1. set the vs code default sell to be “git bash” by the following steps:  
   a. Open the command palette using Ctrl + Shift + P.   
   b. Type - Select Default Profile   
   c. Select Git Bash from the options   
   d. Click on the + icon in the terminal window   
   e. The new terminal now will be a Git Bash terminal.   
   f. Give it a few seconds to load Git Bash
2. Print your git version: **git –version**
3. Download the last version of git (you might need to close vs code before installation)
4. Print the new version: **git –version**
5. Graphical user interface, text

   Description automatically generated
6. Set configuration by:  
   **git config –global user.name “Yossi Arye”  
   git config –global user.email** [**aryeyossi@gmail.com**](mailto:aryeyossi@gmail.com)   
   **git config –global core.editor “code --wait”** (the shell will keep watching while the config file stays open)
7. Open the git config file (.gitconfig) by **git config –global -e**
8. Graphical user interface, application

   Description automatically generated
9. A screenshot of a computer

   Description automatically generated with low confidence
10. For windows: **git config –global core.autocrlf true**  
    For mac/linux: **git config –global core.autocrlf input**
11. create git repository:  
    **mkdir myRepo   
    cd myRepo   
    git init   
    ls -a**
12. Graphical user interface

    Description automatically generated with medium confidence
13. A screenshot of a computer

    Description automatically generated with low confidence
14. A screenshot of a computer

    Description automatically generated with low confidence
15. After committing, the staging area keeps the snapshot
16. “git add file1” command is for adding, changing and removing file1
17. Graphical user interface, text, application, chat or text message

    Description automatically generated
18. In every commit git show not only the delta, but the full code. But behind the scenes, it stores the code in efficient way by compressing the content and not storing duplicate content
19. Create and write to new files:  
    **echo console.log\(Hello World!\) > file1.js  
    echo console.log\(Hello World!\) > file2.js**
20. **ls** to see files are added
21. **cat** to see the file content
22. **git status** to show the status of the files in our working directory (we will see all the unstaged files listed in red and all the staged files in green)
23. Staging changes: **git add file1.js file.js** for staging file1 and file2  
    or **git add \*js** for staging all the files that their extension is js  
    or **git add .** for staging all files
24. **git status** to see file1.js and file2.js in “new file” list, colored in green
25. Fix the code console.log(“Hello World!”) in file1.js via vs code editor (or via terminal)
26. **git status** to see file1.js and file2.js in “new file” list, colored in green. And file1.js in “modified” list colored in red
27. Add file1.js and check status (we should see only “new file” list, colored in green)
28. Committing changes:  
    **git commit -m “initial commit”**  
    The commit will clear the working directory and create a snapshot in the repository.
29. Another way is to enter **git commit** and the vs code will open the commit file.   
    in the first line will be a short description and in the second line a long description.  
    (the lines start with # will not be shown in the commit message)

Text

Description automatically generated

1. After commit we will see the number of file have changed and the number of the insertion (lines)
2. We can add file to staging and commit it by one command **git commit -a -m “commit message”** or **git commit -am “commit message”**
3. Removing file:  
   **rm file2.js**  
   file2.js is not anymore in our working directory, but it still in the staging area, since we didn’t add the changes to the staging area.
4. We can see the file in the staging area by **git ls-files**.
5. For adding it to the staging area we will use **git add file2.js** and we will see it by **git status** in the “deleted” list, colored in green.
6. We can use git command for deleting file and adding it to the staging area immediately by **git rm file2.js**
7. Renaming file name:  
   **mv file1.js main.js**  
   We will see the changes in the working directory but not in the staging area.  
   To add the changes to the staging area we can use **git add file1.js** and **git add main.js**.
8. We can use git command for renaming file and adding it to the staging area immediately by **git mv file2.js main.js**
9. Ignoring files:  
   We can have files or directories that we don’t want to add to our repository.  
   for that we need to add them to a file name .gitignore.  
   **echo I want to want to ignore it > logs/file.log**  
   **echo logs/ > .gitignore**  
   If we will print **git status**, we will see only the .gitignore in the red files, but not the /logs dir  
   **git commit .gitignore -am “add git ignore”**
10. .gitignore file, will ignore files that added to it while they have not added yet to the staging area, if the files already staged .gitignore will not ignore them.  
    echo this is a test file > file.test  
    git add file.test  
    echo file.test >> .gitignore  
    echo append changes to test >> file.test   
    git status, we will see - in the green list: file.test because we added it, in the red list: file.test and .gitignore, because file.test has added to staging area before it has added to .gitignore .
11. Removing file from staging area but keep it in the working directory:  
    **git rm –cached file.test**  
    We will get error, because there is a different between staging and working directory.  
    For that we will use: **git rm –cached -f file.test**For removing directory we need to add the flag -r **git rm –cached -r <directory name>**  
    Now if we will check the status we won’t see file.test in the red list because it is ignored file.
12. We can use **git status -s** to see the status in a clearer way Graphical user interface, text

