

תרגול Git בסיסי

מפגש 4/5 - חזרה על שנה א'
כותבות התרגיל: יולי לוי וטלי לכיש



התקנות

1. אם עדיין אין לכם - פתחו עכשיו משתמש ב-GitHub.
2. [הורדת Git](#) [כנסו לקישור](#) ומלאו את הפרטים בשביל להירשם. הירשמו עם [המייל של ניצנים](#).
3. [כנסו לקישור](#). [לחצו](#) על הטאב הראשון משמאל: **Windows**, ואז על: [Click here to download](#).



3. קובץ ההתקנה **Git-2.51.1-64-bit.exe** יירד למחשב שלכם (או הגרסה האחרונה עבור Windows).



4. [לחצו](#) על הקובץ כשהורדתו מסתיימת, [ועקבו](#) אחר הוראות ההתקנה.
- שימו** ❤️ - המחשב שקיבלתם מ"ניצנים" מתאים להתקנת גרסת Windows 64 bit. אם אתם עובדים בנוסף בבית על מחשב אחר שמוותקנת עליו מערכת הפעלה אחרת, למשל macOS של Apple - בחרו במחשב זה בהורדת הגרסה המתאימה לו.

תזכורת

- [Pull](#) - "מושך" (מוריד) את הגרסה המעודכנת ביותר מה-Repository המרוחק (שנמצא למשל ב-GitHub) אל המחשב המקומי.

- **Add** – מוסיף קבצים ל-Staging area, שהוא אזור-ביניים שבו בוחרים אילו שינויים ייכנסו ל-Commit הבא.
- **Commit** – שומר "צילום מצב" של הקוד ב-Repository המקומי, יחד עם הודעת תיאור קצרה (message) שמסבירה מה השתנה.
- **Push** – מעלה את כל ה-Commits שבוצעו מהמחשב המקומי אל ה-Repository המרוחק בענן.

קובץ .gitignore – הקובץ הזה מגדיר ל-Git מאילו קבצים להתעלם - כך שלא יהיו במעקב ולא יועלו ל-GitHub. **למה צריך אותו?** יש קבצים שלא קשורים ישירות לפרויקט עצמו ולכן לא נרצה שיעלו ל-GitHub – למשל קבצי הגדרות אישיות של PyCharm (קבצי .idea), קבצים זמניים ש-PyCharm יוצר בעצמו (__pycache__ / *.pyc), או קבצי מערכת. באמצעות .gitignore. אנו מונעים מקבצים מיותרים או אישיים להישמר בהיסטוריית השינויים בקוד או לעלות ל-Repository המשותף, כך שהוא נשאר נקי ומסודר ורק הקוד החשוב משותף לכל הצוות.

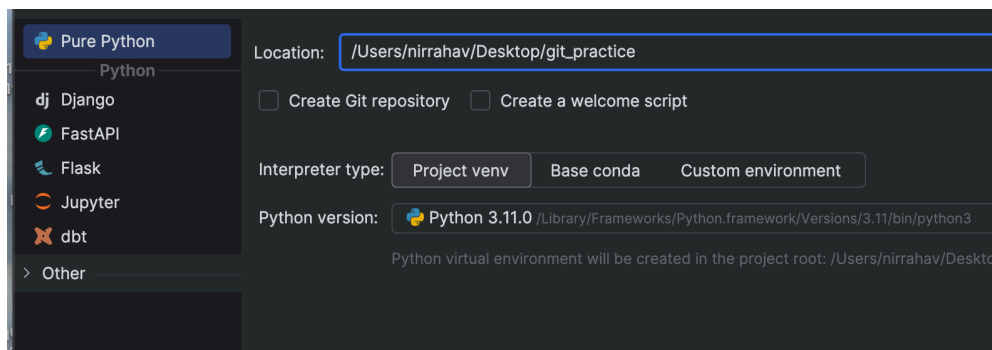
לרשותכם נספח **Git Cheat Sheet** ביחידה 00 | **תומכי למידה** בקודו.



