🚀 מלא ומקיף למתחילים Git מדריך

📑 תוכן עניינים

🎯 חלק 1: מה זה Git ולמה אנחנו צריכים אותו?

🤔 דמיינו את המצב הזה:

אתם כותבים עבודה ב Word. פתאום המחשב קורס ואתם מאבדים הכל. או שאתם רוצים לחזור לגרסה ישנה של העבודה אבל כבר מחקתם אותה.

פותר את הבעיות האלה בקוד Git!

📚 במילים פשוטות Git:

Git הוא כמו **מכונת זמן לקוד** שלכם. הוא:

1. **שומר היסטוריה** - כל שינוי שעשיתם נשמר
2. **מאפשר לחזור אחורה** - טעיתם? אפשר לחזור לגרסה קודמת
3. **מאפשר עבודה משותפת** - כולכם יכולים לעבוד על אותו פרויקט בלי להתנגש
4. **גיבוי אוטומטי**-GitHub (באינטרנט) - הקוד נשמר גם ב

🏗️ המושגים הבסיסיים (אל תפחדו מהשמות!):

1. Repository (repo) - המאגר

זה כמו **תיקייה חכמה** שעוקבת אחרי כל השינויים בקוד.

📋

```
📁 LearnFullStack (זה ה-Repository שלכם)
  ├── 📁 Student1
  ├── 📁 Student2
  └── 📁 Student3
```

2. Commit - שמירה

הוא נקודה שאפשר לחזור אליה commit זה כמו **צילום של הקוד ברגע מסוים**. כל

```
📋
✓
```

```
Commit 1: "הוספתי קובץ HTML ראשון"
Commit 2: "שיניתי את הצבע לכחול"
Commit 3: "הוספתי תמונה"
```

3. Branch - ענף

זה כמו **עולם מקביל** שבו אתם יכולים לנסות דברים בלי לקלקל את הקוד הראשי.

```
📋
✓
```

```
main (הקוד הראשי) ──────────────────────────────
                \
                 student1-branch (העולם שלכם)
```

4. Push - דחיפה

העלאת השינויים-GitHub (לאינטרנט) שלכם מהמחשב שלכם ל.

```
📋
✓
```

```
המחשב שלך ──Push──> GitHub (☁ ענן)
```

5. Pull - משיכה

הורדת השינויים-GitHub (כדי לראות מה אחרים עשו) מ למחשב שלכם.

```
📋
✓
```

GitHub (☁️ ענן) ──Pull──> המחשב שלך

6. Pull Request (PR) - בקשת מיזוג

בקשה להוסיף את השינויים שלכם לקוד הראשי. אבא יבדוק ויאשר.

📋

student1-branch ──PR──> בדיקה של אבא ──> main

💻 חלק 2: התקנת Git וVS Code

שלב 1: הורדת Git

Windows:

1. היכנסו ל: https://git-scm.com/download/win
2. הורידו את הגרסה ל-Windows
3. הפעילו את הקובץ שהורדתם
4. פשוט תלחצו "Next" (ההגדרות ברירת המחדל טובות) - **בכל שלב בהתקנה**
5. בסוף תלחצו "Finish"

Mac:

1. פתחו Terminal (חפשו "Terminal" ב-Spotlight)
2. הקלידו: `git --version` ותלחצו Enter
3. אם Git אישרו - המחשב יציע להתקין, לא מותקן

שלב 2: בדיקה שGit מותקן

1. פתחו **Command Prompt** (Windows) או **Terminal** (Mac)
2. הקלידו:

📋
bash

```
git --version
```

3. אם רואים משהו כמו `git version 2.40.0` - מזל טוב! Git מותקן!

שלב 3: הגדרת Git (חד פעמי!)

פתחו Terminal/Command Prompt (החליפו בשם שלכם!) והקלידו:

bash

```bash
git config --global user.name "השם שלך"
git config --global user.email "האימייל שלך@example.com"
```

דוגמה:

bash

```bash
git config --global user.name "Student1"
git config --global user.email "student1@gmail.com"
```

שלב 4: VS Code מותקן?

אם כבר יש לכם VS Code - אם לא! מעולה:

1. היכנסו ל: https://code.visualstudio.com
2. הורידו והתקינו
3. פתחו את VS Code

שלב 5: התקנת תוספים חיוניים ב-VS Code

1. **פתחו VS Code**
2. `Ctrl+Shift+X` **לחצו על אייקון התוספים** (4 ריבועים בצד שמאל) או
3. **התקינו את התוספים הבאים:**

תוסף 1: Git Graph

- **חיפוש:** `Git Graph`
- **מחבר:** mhutchie
- **למה צריך:** מראה את ההיסטוריה של Git בצורה ויזואלית מדהימה
- **לחצו:** Install

תוסף 2: GitLens

- **חיפוש:** `GitLens`
- **מחבר:** GitKraken
- **למה צריך:** מראה מי שינה כל שורה ומתי, עוזר להבין קוד
- **לחצו:** Install

תוסף 3: Live Server

- **חיפוש:** `Live Server`
- **מחבר:** Ritwick Dey
- **למה צריך:** לפתוח HTML בדפדפן עם refresh אוטומטי
- **לחצו:** Install

תוסף 4: Prettier

- **חיפוש:** `Prettier - Code formatter`
- **למה צריך:** מסדר את הקוד אוטומטית
- **לחצו:** Install

🚀 חלק 3: עבודה עם הפרויקט - צעד אחר צעד

🎯 המבנה שנבנה:

```
📁 LearnFullStack (GitHub Repository)
├── 📁 Student1
│   ├── 📄 README.md
│   ├── 📁 01-HTML-Projects
│   ├── 📁 02-CSS-Projects
│   └── 📁 03-JavaScript-Projects
├── 📁 Student2
│   ├── 📄 README.md
│   └── ... (אותו מבנה)
└── 📁 Student3
    ├── 📄 README.md
    └── ... (אותו מבנה)
```

📋 מומלץ - הסדר שנעבוד בו Workflow:

1. Clone הפרויקט (פעם ראשונה בלבד)
   ↓
2. יצירת Branch אישי
   ↓
3. עבודה על הקוד
   ↓
4. Commit (שמירה מקומית)
   ↓
5. Push (העלאה ל-GitHub)
   ↓
6. Pull Request (בקשה למיזוג)
   ↓
7. אבא בודק ומאשר
   ↓
8. Sync עם main
   ↓
9. חוזרים לשלב 2 לפרויקט הבא

📥 שלב 1: Clone - הורדת הפרויקט (פעם ראשונה!)

מה זה Clone?

Clone = להעתיק את כל הפרויקט מ-GitHub למחשב שלכם.

⚠️ **עושים את זה רק פעם אחת!**

איך עושים Clone?

