test@

כאשר @test כתוב מעל מתודה, והיא מקבלת את האופציה שיהיה אפשר להריץ אותה כtest case.
כלומר ניתן לכתוב test בטרמינל ובכך לבדוק אם כל מתודה שמסומנת אם @test לפניה אם עובדת באופן ראוי.

Branch

ב-Git, ענף (branch) הוא הפניה להיסטוריה נפרדת של הקוד. הענפים מאפשרים לעבוד על תכונות חדשות, תיקוני באגים או שינויים אחרים מבלי להשפיע על הקוד הראשי. הוא מאפשר עבודה בין כמה מפתחים במקביל ומסייע בניהול הגרסאות

---

cat

פקודת טרמינל בlinux שמטרתה להצגת תוכן של קבצים. יכול לקרוא מקובץ אחד או יותר, ומדפיס את התוכן שלהם למסך.

---

cd

להיכנס לתקייה דרך הטרמינל

---

chmod

chmod (קיצור של **change mode**) היא פקודה שמאפשרת **לשנות הרשאות לקובץ או תיקייה**– כלומר, מי יכול **לקרוא (r)**, **לכתוב (w)** או **להריץ (x)** את הקובץ.
יש שלושה סוגי משתמשים :

- **Owner** – המשתמש שיצר את הקובץ.
- **Group** – קבוצת משתמשים ששייכת לקובץ.
- **Others** – כל שאר המשתמשים במערכת.
  - `rwx` = `7` (קריאה + כתיבה + הרצה)
  - `r-x` = `5` (קריאה + הרצה)
  - `0` = ללא הרשאות

---

clone

clone-זה תהליך שבו נוצרת העתקה מלאה של מאגר אל מחשב(GitHub) מהשרת המקומי המשתמש,כולל כל הקבצים תיקיות היסטוריית הגרסאות וסניפים.
בפשטות שכפול מלא של פרויקט מהענן למחשב שלך

---

docker

docker - פלטפורמה להקמה, הפצה והרצה של יישומים בתוך מכולות (Containers), כולל כל התלויות (ספריות, כלי מערכת, הגדרות רשת ומטמון).
מאפשרת "אריזת" היישום עם כל הסביבה שלו והרצתו עקבית ובידודית על כל מחשב או שרת, בלי צורך בהתקנות ידניות.

Echo

בשביל להדפיס למסך נשים את הפקודה Echo ואז נכתוב מה שנרצה שיודפס למסך.
למשל: echo "Hello world" .

---

gedit

פותח קבצי טקסט (קוד,סקריפטים וכו...) כדי שתוכל לקרוא או לערוך אותם, כמו פנקס רשימות (Notepad).

Git

Git הוא כלי לניהול גרסאות. המטרה שלו היא לעזור לנו לעקוב אחרי כל שינוי שאנחנו עושים בקוד, לעבוד על כמה גרסאות במקביל, ולחזור אחורה אם צריך

Git - add

פקודת ה-Add משמשת להוספת שינויים בקבצים לפני ביצוע ה commit
git Add כאשר משנים קובץ בפרויקט, גיט מזהה את השינוי, אך הוא עדיין לא מתועד. הפקודה מעבירה את השינוי לאזור ההכנה (staging) כך שיהיה מוכן לשמירה, אך הוא עדיין לא נשמר בפועל במאגר.

---

git push

משתמשים בפעולה git push כדי להעלות שינויים שביצעתי בrepository המקומי בסביבת הפיתוח בה השתמשתי (inteljj לדוגמה) אל הrepository המרוחק , לדוגמה: GitHub.

---

git replace

הפקודה הזו מאפשרת **להחליף אובייקטים ב-Git באופן זמני**, כלומר להגיד ל-Git: "במקום להשתמש באובייקט הזה (כמו commit, blob, וכו'), תשתמש בזה אחר".

לדוגמה, אפשר להחליף commit בהיסטוריה מבלי לשנות את ההיסטוריה עצמה בפועל — שימושי ל-debug או לניתוח קוד מורכב.

הפקודה אינה שכיחה כל כך מכיוון ש:

- היא מאוד מסוכנת אם לא יודעים מה עושים.
- באופן זמני Git היא לא משנה את ההיסטוריה בפועל, אלא רק את ההתנהגות של.
- לרוב המפתחים אין צורך בזה ביומיום.

מתי כן נוכל להיעזר בה:

- תיקון היסטוריה לצורך בדיקה (מבלי לעשות `rebase`).
- עבודה עם `git filter-branch` או `git fast-export/import`.
- אנליזה של שינויים עמוקים באובייקטים פנימיים.

Git show

מציג מידע מפורט על אובייקט Git כמו commits, branches.

