סדנת Git – פתרון בעיות

בגלל שאנחנו מתחילים עם גיט, יש מספר בעיות שעלולות לקרות לנו ולא נדע איך לתקן. בדיוק בשביל זה כתבנו את המסמך הזה – לזרוק לכם גלגל הצלה ולעזור לכם ברגעים הקשים. איזה חמודים מגשימים, איך הם מתחשבים!

דילוג מהיר:

- האתר **המאוד** שימושי – ohshitgit.com - יש בו המון פתרונות לבעיות נפוצות.
- פתרון קונפליקטים במיזוג
- קונפליקטים לא מובנים עם הקבצים של Visual Studio
- בעיות עם קובץ ה-gitignore
- הפרויקט לא נפתח לנו ב-Visual Studio
- העלנו קבצים ל-master בטעות!
- כל השינויים שעשינו נמחקו!

פתרון קונפליקטים במיזוג

לפעמים נעבוד יחד עם בן/בת הזוג על אותו הקובץ, ובדרך כלל גיט ידע לחבר את השינויים שלנו לבד. עם זאת, כשנעבוד ממש על אותה השורה, הוא לא יידע איזה מהשינויים הוא החשוב יותר, וייווצר קונפליקט.

לכן, כשעובדים על פרויקט - חשוב לעשות commit ו-push באופן תדיר, וגם לבצע pull שמעדכן שינויים מה-repo המרוחק, כדי להימנע מקונפליקטים מסובכים כמה שניתן. נשתדל גם לחלק מראש בינינו את הקבצים שעליהם נעבוד כדי שלא ייווצרו שינויים אצל שנינו באותו הקובץ.

אבל בסך הכול, קונפליקטים הם חלק טבעי מהחיים...

1. כשננסה לעשות push נקבל שגיאה שדומה לזאת:

```
To gitlab.com:magshimim/my-project.git
 ! [rejected]        develop -> develop (fetch first)
error: failed to push some refs to 'git@gitlab.com:magshimim/my-project.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

כפי שגיט ממליץ – נשלוף את השינויים מה-Remote Repo שלנו אל המחשב שבו הקוד לא מעודכן בעזרת `git pull origin <branch-name>` , ושוב נקבל שגיאה:

```
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From gitlab.com:Magshimim/my-project
* branch develop -> FETCH_HEAD
a608e7b..41af65d master -> origin/develop
Auto-merging Main.cpp
CONFLICT (content): Merge conflict in Main.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

השגיאה אומרת שיש קונפליקט בקובץ, וגיט לא יודע למזג את השינויים בעצמו.

2. נראה באילו קבצים יש קונפליקט בעזרת הפקודה `git status` .

3. נפתח את הקובץ (או קבצים) עם עורך קוד פשוט (אפשר עם Visual Studio, אבל עדיף עם עורך פשוט יותר כמו Sublime Text או Visual Studio Code – השני מומלץ בגלל שהוא מציג את השורות של הקונפליקט בצורה מאוד נוחה). נחפש את השורות שנראות בערך ככה:

```
(Code not in Conflict)
<<<<<<< HEAD
(first alternative for conflict starts here)
Multiple code lines here
=======
(second alternative for conflict starts here)
Multiple code lines here too
>>>>>>> ffa78a89179a16a2dd6ec05403e80df99b717de4
(Code not in Conflict)
```

סתם שתכירו - אם נפתח את זה ב-**Visual Studio Code** זה ייראה אפילו יותר יפה:

```
20  (Code not in Conflict)
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
21  <<<<<<< HEAD (Current Change)
22  (first alternative for conflict starts here)
23  Multiple code lines here
24  =======
25  (second alternative for conflict starts here)
26  Multiple code lines here too
27  >>>>>>> ffa78a89179a16a2dd6ec05403e80df99b717de4 (Incoming Change)
28  (Code not in Conflict)
```

לכל קונפליקט יש מספר אפשרויות למיזוג:

- השאר רק את הגרסה הנוכחית (העליונה)
- השאר רק את הגרסה החדשה (התחתונה)
- השאר את שתי הגרסאות

הגרסה העליונה – זו הגרסה שנמצאת על המחשב שלי (ב-Local Repo), ו**הגרסה התחתונה** – היא הגרסה שעשיתי לה pull מה-Remote Repo.

4. נבחר את הגרסה שנרצה להשאיר, ונמחק את השורות שגיט יצר בשביל הקונפליקט (השורות עם ה-<<<< וה-=======), ונשמור את הקובץ המתוקן שלנו.