    Description automatically generated with medium confidence  
    green M means, we have a modified part in the staging area.  
    green A means, we added a new file to the staging area.  
    red M means, we have a modified part in the working directory.  
    ?? means, we add a new file to the working directory.
13. Comparing changes:   
    **git diff** will compare the staging area version to working directory version.  
    Once we added the working directory to the staging area we will not see any difference.  
    Remember: Also committed changes are in the staging area! So it will compare committed changes to working directory too!  
    Text

    Description automatically generated  
    (a) is the original version, (b) is the new version in the working directory.  
    (-1) means, (-) stands for the original version, (1) the lines in the chunk in the original version (chunk = the part in the file with the changes)  
    (+1, 2) means, (+) stands for the new version, (1, 2) the lines in the chunk in the new version.
14. **git diff --staged** will compare the staging area version to the committed version.
15. Viewing history:  
    **git log** (enter space to see more, enter q to quitText

    Description automatically generated  
      
    **git log –reverse** to see the first commit in the top  
    git log –oneline to see it clearly and shortlyText

    Description automatically generated
16. Viewing history differences:  
    **git show <commit id or part of it if it is unique>Text, chat or text message

    Description automatically generated**anotherwayis using HEAD (HEAD point to the last commit in the branch)  
    **git show HEAD** for the last commit.  
    **git show HEAD~n** (n is the number of the commit before the last commit)  
    **git show HEAD~n:<file name>** to see the last content of the file in the commit**Text

    Description automatically generated**
17. We can show files (blobs) and directories (trees) stored in the git commit database **git ls-tree <commit id> Text

    Description automatically generated**We can see the all git objects by **git show <object id>** command (commit, tree, blob and tags)  
    ****
18. Unstaged files:  
    **git restore –staged <file name or pattern>**The different between **git restore –staged** to **git rm –cached**  
    Is that **git restore –staged** change the staging area to be as the last commit (HEAD)  
    but **git rm –cached** will remove from staging area the file with the changes that are different from the last commit, such the file in the working directory won’t be the same as the last commit.
19. **Text

