

## לי"ע Git מתקדם

לקראת פרויקט Arcade

כותב: ניר רהב



שלום לכולם!

במהלך הסדנה הקודמת הכרתם את הפקודות הבסיסיות של Git - יצירת שינויים (Add), שמירה שלהם (Commit), ושליחה לענן (Push / Pull).

עד עכשיו עבדתם בעיקר לבד על פרויקטים אישיים, אבל בעולם האמיתי - רוב העבודה נעשית בצוותים. בדיוק בשביל זה נועדה הסדנה הזו.

Git הוא לא רק כלי לשמירה על גרסאות - הוא כלי המאפשר תיעוד של הפרויקט, שיתוף פעולה חכם ועבודה במקביל של כמה מפתחים, מעקב אחר שינויים בפרויקט ובקרה על הקוד שלו, גיבוי לפרויקט וחזרה לגרסאות קודמות במידת הצורך.

במהלך הסדנה הקרובה תלמדו פקודות מתקדמות שיעזרו לכם לעבוד בצורה מקצועית יותר, להבין איך להתמודד עם טעויות, ולשלוט טוב יותר בפרויקטים שלכם.

### Clone

תהליך של יצירת עותק מקומי במחשב של repository אחר שנמצא ב-GitHub. כלומר, אנו יוצרים עותק מקומי מלא של הפרויקט שנמצא באותו repo כולל כל הקבצים, היסטוריה והענפים (נלמד זאת בהמשך) של אותו ה-repository. לאחר שעשינו את העותק המקומי, ניתן לעבוד על הפרויקט במחשב האישי, לבצע שינויים ולשמור אותם באמצעות Commits. כיצד נבצע Clone לתוך המחשב שלנו?

נפתח את האתר של GitHub וניגש ל-repository שעליו נרצה לבצע את הפעולה. כעת קיימות שתי אופציות:





1. מדובר ב-repository, חדש ולכן אין קבצים.

2. קיימים קבצים ב-repository.

נתחיל עם אופציה מספר אחת.

בחלונית של Quick setup נעתיק את הקישור המופיע - על ידי לחיצה על כפתור העתקה.

**Quick setup — if you've done this kind of thing before**

 Set up in Desktop
 or
  HTTPS
  SSH
 


Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

<> Code

במידה וקיימים קבצים ב-repository, והוא אינו ריק - נלחץ על

**git-practice** (Public)

Pin Watch 0 Fork 0 Star 0


main 1 Branch 0 Tags

Go to file Add file <> Code

nirrahav first commit 1bd087b · now 1 Commit

main.py first commit now

README

  
**Add a README**  
Help people interested in this repository understand your project by adding a README.  
[Add a README](#)

**About**  
No description, website, or topics provided.



**Activity**  
0 stars  
0 watching  
0 forks

**Releases**  
No releases published  
[Create a new release](#)


**Packages**  
No packages published  
[Publish your first package](#)

נעתיק את הקישור שקיים שם על ידי לחיצה על **כפתור ההעתקה**.


Local Codespaces


 Clone 

HTTPS SSH GitHub CLI



Clone using the web URL.

