**שלב א׳ – הכרה והתקנה של Git + הגדרה ראשונית של המשתמש שלנו**

**אז מה זה בעצם Git ולמה כדאי להשתמש בזה?**

Git היא מערכת חכמה לניהול גרסאות, למעשה היא יודעת לנהל גרסאות של קבצים ומסמכים בכלל אך מאחר שאנו עוסקים בפיתוח קוד, כמובן שנתייחס אליה כמערכת לניהול גרסאות קוד.  
Git הוא מעין מאגר העוקב אחר השינויים בקבצי הקוד של הפרויקט שלנו. כל שינוי המבוצע מעודכן במאגר וביחד עם השינוי גם תיאור קצר המפרט על אותו שינוי וכמובן מי המפתח האחראי על השינוי. המונח המקצועי של כל שיגור שכזה הוא "Commit".   
Git מאפשרת לנו לעבוד בצוות, כל אחד על חלקו ולשגר את השינויים למאגר אחד המכיל את כל השינויים של כולם, שמו המקצועי של המאגר הוא "Repository".  
אז מדוע להשתמש בGit? הרי שיגור "הודעה" כמובן בשמה המקצועי (Commit) אחרי כל שינוי בקוד לא נשמע הליך מהנה ואידאלי במיוחד. נכון, אבל מתרגלים לזה מהר מאוד וביום שהפרויקט שלכם ישתבש ותהיה לכם האפשרות לחזור לגרסאות ישנות יותר במהירות ובפשטות אתם תברכו על כל Commit שעשיתם, בדיוק כמו שנזקקתם בפעם הראשונה לגיבוי כלשהו שיצרתם במהלך חייכם.  
השימוש ב-Git יעזור לכם לעבוד כיחיד ובפרט כצוות באופן מסודר וטוב יותר.

**התקנת Git**

ראשית ניכנס ל[כאן](https://git-scm.com/downloads) ונוריד את ההתקנה בהתאם למערכת שלנו.

**לינוקס**

הריצו את הפקודה הבאה:

$ sudo yum install git-all

**Mac**

לחיצה כפולה על .pkg והתקנה פשוטה.

**חלונות**

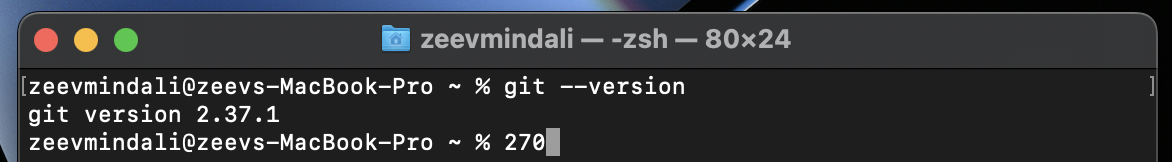
הרצת התקנה פשוטה כמו כל תוכנה רגילה.

**חשוב**: במדריך זה אנו נעבוד עם Git באמצעות חלון הbash זהו חלון הפקודות של Git, לכן, במהלך ההתקנה יש לוודא כי האפשרות "Use Git Bash only" מסומנת.

**לאחר ההתקנה נוודא שהכל תקין**

פתחו את Git Bash והריצו את הפקודה הבאה:

git --version



קיבלתם את מספר הגרסא המותקנת לכם על המחשב? ברכות! אפשר להתחיל עם הכיף האמיתי

**הגדרת המשתמש האישי שלנו**

כפי שכבר אמרתי, בדרך כלל העבודה עם Git מתבצעת בצוות, לכן מאוד חשוב שנדע מי אחראי ל-Commit מסוים (ואם הוא גרם לבאג אולי אפילו לנזוף בו קלות , מי העלה גרסא מסויימת וכדומה).  
לכל "חבר" ב-Repository יש שם משתמש וכתובת מייל המוגדרים ב-Git.  
על מנת להגדיר את שם המשתמש שלנו ב-Git נריץ את הפקודה הבאה:

git config --global user.name "Zeev"

ועל מנת להגדיר את כתובת המייל שלנו נריץ את הפקודה הבאה:

git config --global user.email "zeev@example.com"

כיצד נוודא שההגדרות נקלטו?

נריץ את הפקודה הבאה:

git config --list

מיד נקבל את מגוון ההגדרות של Git וחלק מהן זה שם המשתמש שלנו וכתובת המייל שלנו.