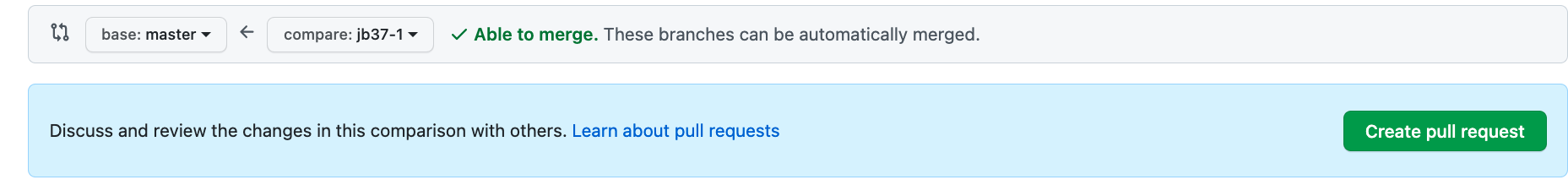
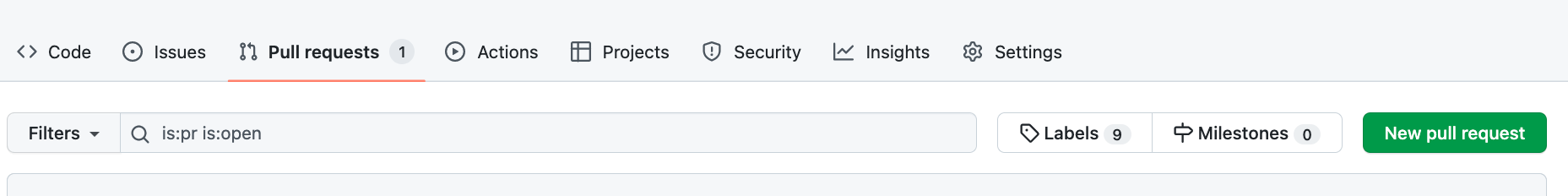
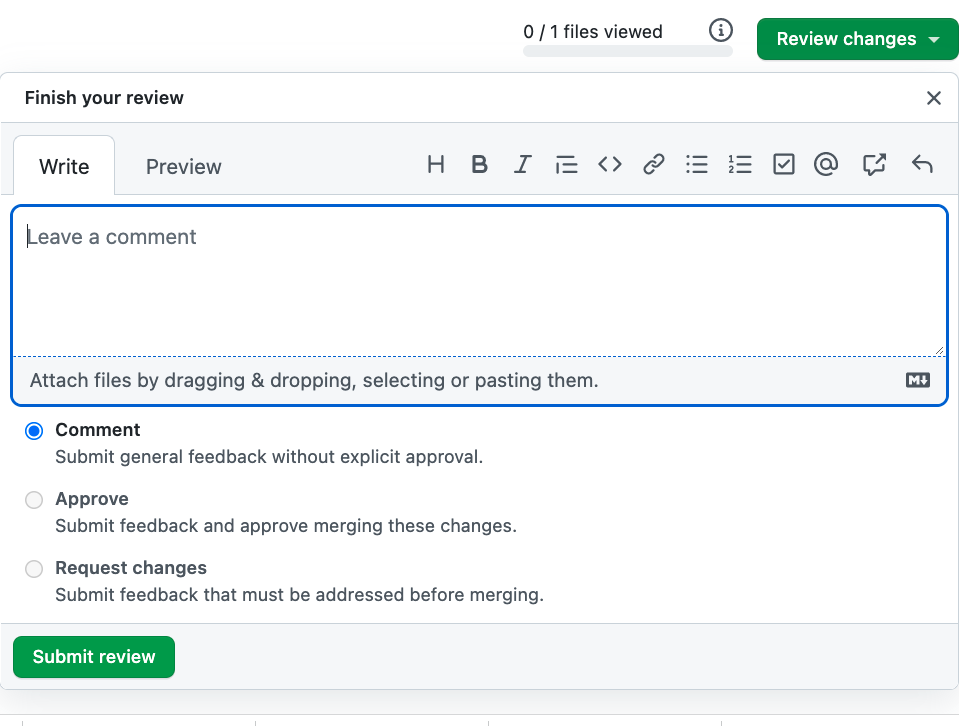
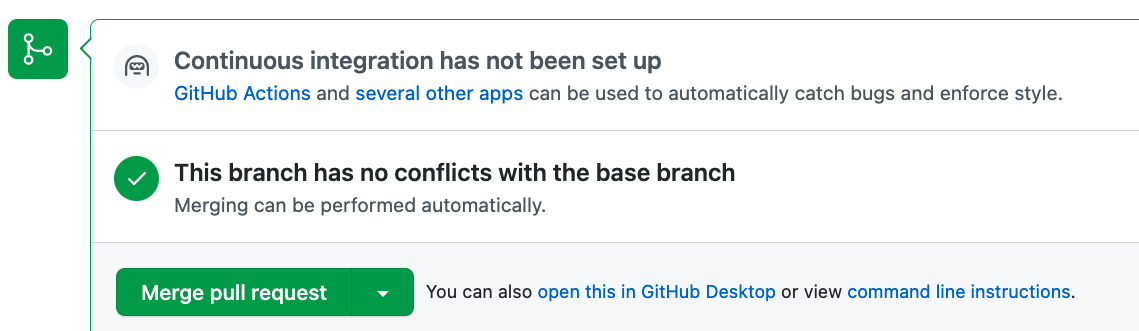
    Description automatically generated**
20. Discard files from working directory  
    The working directory will be the same as the staging area  
    **git restore <file name or pattern>**  
    **git restore .** for all not new files  
    **git clean -f** for all files including new   
    **git clean -f -d** for all files and directories including new
21. Restore file from previous version  
    **git restore –source=<commit id> <file name>Text, application, chat or text message