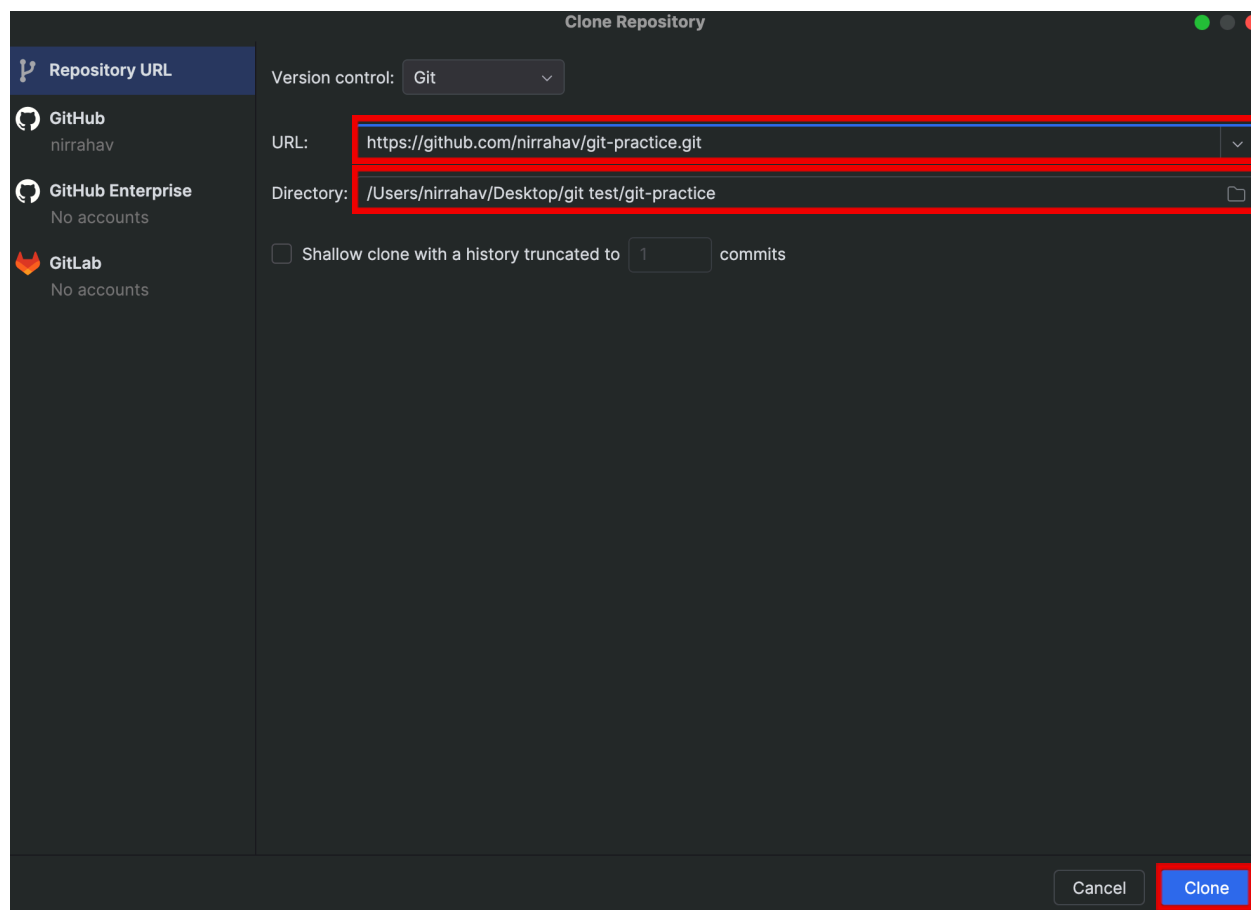
 Open with GitHub Desktop

 Download ZIP

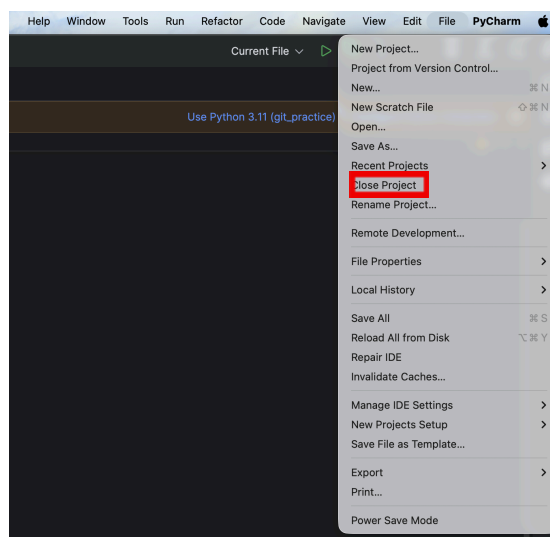
נפתח את PyCharm במחשב, ונלחץ על **Clone Repository** בעמוד פתיחה של PyCharm.

New Notebook New Script New Project Open **Clone Repository**

בחלון שייפתח - הדביקו את הקישור שקיבלתם, ובחרו את מיקום התיקייה שבה תרצו לשמור את הפרויקט, ולאחר מכן לחצו על **Clone** בתחתית המסך מצד ימין.



לאחר שלוחצים על Clone, סביבת הפיתוח PyCharm תוריד למחשב שלכם את הקבצים הרלוונטיים, תיצור repo מקומי?, ויפתח לכם את הפרויקט במיקום שבחרתם.  
שימו ❤️ - בהנחה ונפתח פרויקט PyCharm אין מה להיכנס ללחץ, לחצו File -> Close Project.



