ללמוד לעבוד עם גיט

סיכום זה מבוסס על ווביר שהשתתפתי בו של קהילת כולא לייק, בהנחיית יונתן אייזנשטיין.

להלן קישור להקלטה ביוטיוב: https://www.youtube.com/watch?v=pnik7cUrWSQ

Topics

- What is - Version control, Git, Github
- Base commands - git init, git add, git commit
- Track changes commands - git status, git diff, git log
- New Github project
- Branching - git branch, git checkout
- Understanding CR's and PR's
- Resolving conflicts
- Conventions
- Summery + QA

Version control system

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done to the code.

Usually, version controls systems record all file changes into a special "database" called a repository. There we can see all the file's history, file changes, who changed them, and even why.

Benefits

- We control all changes made to the files.
  And if needed, we can revert back the project.
- Every team member has their copy of the project,
  and can add code without overriding other members.
- Informs us about Who, What, When, Why changes
  have been made.
- All team members can access the projects
  from anywhere.

Git

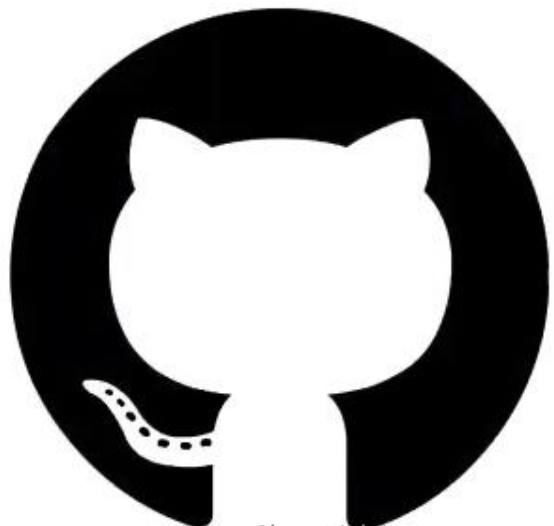
Is the most populer vesrion control system in the world.

Why?

- Free
- Open source
- Fastest
- Scalable
- Easy to learn

Github

It is the most famous platform for hosting projects using git.
And provides extra features to make our life better.



First steps

First, we need to configure git with our credentials.
that way, git will identify us and sign changes we made with our credentials

```
-> git config --global [user.name || user.email ]
```

And optional but recomanded to configure the main editor

```
-> git config --global core.editor "code --wait"
```

for configuring vscode

First steps

To start working with git, we need to create a repository for the project. the command to create an empty repository is

```
-> git init
```

we need to run this command when we are in our projects folder. Git will create an empty repo in the folder (we can see it in the .git hidden file), and git will start tacking for changes.

```
dev/efrohim/git at ⎈ docker-desktop
➜ mkdir gitExample

dev/efrohim/git at ⎈ docker-desktop
➜ cd gitExample

efrohim/git/gitExample at ⎈ docker-desktop
➜ code .

efrohim/git/gitExample at ⎈ docker-desktop
➜
```

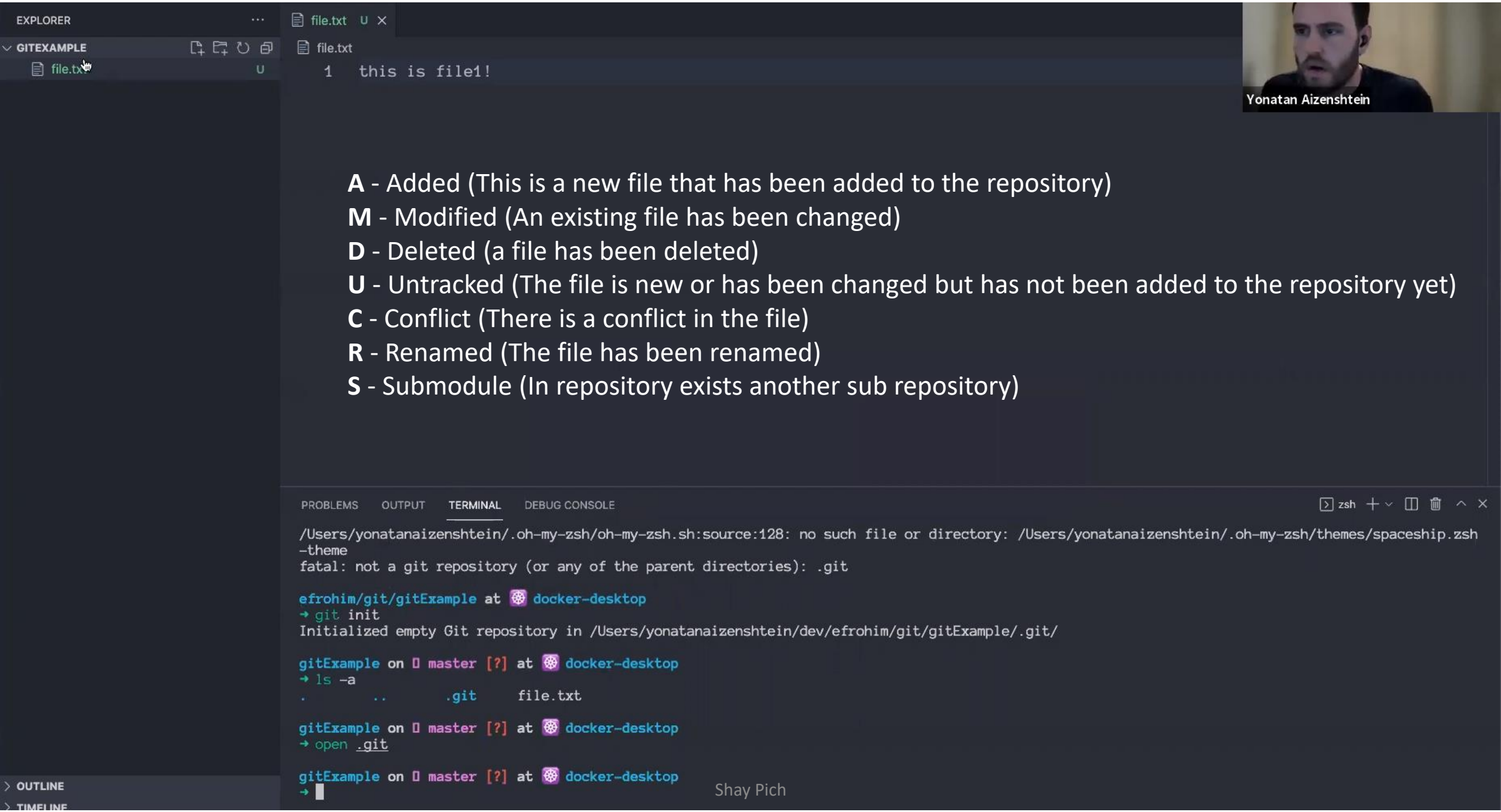
יצירת תיקיה חדשה מהקומנד ליין ופתיחתו ב- vscode. יצירת קובץ חדש בתיקיה זו.

```
efrohim/git/gitExample at ⎈ docker-desktop
➜ ls

efrohim/git/gitExample at ⎈ docker-desktop
➜ echo "file1" >> file.txt

efrohim/git/gitExample at ⎈ docker-desktop
➜ ls
file.txt

efrohim/git/gitExample at ⎈ docker-desktop
➜ clear
```

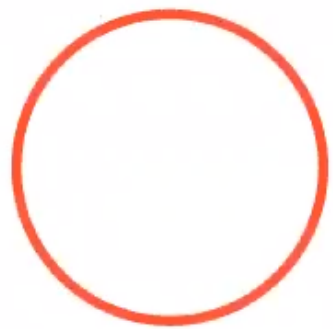

EXPLORER
GITEXAMPLE
file.txt U
file.txt U
file.txt
1 this is file1!
Yonatan Aizenshtein
A - Added (This is a new file that has been added to the repository)
M - Modified (An existing file has been changed)
D - Deleted (a file has been deleted)
U - Untracked (The file is new or has been changed but has not been added to the repository yet)
C - Conflict (There is a conflict in the file)
R - Renamed (The file has been renamed)
S - Submodule (In repository exists another sub repository)
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
zsh
/Users/yonatanaizenshtein/.oh-my-zsh/oh-my-zsh.sh:source:128: no such file or directory: /Users/yonatanaizenshtein/.oh-my-zsh/themes/spaceship.zsh-theme
fatal: not a git repository (or any of the parent directories): .git
efrohim/git/gitExample at docker-desktop
➜ git init
Initialized empty Git repository in /Users/yonatanaizenshtein/dev/efrohim/git/gitExample/.git/
gitExample on master [?] at docker-desktop
➜ ls -a
. .. .git file.txt
gitExample on master [?] at docker-desktop
➜ open .git
gitExample on master [?] at docker-desktop
➜
OUTLINE
TIMELINE

Shay Pich

Workflow

When we work on our project, we all have a similar workflow. We make changes to our files. Then we want to save them to our repository. working with git, we have another special step between the file changes to saving the into the repo called **the staging area**

Project



Staging area

Repository

Basic commands

->git add <filename>

Basic commands

-> git add file 1

אם מבצעים שינויים בקובץ המקורי אז נצטרך לעשות שוב add.