דרך 1: דרך VS Code (הכי פשוט!)

1. **פתחו VS Code**
2. **לחצו על `Ctrl+Shift+P` (Windows) או `Cmd+Shift+P` (Mac)**
   - יפתח לכם תפריט למעלה (Command Palette)
3. **הקלידו: `Git: Clone`**
   - בחרו באפשרות הזו מהרשימה
4. **הדביקו את הכתובת הזו:**

📋

https://github.com/liorstr1/LearnFullStack.git

5. **בחרו תיקייה במחשב** - איפה תרצו לשמור את הפרויקט
   - מומלץ: שולחן העבודה או `Documents/Projects`

6. **המתינו** - VS Code מוריד את כל הקוד
7. **לחצו "Open"** כשזה מציע לפתוח את הפרויקט

דרך 2: דרך Terminal (אלטרנטיבה)

1. פתחו Terminal/Command Prompt
2. נווטו לתיקייה שתרצו (לדוגמה שולחן העבודה):

bash

```bash
cd Desktop
```

3. הקלידו:

bash

```bash
git clone https://github.com/liorstr1/LearnFullStack.git
```

4. נכנסים לתיקייה:

bash

```bash
cd LearnFullStack
```

5. פותחים ב-VS Code:

bash

```bash
code .
```

✅ איך יודעים שזה עבד?

- במיקום שבחרתם `LearnFullStack` תראו תיקייה חדשה בשם
- ב-VS Code תראו את כל התיקיות בצד שמאל (Explorer)
- למטה ב-VS Code תראו את שם ה-branch (כנראה `main` או `master`)
- בשורת הסטטוס התחתונה Git יהיה לכם סמל של

🌿 אישי Branch שלב 2: יצירת

למה צריך Branch אישי?

זה כמו **חדר עבודה אישי** שלכם. כל אחד עובד בחדר שלו, ואז מציג את העבודה לאבא Branch.

יתרונות:

- לא מפריעים אחד לשני
- אפשר לנסות דברים בלי פחד
- אבא יכול לבדוק כל אחד בנפרד
- אפשר לעבוד על כמה דברים במקביל

איך יוצרים Branch?

דרך 1: דרך VS Code (הכי פשוט!)

1. **למטה משמאל ב-VS Code** תראו את שם ה-branch הנוכחי (כנראה `main`)
   - 🌿 זה ליד סמל של שתי ענפים
2. **לחצו על השם** (main)
3. בחרו - **יפתח תפריט למעלה**: **"Create new branch..."**
4. **תנו שם ל-Branch** - לפי הכללים הבאים:

כללי שמות Branch:

📋 ✓

<שם-התלמיד>/<שם-הפרויקט>

דוגמאות טובות:

✅ student1/html-cv
✅ student2/css-basics
✅ student3/js-calculator
✅ student1/lesson-1-2-css

דוגמאות רעות:

❌ my-branch
❌ test123
❌ asdfsdf
❌ work

למה החלוקה הזו?

- `student1` = מזהה מי עושה את העבודה
- `html-cv` = מה הפרויקט
- `/` = מפריד בין השם לפרויקט

5. **לחצו Enter** - Branch נוצר ואתם כבר נמצאים בו!

דרך 2: דרך Terminal

1. פתחו Terminal ב-VS Code (`Ctrl+~` או View → Terminal)
2. ודאו שאתם על `main`:

bash

```
git checkout main
```

3. משכו את העדכונים האחרונים:

bash

```
git pull origin main
```

4. צרו Branch חדש:

bash

```
git checkout -b student1/html-cv
```

(בשם הפרויקט `html-cv`-בשם שלכם ו `student1` החליפו)

הסבר הפקודה:

- `git checkout` = עבור ל-branch
- `-b` = צור branch חדש
- `student1/html-cv` = השם של ה-branch

✅ איך יודעים שה-Branch נוצר?

- למטה ב-VS Code תראו את השם החדש של ה-Branch במקום `main`
- בטרמינל - תראו רשימה של כל ה-Branches, תסמן את ה-Branch הנוכחי (*) אם תקלידו `git branch`
- ב-Source Control תראו את השם (סמל של שלושה עיגולים מחוברים)

📝 שלב 3: עבודה על הקוד

מבנה התיקיות שלכם:

כל תלמיד צריך ליצור את המבנה הבא **בתוך הפרויקט**:

📋

```
📁 LearnFullStack
└── 📁 StudentX (Student1/2/3-החליפו ב)
    ├── 📄 README.md
    ├── 📁 01-HTML-Projects
    │   ├── 📄 my-cv.html
    │   ├── 📄 basic-page.html
    │   ├── 📄 lists.html
    │   └── ...
    ├── 📁 02-CSS-Projects
    │   ├── 📁 css-basics
    │   │   ├── 📄 index.html
    │   │   └── 📄 style.css
    │   └── ...
    └── 📁 03-JavaScript-Projects
        └── ...
```

איך יוצרים את המבנה?

אם אתם Student1 (דוגמה):

1. **ב-VS Code, בExplorer (צד שמאל)**, לחצו ימני על `LearnFullStack`
2. **בחרו: "New Folder"**
3. **תנו שם:** `Student1` (לפי מה שאבא קבע - 1, 2 או 3)
4. **לחצו ימני על `Student1`** וצרו:
   - **קובץ חדש:** `README.md`
   - **תיקייה חדשה:** `01-HTML-Projects`
   - **תיקייה חדשה:** `02-CSS-Projects`
   - **תיקייה חדשה:** `03-JavaScript-Projects`
5. **בתוך `01-HTML-Projects`** תתחילו לעבוד על הפרויקטים מהשיעורים

דוגמת README.md:

פתחו את הקובץ `README.md` וכתבו:

📋

markdown

Student1 - Learning Journey 🚀

About Me
- **Name:** [השם שלך]
- **Age:** 17
- **Location:** Haifa, Israel
- **Started:** January 2025

Current Progress
- ✅ HTML Basics - Completed!
- 🔄 CSS Fundamentals - In Progress
- ⏳ JavaScript - Coming Soon
- ⏳ React - Future

Projects Completed

HTML Projects
1. [My CV](01-HTML-Projects/my-cv.html) - Personal CV website with semantic HTML
2. [Basic Page](01-HTML-Projects/basic-page.html) - First HTML page
3. [Lists Practice](01-HTML-Projects/lists.html) - Working with lists
4. [Complete Page](01-HTML-Projects/complete-page.html) - Full page with all tags

CSS Projects
1. Coming soon...

JavaScript Projects
1. Coming soon...

