I'm not a bot

I made a critical mistake by rebasing my branch with `git rebase -i --root` without fully understanding its implications. I had mistakenly merged the first commit differing from the master branch, which was reflected in the default GitHub view for Windows users. As a result, I experienced issues trying to remove unwanted commits and restore the original commit history. To rectify the situation, I started by deleting my local repository and re-cloning from the remote. However, this led to an unexpected outcome - the most recent unneeded commit was re-added to the master branch. Frustrated but determined, I explored alternative solutions to overcome Git's stubbornness. After exhaustively searching for a solution, I discovered that using `git reset --hard ` didn't work as expected. Instead, I tried updating another branch to master and then attempting `git rebase -i `, only to be met with disappointment when the commit was unexpectedly restored. It wasn't until I stumbled upon the `git rebase -i ` command with the `d` option that I finally found a way to remove the unwanted commit without losing any data. By carefully following this approach, I was able to successfully delete the commit from my branch history and restore the original commit timeline. I installed Node.js from nodejs.org but npm is not recognized in the normal command prompt. I found out that it was installed in C:\Program Files\NodeJS and when I opened a command prompt there, npm worked fine. I added the path to PATH variable, but still got the error. After researching, I found that one of the GitHub issues suggested restarting my machine which didn't work for me either. However, I noticed that if I used the "Node.js Command Prompt" from the program files, then npm was recognized. This led me to suspect an issue with PATH variable or some installed application like Chocolatey, which might have added a quote causing problems. Temporarily reverting back to a previous commit in Git involves checking out the desired commit. For example, you can use `git checkout 0d1d7fc32` to detach your HEAD and go back to that specific commit. If you want to make commits while there, create a new branch using `git checkout -b old-state 0d1d7fc32`. To get rid of unpublished commits, you can reset the repository with `git reset --hard 0d1d7fc32`, but be careful as this will destroy any local modifications. If you have published the work and don't want to rewrite history, reverting the commits is your only option in many enterprise organisations where protected branches prevent rewriting history. Reverting a commit in Git creates a new commit with the reverse patch to cancel out the original one, without changing the existing history. To revert a merge commit or a range of commits, use `git revert -m 1 ` for merges or `git revert HEAD~2..HEAD` for a range of commits. You can also manually squash them afterwards using `rebase -i`. If you decide not to revert after all, you can undo the changes with `git revert` again or reset back to before the revert. you dont need that code any longer. These cases all call for git revert. The git revert command does just what you might expect. It reverts a single commit by applying a reverse commit to the history. Sometimes you need to revert several commits to completely undo a change. You can use -no-commit, or you can use -n to tell Git to perform the revert, but stop short of committing the change. This lets you combine all the revert commits into one commit, which is useful if you need to revert a feature that spans several commits. Make sure that you revert commits in reverse order-the newest commit first. Otherwise, you might confuse Git by trying to revert code that doesn't exist yet. I am using a .NET console app to test SQL Server 2019 database connection and get following error message: A connection was successfully established with the server, but then an error occurred during the login process. (provider: SSL Provider, error: 0 - The certificate chain was issued by an authority that is not trusted.) The user id and password are valid. The C# code is not using any encryption. I have not configured any certificate in SQL Server configuration. Is certificate required in SQL Server 2019? Can I get a self generated cert from the SQL Server? If no self-generated cert available, where can I get a valid certificate for free? While the first and selected answer is technically correct, there's the possibility you have not yet retrieved all objects and refs from the remote repository. If that is the case, you'll receive the following error: $ git checkout -b remote_branch origin/remote_branch fatal: git checkout: updating paths is incompatible with switching branches. Did you intend to checkout 'origin/remote_branch' which can not be resolved as commit? Solution If you receive this message, you must first do a git fetch origin where origin is the name of the remote repository prior to running git checkout remote_branch. Here's a full example with responses: $ git fetch origin remote: Counting objects: 140, done. remote: Compressing objects: 100% (30/30), done. remote: Total 69 (delta 36), reused 66 (delta 33) Unpacking objects: 100% (69/69), done. From e6ef1e0..5029161 develop -> origin/develop * [new branch] demo -> origin/demo d80f8d7..359eab0 master -> origin/master $ git checkout demo Branch demo set up to track remote branch demo from origin. Switched to a new branch 'demo' As you can see, running git fetch origin retrieved any remote branches we were not yet setup to track on our local machine. From there, since we now have a ref to the remote branch, we can simply run git checkout remote_branch and we'll gain the benefits of remote tracking. Page 2 Though there are already a lot of answers and the commands below are supposed to work: git pull or git fetch git checkout But in my case, i was not able to get all the remote branches, though git pull or git fetch was not throwing any error, but when i ran git branch -a, it was showing my local branches and one remote branch, which was not even right. To fix that, i simply removed the remote: git remote remove origin then add the remote again: git remote add origin Then, run the above commands: git pull or git fetch and then git checkout Even though utf8_decode is a useful solution, I prefer to correct the encoding errors on the table itself. In my opinion it is better to correct the bad characters themselves than making "hacks" in the code. Simply do a replace on the field on the table. To correct the bad encoded characters from OP : update set = replace(, "Ã«", "ë") update set = replace(, "Ã", "à") update set = replace(, "Ã¬", "ì") update set = replace(, "Ã¹", "ù") Where is the name of the mysql table and is the name of the column in the table. Here is a very good check-list for those typically bad encoded windows-1252 to utf-8 characters -> Debugging Chart Mapping Windows-1252 Characters to UTF-8 Bytes to Latin-1 Characters. Remember to backup your table before trying to replace any characters with SQL! [I know this is an answer to a very old question, but was facing the issue once again. Some old windows machine didnt encoded the text correct before inserting it to the utf8_general_ci collated table.] As for your first question: "if item is in my_list:" is perfectly fine and should work if item equals one of the elements inside my_list. The item must exactly match an item in the list. For instance, "abc" and "ABC" do not match. Floating point values in particular may suffer from inaccuracy. For instance, 1 - 1/3 != 2/3. As for your second question: There's actually several possible ways if "finding" things in lists. Checking if something is inside This is the use case you describe: Checking whether something is inside a list or not. As you know, you can use the in operator for that: 3 in [1, 2, 3] # => True Filtering a collection That is, finding all elements in a sequence that meet a certain condition. You can use list comprehension or generator expressions for that: matches = [x for x in lst if fulfills_some_condition(x)] matches = (x for x in lst if x > 6) The latter will return a generator which you can imagine as a sort of lazy list that will only be built as soon as you iterate through it. By the way, the first one is exactly equivalent to matches = filter(fulfills_some_condition, lst) in Python 2. Here you can see higher-order functions at work. In Python 3, filter doesn't return a list, but a generator-like object. Finding the first occurrence If you only want the first thing that matches a condition (but you don't know what it is yet), it's fine to use a for loop (possibly using the else clause as well, which is not really well-known). You can also use next(x for x in lst if ...) which will return the first match or raise a StopIteration if none is found. Alternatively, you can use next((x for x in lst if ...), [default value]) Finding the location of an item For lists, there's also the index method that can sometimes be useful if you want to know where a certain element is in the list: [1,2,3].index(2) # => 1 [1,2,3].index(4) # => ValueError However, note that if you have duplicates, .index always returns the lowest index:..... [1,2,3,2].index(2) # => 1 If there are duplicates and you want all the indexes then you can use enumerate() instead: [i for i,x in enumerate([1,2,3,2]) if x==2] # => [1, 3] You can use grep tool to search recursively the current folder, like: grep -r "class foo" .ripgrep can be used to search for specific strings of text in files across your entire Linux system, and it's arguably faster than other tools like GNU grep, ucg, ag, sift, or ack. if you're looking to find a particular phrase within all files on your machine, you should use the following command: rg "class foo" . ripgrep supports various parameters such as -i for case-insensitive searching and -n to display line numbers of matches. you can also increase context with --context option. The recommended settings are usually -C5 and --color=auto. ripgrep is a powerful tool for text searches on linux systems, but it may require some learning curve compared to other tools.