Basic commands

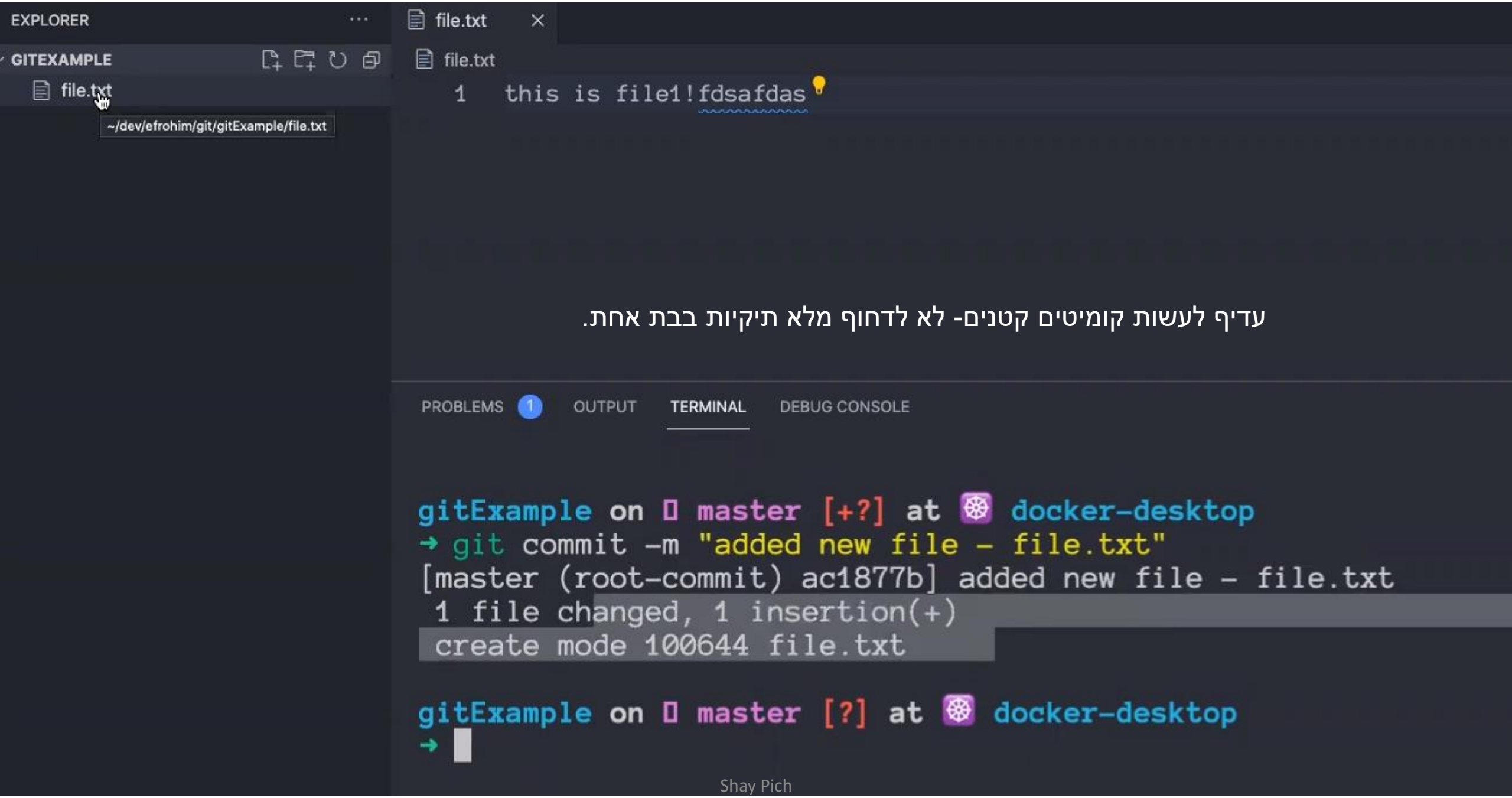

```
-> git add file 1
-> git commit -m “my message”
```

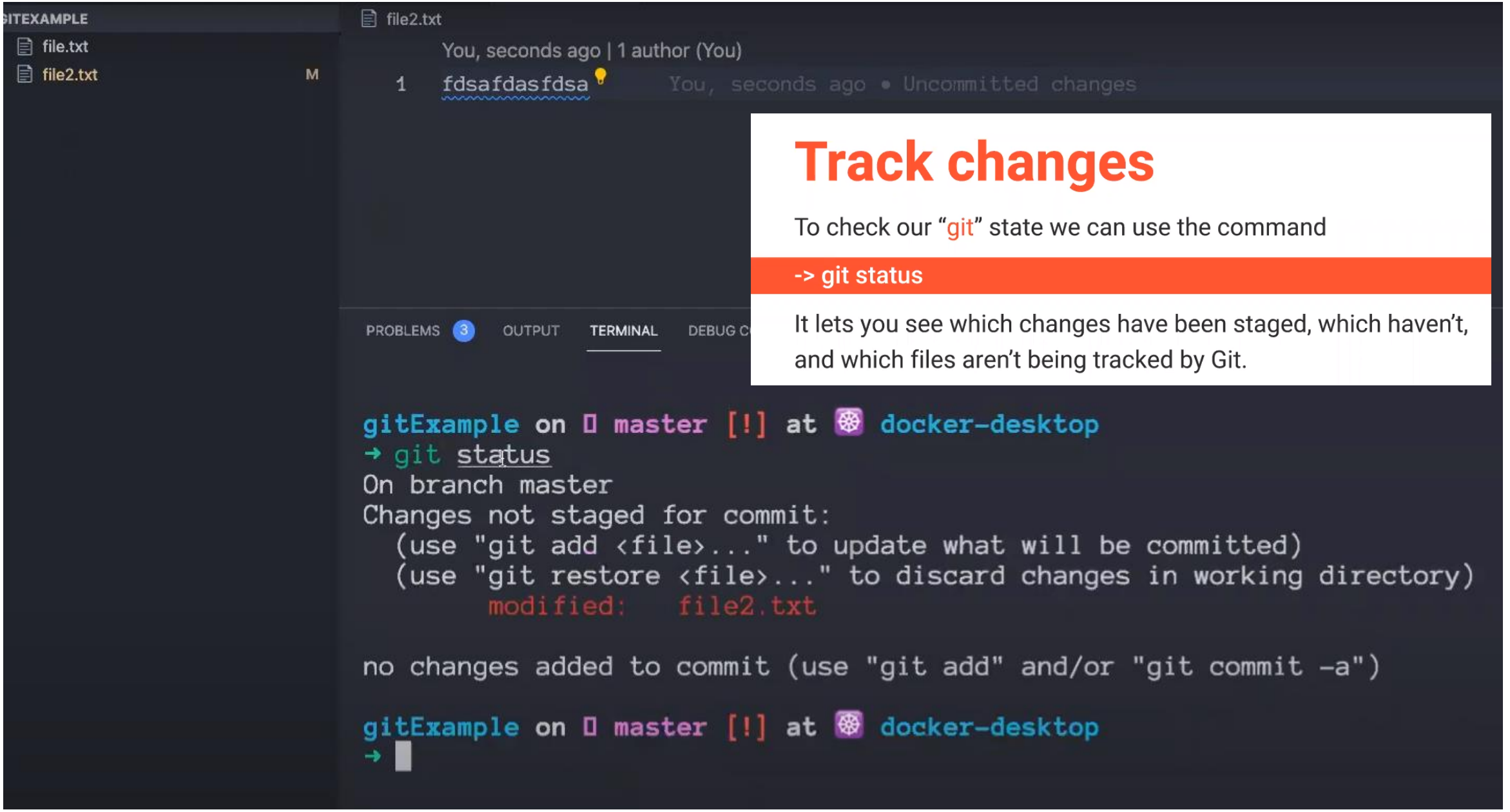
למשל "ערכנו את הקובץ". צריך שזה יהיה אינפורמטיבי.

Basic commands

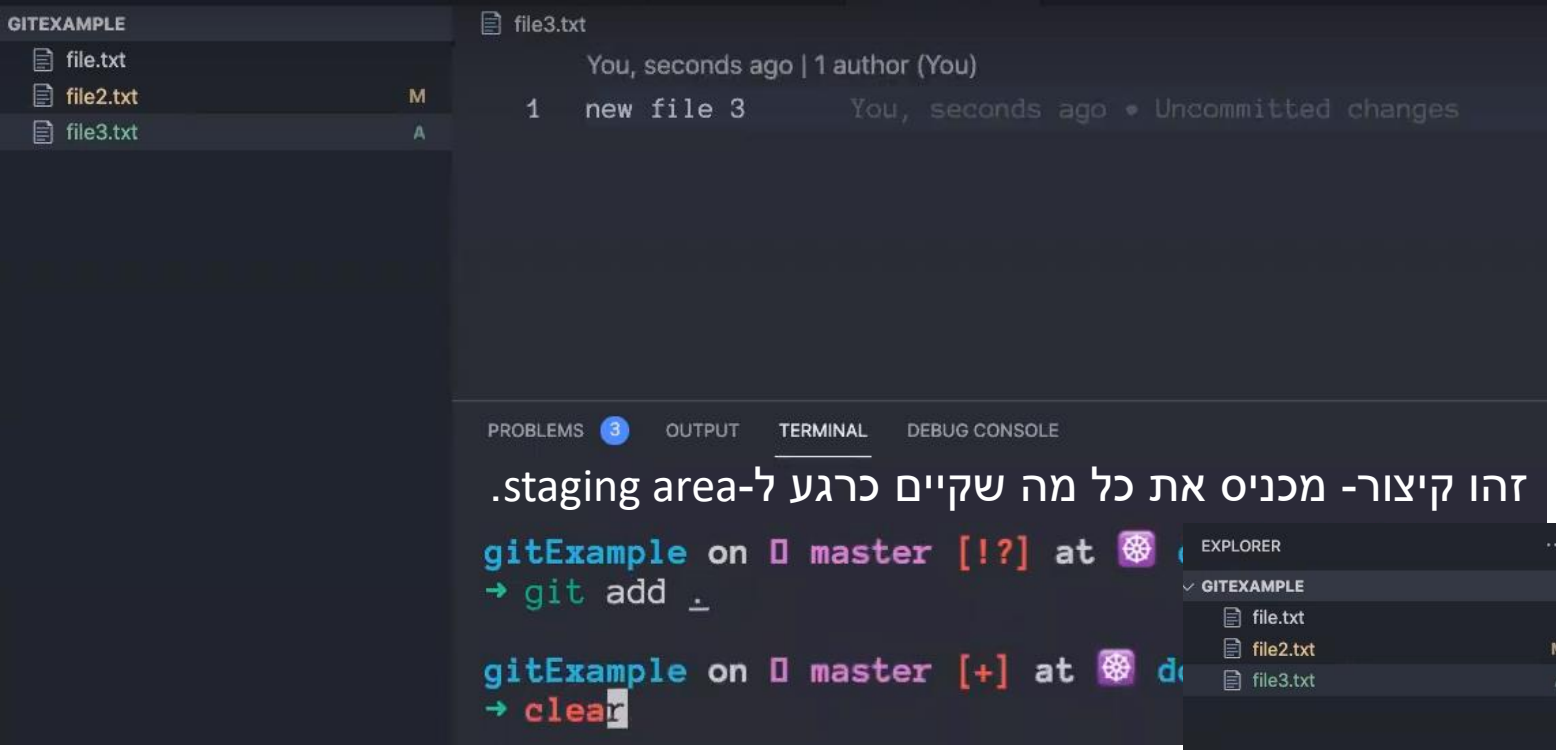
-> git add file 1

-> git commit -m “my message”