**טיפ קטן לסיום…**

אם תרצו לנקות את מסך ה-Bash, הפקודה הבאה תעשה זאת :

clear

**שלב שני – יצירת Repository, קובץ gitignore והוספת קבצי הפרויקט**

**יצירת Repository**

כפי שכבר אמרתי, Git מנהלת מעקב אחר הפרויקט שלכם. כל הקסם הזה מתרחש בתיקייה שבדרך כלל תהיה נסתרת ועם שם די צפוי .git   
על מנת ליצור תיקייה זו, בשמה המקצועי – Repository, עלינו "לכוון" את Git להסתכל על התיקייה שבה נרצה לנהל את ה-Repository שלנו.  
ניתן לעשות זאת על ידי הפקודה cd:

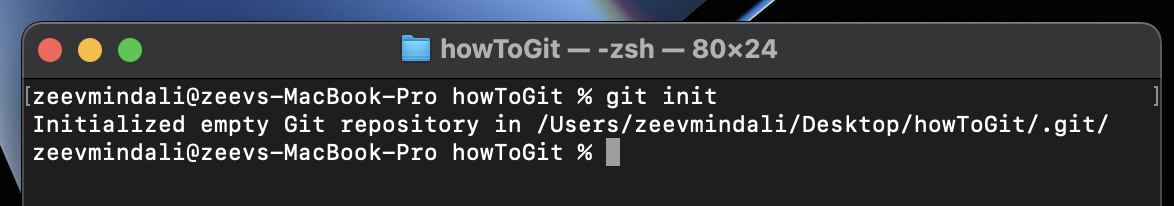
cd /users/zeevmindali

טיפ למשתמשי חלונות מקש ימני בתיקייה הרצויה ובחירה ב-"Git Bash Here" תעשו את אותה הפעולה.

מלבד הכתובית הצהובה מצד ימין, נוכל לקבל את התיקייה עליה Git "צופה" באמצעות הפקודה pwd:

פקודת pwd ב-Git

כעת, נוכל ליצור את ה-Repository שלנו על ידי הפקודה git init :



פקודת git init

כפי שאתם רואים, מיד קיבלנו הודעה שיצרנו Repository בתיקיה howToGit

כעת, לפני שנעבור להוספת קבצי הפרויקט שלנו ל-Repository שלנו, אני רוצה שתבינו כיצד Git עובדת.  
אני רוצה שתחשבו על Git כבעלת 3 תחנות עיקריות:  
Working Copy, Staging Copy, Committed Copy.

**Working Copy**

ה-Working Copy זוהי התחנה המסמלת את הקבצים אשר נמצאים בתיקיית הפרויקט שלנו, על המחשב שלנו.  
כאשר אנו עובדים על קובץ מסוים ושמרנו את השינויים, ה-Working Copy שלנו השתנה.

**Staging Copy**

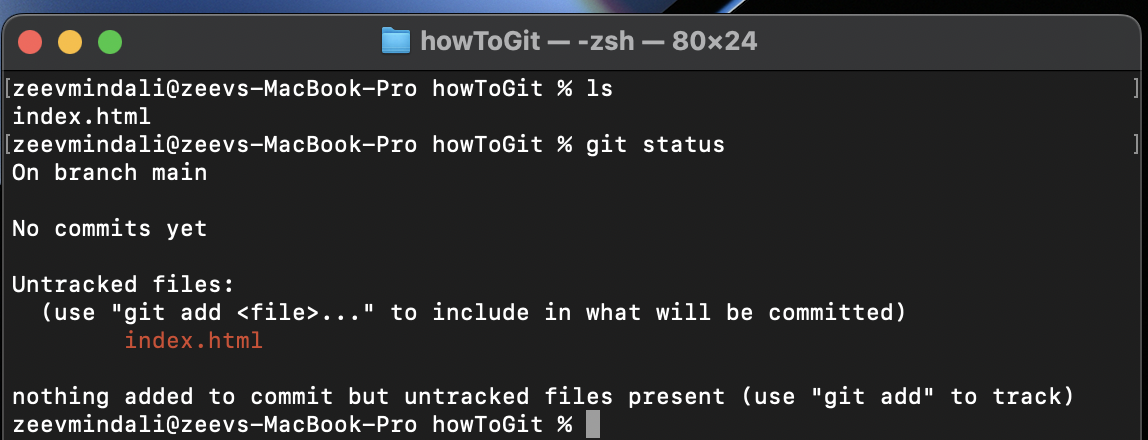
כאשר ביצענו שינוי, אנו צריכים "להוסיף" את השינויים שביצענו למעקב של Git.  
ה-Staging Copy הוא האיזור שבו נמצאים הקבצים שהועלו. יש לשים לב – עדיין לא עשינו Commit לקבצים אלו!