Goals for This Month
- [ ] Complete all HTML exercises
- [ ] Master CSS Flexbox
- [ ] Build a responsive portfolio
- [ ] Learn JavaScript basics

Skills Learned
- HTML5 semantic tags
- CSS styling and layouts
- Git and version control
- VS Code productivity

Resources I'm Using

- [MDN Web Docs](https://developer.mozilla.org)
- [W3Schools](https://www.w3schools.com)
- Course materials from Dad

Notes
- Remember to commit often!
- Always test in browser before pushing
- Ask Dad when stuck

Last Updated: [תאריך]

💾 שלב 4: Commit - שמירת השינויים

מה זה Commit?

Commit = צילום של הקוד **כרגע**. זה שומר את כל השינויים שעשיתם.

למה זה חשוב?

- נקודת שחזור - אפשר לחזור אליה
- היסטוריה - רואים מה עשיתם מתי
- מתעד מה השתנה commit תיעוד - כל

מתי עושים Commit?

הכלל הזהב: תעשו Commit אחרי כל חלק משמעותי שהשלמתם!

✅ **דוגמאות טובות למתי לעשות Commit:**

- סיימתי להוסיף header לדף
- סיימתי לעצב את ה-navigation
- תיקנתי באג בקוד
- הוספתי תמונות לפרויקט
- השלמתי section מלא
- סיימתי פיצ'ר מסוים

❌ **מתי לא לעשות Commit:**

- אחרי כל שורת קוד (יותר מדי!)
- אחד ענק בסוף היום (פחות מדי!) Commit
- כשהקוד לא עובד (תמיד תעשו Commit על קוד עובד!)
- באמצע עבודה על פיצ'ר

איך עושים Commit?

דרך 1: דרך VS Code (הכי פשוט!)

1. **לחצו על אייקון ה-Source Control** (שלושה עיגולים מחוברים בצד שמאל)
   - או לחצו `Ctrl+Shift+G`
2. **תראו רשימה של כל הקבצים שהשתנו:**
   - `U` = Untracked (קובץ חדש)
   - `M` = Modified (שונה)
   - `D` = Deleted (נמחק)
   - `A` = Added (נוסף)
3. **Stage הקבצים** (הכנה ל-Commit):
   - **אפשרות 1:** לחצו על + הגדול ליד "Changes" - כל הקבצים
   - **אפשרות 2:** לחצו על + ליד כל קובץ ספציפי - קבצים ספציפיים
4. **כתבו הודעת Commit** בתיבה למעלה (Message):

כללי כתיבה להודעות Commit:

✅ הודעות טובות:

"Add basic HTML structure for CV"

"Style navigation bar with CSS"

"Fix broken link in footer"

"Add profile image to about section"

"Complete lesson 1.1 exercises"

"Update README with progress"

❌ הודעות רעות:

"update"

"changes"

"aaa"

"test123"

"work in progress"

"stuff"

"fix"

הפורמט המומלץ:

<פועל באנגלית> <מה עשיתם>

תבנית:
[Action] [What you did]

פעלים נפוצים:
- Add: הוספתם משהו חדש
- Update: עדכנתם משהו קיים
- Fix: תיקנתם באג
- Remove: מחקתם משהו
- Refactor: שיפרתם קוד קיים
- Style: שינויי עיצוב
- Complete: השלמתם משהו

דוגמאות מעולות:
✅ "Add contact form to CV page"
✅ "Update navigation styling with Flexbox"
✅ "Fix typo in about section"
✅ "Remove unused CSS rules"
✅ "Complete all HTML exercises from lesson 1.1"

5. **לחצו על ה-✓ (Commit)** למעלה
   - או לחצו `Ctrl+Enter`
6. **תראו הודעה** - Commit נוצר!

דרך 2: דרך Terminal

bash

```
# 1. ראו מה השתנה
git status

# 2. הוסיפו קבצים ל-Stage
git add .                    # הכל
git add my-cv.html           # קובץ ספציפי
git add 01-HTML-Projects/    # תיקייה ספציפית
git add *.html               # כל קבצי HTML

# 3. עשו Commit
git commit -m "Add basic HTML structure for CV"

# קיצור: Stage + Commit בפקודה אחת (רק לקבצים שכבר tracked)
git commit -am "Update navigation styling"
```

✅ איך יודעים שה-Commit עבד?

- ב-Source Control של VS Code לא תראו יותר שינויים (Clean Working Tree)
- אם תלחצו על **Git Graph** (סמל של גרף למטה או View → Command Palette → "Git Graph: View Git Graph") - תראו את ה-Commit החדש
- בטרמינל, תקלידו `git log --oneline` - תראו את ההיסטוריה

☁️ שלב 5: Push - העלאה ל-GitHub

מה זה Push?

Push = להעלות את כל ה-Commits שעשיתם מהמחשב שלכם ל-GitHub (לענן).

למה צריך Push?

- גיבוי הקוד שלכם באינטרנט ☁️
- אבא (ואחרים) יכולים לראות את העבודה 👀
- אפשר לפתוח Pull Request ✅
- בטוח מפני תקלות במחשב 💾

איך עושים Push?

דרך 1: דרך VS Code (הכי פשוט!)

1. **פתחו Source Control** (`Ctrl+Shift+G`)
2. **לחצו על ה-... (שלוש נקודות) למעלה**
3. **בחרו: `Push`**
4. **בפעם הראשונה** - VS Code יבקש אישור:
   - "The branch 'student1/html-cv' has no upstream branch. Would you like to publish this branch?"
   - **לחצו: "OK" או "Yes"**

    - זה יוצר את ה-Branch ב-GitHub
5. **המתינו** - VS Code מעלה את הקוד
6. **תראו הודעה** - Push הצליח!

דרך 2: דרך Terminal

bash

```
# בפעם הראשונה (כשה-Branch חדש):
git push -u origin student1/html-cv

# מהפעם השנייה והלאה (אחרי ה--u):
git push
```

הסבר הפקודה:

- `git push` = דחוף את ה-Commits
- `-u` = set upstream (קשר את ה-Branch המקומי ל-GitHub)
- `origin` = השם של ה-remote (GitHub)
- `student1/html-cv` = שם ה-Branch

⚠️ שגיאות נפוצות ופתרונות:

שגיאה 1: "failed to push some refs"

סיבה: מישהו אחר Push או ש, לפניכם, GitHub יותר עדכני מהמחשב שלכם.

פתרון:

bash

```
# 1. משכו עדכונים
git pull origin student1/html-cv

# 2. אם יש conflicts - תפתרו אותם (נסביר בהמשך)

# 3. נסו שוב Push
git push
```

שגיאה 2: "fatal: The current branch has no upstream"

סיבה: זו הפעם הראשונה שאתם דוחפים את ה-Branch הזה.

פתרון:

bash

```
git push -u origin <branch-name>
```

שגיאה 3: "Permission denied"

סיבה: אין לכם הרשאות, או צריך authentication.