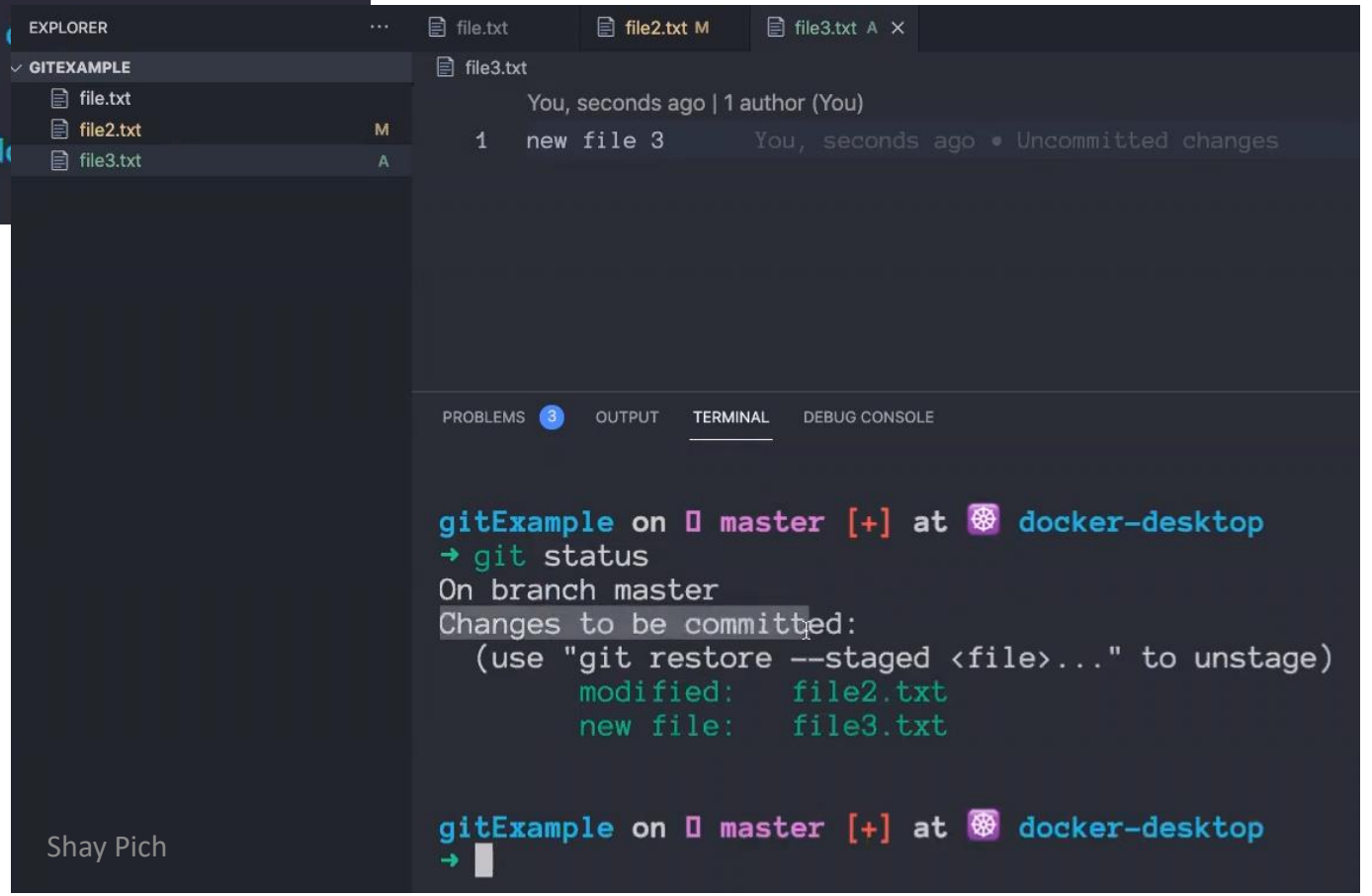

EXPLORER
GITEXAMPLE
file.txt
~/dev/efrohim/git/gitExample/file.txt
file.txt
file.txt
1 this is file1!fdsafdas
עדיף לעשות קומיטים קטנים- לא לדחוף מלא תיקיות בבת אחת.
PROBLEMS 1
OUTPUT
TERMINAL
DEBUG CONSOLE
gitExample on master [+?] at docker-desktop
➜ git commit -m "added new file - file.txt"
[master (root-commit) ac1877b] added new file - file.txt
1 file changed, 1 insertion(+)
create mode 100644 file.txt
gitExample on master [?] at docker-desktop
➜



סטטוס- בא לעדכן מה מצב הרפוסיטורי שלנו. M- modified.



זהו קיצור- מכניס את כל מה שקיים כרגע ל-staging area.

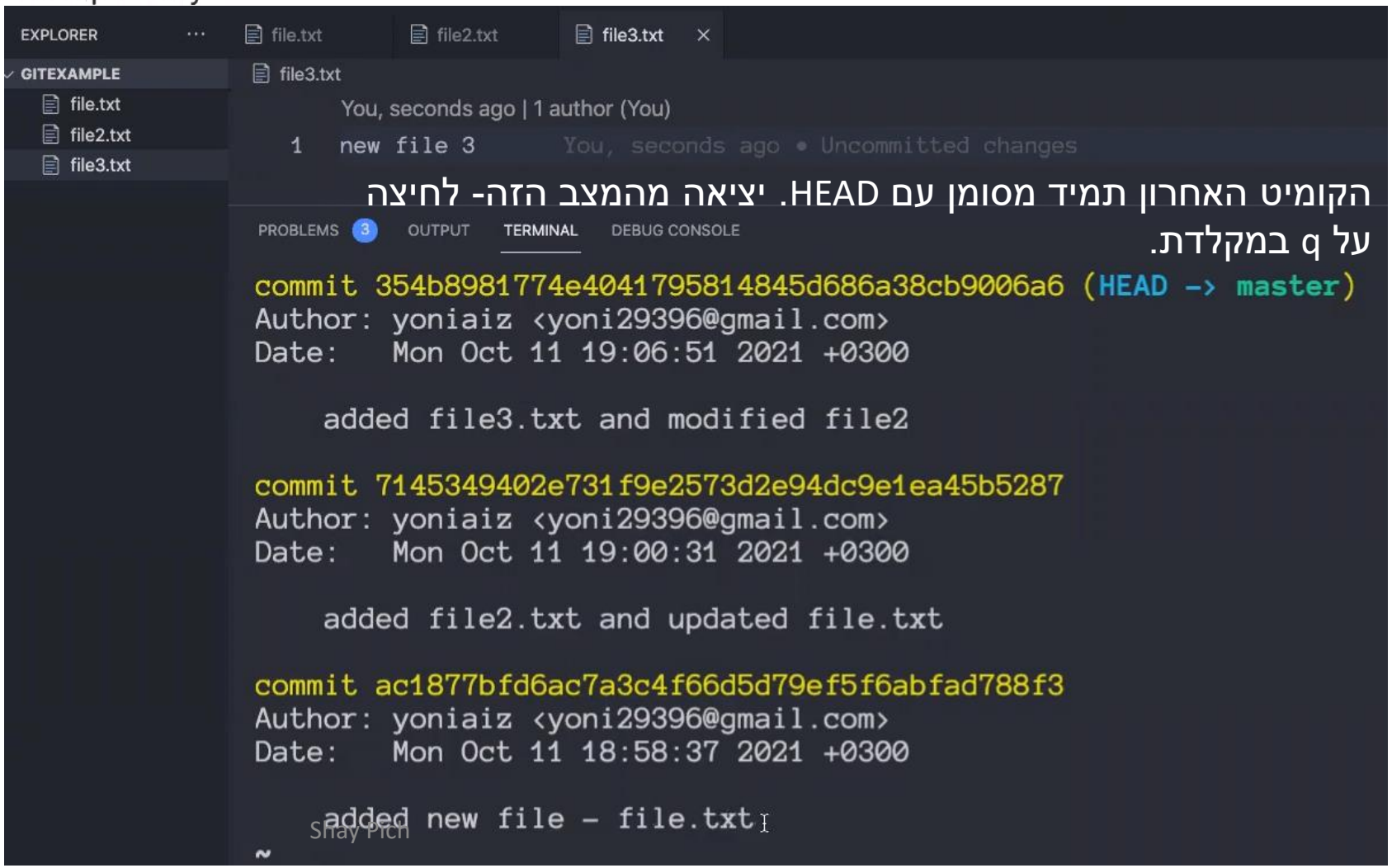


Track changes

To see all our commit we can use the command

```
-> git log
```

It will output a list with all commit history on the repository
(to exit from the command press Q)



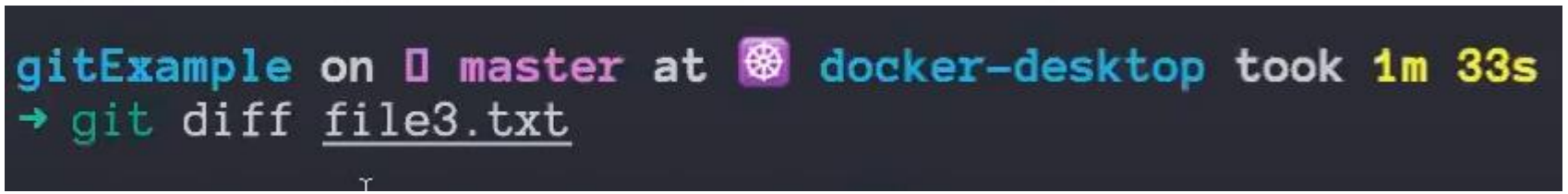
הקומיט האחרון תמיד מסומן עם HEAD. יציאה מהמצב הזה- לחיצה על q במקלדת.