**Committed Copy**

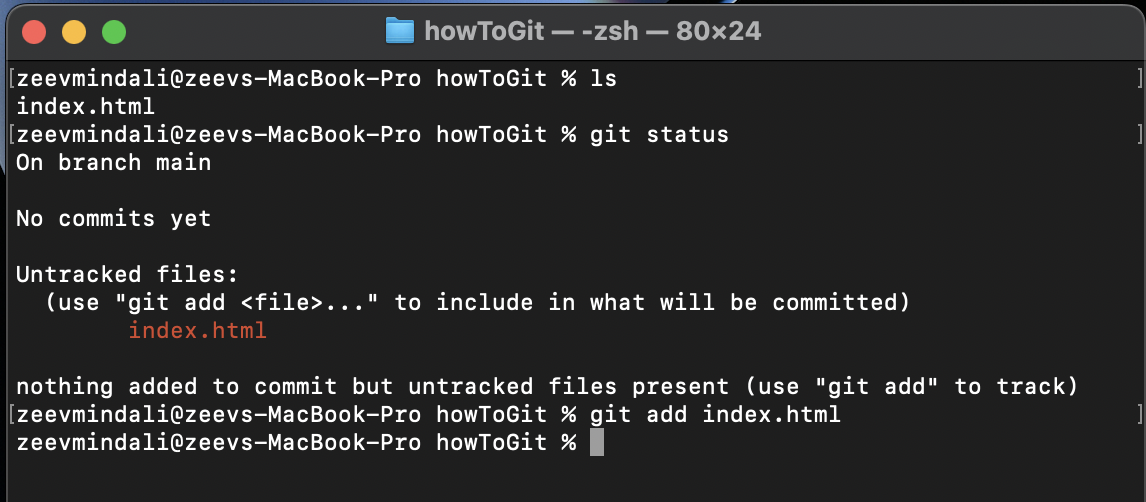
ה-Committed Copy זוהי התחנה בה נמצאים הקבצים שעברו commit  
מרגע שביצענו Commit לקבצים, השינוי "נדחף" אל ה-Repository ונוסף Commit חדש למערכת.  
במילים אחרות, נוצר "העתק" של כל הקובץ עם השינוי החדש בתוספת הודעה קצרה המפרטת על השינוי שביצענו.

**העלאת קבצי הפרויקט שלנו למאגר של Git בפעם הראשונה**

כרגע יש לנו מאגר "ריק" שרק אתחלנו אותו על ידי הפקודה git init אבל הוא איננו "עוקב" אחר הפרויקט שלנו. על מנת שיעשה זאת, עלינו להוסיף את הקבצים.  
לצורך הדוגמא, נאמר והפרויקט שלנו מכיל כרגע קובץ אחד ויחיד בשם index.html.  
באפשרותנו להבין מה קורה ב-Repository שלנו על ידי הפקודה git status:



מה בעצם אנחנו רואים כאן? git מודיעה לנו שיש לנו קבצים שהיא "לא מכירה" ("Untracked files"), הקובץ הוא כמובן, index.html.  
כרגע index.html נמצא בתחנת "Working Copy".  
כיצד נוסיף אותו ל-Repository שלנו? על ידי הפקודה git add:

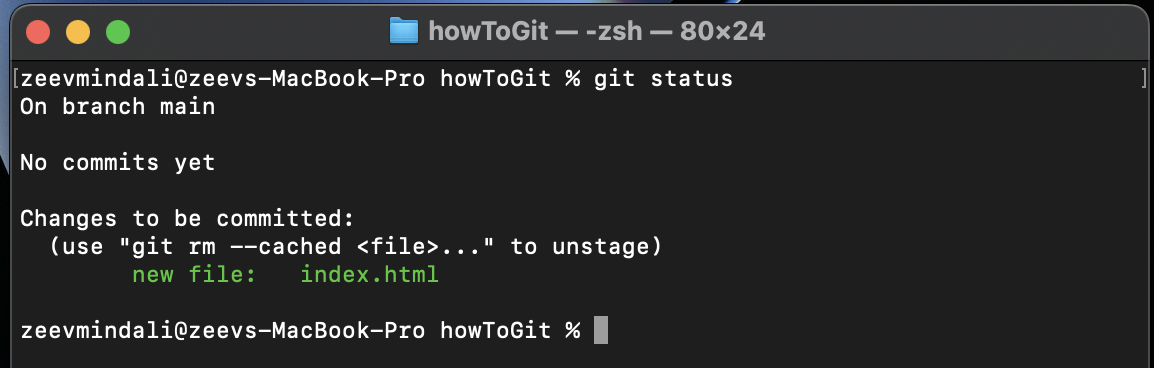


פקודת git add

מה נעשה כאשר הפרויקט שלנו יכיל מאות/אלפי קבצים? בטח שלא נרשום כל קובץ בנפרד, הפקודה הבאה תעשה זאת עבורכם:

git add .

כן, נקודה, קולטים?

כעת נבדוק את מצב ה-Repository שלנו שוב: ****

ירוק זה כבר יותר טוב   
הצבע הירקרק מבשר לנו כי הקבצים שלנו עברו "תחנה" אל Staging Copy אפשר לראות את זה כ-"חצי דרך".

**קובץ gitignore.**

סביר להניח כי ברוב הפרויקטים שלנו, ישנם קבצים/תיקיות שנרצה ש-Git לא תעקוב אחריהם והם יועלו ל-Repository שלנו, לדוגמא, אין לנו צורך בספריית node\_modules בפרויקט ריאקט.  
קובץ gitignore. הוא הפתרון.  
יצירת הקובץ מתבצעת על ידי הפקודה touch:

touch .gitignore

כעת נוכל להבחין כי Git יצרה עבורנו קובץ חדש בתיקיית הפרויקט שלנו   
(סביר להניח שהקובץ ללא שם)   
נערוך את הקובץ ונוסיף את הקבצים/התיקיות שנרצה ש-Git תתעלם מהן באופן הבא:

.node\_modules

/zeev.txt

השורה הראשונה תגרום להתעלמות מתיקייה שלמה בשם node\_modules והשורה השנייה תגרום להתעלמות מקובץ בשם zeev.txt   
ניתן ולפעמים אף רצוי להוסיף את קובץ gitignore ל-Repository שלנו, כל מקרה לגופו.