פתרון:

1. ודאו שאבא הוסיף אתכם כ-Collaborators
2. ודאו שהגדרתם Git עם username ו-email
3. ייתכן שתצטרכו Personal Access Token (אבא יעזור)

✅ איך יודעים שה-Push עבד?

1. **היכנסו ל-GitHub:** https://github.com/liorstr1/LearnFullStack
2. **לחצו על Branches** (או dropdown ליד "main")
3. **תראו את ה-Branch שלכם ברשימה!** 🎉
4. תראו את הקוד שלכם - **לחצו עליו**!
5. **תראו את כל ה-Commits שעשיתם**

🔀 שלב 6: Pull Request - בקשה למיזוג

מה זה Pull Request (PR)?

PR = בקשה לאבא: "תבדוק את הקוד שלי ותאשר להוסיף אותו ל-main"

זה **לא מוסיף אוטומטית** - אבא צריך לבדוק ולאשר!

למה צריך PR?

- בדיקת קוד (Code Review) 🔍
- למידה מפידבק 📚
- שמירה על איכות הקוד ✨
- תרגול עבודת צוות 👥

איך פותחים Pull Request?

ב-GitHub:

1. **היכנסו ל:** https://github.com/liorstr1/LearnFullStack
2. תראו הודעה צהובה למעלה - **אחרי Push**:

student1/html-cv had recent pushes (X minutes ago)
[Compare & pull request]

לחצו על הכפתור הירוק!

אם אין הודעה:

- לחצו על **Pull requests** למעלה
- לחצו על **New pull request** (כפתור ירוק)
- בחרו:
  - **base:** `main` ← לאן אנחנו רוצים למזג
  - **compare:** `student1/html-cv` ← מה אנחנו רוצים למזג
- לחצו

3. **מלאו את הפרטים:**

Title (כותרת):

[Student1] HTML CV Project - Lesson 1.1 Complete
[Student2] CSS Basics - Lesson 1.2 Exercises
[Student3] JavaScript Calculator - First Project

Description (תיאור):

markdown

📋 מה עשיתי?
- יצרתי את דף ה-HTML שלי ב CV
- הוספתי פרטים אישיים, השכלה, מיומנויות
- השתמשתי בכל התגים מהשיעור
- בדקתי שהכל עובד בדפדפן
- הוספתי תמונות וקישורים

📁 קבצים שהוספתי:
- `Student1/01-HTML-Projects/my-cv.html`
- `Student1/01-HTML-Projects/basic-page.html`
- `Student1/README.md`

✅ Checklist:
- [x] הקוד עובד בלי שגיאות
- [x] השתמשתי ב HTML סמנטי
- [x] כל התגים נסגרים
- [x] הקוד מסודר וקריא
- [x] בדקתי בדפדפן

📸 Screenshots (אופציונלי):
[אם יש לכם צילום מסך של הדף - הדביקו כאן]

📝 Notes:
- זה הפרויקט הראשון שלי!
- אשמח לפידבק על המבנה
- יש לי שאלה על תגי header - נדבר בשיעור הבא

🎯 What I Learned:
- HTML semantic tags
- Document structure
- Forms and inputs
- Lists and tables

4. **לחצו על `Create pull request`** (כפתור ירוק)
5. **תראו את ה-PR שנוצר!** 🎉

מה קורה אחרי זה?

תהליך ה-Review:

אתם פותחים PR
↓
אבא מקבל התראה 📧
↓
אבא בודק את הקוד 👀
↓

```
┌──────────────────────────────────┐
│ אבא יכול לעשות אחד מאלה:         │
├──────────────────────────────────┤
│ 1. ✅ Approve - אישור            │
│ 2. 💬 Comment - הערות            │
│ 3. 🔄 Request Changes - שינויים  │
└──────────────────────────────────┘
```

↓
אם צריך שינויים:
- תתקנו בקוד שלכם
- תעשו Commit חדש
- תעשו Push
- ה-PR יתעדכן אוטומטית!

↓
אבא מאשר (Merge) ✅
↓
הקוד שלכם נכנס ל-main! 🎉
↓
אתם מקבלים הודעה

איך לראות את ה-PR שלכם:

1. **היכנסו ל-GitHub**
2. למעלה **"Pull requests" לחצו על**
3. **תראו את כל ה-PRs** - שלכם יהיה פתוח (Open)
4. תראו - **לחצו עליו**:
    - את כל ה-Commits
    - את כל הקבצים שהשתנו
    - הערות של אבא
    - היסטוריית השיחה

🔄 שלב 7: Sync - עדכון מה-Main

למה צריך Sync?

אחרי שה-PR **צריך לעדכן את המחשב שלכם** - שלכם אושר, או כשמישהו אחר העלה קוד.

למה?

- לראות את העבודה של האחרים
- לקבל את השינויים האחרונים
- להימנע מקונפליקטים
- לעבוד על קוד עדכני

איך עושים Sync?

דרך 1: VS Code

1. **עברו ל-Branch `main`**:
   - למטה משמאל-Branch לחצו על שם ה
   - בחרו `main` מהרשימה
2. **משכו עדכונים** (Pull):
   - **אפשרות 1:** לחצו על הסמל של Sync (חצים מעגליים) למטה מימין
   - **אפשרות 2:** Source Control → ... → Pull
3. **תראו הודעה** - Pull הצליח!
4. **עכשיו אתם מעודכנים!**

דרך 2: Terminal


bash

```
# 1. חזרו ל-main
git checkout main

# 2. משכו עדכונים
git pull origin main

# 3. תראו רשימה של מה השתנה
```

✅ עכשיו אתם מעודכנים עם כולם!

מה הלאה?

- צרו → Branch חדש מ-main אם רוצים לעבוד על פרויקט חדש
- קיים → עברו אליו Branch-אם רוצים להמשיך ב

📊 חלק 4: Git Graph - הדרך הויזואלית לראות את ההיסטוריה

מה זה Git Graph?

Git Graph הוא תוסף ל-VS Code שמראה את כל ההיסטוריה של הפרויקט בצורה ויזואלית - כמו **עץ משפחה של הקוד**.

איך משתמשים ב Git Graph?