## קובץ requirements.txt

מה זה requirements.txt, ולמה בכלל צריך את זה?

כשעובדים על פרויקט python, כמעט תמיד משתמשים בספריות חיצוניות למשל (flask, pandas, numpy). לכל מחשב יכולות להיות גרסאות שונות של אותן ספריות. ללא תיאום - הקוד יעבוד אצלך, אבל ייפול אצל חבר/ה לצוות או במחשב בית-הספר.

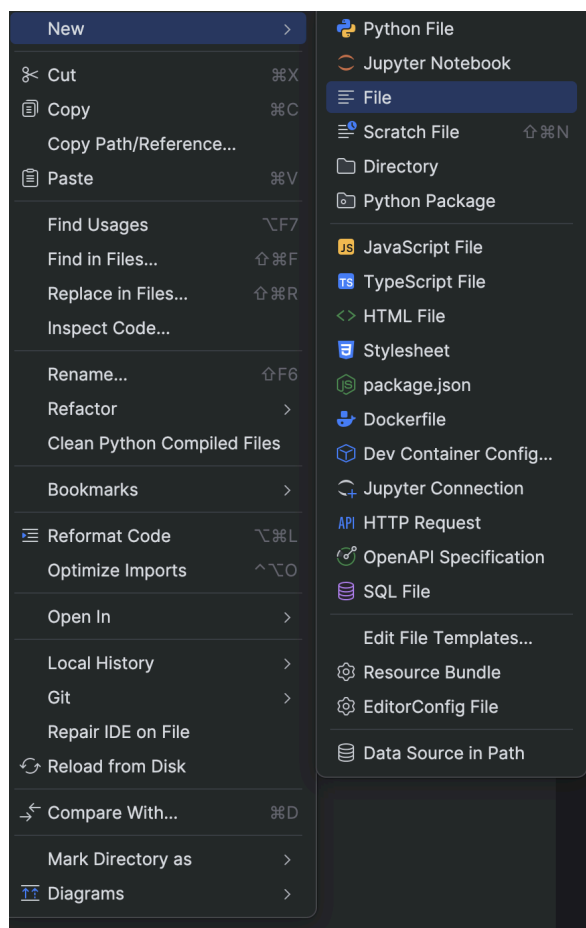
requirements.txt הוא מעין "רשימת קניות" של הפרויקט: הוא אומר למחשב בדיוק אילו ספריות צריך, ואילו גרסאות.

## זה פותר 3 בעיות אמיתיות:

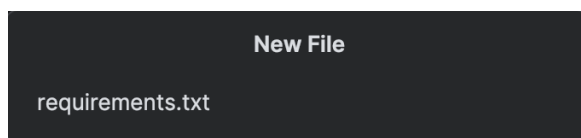
1. שחזוריות: כל אחד שמוריד את הפרויקט יכול להתקין אותו סט חבילות, ואז הקוד יתנהג אותו דבר.
2. שיתוף צוותי: כולם עובדים עם אותה סביבת עבודה ← פחות "אצלי זה לא עובד".
3. Deploy ובדיקות: בעזרת GitHub אפשר להגדיר תהליכים אוטומטיים שבודקים את הקוד, ומעלים אותו לשרת אמיתי באינטרנט לשימוש רחב.

## אז איך עושים את זה?

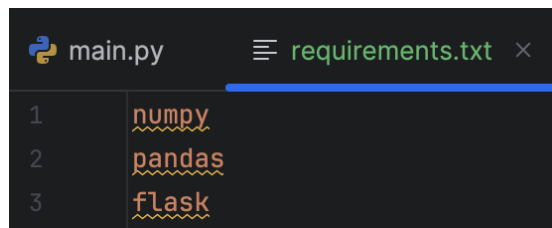
פותחים את הפרויקט ב-PyCharm, לוחצים מקש ימני בעכבר File -> New.



כעת נפתח קובץ חדש בשם requirements.txt (פתיחת הקובץ זהה לפתיחת קובץ python חדש).



כעת לקובץ החדש נוסיף את כל הספריות אשר נעבוד איתם בפרויקט שלנו.