אז איך מבצעים Commit? קצת סבלנות! בפרק הבא

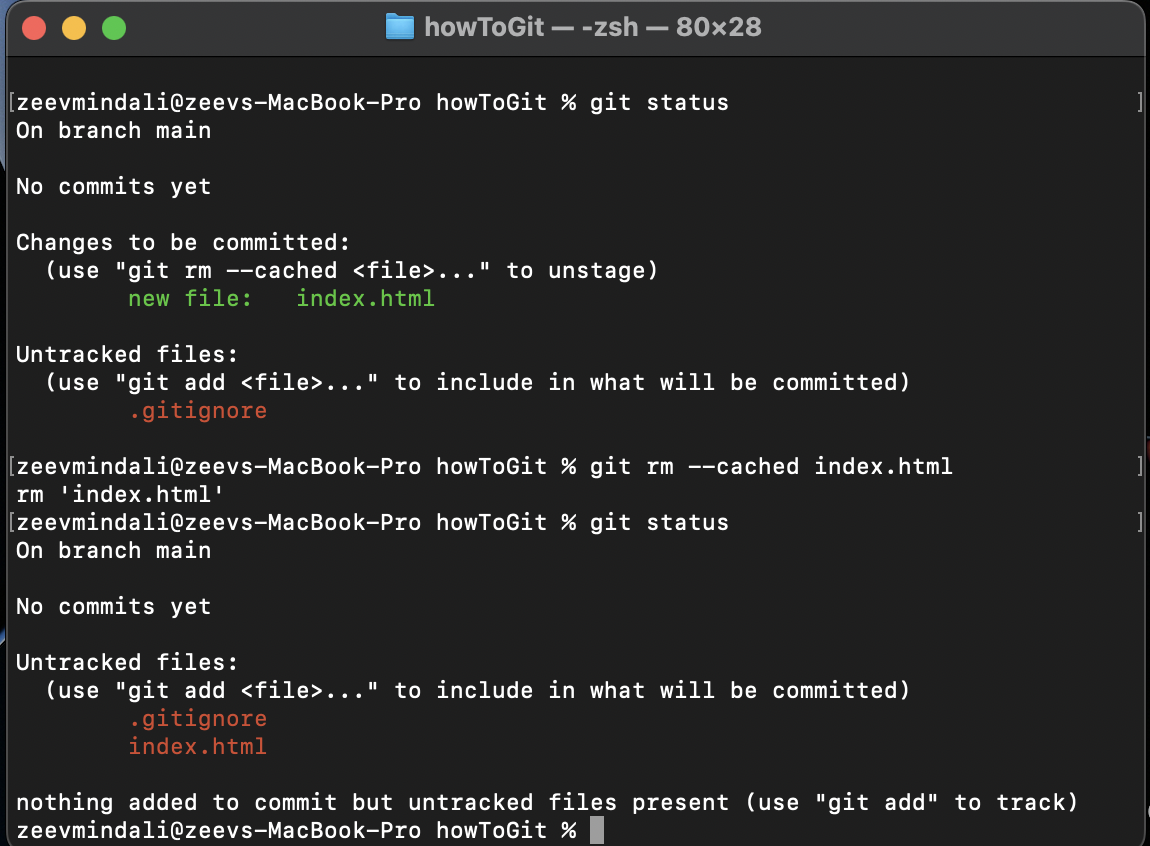
**שלב שלישי – Unstage לקובץ, יצירת Commit וצפייה בקומיטים קודמים**

**Unstageלקובץ**

זכרו, כעת יש לנו קובץ שנמצא ב-Staging Copy אך מה אם התחרטנו ואנו מעוניינים להחזיר אותו אל ה-Working Copy שלנו?  
בבקשה:

git rm --cached index.html

כעת ניתן לשים לב שחזרנו "צעד אחורה" בדרך אל ה-Commit המיוחל:



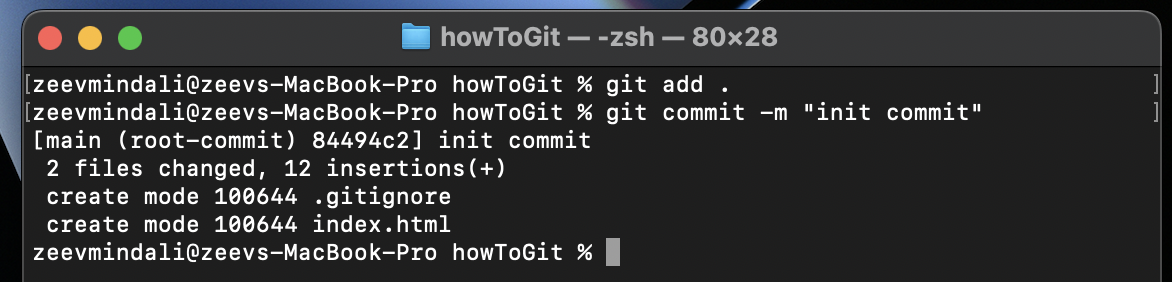
unstage לקובץ

**יצירת Commit**

כמובן שכעת, על מנת לעשות Commit נוסיף את הקובץ שוב על ידי git add.   
על מנת לבצע Commit, נשתמש בפקודה הבאה:

git commit -m "init commit"

כמובן שבין המרכאות נכתוב את מהות ה-Commit הנוכחי, מה בידיוק בוצע.



