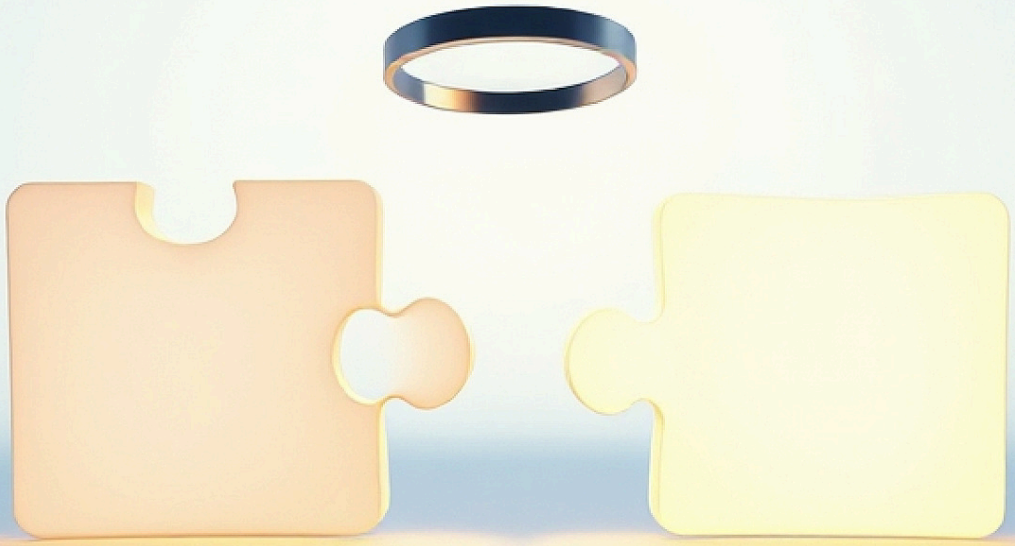
What is a code merge conflict?



www.yzokul.com.tr

What is a code merge conflict?



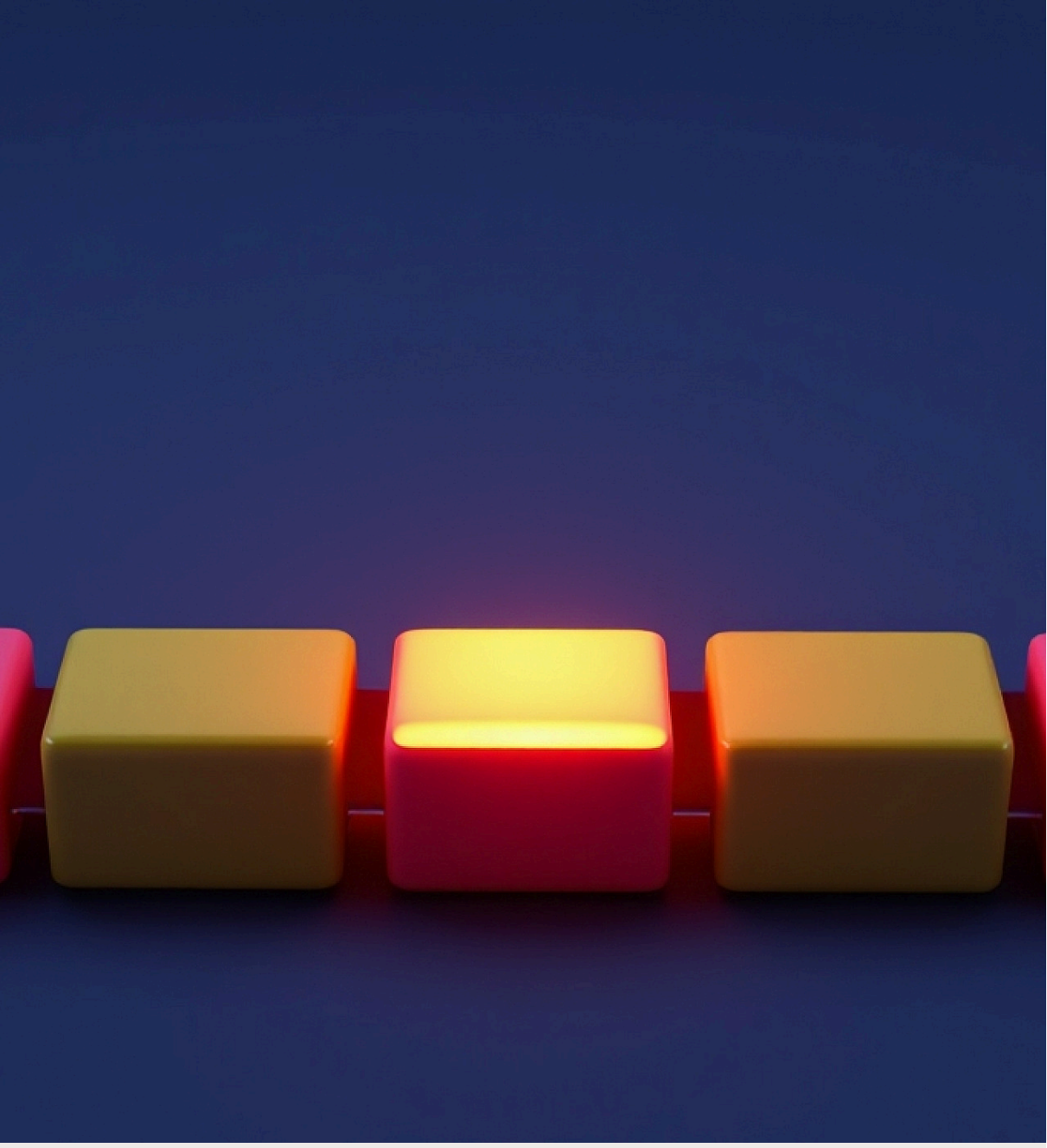
A code merge conflict happens when two sets of changes to the same code cannot be combined automatically, so a person must decide what to keep. This answer book explains the idea from the beginning, using plain words. You will see how version control and branching work, why conflicts appear, how to resolve one by hand, and how to check your work afterward. The short answer: a conflict is a request for a human decision, not a disaster.