GitHub

היא פלטפורמת מפתחים קניינית המאפשרת למפתחים ליצור, לאחסן, לנהל ולשתף את הקוד שלהם

Gradle

הוא כלי אוטומציה לבנות המשמש להידור, בדיקה וניהול של תלות בפרויקטים של Java, Kotlin ו-Groovy, תוכנה, במיוחד בשפות כמו.
ומציע גמישות גבוהה Kotlin או Groovy - הוא משתמש בקבצי תצורה שנכתבו ב בהגדרת תהליכי בנייה מותאמים אישית.
נמצא בשימוש נרחב בפיתוח אנדרואיד ונחשב תחליף מודרני לכלים ישנים Gradle יותר כמו Ant ו-Maven.

HTTP

הוא פרוטוקול תקשורת המשמש להעברת מידע בין דפדפן אינטרנט (כמו Chrome) לשרת אינטרנט.
בקיצור:
HTTP היא השפה שבה משתמשים דפדפנים ושרתים כדי לשלוח ולקבל דפי אינטרנט, תמונות, סרטונים ועוד.
לדוגמה - כאשר אתה מבקר באתר, הדפדפן שלך שולח בקשת HTTP לשרת, אשר לאחר מכן מגיב עם תוכן העמוד.

---

if ב-Bash

לבצע (true), משמש כדי לבדוק תנאי – ואם הוא נכון `if`, (שפת סקריפטים בלינוקס) Bash־ב
פעולה מסוימת.

```
if [ condition ]; then
commands to run if the condition is true
fi
```

JUnit

הוא framework לבניית בדיקות לקוד Java.
נשתמש ב-JUnit כאשר נרצה לבדוק קוד מול מספר מקרים אפשריים שאנחנו לא מעוניינים שיקרו (לדוגמה, עבור קוד שמבצע חילוק ניתן פרמטר שמחלק ב-0 כדי לוודא את תקינות הקוד), או כאשר נבנה על קוד ישן יותר, אשר אמור להתאים לאותם מקרי קצה, ובנוסף עבור אוטומציה ונוחות.
ב-Java, נבנה מחלקת Test אשר תשתמש ב-JUnit, ולפעולות הבדיקה נוסיף אנוטציה של @Test על מנת לסמן כי אלו פעולות בדיקה של JUnit.
לאחר מכן, נשתמש ב-Assertions (ראו https://docs.junit.org/5.0.1/api/org/junit/jupiter/api/Assertions.html) על מנת להוסיף את הבדיקות הרצויות, כך שה-Assert יזרוק שגיאה אם הקוד לא עבר בדיקה, ו-JUnit יציג לנו את התוצאה של הבדיקות.
נצרף כעת דוגמה לפונקצית בדיקה עבור פונקציה isPalindrome:

```
import org.junit.Test;
import static org.junit.Assert.*;

public class AppTest {
   @Test
   public void testIsPalindrome() {
      assertFalse(App.isPalindrome("abc"));
      assertTrue(App.isPalindrome(""));
      assertTrue(App.isPalindrome("TACOCAT"));
      assertFalse(App.isPalindrome("TEST"));
   }
}
```

---

ls

הסבר:
מציגה למשתמש את כל האברים הנמצאים התקייה הנוכחית terminal פקודה ב.
שונות ls קיימים פקדוות:

1. ls -r: מראה את האיברים מסדר הפוך (r-reverse)

2. ls -a: מראה את כל האיברים(כולל את המוסתרים/בעלי נקודה בתחילת שמם) הנמצאים בתקייה (a-all)

3. ls -l: מראה מידע מפורט הרבה יותר על האיברים המוצגים(כמו:שם המשתמש, סוג איבר, גודלו, (l-long) (כמות הקישורים הקשחים, זמן השימוש האחרון ועוד

<u>מבנה ls -l:</u>

1) (שם בקובץ, ברשאות במשתמש, הרשאות הקבוצה והרשאות אחרים(בסדר הזה
2) קישורים(כמות הקישורים הקשים
3) המשתמש
4) הקבוצה
5)גודל הקובץ(בביטים)
6) זמן השינוי האחרון(שנה ושעה)
7) שם בתקיה\הקובץ

<u>דוגמה:</u>

/drwxr-xr-x 1 user group 0 May 28 10:00 folder

--(7)----------(6)-------(5)-----(4)---(3)--(2)-----(1)

הרשאות וקיצד לשנות אותם:

r - לקרוא

w - לכתוב

x - להפעיל

במבנה כזה ש:

```
rwxr-xr-x
```

owner-read write and execute

group-read and execute

other- execute

<u>שינוי הרשאות:</u>

<u>דרך א':</u>

chmod [u(user)/g(group)/o(others)/a(all)] + [r(read)/w(write)/e(execute)] file name

דוגמה: chmod u +w file.txt(נותן למשתמש(user) לכתוב)(write) בקובץ file.txt

<u>דרך ב':</u>

chmod xxx (כאשר כל מספר מציג את השאות של כל קבוצה בסדר בהם הם מוצגים(משתמש, קבוצה, אחרים) בדרך לש ביתים!)