5. נשמור את השינויים שלנו בעזרת **add & commit**, נעשה **push** לשינויים, ושלום על ישראל!

[6. כמובן נעשה `git pull` במחשב השני כדי לוודא שיש אצלו את השינויים המתוקנים.]

קונפליקטים לא מובנים עם הקבצים של Visual Studio בעיות עם קובץ ה-gitignore

שתי הבעיות האלה הן כנראה אותה הבעיה.

ניסיתם לעשות push לקוד שלכם, אבל אתם מקבלים קונפליקטים לא מובנים. ניסיתם לראות באיזה קבצים יש לכם קונפליקט, ונראה שזה קורה בכל מיני קבצים מוזרים כמו עם סיומות מוזרות, כמו exe. , db. , obj. וכו'. **סימן שיש לכם בעיה עם קובץ ה-gitignore!**

כנראה לא עקבתם אחרי ההוראות מספיק בתשומת לב, או שפספסתם איזה משהו ואין לכם מושג מה. טוב, לא נורא – אנחנו לא נעלבים :) בואו נלמד איך לפתור בעיות עם ה-gitignore.

1. בדקו שהשם של הקובץ הוא gitignore. (עם נקודה בהתחלה!)

2. בדקו שהקובץ מתאים להגדרות הפרויקט שלכם (Visual Studio ושפת c++). חפשו באינטרנט קובץ מתאים והשוו אליו אם אתם לא בטוחים.

3. אם הייתה תקלה עם הקובץ – תקנו אותו **ב-master** (אפשר בעזרת עריכה של הקובץ דרך הממשק באתר), ושמרו את השינויים.

אנחנו עורכים אותו ב-master כי אנחנו רוצים שה-master יהיה מעודכן עם קובץ ה-gitignore המתאים, וממנו נמשוך את השינויים לשאר ה-branch-ים.

4. נעבור ל-develop, ונמשוך את השינויים שעשינו ב-master בעזרת הפקודה: `git pull origin master` ונוודא שהקובץ אכן עודכן בתיקייה.

בגלל שכבר עשינו בעבר add & commit לקבצים הלא-רצויים האלה (הקבצים ש-gitignore אמור להתעלם מהם) - הם נשמרו בהיסטוריה של ה-repo שלנו. לכן נרצה למחוק את הקבצים האלה משם.

5. נריץ את הפקודה `git rm -r --cached .` (שימו לב לנקודה בסוף).

הפקודה הזו מוחקת את **כל הקבצים** (זו משמעות הנקודה) בצורה **רקורסיבית** (הדגל r–) **מתוך ה-cache** של גיט (כל הקבצים חוזרים למצב **Untracked**). עכשיו צריך –

6. להוסיף חזרה את כל הקבצים בעזרת `git add .` - שמוסיפה הכול (חוץ מהקבצים ש-gitignore אמר להתעלם).

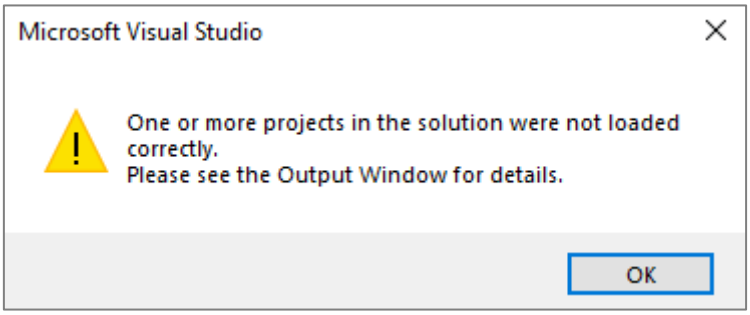
7. שמרו את השינויים בעזרת `git commit -m ".gitignore fix"` , ועשו push ל-**develop**.

8. עשו pull מה-develop **בכל branch שעליו אתם עובדים** כדי לסנכרן את השינויים שעשינו.

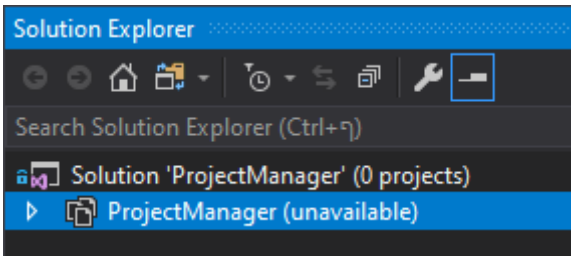
זהו! עכשיו כשננסה לעשות **push** או **pull** לא אמורים להיות לנו קונפליקטים מוזרים.