Code is just text stored in files that tells a computer what to do. A software project is simply a folder full of these files, often organized into subfolders. Each file has a name and contents you can open and read. When people work on a project together, they change these files over time. Keeping track of those changes is the job of a version control tool.



People often want to keep old versions of their files so they can go back or compare. Version control is a tool that remembers every saved version for you, so nothing is lost. Instead of copying folders by hand, you let the tool record each change. This makes it easy to see what changed, when, and by whom. It also lets several people work on the same project without overwriting each other's work.



A commit is a saved snapshot of your files together with a short message about what changed. Each commit is a point you can return to later, like a bookmark in the project's history. For example, you might commit with the message fix login button color. The version control tool stores that snapshot and links it to the previous one. Over time, commits form a chain that records how the project grew.



A branch is a separate line of work that starts from a commit. People make branches so they can try changes without disturbing the main version of the project. For example, you create a branch called new-menu to experiment with a redesign. The main branch stays as it was, and your branch grows on its own. When the work is ready, you can bring it back into the main line.



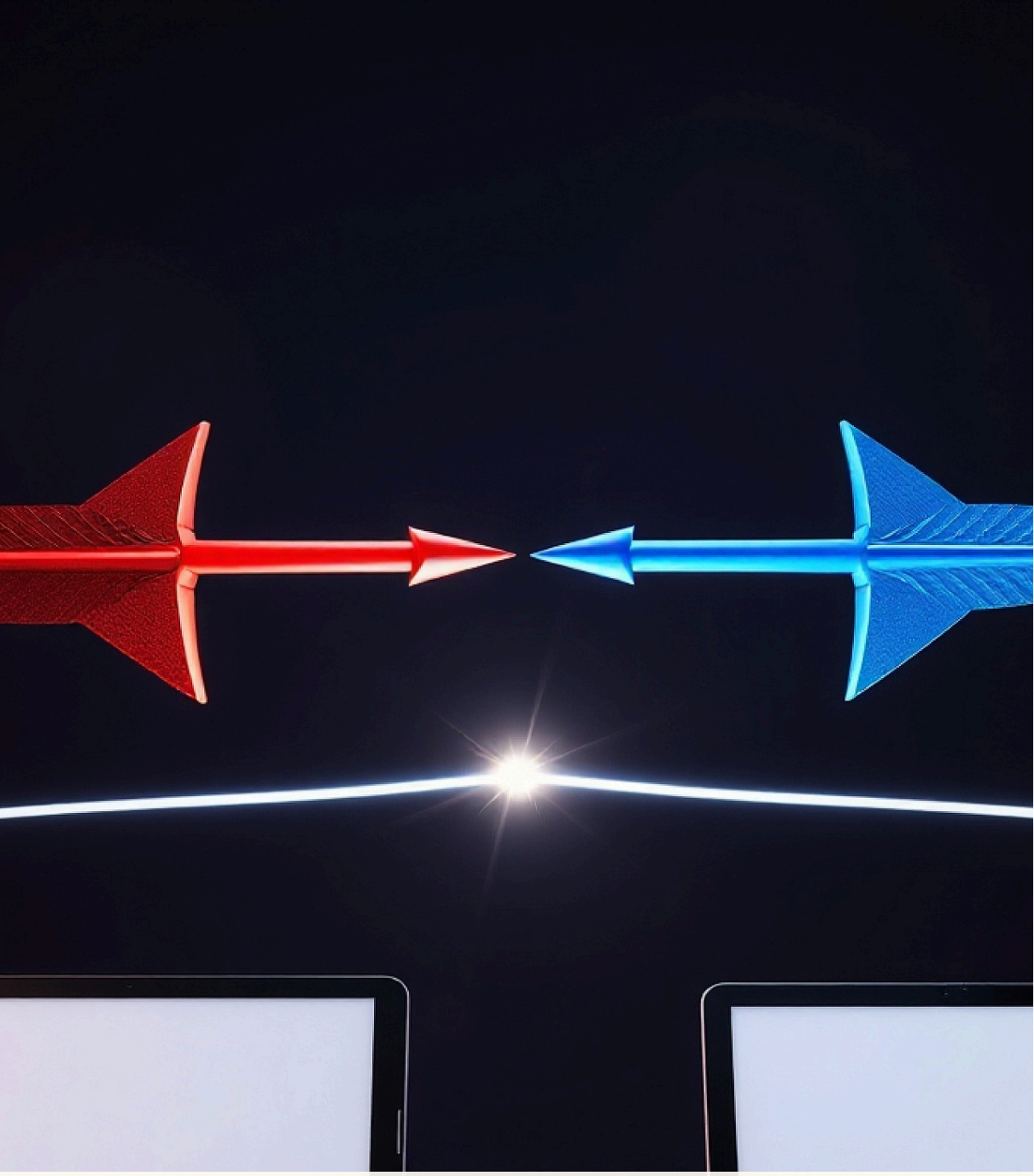
Picture two branches growing from the same starting commit, each with its own new commits. This is what it means to have two lines of development. One person might add a feature on one branch while another fixes a bug on a different branch. Both branches share the same history up to the point where they split. After that, each branch records its own separate changes.



A merge takes the changes from one branch and adds them to another. The version control tool tries to combine the changes for you automatically. For example, you merge your new-menu branch into the main branch so everyone gets the redesign. The tool looks at what changed on each side since they split. If it can combine everything cleanly, the merge finishes without asking you anything.



If the two branches changed different parts of the files, the tool combines them automatically with no problem. For example, one branch edits the top of a file while the other edits the bottom. The tool simply keeps both edits, because they do not overlap. Most merges happen this way, quietly and without any conflict. You often do not even notice that a merge took place.



The key case is when both branches changed the very same line in different ways. For example, one branch sets the title to Welcome while the other sets it to Hello. The tool cannot know which version you want, because both are valid changes. So it stops and asks you to decide. This situation is exactly what leads to a merge conflict.