לדוגמה: chmod 755 - נותן rwx-rxr-x (משתמש כל ההרשאות, קבוצה + אחרים יכול לקרוא ולהפעיל)

דגש: r =4, w=2, x=1 חיבור וחיסור של כל אחד יוסף ויוריד הרשאות לכל מספר

Man - manual

נועד להצגת מדריך עזרה (תיעוד) לפקודות לינוקס/יוניקס. זהו בעצם ספר ההוראות המובנה של מערכת ההפעלה, שמסביר איך להשתמש בפקודות ובכלים.

מבנה המסמך שמוצג:

- **NAME** – שם הפקודה ותיאור קצר.
- **SYNOPSIS** – איך להשתמש בפקודה (תחביר).
- **DESCRIPTION** – פירוט מה הפקודה עושה.
- **OPTIONS** – פירוט כל הדגלים/פרמטרים שניתן להשתמש בהם.
- **EXAMPLES** – לפעמים יש דוגמאות שימוש.

man [שם_הפקודה] - לדוגמא: man ls"ls" יציג את המדריך המלא לפקודה

---

marco and polo

- **מטרת הפונקציות:**
- `marco`: שומרת את הנתיב של התיקייה הנוכחית (current working directory).
- `polo`: מחזירה את המשתמש לתיקייה שנשמרה קודם על ידי `marco`.

שימוש בפונקציות האלה מאפשר ניווט יעיל במערכת קבצים מורכבת, ללא צורך לרשום שוב ושוב נתיבים ארוכים

דוגמא למימוש בBash:

```
marco(){
# שמירה על הנתיב במשתנה    "export MARCO_DIR="$PWD
}
polo() {
#חזרה לנתיב השמור   " cd "$MARCO_DIR
}
```

---

mkdir

הפקודה **make directory** (**mkdir**) היא פקודה ליצירת תיקיה חדשה בטרמינל

MongoDB

הוא מסד הנתונים מוביל ב- NoSQL הפותח על ידי אליוט הורוביץ וגרסתו הראשונה שוחררה ב-2009.

- הוא מסוג Document-Oriented כלומר המידע בתוכו נשמר באמצעות key-value אבל הערכים במסמך (בדרך כלל JSON או XML).

- הוא מסד נתונים מקומי כלומר מאוחסן במחשב אך ניתן ליצור גם מסד בענן באמצעות MongoDB Atlas

---

nano

פקודה זו פותחת את הקובץ שצוין בעורך הטקסט הננו. אם הקובץ לא קיים, ננו תיצור אותו עבורך.
אופן השימוש הוא בחלון הCMD כדלהלן:
[nano [filename

---

pwd

pwd - Print Working Directory
פקודת Bash שמדפיסה את הנתיב (path) מתיקיית השורש (root directory) לתיקייה הנוכחית.

Repository

- **מחסן של הקוד וההיסטוריה שלו.**

יכול להיות מקומי או מרוחק.

נוצר ע"י git init.

Terminal

הטרמינל הוא ממשק שורת פקודה (CLI - Command Line Interface), שמאפשר למשתמש להפעיל פקודות ישירות מול מערכת ההפעלה, במקום להשתמש בממשק גרפי (GUI).
באמצעות הטרמינל ניתן ליצור ולמחוק קבצים ותיקיות, להריץ קוד, לנווט בין תיקיות, להשתמש לניהול גרסאות, להתקין תוכנות, להריץ סקריפטים, לנהל הרשאות, להתחבר לשרתים Gitב־מרוחקים, ולעקוב אחר תהליכים שרצים במחשב. הוא חיוני לעבודה בסביבות פיתוח מתקדמות, לאוטומציה של משימות ולשליטה מלאה על סביבת העבודה בצורה מדויקת ומהירה.

TOKEN

ב־Git (ובמיוחד עם GitHub), **טוקן** הוא מין "סיסמה מיוחדת" שמחליפה את הסיסמה הרגילה שלך כשאתה עובד עם פקודות כמו `git push` או `git pull`. במקום להקליד את הסיסמה שלך בכל פעם, אתה יוצר טוקן – מחרוזת ארוכה של תווים – ומכניס אותו כש-Git מבקש סיסמה. זה הרבה יותר בטוח, במיוחד אם אתה משתמש באימות דו-שלבי (2FA). הטוקן נוצר דרך ההגדרות של החשבון שלך בגיטהאב, אפשר לקבוע לו הרשאות (למשל רק קריאה או גם כתיבה) והוא תקף לתקופה מסוימת. חשוב לשמור אותו, כי הוא יוצג רק פעם אחת.