1. פתיחת Git Graph:

דרך 1: לחצו על סמל הגרף בשורה התחתונה (Status Bar)

דרך 2:

- `Ctrl+Shift+P` (Command Palette)
- הקלידו: `Git Graph: View Git Graph`
- Enter

2. מה תראו ב Git Graph:

```
┌─────────────────────────────────────────────────────────────────────────────┐
│ Git Graph                                    🔄 ⚙️ ❌                       │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│                                                                             │
│  main    ●──────●──────●──────●──────●                                      │
│           \      \                                                          │
│  student1     ●──────●──────●  (PR merged)                                  │
│                \                                                            │
│  student2          ●──────●──────● (working)                                │
│                                                                             │
│                                                                             │
│ Commits: [רשימת commits]                                                    │
│                                                                             │
│  ┌─ Commit Details ──────────────────────────────────────────────────┐      │
│  │ Author: Student1                                                  │      │
│  │ Date: 2025-01-15 14:30                                            │      │
│  │ Message: Add basic HTML structure for CV                          │      │
│  │                                                                   │      │
│  │ Changed files:                                                    │      │
│  │   M  Student1/01-HTML-Projects/my-cv.html                         │      │
│  │   A  Student1/README.md                                           │      │
│  └───────────────────────────────────────────────────────────────────┘      │
└─────────────────────────────────────────────────────────────────────────────┘
```

3. אלמנטים ב Git Graph:

גרף ענפים (Branch Graph):

- **נקודות (●)** = Commits
- חיבורים בין = **קווים** commits
- **צבעים שונים** = branches שונים
- איפה כל = **ראשי ענפים** branch נמצא עכשיו

רשימת Commits:

- לחיצה על commit → רואים פרטים
- גלילה למטה → commits ישנים יותר

פרטי Commit:

- מי עשה
- מתי
- מה ההודעה
- אילו קבצים השתנו

4. פעולות שאפשר לעשות:

לחיצה ימנית על Commit:

- **Checkout Commit** - חזרה לcommit ספציפי
- **Cherry Pick** - העתקת commit לbranch אחר
- **Revert Commit** - ביטול commit
- **View Commit Details** - פרטים מלאים

לחיצה ימנית על Branch:

- **Checkout Branch** - מעבר לbranch
- **Merge into Current Branch** - מיזוג
- **Delete Branch** - מחיקה

📖 קריאת הגרף:

📋✓

```
main        ●———●———●———●———●———●
            \       /       \
student1        ●———●——┘         \
                        \
student2                ●———●
```

פענוח:

1. main התחיל
2. student1 יצר branch, עבד (2 commits), ו-merged חזרה
3. student2 יצר branch והוא עדיין עובד

💡 **טיפים לשימוש:**

1. **תסתכלו על Git Graph לפני Push** - לראות שהכל בסדר
2. **אחרי Pull** - לראות מה השתנה
3. **לפני מיזוג Branches** - להבין את המצב
4. **כשיש קונפליקט** - לראות מאיפה זה בא

🎯 חלק 5: כללי עבודה והנחיות

📋 שנעבוד לפיו Workflow:

תהליך עבודה סטנדרטי (כל פרויקט):

📅 תחילת יום / פרויקט חדש:
├─ 1. git checkout main
├─ 2. git pull origin main (עדכון)
├─ 3. git checkout -b student1/new-project (branch חדש)
└─ 4. התחילו לעבוד!

⚒️ במהלך העבודה:
├─ כתבתם קוד
├─ בדקתם שזה עובד
├─ git add .
├─ git commit -m "הודעה ברורה"
├─ המשיכו לעבוד...
└─ חזרה על 1-5 כמה פעמים

🌆 סוף יום / סיום פיצ'ר:
├─ 1. git add .
├─ 2. git commit -m "סיכום מה עשיתם"
├─ 3. git push (העלאה לGitHub)
└─ 4. פתחו PR ב-GitHub

📬 אחרי אישור PR:
├─ 1. git checkout main
├─ 2. git pull origin main (קבלו את השינויים)
└─ 3. חזרה להתחלה לפרויקט הבא!

📜 כללי הזהב:

1. Commit Messages:

✅ DO:
- "Add login form to homepage"
- "Fix navigation bar alignment"
- "Update README with installation steps"

❌ DON'T:
- "update"
- "fix stuff"
- "work in progress"

2. Branch Names:

✅ DO:
- student1/html-cv
- student2/css-flexbox
- student3/js-calculator

❌ DON'T:
- test
- my-branch
- asdf123

3. מתי לעשות Commit:

✅ DO:
- אחרי השלמת פיצ'ר קטן
- אחרי תיקון באג
- כל שעה-שעתיים של עבודה
- לפני סוף יום

❌ DON'T:
- אחרי כל שורת קוד
- אחד ענק בסוף Commit
- כשהקוד לא עובד

4. מה לא להעלות (gitignore):

📋
✓

❌ לעולם אל תעלו:
- node_modules/ (תיקיות dependencies)
- .env (קבצי סביבה עם secrets)
- .DS_Store (קבצי Mac)
- *.log (קבצי log)
- personal-notes.txt (הערות אישיות)

✅ תמיד תעלו:
- קבצי קוד (.html, .css, .js)
- README.md
- תמונות שחלק מהפרויקט
- קבצי הגדרה של הפרויקט

🔒 **כללי אבטחה:**

📋
✓

⚠️ אל תעלו ל GitHub:

1. סיסמאות
2. API Keys
3. Tokens
4. פרטי כרטיס אשראי
5. מידע אישי רגיש

אם בטעות העליתם משהו רגיש:

1. ספרו לאבא מיד
2. נמחק מההיסטוריה
3. נשנה את הסיסמה/key

🚀 מתקדמות Git חלק 6: פקודות

📚 רשימה מלאה של פקודות:

ניווט בין Branches:

```bash
```

```bash
# ראה את כל ה-Branches (מקומיים)
git branch

# ראה את כל ה-Branches (כולל remote)
git branch -a

# עבור ל-Branch אחר
git checkout <branch-name>
git checkout main
git checkout student1/html-cv

# צור Branch חדש ועבור אליו
git checkout -b student1/new-project

# מחק Branch מקומי
git branch -d student1/old-project    # מחיקה רגילה
git branch -D student1/old-project    # מחיקה כפויה

# שנה שם ל-Branch
git branch -m old-name new-name

# ראה branch נוכחי
git branch --show-current
```

עבודה עם שינויים:

bash

```
# ראה סטטוס (מה השתנה)
git status
git status -s                    # גרסה קצרה

# הוסף קבצים ל-Stage
git add <file>                   # קובץ אחד
git add .                        # הכל בתיקייה נוכחית
git add *.html                   # כל קבצי HTML
git add src/                     # תיקייה שלמה

# הסר קבצים מ-Stage (אם טעית)
git reset <file>
git reset                        # הכל

# ראה הבדלים
git diff                         # שינויים לפני Stage
git diff --staged                # שינויים אחרי Stage
git diff <branch1> <branch2>     # הבדלים בין branches

# Commit
git commit -m "Your message"
git commit -am "Message"         # Stage + Commit ביחד (רק tracked files)
git commit --amend               # שנה את ה-commit האחרון

# ראה היסטוריית Commits
git log
git log --oneline                # גרסה קצרה
git log --graph                  # עם גרף ויזואלי
git log --oneline --graph --all  # הכל ביחד
git log --author="Student1"      # commits של מישהו ספציפי
git log --since="2 weeks ago"    # commits מהשבועיים האחרונים
git log <file>                   # היסטוריה של קובץ ספציפי

# ראה commit ספציפי
git show <commit-hash>
git show HEAD                    # ה-commit האחרון
```

