLab Markdown)

לאחר שעשינו את כל הcommitים וסיימנו לעבוד,  נוכל להצהיר שהMR  שלנו כבר לא Work in progress,  נעשה זאת ע"י לחיצה על הכפתור  Resolve WIP status:   
  
****

הדבר יסיר את תגית הWIP מהבקשה שלנו, וכעת ניתן לבצע את המיזוג. כשנמזג זה יסגור לנו **גם**את הMR, **גם**את הissue **וגם** את הbranch(אם בחרנו כך). זה הכל!

תנסו את השיטה הזאת ותראו אם זה נוח לכם, סיכויים טובים שכן ☺

תרגול GIT מעשי

[תרגול gitlab מעשי](https://learngitbranching.js.org/)