Track changes

To see changes between commits we can use the command

```
-> git diff <commit hash1> <commit hash2>
```

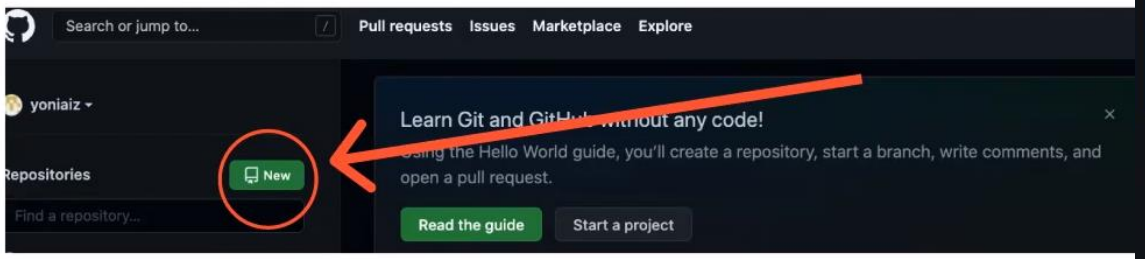
It will output all changes between the selected commits



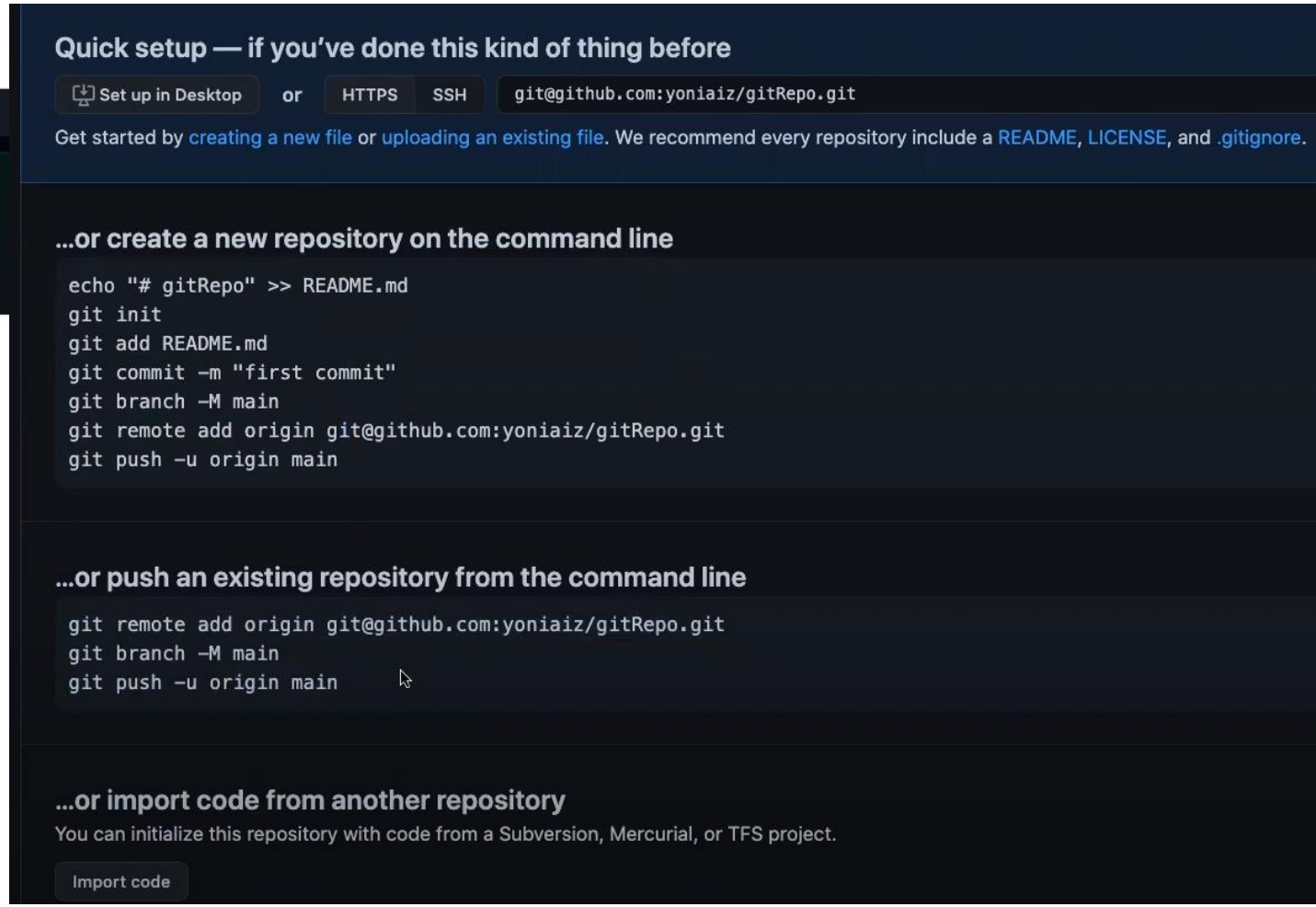
בעקרון לא שימושי מכיוון שיש לנו את גיט, אבל זה משמש לבדיקת
שוני בין היסטוריית קומיטים של קובץ מסוים.

New Github repo

To create a new Github repo we need to login into Github
And press on the New button on the left side



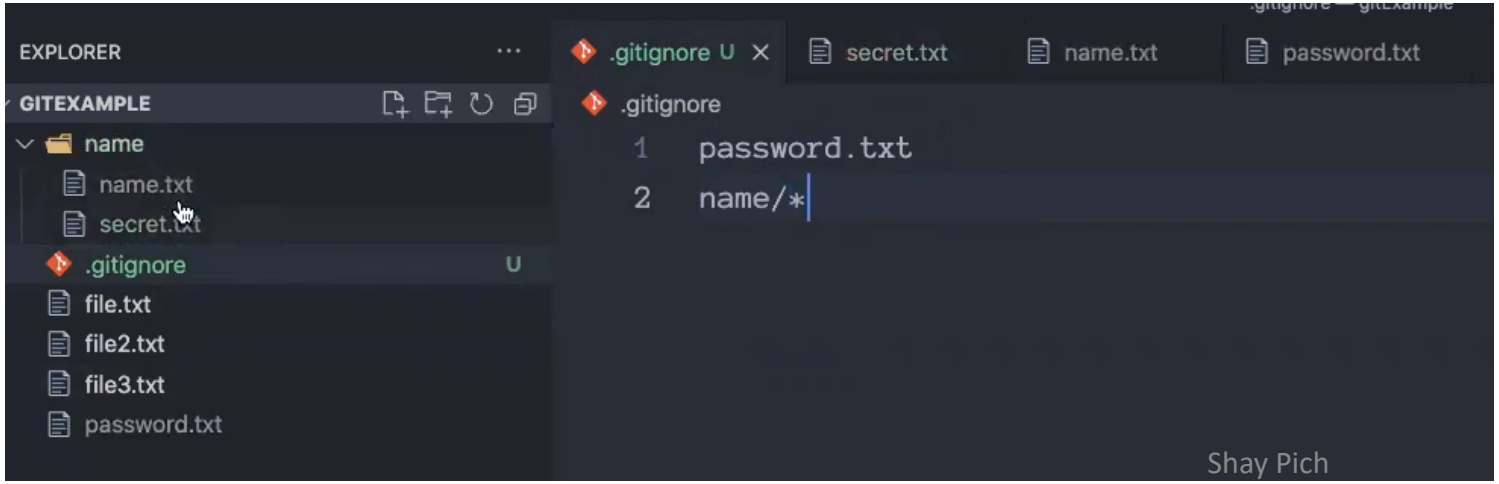
How SSH works?

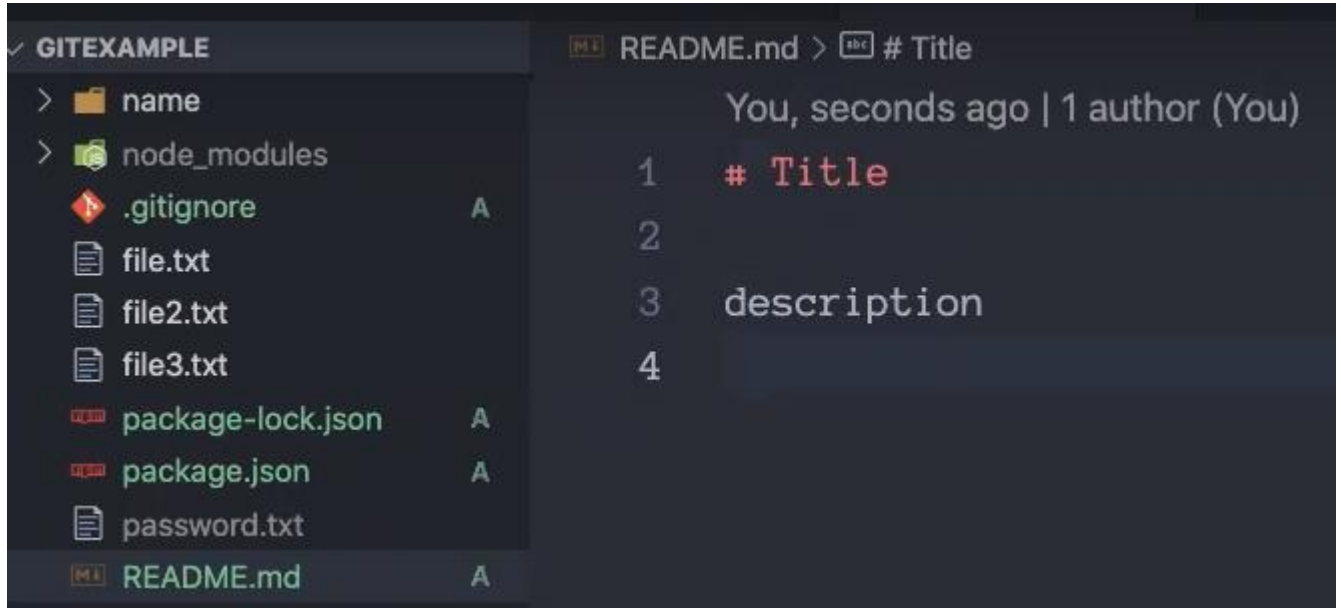


האפשרות השלישית עבור אחד שכבר קיים אצלנו במחשב- מדביקים כל פקודה אחד אחד לעורך או בשורת הפקודה.
לאחר מכן הכל כבר נראה בגיטהאב.

Ssh- בשביל שגיטהאב יזהה שהמחשב הוא באמת שלי למען תקשורת- נעשה על ידי מפתח אחד במחשב ואחד אצל גיטהאב.

gitignore- נשייך אליו קבצים שאנחנו אף פעם לא נרצה שיישמרו בתוך הרפוזיטורי שלנו, שהם יהיו רק על המחשב שלנו ולא על השרת (למשל קבצי סיסמאות או קבצים עצומים גנריים).