סנכרון עם GitHub:

bash

```bash
# Pull (משיכה)
git pull                        # מה-branch הנוכחי
git pull origin main            # מ-main ספציפית
git pull origin <branch-name>   # מ-branch ספציפי

# Push (דחיפה)
git push                        # ל-branch upstream
git push origin <branch-name>   # ל-branch ספציפי
git push -u origin <branch-name>  # set upstream וגם push
git push --force                # דחיפה כפויה (זהירות!)

# Fetch (הורד מידע בלי למזג)
git fetch                       # הכל
git fetch origin                # מ-origin
git fetch origin <branch-name>  # branch ספציפי

# ראה remotes
git remote -v
git remote show origin
```

חזרה אחורה / ביטול שינויים:

bash

```
# בטל שינויים בקובץ (לפני Stage)
git checkout -- <file>
git restore <file>              # גרסה חדשה יותר

# בטל Stage
git reset HEAD <file>
git restore --staged <file>         # גרסה חדשה יותר

# חזור ל-commit קודם
git reset --soft HEAD~1             # שמור שינויים
git reset --mixed HEAD~1            # בטל stage אבל שמור שינויים
git reset --hard HEAD~1             # מחק הכל (זהירות!)

# חזור ל-commit ספציפי
git reset --hard <commit-hash>

# Revert (יצירת commit חדש שמבטל commit ישן)
git revert <commit-hash>            # בטוח יותר מ-reset
```

Branch Management:

bash

```
# מיזוג (Merge)
git merge <branch-name>             # מזג branch אחר לנוכחי
git merge --no-ff <branch-name>     # מיזוג עם merge commit תמיד

# Rebase (סידור מחדש של commits)
git rebase main                     # העבר את ה-commits שלי על main
git rebase --continue               # המשך אחרי פתרון conflict
git rebase --abort                  # בטל rebase

# Cherry-pick (העתקת commit ספציפי)
git cherry-pick <commit-hash>
```

ניקוי וסדר:

bash

```bash
# נקה קבצים לא tracked
git clean -n                    # הצג מה יימחק (dry run)
git clean -f                    # מחק קבצים
git clean -fd                   # מחק קבצים ותיקיות

# נקה branches ישנים
git branch --merged             # ראה branches ש-merged
git branch -d <branch-name>     # מחק branch שכבר merged

# Stash (שמור שינויים זמנית)
git stash                       # שמור והסתר שינויים
git stash list                  # ראה רשימת stashes
git stash pop                   # החזר את השינויים
git stash apply                 # החזר בלי למחוק מה-stash
git stash drop                  # מחק stash
```

מידע ועזרה:

bash

```bash
# מידע על הפרויקט
git config --list               # כל ההגדרות
git config user.name            # השם שלך
git config user.email           # האימייל שלך

# עזרה
git help                        # רשימת פקודות
git help <command>              # עזרה על פקודה ספציפית
git <command> --help            # אותו דבר
```

💡 חלק 7: טיפים וטריקים

🎓 Best Practices:

1. Commit Often, Push Once a Day (או יותר):

```
✅ עשו Commit כל שעה-שעתיים
✅ עשו Push לפחות פעם ביום
✅ עשו Push לפני שמכבים את המחשב
```

2. תמיד Pull לפני Push:

bash

```bash
# תרגיל טוב:
git pull origin main
# עכשיו תעבדו עם קוד עדכני
git push
```

3. בדקו שהקוד עובד לפני Commit:

```
❌ אל תעשו Commit על קוד שלא עובד
✅ רצו את האתר/אפליקציה לפני Commit
✅ תקנו שגיאות לפני Commit
```

4. README.md תמיד מעודכן:

markdown

```
✅ עדכנו את ה-README אחרי כל פרויקט
✅ רשמו מה למדתם
✅ הוסיפו screenshots
```

🔧 Shortcuts ב-VS Code:

```
Ctrl+Shift+G      - פתח Source Control
Ctrl+Shift+P      - Command Palette
Ctrl+`            - פתח/סגור Terminal
Ctrl+B            - הסתר/הצג Sidebar
Ctrl+K Ctrl+S     - Keyboard Shortcuts
F5                - Refresh browser (כשב-Live Server)
```

📝 Git Aliases (קיצורים):

הוסיפו את אלה ל-Git config שלכם:

bash

```bash
# הגדרת aliases
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.lg "log --oneline --graph --all"

# עכשיו תוכלו לכתוב:
git st      # במקום git status
git co main # במקום git checkout main
git br      # במקום git branch
git lg      # log יפה
```

❓ חלק 8: FAQ - שאלות נפוצות

🤔 שאלות כלליות:

מה ההבדל בין Git ו-GitHub?

Git = התוכנה במחשב שלך (מנהל גרסאות)
GitHub = האתר באינטרנט (אחסון הקוד)

דוגמה:
Git כמו Word במחשב שלך
GitHub כמו Google Drive לשמירת המסמכים

ש: מה קורה אם אני מוחק קובץ בטעות?

bash

```bash
# אם לא עשית commit:
git checkout -- <file>

# אם עשית commit:
git log -- <file>          # מצא את ה-commit האחרון של הקובץ
git checkout <commit> -- <file>  # החזר את הקובץ
```

ש: איך אני יודע על איזה branch אני?

bash

```bash
# דרך 1:
git branch    # הכוכבית (*) מסמנת

# דרך 2:
git branch --show-current

# דרך 3:
```

למטה משמאל VS Codeתסתכלו ב

ש: שכחתי מה עשיתי ב-commit האחרון?

bash

```
git show            # כל הפרטים
git log -1          # רק המידע
git diff HEAD~1     # ההבדלים
```

🐛 בעיות נפוצות:

בעיה: "You have diverged branches"

```
יש: סיבה commits ב-GitHub שאין במחשב שלכם
      ויש commits ב-GitHub במחשב שאין ב

פתרון:
git pull origin <branch>    # נסו למזג
# יש אם conflicts - תפתרו
git push
```

בעיה: "Merge Conflict"

```
מה זה: שני אנשים שינו את אותה השורה

איך נראה בקובץ:
<<<<<<< HEAD
הקוד שלי
=======
הקוד של מישהו אחר
>>>>>>> other-branch

איך לפתור:
1. פתחו את הקובץ
2. מחקו את הסימנים (<<<<, ====, >>>>)
3. תשמרו מה שאתם רוצים
4. git add <file>
5. git commit -m "Resolve conflict"
```

בעיה: "detached HEAD state"