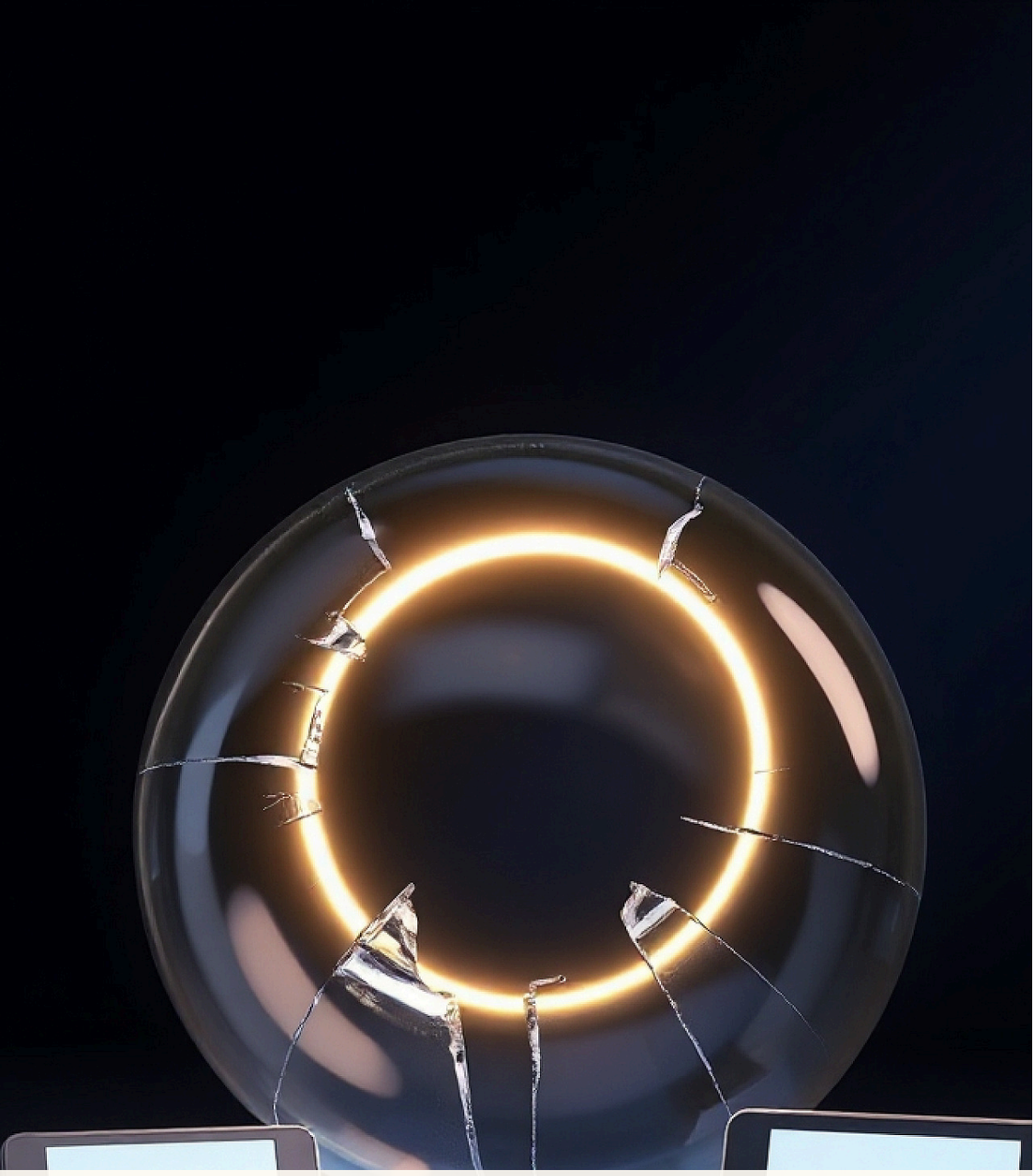
That stop is called a merge conflict. It is the tool saying it needs a human decision, not a sign that something is broken. The word conflict sounds dramatic, but it simply means the automatic merge could not finish. Nothing has been thrown away, and no work is lost. The tool pauses and waits for you to choose what the final file should contain.



The tool shows the conflict inside the file using special markers. One marker separates your version from the other version, and another marker ends the other version. For example, you might see a block labeled with your branch name above one set of lines and the other branch name above another set. You can read both options clearly and compare them side by side.