---

zsh

רק עם השלמה חכמה, תוספים ועיצובים bash היא מעטפת טרמינל מתקדמת, כמו

בנוסף היא ברירת המחדל של macOS

Bash

באש היא מעטפת פקודות (ובעצם שפת תכנות) שפותחה בשנת 1989 כחלק מפרוייקט GNU.
במערכות linux נעשה שימוש בבאש, המותקן כברירת מחדל ופועל על הטרמינל, ומאפשר למשתמש "לתקשר" עם המחשב באמצעות פקודות. באש הוא מה שנקרא באנגלית "shell", שכן הוא פועל מסביב לגרעין של מערכת ההפעלה.

בהקשר של linux, כשאנו רושמים פקודה בטרמינל, באש מחפש בתוך הזיכרון של המחשב פקודה קיימת מסוג זה, ואם זו קיימת הוא מבצע אותה. דוגמאות לשימושים של באש הם ניהול קבצים, התקנת תוכנות, וניהול הרשאות.
חלק ממה שהופך את באש לעוצמתי, הוא העובדה שהוא פועל כשפת תכנות לכל דבר. כלומר, במקום לכתוב פקודות בטרמינל באופן ידני, ניתן לכתוב תוכניות קוד שיבצעו פקודות אוטומטית.

Forking - פיצול

מטרת הפיצול: ייצור שכפול של פרויקט קוד מקור (repository) של מישהו אחר ועבודה עליו. בעצם זהו ענף פיתוח של תוכנה אשר נפרד מהתוכנה המקורית במטרה לעבוד עליו בנפרד.
כיצד מבצעים זאת (בgithub):

- היכנס ל-GitHub (github.com) והתחבר לחשבונך.
- מצא את המאגר שברצונך לבצע עליו Fork (למשל `https://github.com/some-user/some-repo`).
- בפינה הימנית העליונה של עמוד המאגר, לחץ על Fork.
- בחר את החשבון שלך כדי ליצור את ה-Fork.
- כעת יש לך עותק משוכפל של המאגר בחשבון שלך

Repository

מדובר בתיקייה שבה נשמרים:

1. **הקבצים של הפרויקט שלך** (קוד, קבצי תיעוד, תצוגות וכו').
2. **היסטוריית השינויים** של כל הקבצים – כולל מי שינה מה, מתי ולמה.
3. משתמש בו כדי לעקוב אחרי הכל – נשמר בתיקייה מוסתרת שנקראת Git-**מטא-דאטה** ש .git.

ישנם שני סוגי ריפוזיטורי:

1. Local repository - ריפוזיטורי מקומי - הוא יושב על המחשב שלך ואתה עובד עליו ישירות
2. Remote repository - כמו (Github עותק של אותו פרויקט בשרת מרוחק, מה שמאפשר עבודה בקבוצות ושיתוף פעולה עם אנשים אחרים.) וכו..

בתוך כל ריפוזיטורי יש:

- **קבצי הפרויקט** (קוד, תמונות, קבצי הגדרות וכו')
- **היסטוריית קומיטים** (כל שינוי שתיעדת עם git commit)

- **ענפים (branches)** – מאפשרים לעבוד על כמה גרסאות במקביל
- **תגיות (tags)** – לסימון גרסאות חשובות (למשל: v1.0)
- **staging area** – אזור שבו שינויים "מחכים" לפני שהם נשמרים בקומיט

<u>ניצור ריפוזיטורי באמצעות הפקודה:</u>
git init
נשכפל רפוזיטורי קיים באמצעות הפקודה:
git clone <ur>l

- **עשה** – כל פעם שאתה עושה שינוי **git add ואז git commit**.
- **עשה** – אם אתה רוצה לשתף את העבודה שלך **git push**.
- **עשה** – אם מישהו אחר עשה שינויים **לפני שאתה מתחיל לעבוד git pull**.
- עוזר למנוע מעקב אחרי קבצים שאתה לא רוצה (כמו סיסמאות, gitignore. הקובץ קבצי בנייה ועוד).

<u>מתי צריך remote repository?</u>

כשעובדים בצוות (כדי שכולם יראו את הקוד).

- כשאתה רוצה גיבוי בענן.
- כשאתה משתף פרויקט עם העולם (Open Source).
- כשאתה יוצר פורטפוליו של הפרויקטים שלך.

שמירת גרסה - Commit

זה שמירת גרסה (עדכון) לפרויקט מסויים.
1. משנים קובץ בפרויקט.
2. מוסיפים שינויים לרשימת המתנה.
3. מבצעים commit וזה שומר את השינוי.

commit נשמר עם תאריך שינוי, שם המבצע, והערות שמסבירות את השינוי.
זה מאפשר עבודה מסודרת ומעקב אחרי כל שינוי שבוצע בקוד. (כולל שינויים ישנים יותר).