מה זה: חזרתם ל branch ישן ואתם לא על commit

פתרון 1 (רק להסתכל):
```
git checkout <branch-name>   # חזרו ל-branch
```

פתרון 2 (רוצים לעבוד מכאן):
```
git checkout -b new-branch   # צרו branch חדש מהנקודה הזו
```

נספחים

נספח א': פקודות Git לפי תרחישים

רוצה להתחיל פרויקט חדש:

bash
```bash
git checkout main
git pull origin main
git checkout -b student1/project-name
# עבדו על הקוד
git add .
git commit -m "Initial project setup"
git push -u origin student1/project-name
```

רוצה להמשיך לעבוד על פרויקט קיים:

bash

```bash
git checkout student1/project-name
git pull origin student1/project-name
# עבדו על הקוד
git add .
git commit -m "Continue working on feature"
git push
```

רוצה לראות מה קרה מאז אתמול:

bash

```bash
git log --since="yesterday"
git log --since="2 days ago"
git log --author="Student1" --since="1 week ago"
```

טעיתי ורוצה לבטל:

bash

```bash
# ביטול שינויים לפני commit:
git checkout -- <file>

# ביטול commit אחרון (שמירת שינויים):
git reset --soft HEAD~1

# ביטול commit אחרון (מחיקת שינויים):
git reset --hard HEAD~1

# ביטול commit ישן (יצירת commit חדש):
git revert <commit-hash>
```

נספח ב': Git Config מומלץ

bash

```
# הגדרות בסיסיות
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

# עורך ברירת מחדל
git config --global core.editor "code --wait"

# צבעים
git config --global color.ui auto

# Aliases שימושיים
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.unstage 'reset HEAD --'
git config --global alias.last 'log -1 HEAD'
git config --global alias.lg "log --oneline --graph --all --decorate"
git config --global alias.visual "log --oneline --graph --all"

# Pull עם rebase (מתקדם)
git config --global pull.rebase true

# התנהגות push
git config --global push.default current
```

נספח ג': .gitignore מקיף

צרו קובץ `.gitignore` בשורש הפרויקט:

gitignore

```
# ============================
# קבצי מערכת הפעלה
# ============================
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db
Desktop.ini

# ============================
# IDE / Editors
# ============================
# VS Code
.vscode/
*.code-workspace

# JetBrains (WebStorm, etc)
.idea/
*.iml
*.iws
*.ipr

# Sublime Text
*.sublime-project
*.sublime-workspace

# Vim
*.swp
*.swo
*~

# ============================
# Node.js
# ============================
node_modules/
npm-debug.log*
yarn-debug.log*
yarn-error.log*
```

```
lerna-debug.log*
.pnpm-debug.log*

# ============================
# Environment Variables
# ============================
.env
.env.local
.env.development.local
.env.test.local
.env.production.local
.env.*.local

# ============================
# Build / Distribution
# ============================
dist/
build/
*.min.js
*.min.css
.cache/
.parcel-cache/
.next/
.nuxt/
out/

# ============================
# Logs
# ============================
*.log
logs/
*.log.*

# ============================
# Testing
# ============================
coverage/
.nyc_output/
*.lcov

# ============================
```

```
# Personal / Temporary
# ============================
todo.txt
notes.txt
scratch/
temp/
tmp/
personal/
private/

# ============================
# Dependencies
# ============================
bower_components/
jspm_packages/

# ============================
# Archives
# ============================
*.zip
*.tar.gz
*.rar

# אבל השאירו את התמונות והקבצים שצריך!
!src/assets/**
!public/**
```

🎓 מונחון Git בעברית ובאנגלית

עברית	English	הסבר
מאגר	Repository (Repo)	התיקייה עם כל הקוד וההיסטוריה
שמירה	Commit	נקודת שחזור בזמן
ענף	Branch	קו עבודה נפרד
דחיפה	Push	העלאה לענן
משיכה	Pull	הורדה מהענן
מיזוג	Merge	חיבור שני ענפים
קונפליקט	Conflict	התנגשות בשינויים
התמחות	Stage	הכנה לcommit
מרוחק	Remote	השרת (GitHub)
קלון	Clone	העתקה של הפרויקט
מזלג	Fork	העתקה של פרויקט לחשבון שלך
בקשת משיכה	Pull Request (PR)	בקשה למזג שינויים
ראש	HEAD	המצב הנוכחי
תגית	Tag	סימון גרסה
מחסן	Stash	שמירה זמנית

🔧 מקיף Troubleshooting - פתרון בעיות

בעיה: "error: Your local changes would be overwritten by merge"

משמעות: יש לכם שינויים לא שמורים שיימחקו במיזוג

פתרון 1 - שמרו את השינויים:

bash

```
git stash
git pull
git stash pop
```

פתרון 2 - commit את השינויים:

bash

```
git add .
git commit -m "Save local changes"
git pull
```

פתרון 3 - מחקו את השינויים (זהירות!):

bash

```bash
git checkout -- .
git pull
```

בעיה: "fatal: refusing to merge unrelated histories"

משמעות: שני repos שלא קשורים

פתרון:

bash

```bash
git pull origin main --allow-unrelated-histories
```

בעיה: "fatal: detected dubious ownership"

משמעות: בעיית הרשאות (נפוץ ב-Windows)

פתרון:

bash

```bash
git config --global --add safe.directory /path/to/LearnFullStack
```

בעיה: "error: failed to push some refs"

תרחישים שונים:

לפניכם push תרחיש 1: מישהו

bash

```
git pull origin <branch-name>
# אם יש conflicts פתרו
git push
```

תרחיש 2: שיניתם היסטוריה (rebase)

bash

```
# רק אם אתם בטוחים ב-100%!
git push --force-with-lease
```

בעיה: "Permission denied (publickey)"

פתרונות:

1. בדקו username ו-email:

bash

```
git config user.name
git config user.email
```

2. השתמשו ב-HTTPS במקום SSH:

bash

```
git remote -v
git remote set-url origin https://github.com/liorstr1/LearnFullStack.git
```

3. Personal Access Token (אבא יעזור):

- GitHub → Settings → Developer settings → Personal access tokens
- צרו token חדש
- השתמשו בו כסיסמה

בעיה: המון merge conflicts

אסטרטגיה לפתרון:

bash

```bash
# 1. ראו כמה קבצים יש ב conflict
git status

# 2. פתרו אותם אחד אחד
# פתחו כל קובץ ב-VS Code

# 3. VS Code יציע לכם אפשרויות:
# - Accept Current Change (שלכם)
# - Accept Incoming Change (שלהם)
# - Accept Both Changes (שניהם)
# - Compare Changes (השוואה)

# 4. אחרי פתרון קובץ:
git add <file>

# 5. אחרי שפתרתם הכל:
git commit -m "Resolve merge conflicts"
```