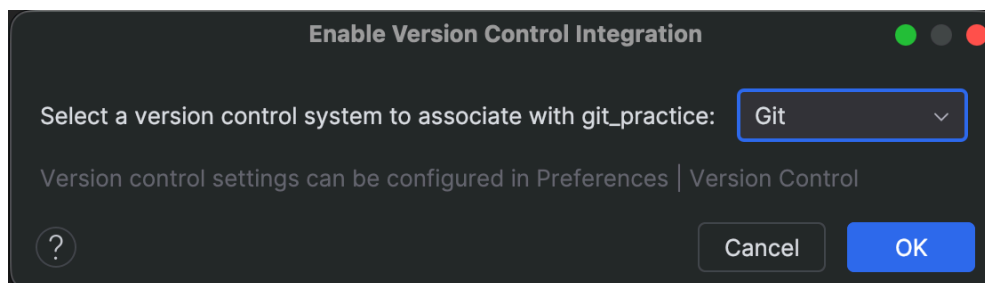
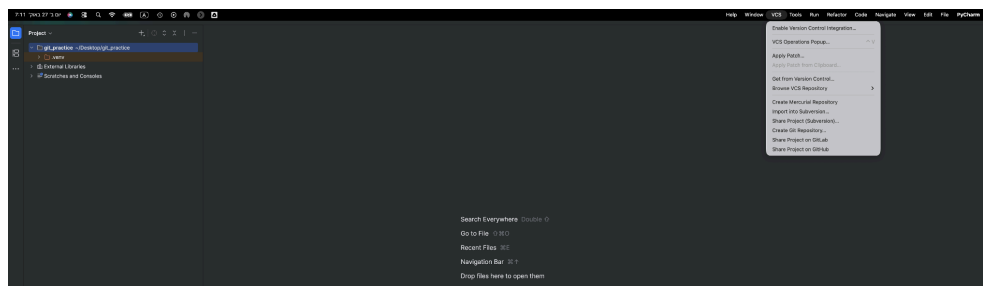
שימו ❤️ - כל השלבים מופיעים **במצגת המפגש**, אתם מוזמנים להיעזר בה.

הפרויקט הראשון שלכם ב-GitHub (:

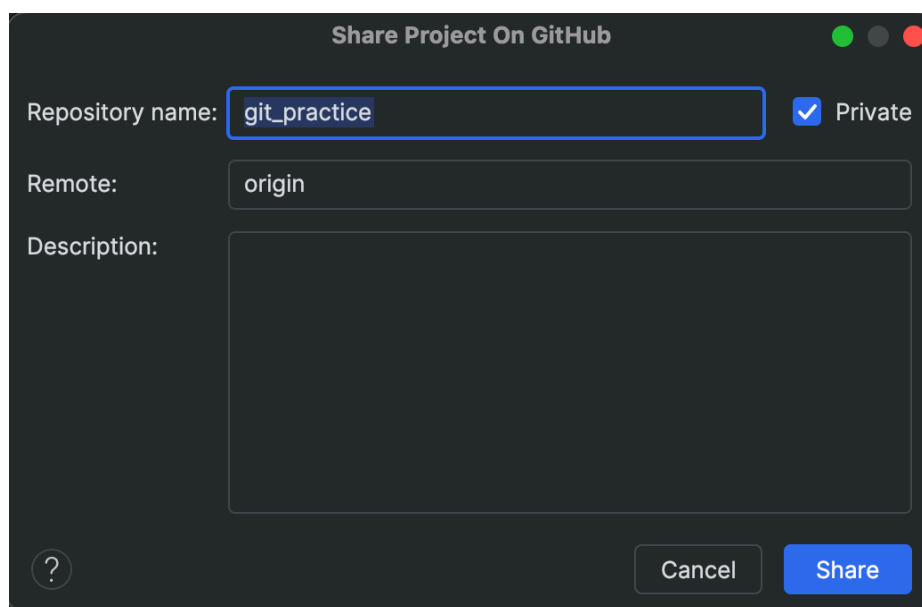
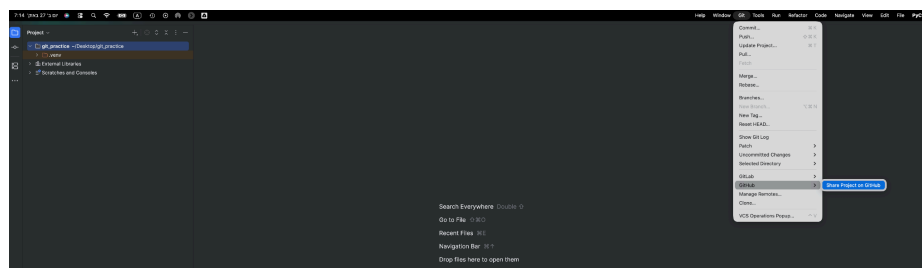
1. **פתחו** פרויקט חדש ב-PyCharm בשם `git_practice`.



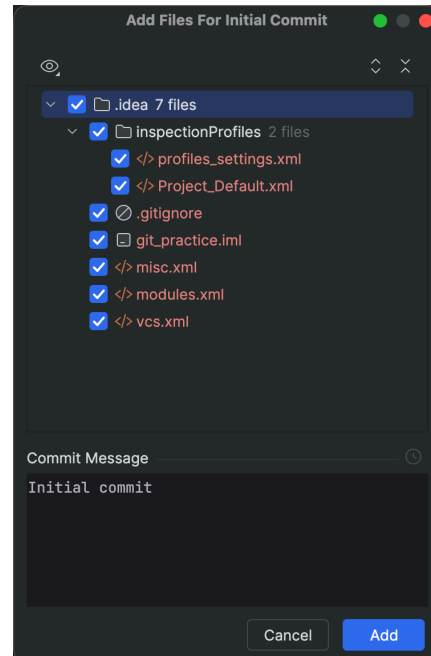
2. אתחול Git בפרויקט (Git → Enable Version Control Integration → VCS).