יצירת commit

**ביצוע Commit מהיר**

במידה ונרצה לבצע Commit "מהיר", ולדלג על הוספת הוקבץ על ידי פקודת git add, נוכל לעשות זאת על ידי הפקודה הבאה:

git commit -am "index.html"

**הערה:** אישית, אני פחות ממליץ לעשות זאת מאחר ואין לנו את האופציה לעשות unsgate לקובץ.

**צפייה בקומיטים קודמים**

על מנת לצפות ברשימת הקומיטים שבוצעו במאגר הנוכחי, נוכל לעשות זאת על ידי הפקודה הבאה:

git log

על מנת לצפות בקומיטים שנוצרו על ידי משתמש מסוים:

git log --author "David"

כמובן ש-zeev mindali זה שם המשתמש.

במידה ויהיו לנו מספר רב של קומיטים, סביר להניח שלא פעם נרצה לקבל רק מספר קומיטים מצומצם, למשל את ה-10 האחרונים:

git log -n 10

**שלב רביעי – השוואת גרסאות, פקודת Checkout, מחיקת/העברת קבצים ועבודה עם GitHub**

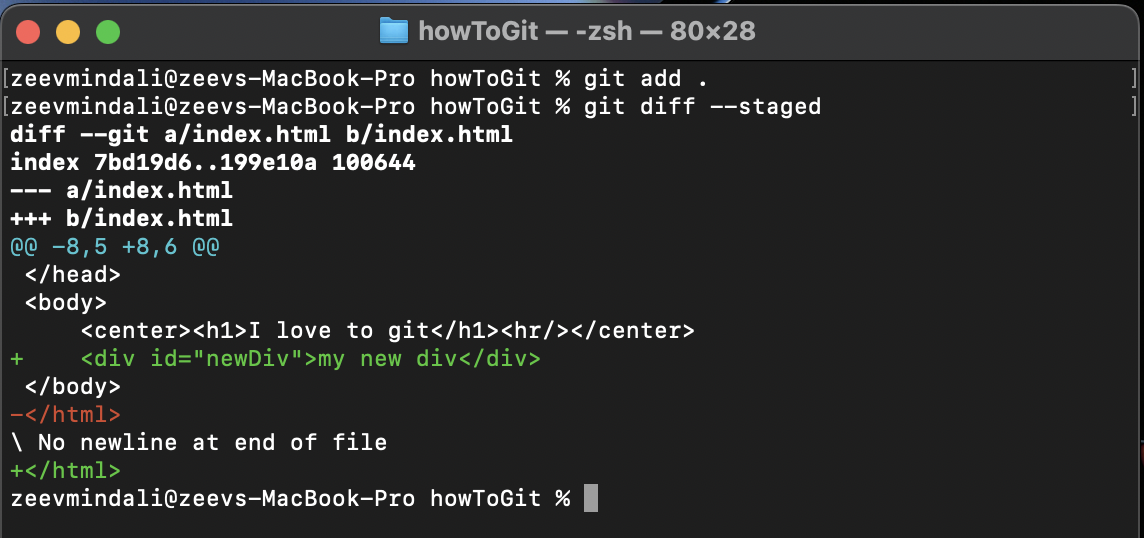
**השוואת גרסאות**

**השוואה בין ה-Working Copy ל-Staging Copy**

כפי שהזכרתי, תחנת הביניים שלנו נקראת Staging Copy, בתחנה זו נמצאים הקבצים שהוספנו אך לא עשינו להם dCommit   
במידה וביצענו שינוי ב-Working Copy, הוספנו אותו ונרצה להשוות בין קובץ הנוכחי לקובץ שנמצא במאגר שלנו, נוכל לעשות זאת עם הפקודה הבאה:

git diff --staged

לצורך דוגמא הוספתי div חדש למערכת, הוספתי את הקובץ ולאחר מכן ביצעתי את הפקודה הנ"ל, זהו הפלט:



פקודת diff --staged

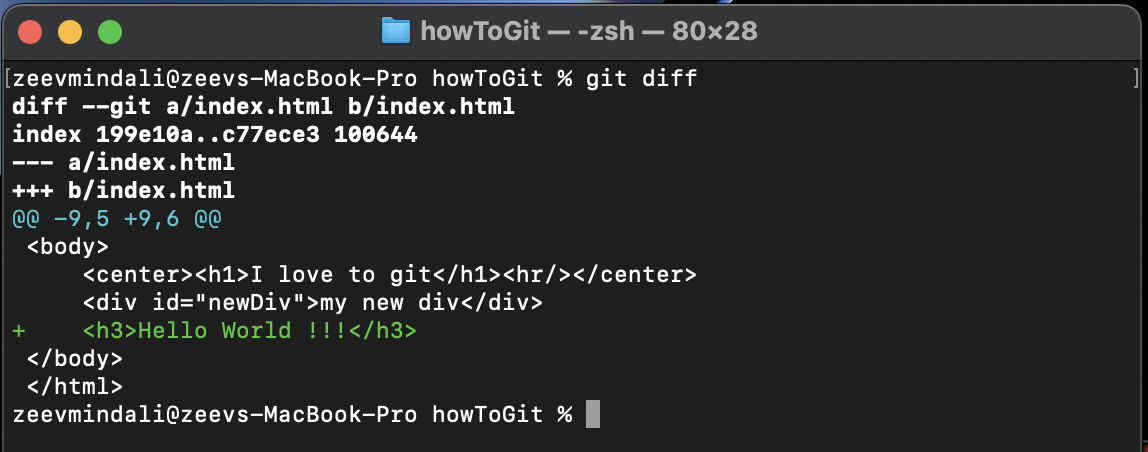
**השוואה בין ה-Working Copy ל-Repostiory**

על מנת להשוות בין הגרסא שיש לנו ב-Working Copy לבין ה-Repostiory, נשתמש בפקודה הבאה:

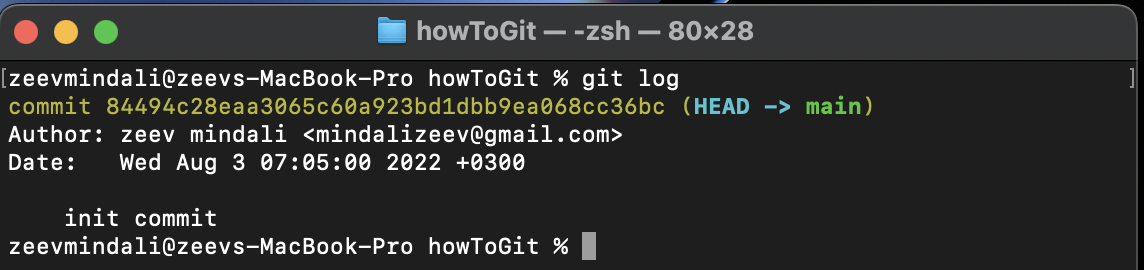
git diff

לצורך הדוגמא, הוספתי שורה לקובץ index.html, ביצעתי הוספה וקומיט ולאחר מכן את הפקודה הנ"ל, זהו הפלט:

ניתן לראות כי Git מראה לנו ששורה חדשה נוספה.

****

**פקודת checkout**

פקודת Checkout מאפשרת לנו "להחזיר" את הקובץ שלנו ל-Commit מסוים.  
כפי שהוסבר, פקודת git log תחזיר לנו את הקומיטים שבוצעו, ניתן לראות כי על יד כל קומיט ישנה מחרוזת בצהוב, זהו המזהה של אותו קומיט.  
דוגמא:

דוגמא לפלט git log

על מנת להחזיר קובץ מסוים לקומיט כלשהו, נשתמש בפקודת checkout באופן הבא:

git checkout 84494 -- index.html

כפי שרואים אין אנו נדרשים לכתוב את כל המחרוזת, מספר תוים מתחילת המחרוזת יספיקו.  
לאחר פקודה זו נוכל לראות שהקובץ חזר לקדמותו אל ה-Working Copy.  
**חשוב מאוד:**יש לשים לב שהקובץ נמצא כרגע ב-Working Copy שלנו, על מנת "לשמור" אותו יש לבצע הוספה וקומיט.

**מחיקת/העברת קבצים**

**מחיקת קובץ**

על מנת למחוק קובץ מה-Working Copy שלנו יש להשתמש בפקודה git rm באופן הבא:

git rm index.html

**העברת קובץ**

על מנת להעביר קובץ, בדוגמא זו לתיקייה כלשהי, יש להשתמש בפקודה git mv באופן הבא:

git mv index.html folder/index.html

**עבודה עם github**

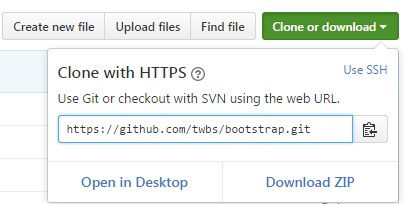
אז מהו github ? github הוא מעין רשת חברתית של Git.  
לאחר רישום קצרצר לאתר, נוכל ליצור Repostiory מרוחק ולדחוף אליו את ה-Repostiory המקומי שלנו.  
GitHub מאפשר לנו לשתף את הפרויקטים שלנו עם הציבור הרחב וכמובן לנהל את המאגר שלנו בצורה קלה.