דחיפת שינויים לרפוזיטורי שלנו שכבר קיים- אז אין צורך כבר ב u–. כעת הכל עודכן בשרת גיטהאב.

```
gitExample on  main [⇡] is  v1.0.0 via  v12.22.0 at  docker-desktop
➜ git push origin main
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 1.08 KiB | 1.08 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.com:yoniaiz/gitRepo.git
   7eef8d1..74e413c  main -> main
```

```
gitExample on  main is  v1.0.0 vi
➜ git branch "feature/login-page"
```

```
gitExample on  main is  v1.0.0 via  v12
➜ git checkout feature/login-page
Switched to branch 'feature/login-page'
```

```
gitExample on  feature/login-page is  v1.0.0 via  v12.22.0 at
➜ git commit -am "updated file.txt file2.txt file3.txt"
[feature/login-page 72805ad] updated file.txt file2.txt file3.txt
 3 files changed, 4 insertions(+), 3 deletions(-)
```

```
gitExample on  feature/login-page is  v1.0.
➜ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

חומר חשוב. כאמור ענף ה-main שלנו הוא הענף הראשי והוא זה שמובא ללקוח, לכן הוא רגיש. אזי, נרצה שלא לעבוד עליו ישירות אלא לבחון אותו בענף צד סטרילי. -am זה קיצור לadd&comit + הודעה. השינויים לא יוצגו ב-main.

Branches

Every project is built from a lot of small features combined. In the development process, we develop one feature, then integrate it into the “main” project, and make all pieces work together.

When we develop a feature, we want to make sure we work in an isolated environment from our main project to ensure we won't affect the current working program.

Also many times we work in a team, where each member work on their feature. and we need to make sure that we don't override each others code

To help us with this process Git gives us the command

-> git branch

This command lets us create a new branch that is a snapshot of the current branch we went to a branch from. that way, we can change our code however we want, without affecting the main branch.

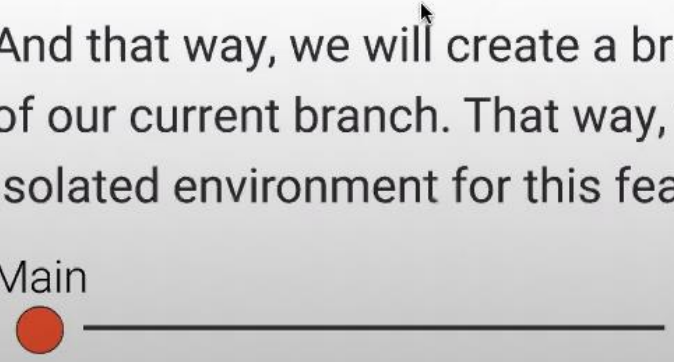
Branches

As we discussed earlier, on the GitHub repo section. We changed our master branch name to main (was optional). So all this time we already worked in a branch, called master or main. and this is our “main” branch

We want to create a new feature for our project, for example, the login/register page. But we want to make sure our main branch stays in its current state.

We can run the command

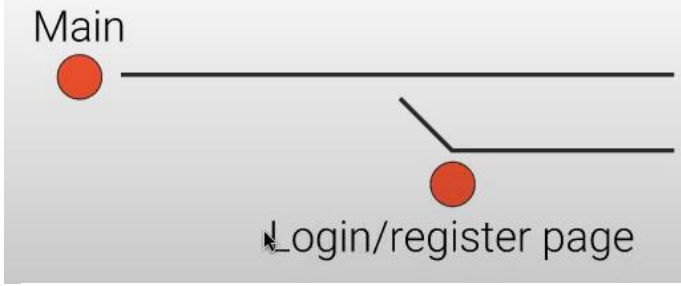
And that way, we will create a branch that is a snapshot of our current branch. That way, we will work in an isolated environment for this feature.



To start working on this new branch we need to run the command

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

To start working on this new branch, we need to run the command.



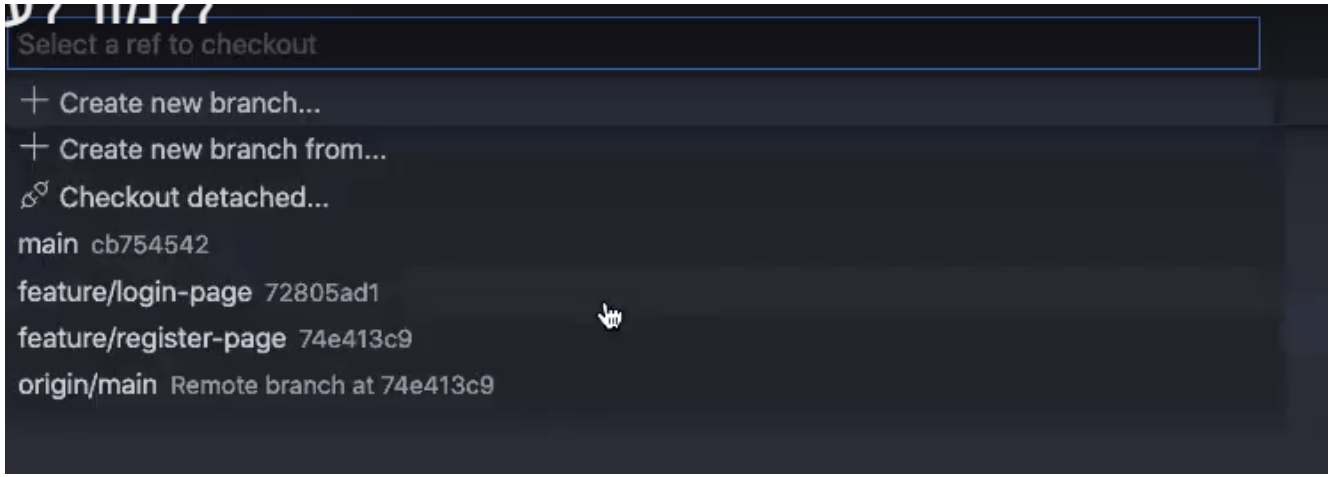
We can keep doing this process as many times that we need, and we can checkout between all branches that we created at any moment in time

To see all branches we can run the command

```
-> git branch -a
```

```
  feature/login-page
* main
  remotes/origin/main
(END)
```

הענף שקיים ברפוזיטורי בגיטהאב



צ'קאאוט ישיר שלא מהקומנד ליין- בראש העורך,
צפייה בכל הענפים שקיימים.

```
gitExample on  main [↑] is 📦 v1.0.0 via ⬢ v12.22.
→ git branch -d feature/register-page
Deleted branch feature/register-page (was 74e413c).
```

מחיקת ענף

```
→ git checkout -b "test-branch"
Switched to a new branch 'test-branch'
```

קיצור של יצירת ענף חדש ותצא אליו

מומלץ למחוק כמה שיותר ענפים ללא משמעות, זאת כדי שהיסטוריית הקוד תהיה כמה שיותר נקיה.

Merge branches

We have few options on how to merge branches.
We will talk about two options.
Option one - merge the branches locally.
Option two - open pull request

יצירת תת-ענף לענף קודם:

```
gitExample on  feature/login-page is 📦 v
➜ git checkout -b "new-login-logic"
Switched to a new branch 'new-login-logic'
```

```
gitExample on  new-login-logic is 📦 v1.0.0 via
➜ git commit -am "created new login logic"
[new-login-logic d2f0f55] created new login logic
 1 file changed, 2 insertions(+), 1 deletion(-)
```

כעת נרצה למזג בשלבים לאחר שינויים בקוד: רושמים את תת הענף שרוצים למזג.

```
gitExample on  feature/login-page is 📦 v1.0.0 v
➜ git merge new-login-logic
Updating fbebb1c..d2f0f55
Fast-forward
 file2.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

ונשאר רק למחוק את תת הענף שכבר לא שימושי:

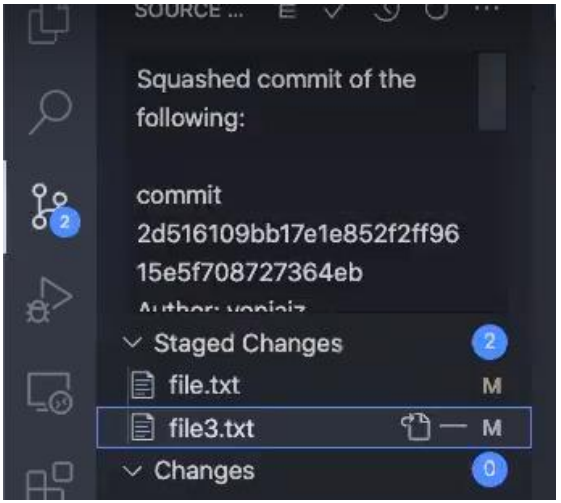
```
gitExample on  feature/login-page is 📦 v1.0
➜ git branch -d new-login-logic
Deleted branch new-login-logic (was d2f0f55).
```

הערה: צריך להיות על הענף שאנו רוצים למזג אליו כדי למזג ענפים לתוכו, ואז רושמים איזה אני רוצה למזג.

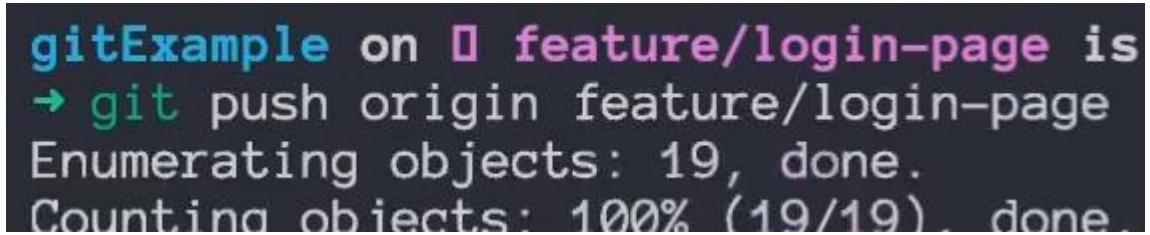
נגיד ועשינו שני קומיטים או יותר בתת-ענף מסוים. במידה ורוצים למזג את הענף בצורה נקיה בלי כל הקומיטים שיצרנו- פשוט למחוק את כולם לקומיט אחד של המיזוג- נרשום:

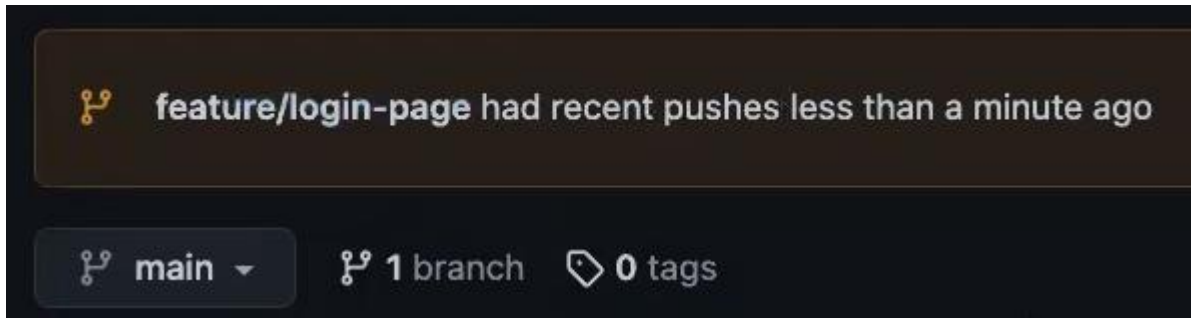
```
gitExample on  new-logic-v2 is 📦 v1.0.0 via v1
➜ git merge --squash new-logic-v2
Updating d2f0f55..2d51610
Fast-forward
Squash commit -- not updating HEAD
 file.txt  | 3 ++-
 file3.txt | 3 ++-
 2 files changed, 4 insertions(+), 2 deletions(-)
```

כעת גיט מאפשר לנו לעשות את הקומיט שלאחר המיזוג עם הודעה גנרית שכבר מופיעה:

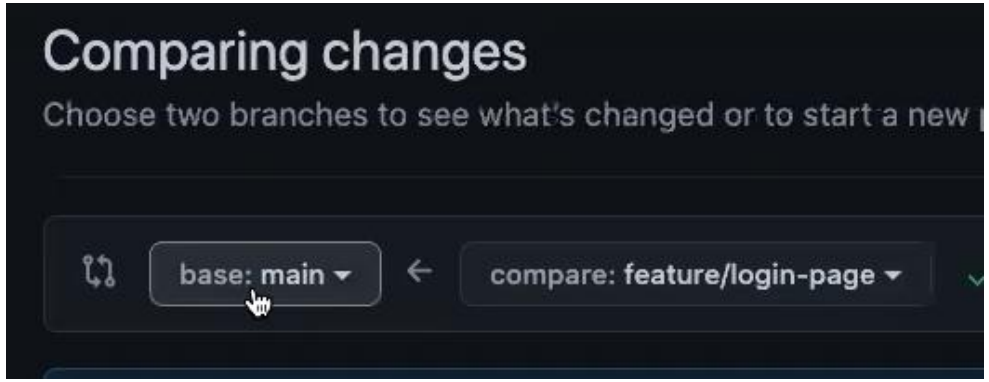


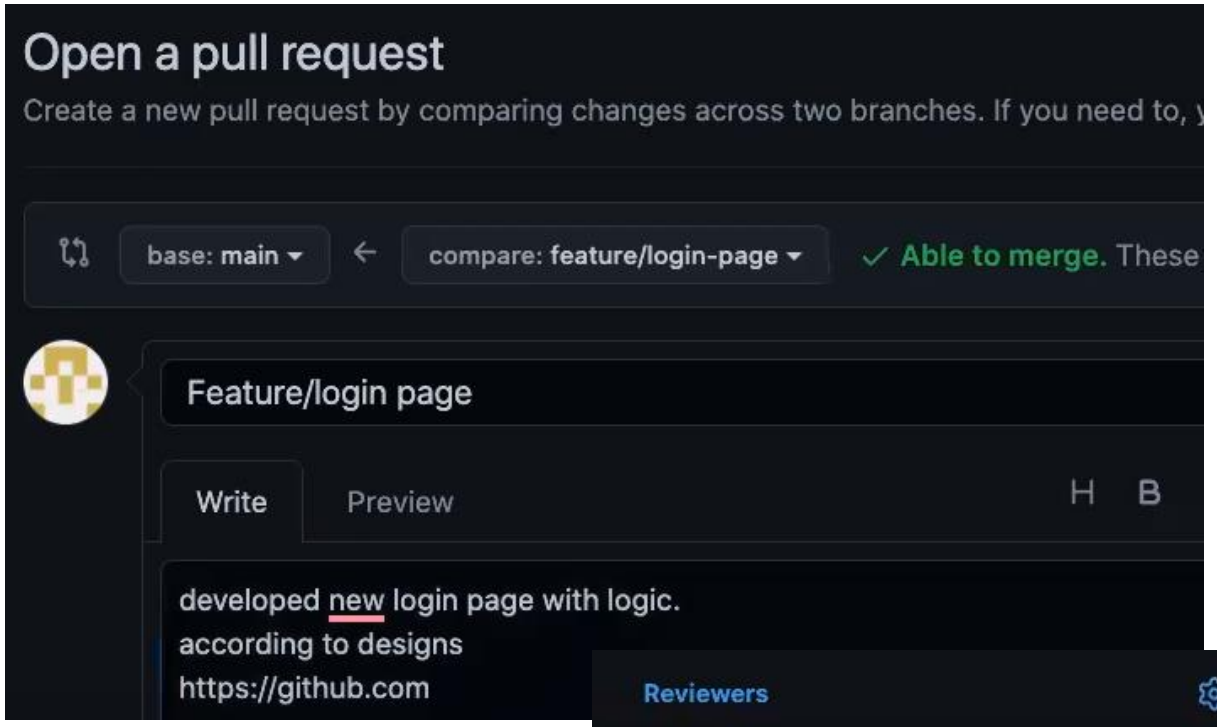
או הודעה משלנו:



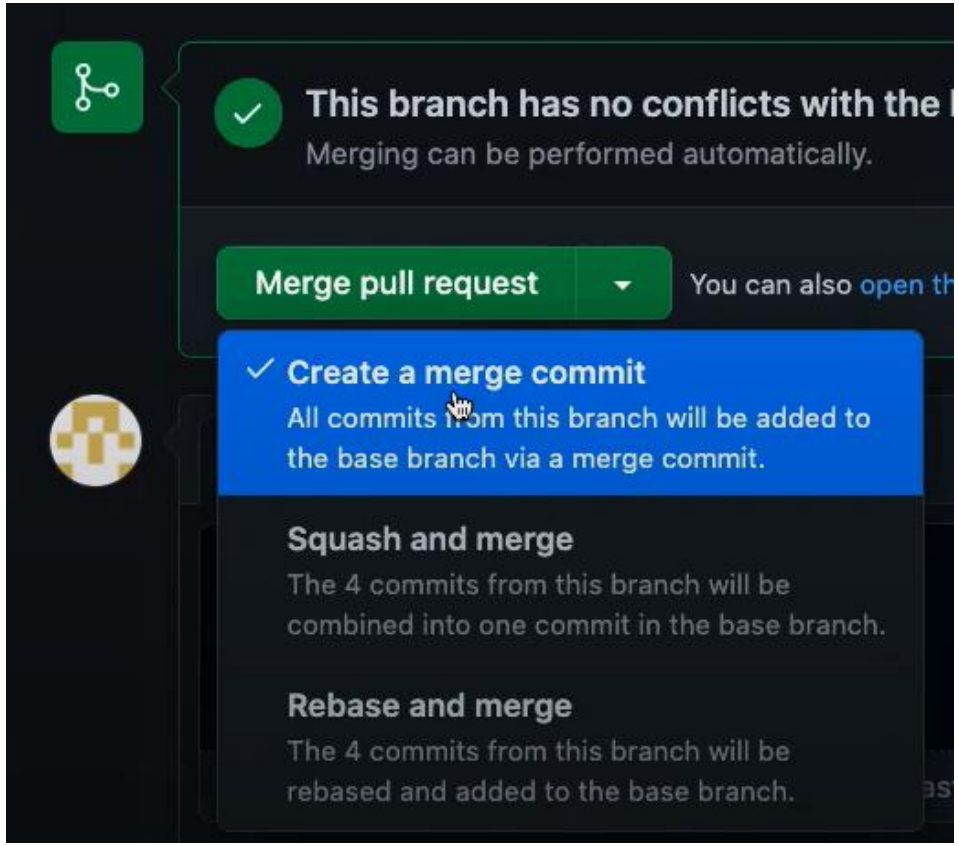


בפול ריקווסט: מה שרוצים למזג לאן למזג



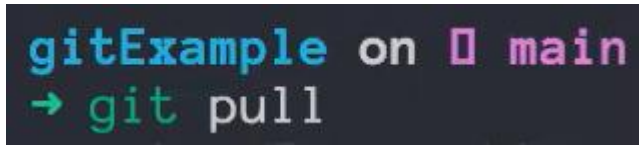


כאמור בסביבות עבודה כאשר אנו רוצים למזג ענפים לענפים ראשיים נצטרך לעשות pull request. זה מאוד נפוץ בפרויקטים של קוד פתוח למשל כדי שלא נהרוס קוד. לאחר קבלת אישור ניתן למזג את הקוד.