הפרויקט לא נפתח לנו ב-Visual Studio

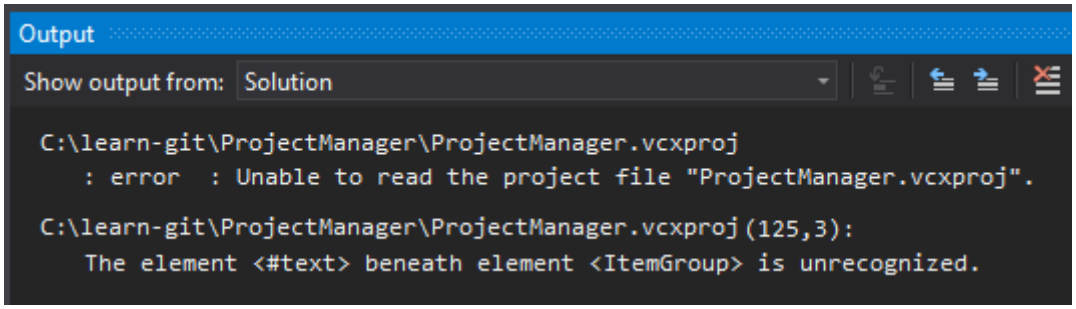
עשינו pull לפרויקט מתוך ה-develop, ופתאום כשאנחנו מנסים לפתוח את הפרויקט עם Visual Studio – אנחנו מקבלים את השגיאה הבאה:



בחלונית בצד זה נראה ככה:



אם נפתח את החלונית Output ונעביר את המצב של Show output from: למצב **Solution**, נראה משהו כזה:



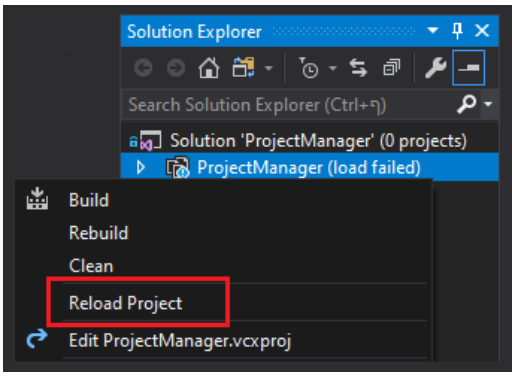
אפשר לראות שיש בעיה עם הקובץ **vcxproj.** שלנו. הקובץ הזה מכיל הגדרות של ה-**Solution** שלנו. לדוגמה, קבצים שטעונים לפרויקט, הגדרות קמפול ועוד. בהודעה השנייה אפשר לראות את השגיאה המדויקת שיש בקובץ.

1. נפתח את הקובץ הזה ע"י הקלקה על השורה השנייה (השורה עם הודעת השגיאה). הקובץ ייפתח ב-Visual Studio.

2. נתקן את השגיאה. נסו להבין איך הקובץ בנוי, או חפשו בגוגל את השגיאה שלכם אם אתם לא מצליחים לבד.

לידע כללי: השגיאה הזו התרחשה בגלל שבמחשבים שונים לפעמים משנים את ההגדרות של הפרויקט, או שמוסיפים קבצים, ואז השינויים נכנסים לקובץ ה-vcxproj של Visual Studio. כשגיט ממזג את השינויים, הוא לפעמים ממזג אותם בצורה לא טובה, ובכך הורס המבנה של הקובץ. לכן במצב הזה – אנחנו צריכים לתקן את הקובץ ידנית.

3. אחרי שתיקנו את הקובץ, נלחץ עם הלחצן הימני על הפרויקט שלנו בחלונית **Solution Explorer**, ונלחץ על **Reload Project**:



4. אחרי שהפרויקט עלה, נשמור את השינויים ע"י **add & commit**, ונעשה להם **push**.

העלנו קבצים ל-master בטעות!

מה שיפה בגיט זה שכל דבר שעשינו אפשר לתקן. זו הסיבה שבגללה אנחנו משתמשים במערכת ניהול גרסאות. אפילו שגיאה נוראית ועצובה כמו לעשות push ל-master (כי ה-master הוא קדוש!) אפשר לתקן יחסית בקלות.