**פקודת clone**

פקודת Clone מאפשרת לנו ליצור עותק של Repostiory מרוחק על המחשב שלנו.  
בדוגמא זו המאגר הוא של Bootstrap והוא נמצא ב-GitHub.  
על מנת לקבל העתק של המאגר נפתח את חלון ה-Bash ונשתמש בפקודה הבאה:

git clone https://github.com/twbs/bootstrap.git

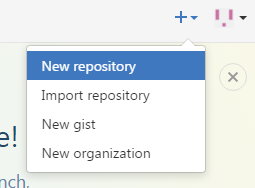
את הכתובת של ה-Repostiory העתקתי מהעמוד של המאגר ב-GitHub.



GitHub – clone url

**יצירת Repostiory ב-GitHub**

לאחר ההרשמה ל-GitHub נוכל ליצור מאגר חדש על ידי הקישור היעודי לכך:



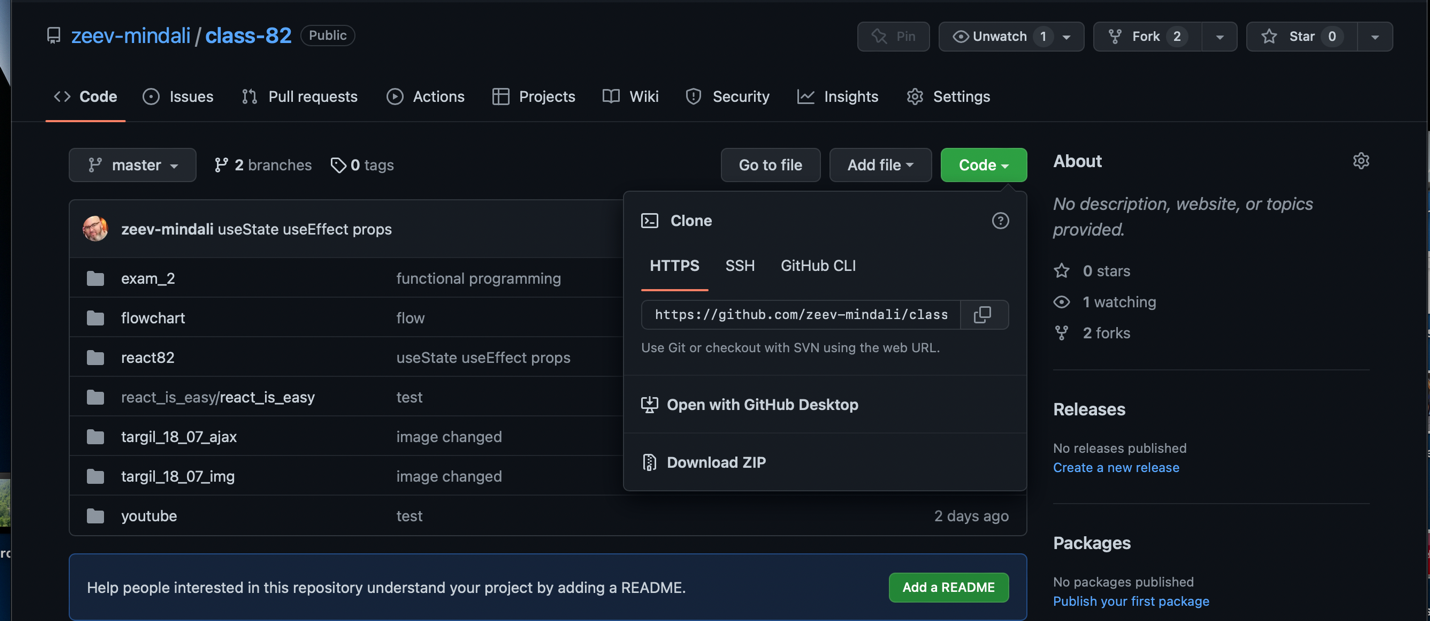
במסך שיפתח לנו נמלא את הפרטים הבאים:

repository name - השם של המאגר שלנו. מאחר ומאגר זה הוא תחת השם משתמש שלנו נוכל לבחור איזה שם שנרצה.  
description - כמה מילים על המאגר שלכם – אופציונלי.  
public/private - מאגר ציבורי או פרטי (כלומר שכולם יראו, טוב לתיקיית עבודות או פרטי פרוייקט אישי שלכם)

שליחת הטופס תיצור את המאגר, מגניב!

**שלב חמישי – פקודות push + pull , branch   
GUI Clients**

**פקודת push**

אז יש לנו מאגר מקומי ומרוחק, על מנת לדחוף את המאגר המקומי למאגר המרוחק נצטרך להגדיר את ה-Remote שלנו על ידי הפקודה הבאה:  
ניתן לקבל את הקישור ע״י גיט עצמו תחת הכפתור code

git remote add origin https://github.com/zeev-mindali/class-82.git

\*כמובן שיש לשנות את הערכים המתאימים בהתאם למאגר המרוחק שלכם.

origin הוא השם של ה-Remote שלנו.

על מנת לדחוף את המאגר המקומי שלנו למאגר המרוחק נשתמש בפקודה הבאה:

git push -u origin master

Master זהו השם של הענף הראשי שלנו.  
אז מה זה ענף?