    Description automatically generated**

**Remote repository**

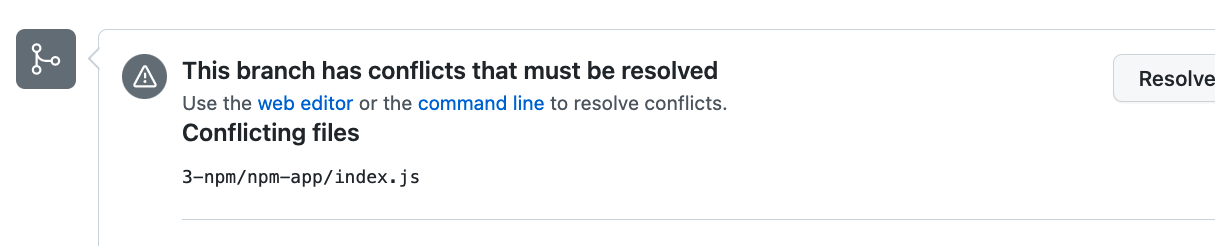
1. Sign up to github
2. If you want to create a new repository follow the instructions of github, you might have problems to add a remote origin to your local, in this case you can follow the instructions in https://levelup.gitconnected.com/fix-password-authentication-github-3395e579ce74
3. Go to the repository and under “code” tab copy the http url
4. Go to the place when you want to clone the repo on your local, and run “git clone <http url>”. If you are not logged in to github from the vs code you will asked to do this
5. Make sure the repo cloned by cd into the cloned folder and run ls command
6. Create a new branch by running the command “git checkout -b <branch name>”
7. Edit a file, add it to staging and create a commit.
8. Run “npm push”, for push the changes to the remote repo.

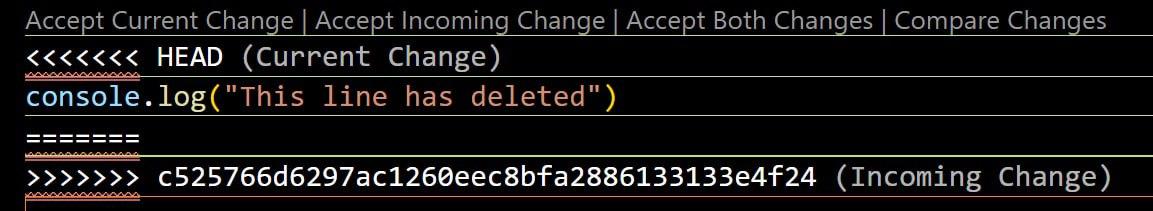
You should get error:  
fatal: The current branch <branch name> has no upstream branch.   
To push the current branch and set the remote as upstream, use

1. Run git push --set-upstream origin <branch name>
2. Go to github and create pull request to merge your remote branch to the main branch (master)  
   
3. You can add reviewers to the pr which can approve or add comments to the pr  
   
4. To merge the branch into master, you can merge it via the following feature
5. In your local, checkout into master and run “git pull”

**Conflicts**

1. A conflict arises **when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other**. Conflicts will most likely happen when working in a team environment.
2. To demonstrate a conflict create 2 branches, in one delete row and in the other edit the same row. First merge the branch you deleted the row, and then try to merge the other which added the code, you will have a conflict.



1. To solve the conflict go to your local and checkout to the branch you tried to merge. Run “git pull origin master” to bring the conflict to your local.
2. Solve the conflict and create commit and push it to the remote, and try to make PR again.