יש 2 מצבים שבהם אנחנו יכולים להיות:

1 – שמרנו קבצים ב-master הלוקאלי שלנו

עבדנו לוקאלית, ופתאום שמנו לב שכל העבודה שלנו קרתה תחת master ולא ב-branch נפרד. מה שאומר שעוד לא עשינו push לשינויים לשרת. הבעיה היא מקומית בלבד.

כדי לפתור אותה (זו תיבת טקסט, אפשר להעתיק):

```
# create a new branch from the current state of master
git branch <some-new-branch-name>

# remove the last commit from the master branch
git reset HEAD~ --hard
git checkout <some-new-branch-name>

# your commit lives in this branch now :)
```

2 – דחפנו את השינויים שלנו ל-master ב-Remote Repo

עבדנו בתוך branch של פיתוח פיצ'ר בשם Feature/cool-one. עשינו שני commit-ים, והרצנו את הפקודה `git push origin master`. פתאום שמנו לב שעשינו push ל-**master** ולא ל-branch שעליו אנחנו עובדים כרגע. נכנסנו לאתר GitLab וראינו שהשינויים שלנו אכן נמצאים ב-master.

אוי לא! מה עושים עכשיו?!

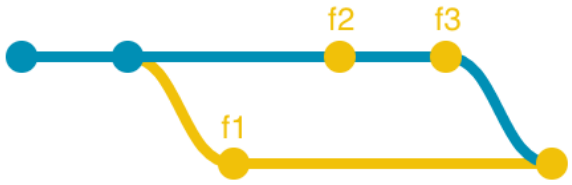
אז פה הפתרון קצת יותר מסובך, אבל נעבור את זה ביחד. כדי לפתור את הבעיה:

נרצה לשמור את השינויים שדחפנו ל-master, לכן -

1. **נעבור ל-branch שבו אנחנו רוצים לשמור את השינויים.**
2. נמזג את master אליו בעזרת הפקודה `git merge master`.

כרגע השינויים שלנו נמצאים גם ב-master, וגם ב-branch שבו רצינו לשמור את השינויים.

ככה ה-repo שלנו נראה כרגע: (העליון – זה ה-master; התחתון – ה-branch שבו אנחנו רוצים לשמור את השינויים)



השינויים שלנו מוזגו מתוך ה-master לתוך ה-branch האחר. עכשיו נרצה למחוק את ה-commit-ים שלנו מה-master.

3. **נעבור ל-master.**
4. נריץ את הפקודה `git revert --no-commit <commit-hash>` על כל commit שדחפנו. במקרה שלנו, עשינו 2 commit-ים, לכן נריץ את הפקודה על f3 ועל f2 (ה-commit-ים שראינו בתרשים למעלה).

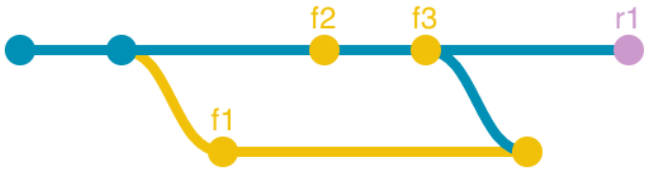
נעשה revert קודם ל-commit האחרון, ואז ללפני אחרון, וכן הלאה:

```
git revert --no--commit <f3-hash>
```

```
git revert --no-commit <f2-hash>
```

5. נשמור את השינויים ע"י `git commit –m "Revert commits"` .

ככה ה-repo שלנו נראה כרגע:



נוצר commit חדש עם השינויים שלנו (בתרשים זה r1). עכשיו אנחנו רוצים למזג את השינויים שעשינו ב-master אל תוך ה-branch השני.

6. **נעבור ל-branch שבו אנחנו רוצים לשמור את השינויים.**
7. נמזג את master אליו בעזרת הפקודה `git merge master` .
8. נריץ שוב revert, והפעם הוא ייצור commit בשבילנו:

```
git revert <r1-hash> -m "Bring back code from commits in master"
```

זהו! תיקנו את הבעיה!

עכשיו נשאר לעשות push לשינויים שלנו אל ה-Remote Repo:

9. מתוך ה-branch השני – נעשה push - `git push origin <other-branch>` .
10. נעבור ל-master – ונעשה push - `git push origin master` .

עכשיו הכול אמור לחזור לקדמותו! איזה כיף!

זה המצב המתוקן של ה-repo שלנו: (f4 זה commit חדש שעשינו ב-branch השני סתם בשביל ההדגמה)

לפחות למדנו לקח חשוב – להריץ git push בתשומת לב גבוהה... ☺

כל השינויים שעשינו נמחקו!