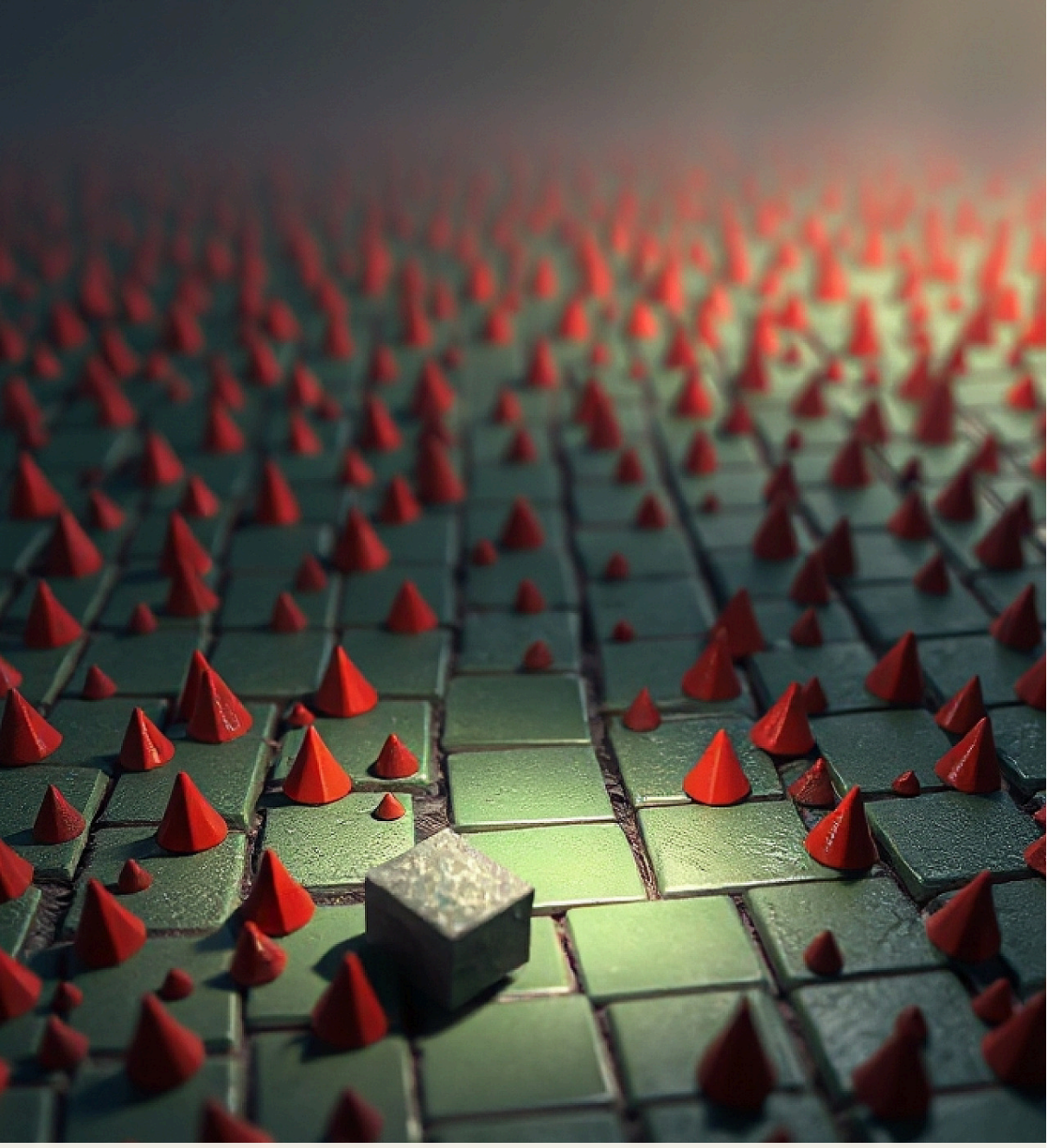
Now you decide what to keep. You can keep one version, keep the other, or combine parts of both into something new. This is a human decision the tool cannot make for you. For example, you might keep the other branch's title but add a line from your own version below it. You edit the file until it contains exactly what you want, then save it.



A conflict does not mean the project is lost or ruined. Nothing is deleted for good, and you can always look back at the old versions in the commit history. If you make a mistake while resolving, you can undo it or start the merge again. The conflict is just a pause for a decision. Treat it as a normal part of working with branches, not as a failure.



After choosing what to keep, you must check that the program as a whole still works correctly. The pieces may each be fine but not fit together. For example, one branch might rename a function while the other still calls the old name. Run the program and its tests to confirm everything behaves as expected. Only then finish the merge and save the result.



Common beginner mistakes include panicking at the markers and deleting both versions by accident. Another mistake is skipping the check afterward, so a broken program slips through. Some people also forget to remove the conflict markers themselves, leaving them in the file. Knowing these traps helps you avoid them. Stay calm, read both sides carefully, and always test the result.



A merge conflict happens when two lines of work change the same part of a file in different ways. A person decides what to keep or how to combine it, then checks that the whole program still works. The project is never lost, because old versions remain in the history. With practice, resolving conflicts becomes a routine and manageable part of software development.