בעיה: "You are in 'detached HEAD' state"

משמעות: חזרתם ל commit ישן

פתרון 1 - רק להסתכל:

bash

```bash
git checkout <branch-name>
```

פתרון 2 - לעבוד מכאן:

bash

```bash
git checkout -b new-branch-name
```

בעיה: commit בטעות קובץ גדול/רגיש

אם עוד לא push:

bash

```bash
git reset --soft HEAD~1
# הסירו את הקובץ
git add .
git commit -m "Your message"
```

אם כבר push:

bash

```bash
# ספרו לאבא מיד!
# צריך לנקות את ההיסטוריה
```

🎯 Git Best Practices - מדריך מתקדם

1. Commit Messages - הסטנדרט

פורמט מומלץ:

```
<type>(<scope>): <subject>

<body>

<footer>
```

Types:

- **feat**: פיצ'ר חדש
- **fix**: תיקון באג
- **docs**: שינוי בdocumentation
- **style**: עיצוב קוד (לא משנה לוגיקה)
- **refactor**: שינוי קוד (לא fix ולא feat)

- **test**: הוספת טסטים
- **chore**: משימות תחזוקה

דוגמאות:

```
feat(auth): add user login functionality

Added login form with email and password validation.
Integrated with backend API.

Closes #42

---

fix(css): correct navigation bar alignment

The navigation bar was not centered on mobile devices.
Fixed by updating flexbox properties.

---

docs(readme): update installation instructions

Added steps for Windows users.
Clarified Node.js version requirements.
```

2. Branch Strategy

שמות ברורים:

```
```

<type>/<description>

Types:
- feature/
- bugfix/
- hotfix/
- experiment/
- refactor/

Examples:
✅ feature/user-authentication
✅ bugfix/login-validation
✅ experiment/new-layout

3. Pull Request Best Practices

תבנית PR טובה:

```markdown
```

🎯 What
Brief description of what this PR does.

🔧 How
How did you implement this?

📸 Screenshots
(if applicable)

✅ Checklist
- [ ] Code works without errors
- [ ] Tested on multiple browsers
- [ ] Updated documentation
- [ ] Added comments to complex code
- [ ] Follows project coding standards

🔗 Related Issues
Closes #42
Related to #37

📝 Notes
Any additional information for reviewers.

4. Code Review Guidelines

כשאתם מקבלים code review:

✅ **עשו:**

- קראו את ההערות בקפידה
- שאלו שאלות אם משהו לא ברור
- תקנו את מה שמבקשים
- תודו על הפידבק

❌ **אל תעשו:**

- אל תקחו ביקורת באופן אישי
- אל תתווכחו על כל דבר קטן
- אל תתעלמו מהערות

כשאתם עושים code review (במקרה של peer review):

✅ **עשו:**

- היו מכבדים ונחמדים

- הציעו פתרונות, לא רק ביקורת
- חגגו דברים טובים
- היו ספציפיים

דוגמה טובה:

נהדר שהשתמשת ב-semantic HTML! 🎉

הצעה לשיפור:
במקום:

אפשר:
<header>

זה יותר נגיש ו-semantic.

5. Repository Organization

מבנה מומלץ:

```
LearnFullStack/
├── .github/
│   ├── ISSUE_TEMPLATE/
│   ├── PULL_REQUEST_TEMPLATE.md
│   └── workflows/
├── Student1/
│   ├── README.md
│   ├── 01-HTML-Projects/
│   ├── 02-CSS-Projects/
│   └── 03-JavaScript-Projects/
├── Student2/
├── Student3/
├── .gitignore
├── LICENSE
└── README.md
```

🚀 Git למתקדמים - מתי תהיו מוכנים

Git Rebase

מה זה: סידור מחדש של commits

bash

```bash
# שמור על היסטוריה ליניארית, rebase במקום merge
git checkout feature-branch
git rebase main

# אם יש conflicts:
# 1. פתרו אותם
git add .
git rebase --continue

# 2. או בטלו:
git rebase --abort
```

Git Cherry-Pick

מה זה: העתקת commit מ branchאחר לאחד

bash

```bash
# רוצים commit מbranch אחר
git cherry-pick <commit-hash>
```

Git Bisect

מה זה: מציאת commit שגרם לבאג

bash

```
git bisect start
git bisect bad              # הנוכחי גרוע commit
git bisect good <commit-hash>  # זה היה טוב commit
# באמצע commits יבדוק Git
# אתם תגידו לו good או bad
git bisect reset            # סיום
```

Git Hooks

מה זה: אוטומציה לפעולות Git

דוגמה: בדיקה אוטומטית לפני commit

bash

```
# .git/hooks/pre-commit

#!/bin/sh
npm run lint
npm test
```

📈 מעקב התקדמות

שבועי:

- [ ] כמה commits עשיתי השבוע?
- [ ] כמה PRs פתחתי?
- [ ] כמה PRs אושרו?
- [ ] האם יש לי branches ישנים למחיקה?

חודשי:

- [ ] מה היחס commits/working days שלי?
- [ ] האם אני משפר את הודעות ה-commit?
- [ ] האם אני עובד לפי ה-workflow?
- [ ] האם השתפרתי בפתרון conflicts?

מדדי הצלחה:

מתחיל:
- 1-2 commits ביום
- branch אחד בכל פעם
- בסיסיות commit הודעות

מתקדם:
- 3-5 commits ביום
- 2-3 branches פעילים
- מפורטות commit הודעות
- עצמאי conflicts פתרון

מקצוען:
- commits עקביים
- multiple branches
- commit messages סטנדרטיים
- code review למישהו אחר

🎉 סיום - אתם מומחי Git!

שיצרנו במיוחד בשבילכם Git-זהו! סיימתם את המדריך המלא ביותר ל.

מה למדתם:

✅ יסודות Git כל הפקודות החשובות ✅ והבנה מעמיקה ✅ Workflow פתרון בעיות ✅ מקצועי ✅ Best practices ✅ טכניקות מתקדמות

מה הלאה:

1. 📚 **תרגלו כל יום** - Git נעשה טבעי רק בתרגול
2. 🎯 **workflowעקבו אחרי ה** - זה יהפוך להרגל
3. 💪 **אל תפחדו לטעות** - זה חלק מהלמידה
4. 🤝 **עזרו אחד לשני** - peer review זה מעולה
5. 🚀 **תהנו מהקוד** - זה רק ההתחלה!

זכרו:

> "Git is hard until it's not." Git קשה עד שהוא לא.

עכשיו לכו לקודד! 💻 🚀

המדריך נוצר עם ❤️ במיוחד עבור:

- *Student1* 🌟
- *Student2* 🌟

- *Student3* 🌟

גרסה 1.0 - Complete Edition 2025 ינואר

May the Git be with you! 🎮