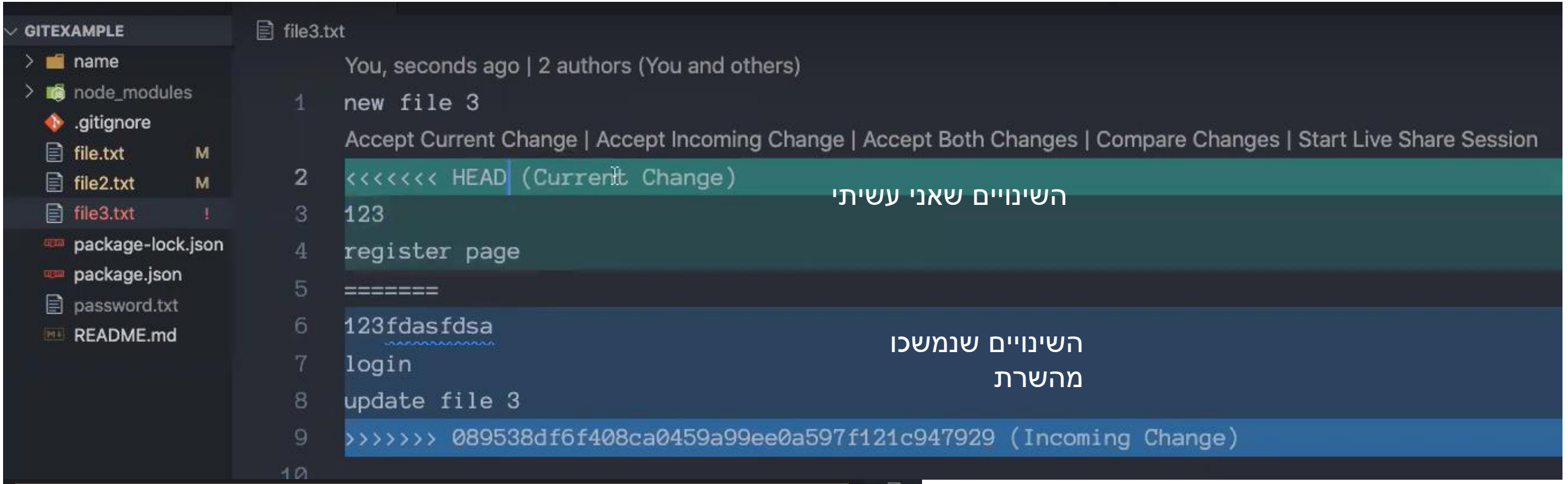


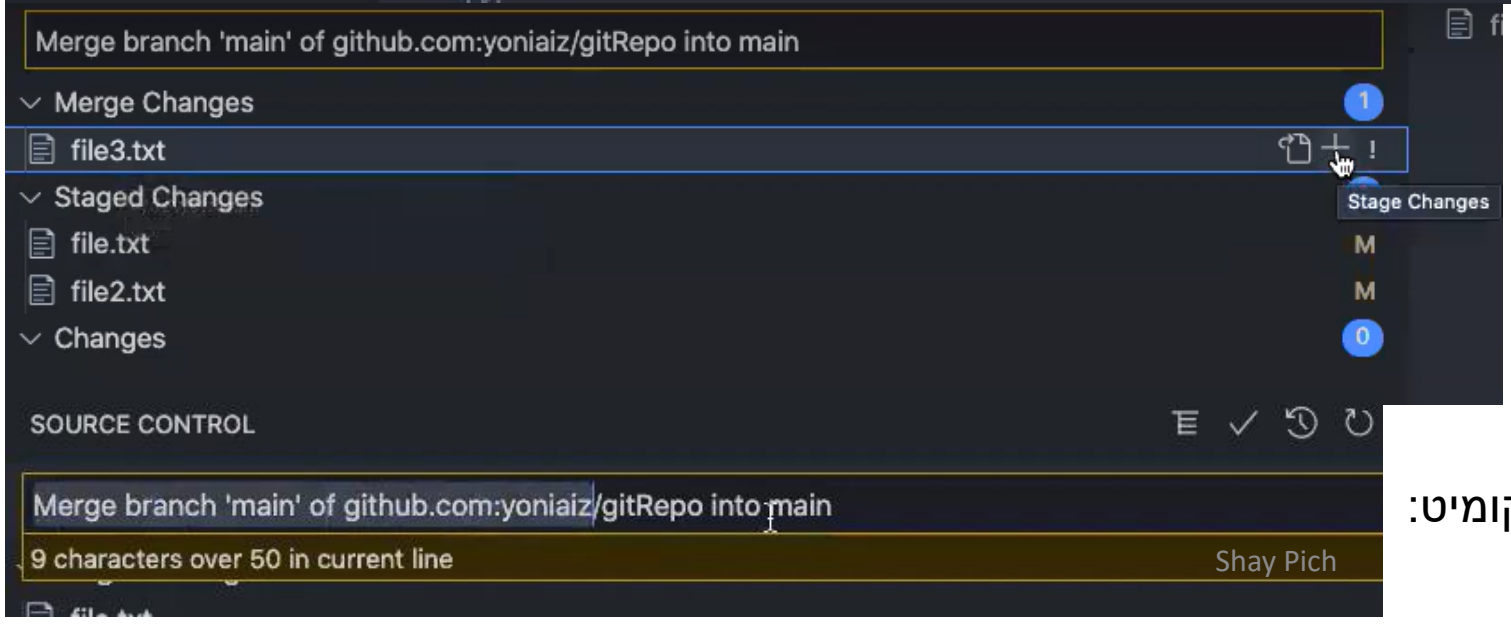
המרג' הרגיל- הכנסת כל הלוגים שהיו על הענף לענף הרצוי

הכנסת כל הקומיטים שעל הענף לקומיט אחד ודחיפה לענף



כעת נשאר לסנכרן בין מה שבשרת לבין מה שבמחשב:

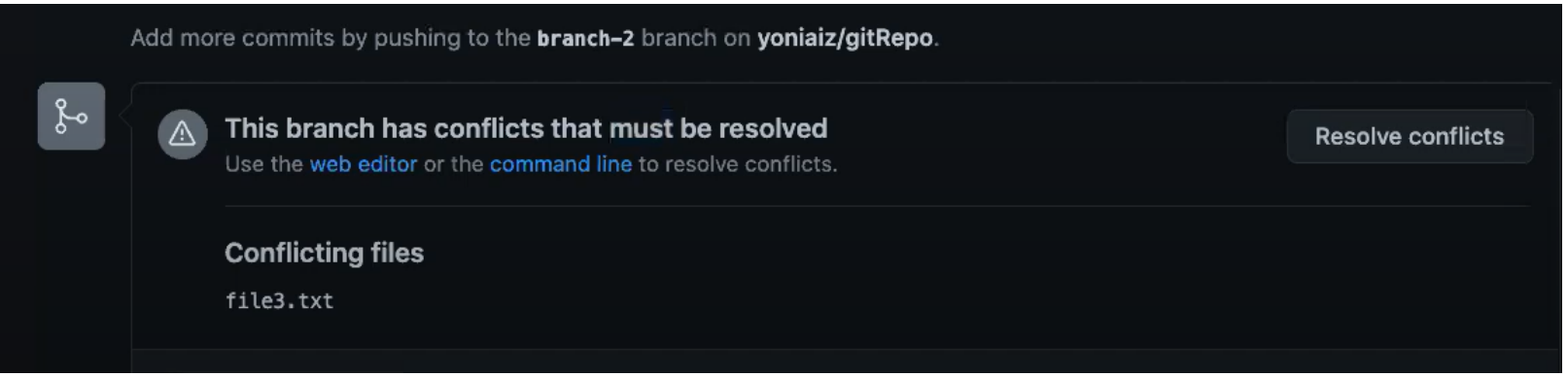




אחרי שבוחרים אחת מאופציות הקונפליקט,
מכניסים לסטייג':

קונפליקט- דיסוננס בין שינויים בשרת למחשב.

ולבסוף עושים קומיט:



נגיד וביצענו שינוי בתיקיה כמה פעמים ולכל שינוי עשינו פול ריקווסט, אם אחד מהם אושר אז לפול ריקווסטים האחרים יווצר קונפליקט. לא כדאי לפתור אותו מהגיטהאב! למה?
כי זה פחות נוח וגם כי אנו רוצים לבדוק שפתרון הקונפליקט עובד! ואז נוכל לדחוף אותו.

(בשני ענפים ערכנו את אותה שורת קוד גיט לא יוכל להחליט מי שורת הקוד הנכונה)

נפתור את הקונפליקט- נייבא את הענף העיקרי ואז נוכל לראות מוחשית את הקונפליקט ולפתור אותו במחשב:

```
→ git pull origin main
```

```
login
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes | Start Live Share Session
<<<<<<< HEAD (Current Change)
fdasfearwfeasfea      You, 3 minutes ago • commit message
=======
something
>>>>>>> f5015d3c39790a46d01220d1674bf8864774f40e (Incoming Change)
```

מעבירים לסטייג' לאחר בחירת האפשרות המתאימה (ע"י ה+ בצד שמאל כמו מקודם) ומבצעים קומיט בהתאם:

```
gitExample on  (git)-[branch-2|merge]- [!] is
→ git commit -am "fixed conflict with origin main"
[branch-2 37d54f7] fixed conflict with origin main
```

```
gitExample on  branch-2 is
→ git push origin branch-2
```

ודוחפים לשרת:

אם מוחקים שורה למשל, עושים קומיט ופול אז השרת של גיט יזהה שהקוד אצלנו "חזר בזמן"
ואז בעצם יקבל את השינוי שעשינו

Conventions - commits

For commit, we have some guidelines to how we should write a good and informative message, to help us in the future. If we want to add a body to the message - recommended but optional
we can run git commit without flags, and it will open our editor in edit commit mode.

כעקרון כל צוות בוחר את הקונבנציות שלו ושיטת העבודה, אבל יש קונבנציות כלליות.

type(scope): subject

BODY
FOOTER

Type

feat - A new feature
fix - A bug fix
docs - Changes in documentation
style - Style changes, formatting, missing semicolons or whitespaces
refactor - code changes that neither fixes a bug or adds a feature
perf - changes that improve performance
test - Add missing tests
chore - changes the build process

Scope

A scope is provided in parentheses after a type. A scope is a phrase describing parts of the code affected by the changes. For example "(userservice)".

Subject

The subject contains a short description of the applied changes

Body (optional)

Must begin one blank line after the description. Can provide additional contextual information.

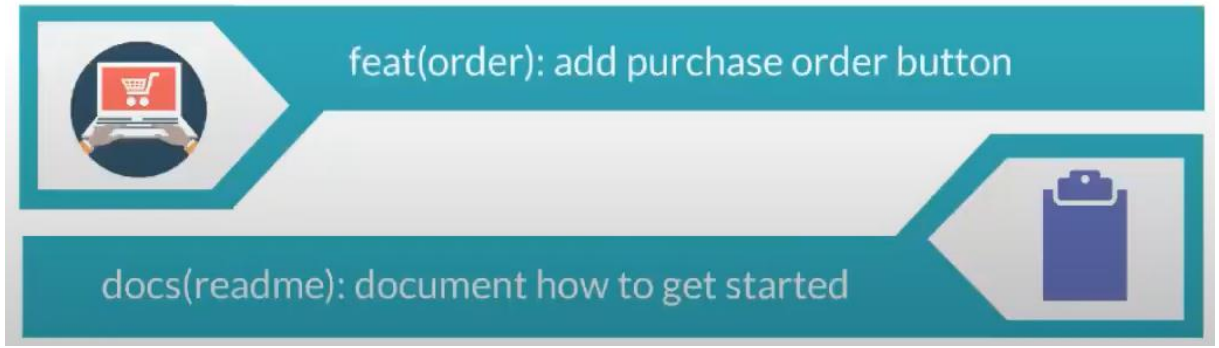
Used to indicate breaking changes. For breaking changes the Body must start with "BREAKING CHANGE".