**חשוב:** אז איך חבר הצוות שלנו מתקין את כל הספריות הנדרשות?

פשוט מאוד, נפתח את הטרימינל ב-PyCharm ונכתוב את הפקודה הבאה:

```
pip install -r requirements.txt
```

```
Terminal Local x + v
(base) nirrahav@macbookpro git-practice % pip install -r requirements.txt
```

✓ וקעת PyCharm יתקין לנו את כל הספריות הרלוונטיות!

## משימה

הקוד הנתון בקישור לגיט, משתמש בספריית requests כדי לשלוח בקשה לאתר שמחזיר בדיחה רנדומלית. אם הכל עובד כמו שצריך – תראו בדיחה שמורכבת משורה ראשונה (שאלה או פתיח) ושורת פאנץ' אחרונה.

💡 טיפ: ספריית requests - ספרייה ששולחת בקשות לאינטרנט ומקבלת מידע מאתרים.

מטרת הקוד היא לא הלוגיקה עצמה, אלא לתרגל הורדה של פרויקט אמיתי, התקנת תלויות, והרצה בסביבה מקומית.

1. בצעו Clone ל-repository.
2. התקינו את החבילות הנדרשות עם `pip install -r requirements.txt`.
3. הריצו את הקוד ובדקו שהוא מציג בדיחה מהאינטרנט.

ליחצו כאן על [הקישור לגיט](#)

## דגשים

### Clone ≠ הורדה רגילה

- פעולה של clone לא רק מורידה קבצים, אלא **מחברת** את המחשב שלכם אל הפרויקט שבענן (repository).
- מהרגע שעשיתם Clone – אתם עובדים על עותק חי שמסנכרן עם GitHub.

- Clone הוא חיבור מתמשך המאפשר עדכונים דו-כיווניים בניגוד להורדת zip שזהו צילום רגעי של הקוד.

### גרסה מקומית מול גרסה בענן (GitHub)

- לאחר Clone, נוצר לכם **עותק מקומי** של הפרויקט על המחשב.
- אתם יכולים לשנות קבצים, לשמור (Commit) ולבדוק את השינויים – אבל הם נשארים אצלכם מקומית.
- רק לאחר Push השינויים נשלחים ל-GitHub, ורק לאחר Pull תקבלו את העדכונים מהצוות שתעבדו איתו.



### עבודה מסונכרנת עם Git

המטרה שלנו בעבודה עם Git היא **להישאר מסונכרנים** – כלומר שכל הצוות יעבוד על אותו קוד מעודכן.

כדי שזה יקרה, אנחנו משתמשים בשתי פקודות עיקריות:

1. git pull – אותה כבר למדנו בסדנת Git הבסיסית.
2. git fetch – אותה נלמד עכשיו.

### מה בעצם קורה כשאנחנו מסתנכרנים?

יש לנו שני מאגרים:

- **Repository מקומי (Clone)** – זה הקוד שעליו אנחנו עובדים במחשב שלנו.
  - **Repository מרוחק (GitHub)** – זה המאגר הראשי של הצוות בענן.
- אנחנו רוצים ששני המאגרים האלה יהיו **באותו מצב**, ושנדע בכל רגע מה השתנה אצל אחרים.

#### Pull – תזכורת

פקודה זו מביאה את כל השינויים מה-repository המרוחק וממזגת אותם אוטומטית לתוך הקוד המקומי שלנו. במילים אחרות – גם מעדכנת, וגם משלבת את השינויים ישר לתוך הקבצים אצלנו.

#### Fetch

git fetch גם מביאה את השינויים מה-repository המרוחק, אבל, לא ממזגת אותם מיד אל תוך הקוד שלנו. היא רק מאפשרת לנו לראות מה השתנה אצל אחרים לפני שאנחנו מחליטים לשלב את זה בעצמנו. זה נותן למפתח שליטה מלאה על תהליך העדכון – מתי ואיך למזג את הקוד החדש.

לסיכום:

- git pull – עושה שתי פעולות במכה אחת (fetch + merge).



- `git fetch` – רק מביא את השינויים, בלי לגעת בקוד המקומי.

### **לקריאה נוספת:**

אפשר להעמיק ולקרוא עוד על הפקודה באתר הרשמי של Git

 <https://git-scm.com/docs/git-fetch>