3. חברו את ה-Repository המקומי ל-Repo המרוחק שלכם ב-GitHub.



4. בחלונית שנפתחה לחצו על **Cancel**.



5. **צרו** קובץ חדש בשם **main.py**, עם הפקודה:

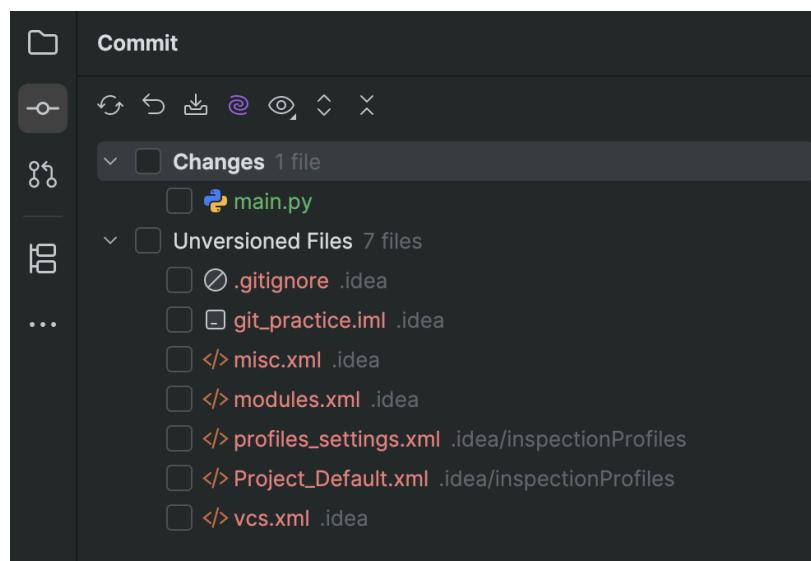
```
print("Hello Git")
```

6. **הריצו** את **main.py**.

7. **פתחו** את ה-**Commit tool window**. שם תראו את האזורים:

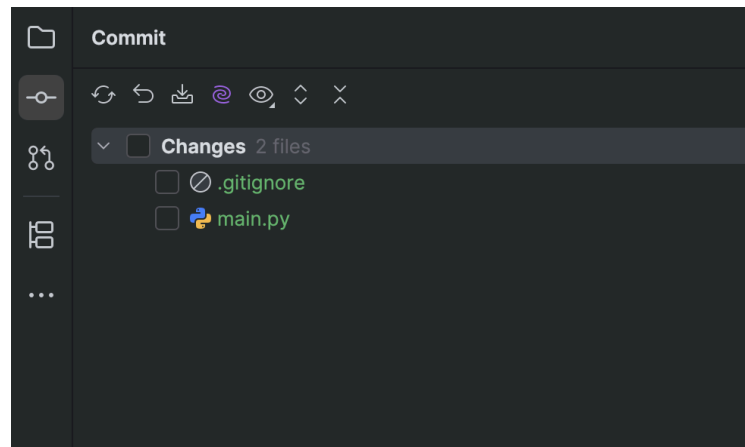
a. Changes עם הקובץ **main.py**.

b. Unversioned files עם הקבצים של תיקיית **idea** (ביניהם גם **.gitignore**).



8. **הוסיפו** את הקובץ `main.py` **בלבד** ל-Staging area בעזרת **Add** (כלומר **סמנו** אותו ב-V)
שימו - אין באמת לחצן Add בממשק של PyCharm, אלא נסמן ב-V ב-**Commit tool window** את הקבצים אותם נרצה לעדכן ב-Commit הבא.
9. בצעו **Commit and Push** ראשון ל-GitHub.
שימו - **לפני** שאתם מבצעים Commit and Push - **כתבו** הודעת Commit מתאימה!
(למשל: `Add main.py with "hello" print`)
10. **כנסו** לאתר GitHub וודאו שאכן נוצר Repo מרוחק.
11. שימו לב שב-Repo ב-GitHub נמצא עכשיו רק `main.py`.
12. **לחצו** על `File → New → File`, צרו קובץ חדש בשם `.gitignore`.
13. **כתבו** בו את השורות הבאות:

`__pycache__`
`pyc.*`
`/idea.`
14. **פתחו** שוב את את ה-**Commit tool window**. עכשיו רואים שרק החלק של Changes מופיע:



15. **סמנו** ב-V את `.gitignore`, ובצעו **Commit and Push**.
זכרו לכתוב הודעת Commit (למשל: `Added .ignore file`) **לפני** ביצוע Commit and Push.
 16. **היכנסו** ל-GitHub ובדקו ב-Repo שמופיעים רק הקבצים: `main.py` ו-`.gitignore`.
- ? שאלת מחשבה:** למה חשוב להוסיף קובץ `.gitignore`. כבר בתחילת הפרויקט?