זה קורה לכל אחד. כל מתכנת, גם אחד עם המון ניסיון, יכול בלהט הרגע, ולגמרי בטעות, להריץ פקודה טיפשית כמו `git reset --hard` , ולמחוק לעצמו את כל השינויים האחרונים שהוא עשה בתיקייה.

וזה נוראי. פתאום כל העבודה שלנו נמחקת לחלוטין, וגם אין לנו אפשרות לשחזר את זה. ואז נמחקת לנו עבודה של כמה שעות/ימים/שבועות. באסה רצינית!

אז לא הכול אפשר לשחזר, במיוחד לא פקודה כל כך מסוכנת.

יש 3 מצבים שבהם קובץ יכול להיות:

Unstaged – לא עשיתם לקובץ `git add <file>` - עם כמה שזה עצוב, אי אפשר לשחזר את השינויים. הרי לא אמרתם לעולם לגיט לשמור את השינויים שלכם, אז אין לכם מאיפה לשחזר אותם. ממש באסה, מצטערים ☹

Staged – עשיתם add, אבל עוד לא עשיתם commit לשינויים שלכם. לא תמיד אפשר לשחזר את זה, אבל יכול להיות שבמצב שלכם זה אפשרי. נניח ועשינו add לקובץ, ואז בטעות מחקנו אותו. אפשר לשחזר את זה – ע"י הרצה של `git checkout .` השינויים יחזרו לשינויים שיש במצב ה-Staged. אבל אם הרצתם `git reset --hard`, אין דרך לשחזר את השינויים, כי הרי עוד לא עשיתם להם commit, אז גיט עוד לא שמר אותם סופית (אלא רק באופן זמני).

Committed – עשיתם commit לשינויים, ואחרי זה הרצתם `git reset --hard`. את המצב הזה אפשר לשחזר (דיי בקלות).

כדי לשחזר את השינויים האלה, נלמד על פקודת קסם חדשה שיש בגיט. הפקודה מאפשרת לנו לראות את **כל** הפעולות האחרונות שעשינו. הפקודה היא `git reflog`.

אם נריץ אותה, נוכל לראות גם את הפקודה האחרונה שהרצנו – זו שמחקה לנו את כל השינויים.

```
f733ab6 (HEAD -> master) HEAD@{0}: reset: moving to HEAD@{1}
7c36a54 HEAD@{1}: commit: test2
f733ab6 (HEAD -> master) HEAD@{2}: checkout: moving from
         f733ab6166792f3b49f171b486ff30d36f786dc5 to master
f733ab6 (HEAD -> master) HEAD@{3}: checkout: moving from master to HEAD@{1}
f733ab6 (HEAD -> master) HEAD@{4}: reset: moving to HEAD
f733ab6 (HEAD -> master) HEAD@{5}: commit (initial): Initial Commit
```

כנראה שאצלכם הפלט יהיה דיי שונה, וכנראה גם הרבה יותר ארוך. מה שמעניין זו הפקודה האחרונה שהרצנו (שנמצאת בראש הרשימה).

אנחנו רוצים לחזור אחורה לשלב שלפני ההרצה של ה-reset. כדי לחזור שלב אחד אחורה – נשתמש בפקודה checkout - `git checkout HEAD@{1}`.

מה שעשינו עכשיו הוא דבר דיי מורכב, אנחנו מתעסקים עם ה-**HEAD** של גיט. זה נושא מתקדם, ולא למדנו עליו. אם אתם מסתבכים – אתם יותר ממוזמנים לקרוא על git head באינטרנט, כדי להבין קצת יותר לעומק מה אנחנו עושים פה.

אם נסתכל עכשיו בתיקייה – השינויים שלנו אמורים לחזור! (בתקווה שהכול עבד כמו שצריך). יכול להיות שנרצה לחזור יותר אחורה (ולא רק ל-HEAD@{1} אלא קצת יותר אחורה) – אז פשוט עשו checkout ל-HEAD@{<>} המתאים.

עכשיו תעתיקו את הקבצים שלכם לתיקייה חיצונית, ליתר ביטחון. אם נהרוס משהו – לפחות הקבצים שלנו עדיין יישמרו בצד.

נחזור חזרה ל-branch שבו היינו לפני ה-checkout, ונדביק את הקבצים שנמחקו לתיקייה. נעשה להם add & commit, ומזל טוב! שחזרנו את הקבצים!

למדנו לקח חשוב – לחשוב טוב טוב לפני שמריצים פעולות קריטיות כמו reset --hard!