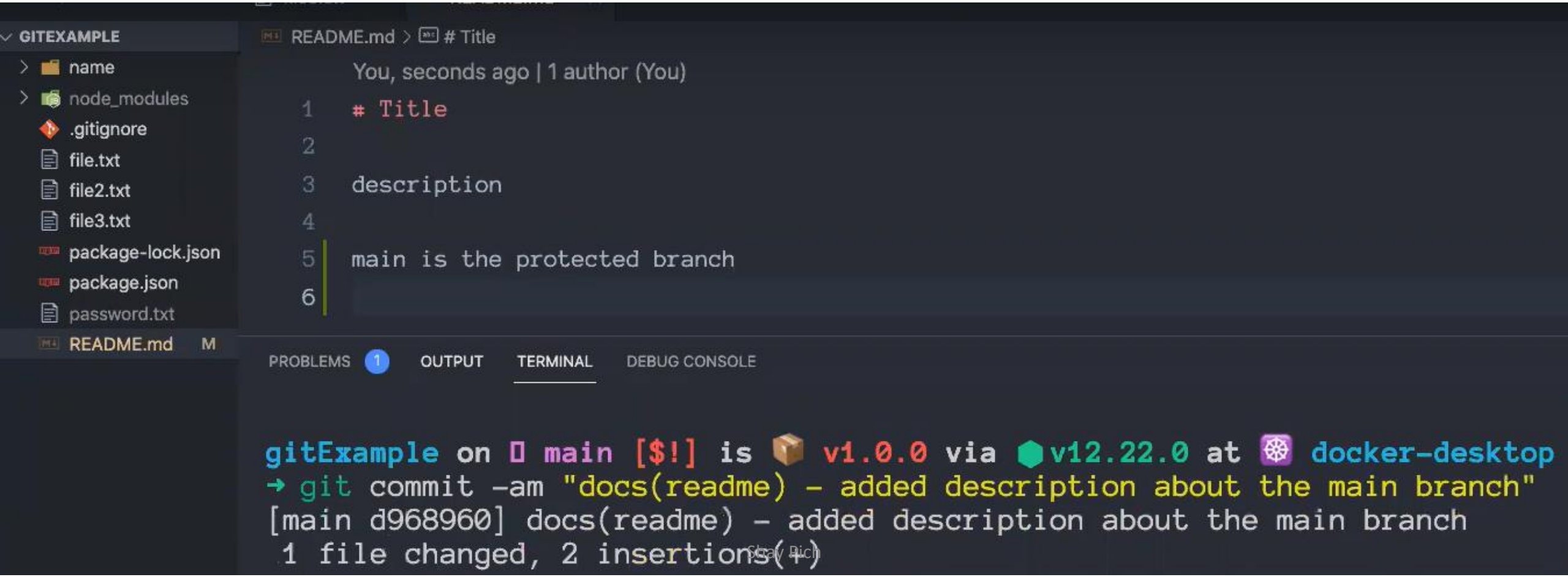
Footer (optional)

Use to reference issues affected by the code changes. For example "Fixes #13"

Can also be used to indicate breaking changes by starting with "BREAKING CHANGE".

דוגמאות.


Example commits
feat(order): add purchase order button
docs(readme): document how to get started


GITEXAMPLE
name
node_modules
.gitignore
file.txt
file2.txt
file3.txt
package-lock.json
package.json
password.txt
README.md M
README.md > # Title
You, seconds ago | 1 author (You)
1 # Title
2
3 description
4
5 main is the protected branch
6
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
gitExample on main [$!] is v1.0.0 via v12.22.0 at docker-desktop
➜ git commit -am "docs(readme) - added description about the main branch"
[main d968960] docs(readme) - added description about the main branch
1 file changed, 2 insertions(+)

Conventions - commits

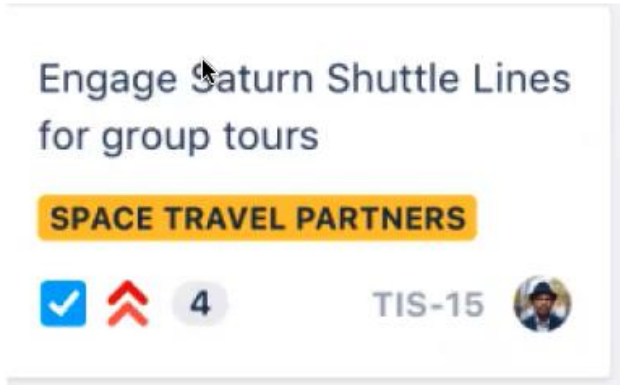
Usually, we work with products like Jira/Trello or even GitHub issues, tickets or tasks created there and assigned to the developers.

Every ticket is created with ID, and it's a good practice to start your commit message with the ID of the ticket.

When we try to debug the commits, we can find the related tickets/tasks in our product and get more details on what has been done.

Conventions - commits

Jira ticket example



The commit message will look something like this:

ג'ירה- פלטפורמת ניהול משימות.

Also If the project is connected with GitHub it will track changes inside the product.

```
gitExample on  main [$] is 📦 v1.0.0 via ⬢ v12.
➜ git commit -am "TIS-15 feat(bla) - bla"
[main 64d7739] TIS-15 feat(bla) - bla
 1 file changed, 3 insertions(+), 1 deletion(-)
```

Conventions - branches

Branch types

Regular/ Long-running branches - branches that not deleted

from the repository And part of the day to day flows

- master/main - the main branch that contains production ready code
- develop (not used in all branching stategies) - containes all development code, get updated by the developers team
- QA (not used in all branching stategies) - Test branch, used by the QA team

אחד מתחת לmain

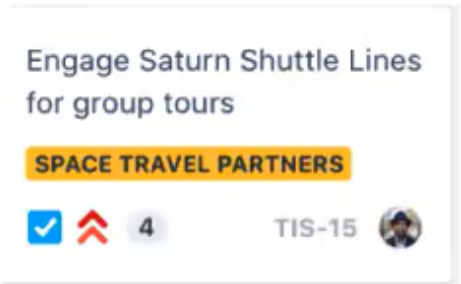
Branch types

Temporary / Short-lived branches - Used for development and deleted when the branch is merged

- Feature branch - used for developing features
- Bug fix - used to fix general bug
- Hot fix - used to fix urgent bug, gets merged to master
- WIP branch or Experimental branches
- Release branches - code ready for release to production

Conventions - branches

A branch name should be informative and short.
The namings conventions can change but are similar to commits - many workplaces use Jira (or similar products) tickets Ids to prefix the branch name - GitHub will also automatically link the branch to the issue (if configured).
For example: TIS-15-engage-Saturn-shuttle. Also, sometimes workplace conventions it to add the developer's name:
Yonatan-TIS-15-engage-Saturn-shuttle



Conventions - branches

Branching strategies

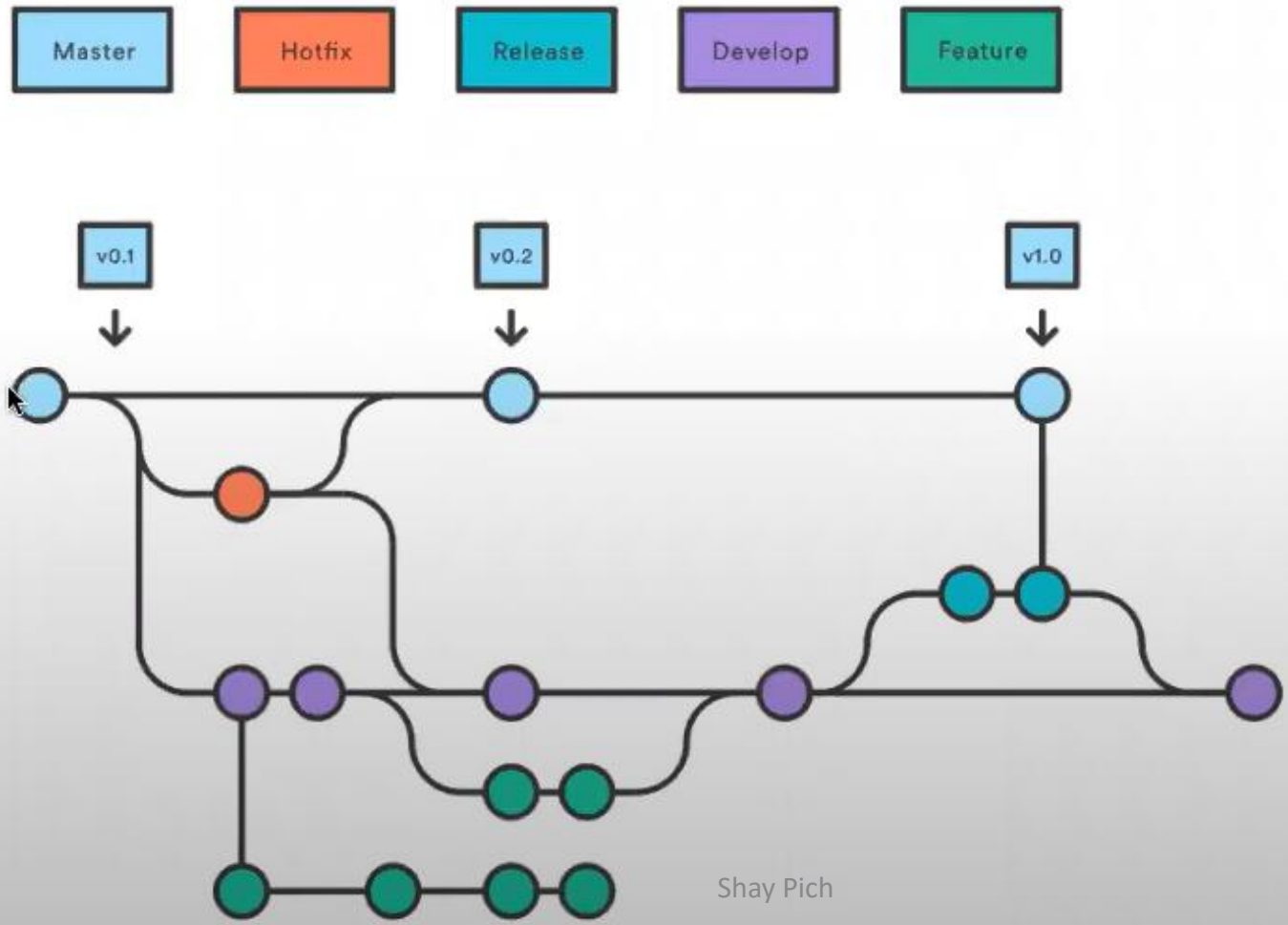
Examples for different strategies:

- Git flow (what we will cover) — ותיק, אך מיושן וקצת מעיק
- GitHub flow — פופולארי
- GitLab flow
- Trunk-based Development — פופולארי

And more

Conventions - branches

GitFlow



Conventions - branches

GitFlow

To work with gitflow i recomand using or
Git GUI helpers like Git Kraken
https://www.gitkraken.com/
or use the extantions git-flow-cheatsheet
git-flow-cheatsheet/