עדכון הפרויקט - עבודה שוטפת

1. **ערכו** את הקובץ כך שידפיס בנוסף להודעה הקודמת הודעה שמספרת על משהו מגניב שעשיתם בחופש.
2. בצעו **Push → Commit → Add** נוסף עם הודעת Commit מתאימה.
3. **בדקו** ב-GitHub שהשינויים מופיעים בהיסטוריית ה-Commits.

דגשים



- עשיתם Push ל-GitHub **ממחשב אחר?** כשתחזרו לעבוד על הפרויקט מהמחשב שלכם, טרם תמשיכו לעבוד - אל תשכחו לבצע **Pull** כדי "למשוך" אליכם למחשב מה-GitHub Repo את גרסת הפרויקט האחרונה הכי עדכנית שעבדתם עליה!
 - הקפידו על **Commit and Push** בסיום כל שינוי.
 - וודאו ב-GitHub שהשינויים מופיעים לאחר ה-Push.
 - בצעו Commit עם **הודעות** ברורות ותמציתיות (פירוט נוסף במצגת ובל"ע שקיבלתם לקראת שיעור זה).
 - הימנעו מהעלאת קבצים מיותרים (כגון `__pycache__`, `.iml`, `__idea/`) באמצעות קובץ `.gitignore`.
 - שימו לב לצבעי שמות הקבצים ב-PyCharm - צבע הקובץ נותן לכם אינדיקציה לגבי הסטטוס שלו (**ירוק** - חדש, **כחול** - שונה, **אפור** - לא מנוהל ע"י Git (ignored)).
- ? שאלה:** מה יהיה הצבע של קובץ שנמחק? מוזמנים לעיין [בטבלת צבעי הסטטוס בקישור](#).

בונוס



חקירה עצמאית - README.md file

- לכל פרויקט טוב יש קובץ **README** שמסביר על הפרויקט ומתעד אותו.
- הוא הדבר הראשון שמפתח אחר יראה כשייכנס ל-Repository שלך ב-GitHub.
- המטרה שלו היא:
- להסביר בקצרה מה הפרויקט עושה.
 - לתת הוראות הרצה והתקנה.
 - לשתף מידע חשוב (תלויות, קרדיט, דוגמאות שימוש).
- 💡 **כלל אצבע:** דמיינו שחבר שלא היה איתכם בשיעור רוצה להריץ את הפרויקט שלכם - כל מה שהוא צריך לדעת צריך להיות כתוב ב-README.

דוגמה:

לפני שתכתבו README בעצמכם, היכנסו [לפרויקט Snake ב-GitHub](#) והסתכלו על קובץ ה-README שלו.

אל תקראו הכל לעומק, רק תציצו כדי להבין איך נראה README אמיתי.

איך כותבים קבצי README?

כדי ש־README יהיה קריא וברור, אנחנו משתמשים בשפת עיצוב פשוטה שנקראת **Markdown**.
 Markdown מאפשרת להוסיף כותרות, רשימות, קוד, קישורים ועוד (כפי שראיתם בדוגמה של Snake) בצורה מאוד קלה
 לכתיבה.

לפניכם טבלה עם חוקי העיצוב הבסיסיים ב־Markdown (כמובן שיש עוד).

What you want to do	Markdown syntax
Main heading	Heading #
Subheading	Heading ##
Bold text	**bold**
Italic text	<i>*italic*</i>
Numbered list	Item .1
Bulleted list	Item -
Inline code	`("print("hello`

מצורף קישור ל-[Markdown Cheat Sheet](#) למידע נוסף (:

כתיבת קובץ README לפרויקט שלכם

צרו קובץ חדש בשם **README.md** בתוך הפרויקט, בעזרת טבלת חוקי העיצוב.

אבני דרך לכתיבה - על הקובץ להכיל:

- כותרת ראשית - שם הפרויקט שלכם.
- תיאור קצר - מה הפרויקט עושה, תוך שימוש בהדגשה (Bold) לפחות פעם אחת.
- דוגמת קוד - הוסיפו שורת קוד אמיתית מתוך הפרויקט שלכם.
- הוספת רשימה (Bulleted list) - עם לפחות שני פריטים על מה אפשר למצוא בפרויקט.