**ענפים**

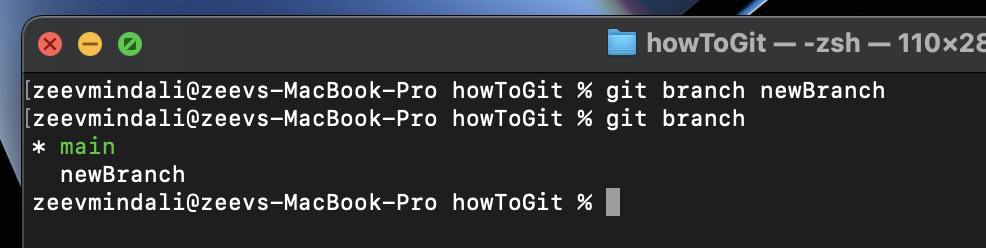
אז מהם בדיוק ענפים (branches) ?  
ענף הוא "גרסה" של ה-Repostiory שלנו.  
לצורך דוגמא, נאמר ופיתחנו מערכת כלשהי, הענף הראשי נקרא master והוא "מכיל" המערכת בגרסתה "הקלאסית".  
אם נרצה להעתיק את המערכת ולהוסיף לה יכולות מיוחדות/מותאמות אישית ללקוח כלשהו או בכלל לעבוד על גרסה חדשה למערכת, כדאי לנו ליצור ענף חדש.  
לאחר סיום העבודה ניתן גם למזג (merge) בין הענפים.

על מנת ליצור ענף חדש, נשתמש בפקודה הבאה:

git branch newbranch

על מנת לקבל את רשימת הענפים שיש ברשותינו:

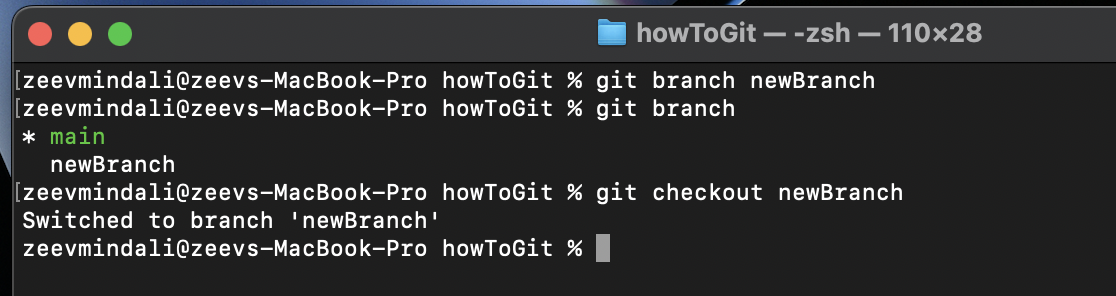
git branch



כפי שניתן לראות בתמונה, ענף אחד צבוע בירוק ובעל כוכבית. המשמעות היא שהענף הזו הוא הענף שאנו עובדים עליו כרגע.

איך נעבור לענף אחר?

git checkout newbranch



ניתן לראות כי גיט בישרה לנו כי ביצענו החלפה לענף בשם newbranch

ניתן ליצור ענף חדש ולעבור עליו באמצעות פקודה אחת בלבד:

git checkout -b version2

כיצד נמחק ענף?

git branch -D newbranch

**פקודת Pull**

לאחר שלמדנו על עבודה עם שרת מרוחק וענפים, נשאלת השאלה מה יקרה אם ביצענו clone למאגר מרוחק, ביצענו שינויים כלשהם אבל הענף המקורי (זה שבשרת המרוחק) עודכן?  
אנו בוודאי נרצה "למשוך" אלינו את השינויים.  
פקודת git pull תעשה זאת:

git pull origin master

originהוא הכינוי לשרת המרוחק שלנו וה master-זהו השם של הענף אותו נרצה "למשוך".

ניתן להוסיף את האיתחול של הרשאות המשתמש ע״י הפקודה:

git config --global credential.helper store

ולבצע משיכה מהמערכת

git pull

במידה והזנו סיסמא או שם משתמש לא נכונים, נערוך פשוט את הקובץ:

.git-credentials

**Gui Clients לגיט**

במדריך זה עבדנו עם Git Bash שהוא חלון הפקודות של גיט.  
כיום, רבים מעדיפים לעבוד עם תוכנות בעלי ממשקים ויזוליים נוחים.  
הכנתי עבורכם רשימה קצרה של כמה Gui Clients חינמיים:

**TortoiseGit**

תוכנה קלילה, די פשוטה ומאוד מתאימה למתחילים.  
יתרון נוסף שהיא גם מתממשקת עם) Windows explorer לא הדפדפן (!כלומר באמצעות המקש ימני תוכלו לבצע פעולות נפוצות.  
תומכת בחלונות בלבד.  
[TortoiseGit](https://tortoisegit.org/)

**SourceTree**

תוכנה מעולה, נפוצה מאוד, בעלת תמיכה בחלונות וMac-  
[SourceTree](https://www.sourcetreeapp.com/)

**GitHub Desktop**

תוכנה מדהימה ומומלצת מאוד מבית GitHub.  
תומכת בחלונות ו-Mac.  
[GitHub Desktop](https://desktop.github.com/)