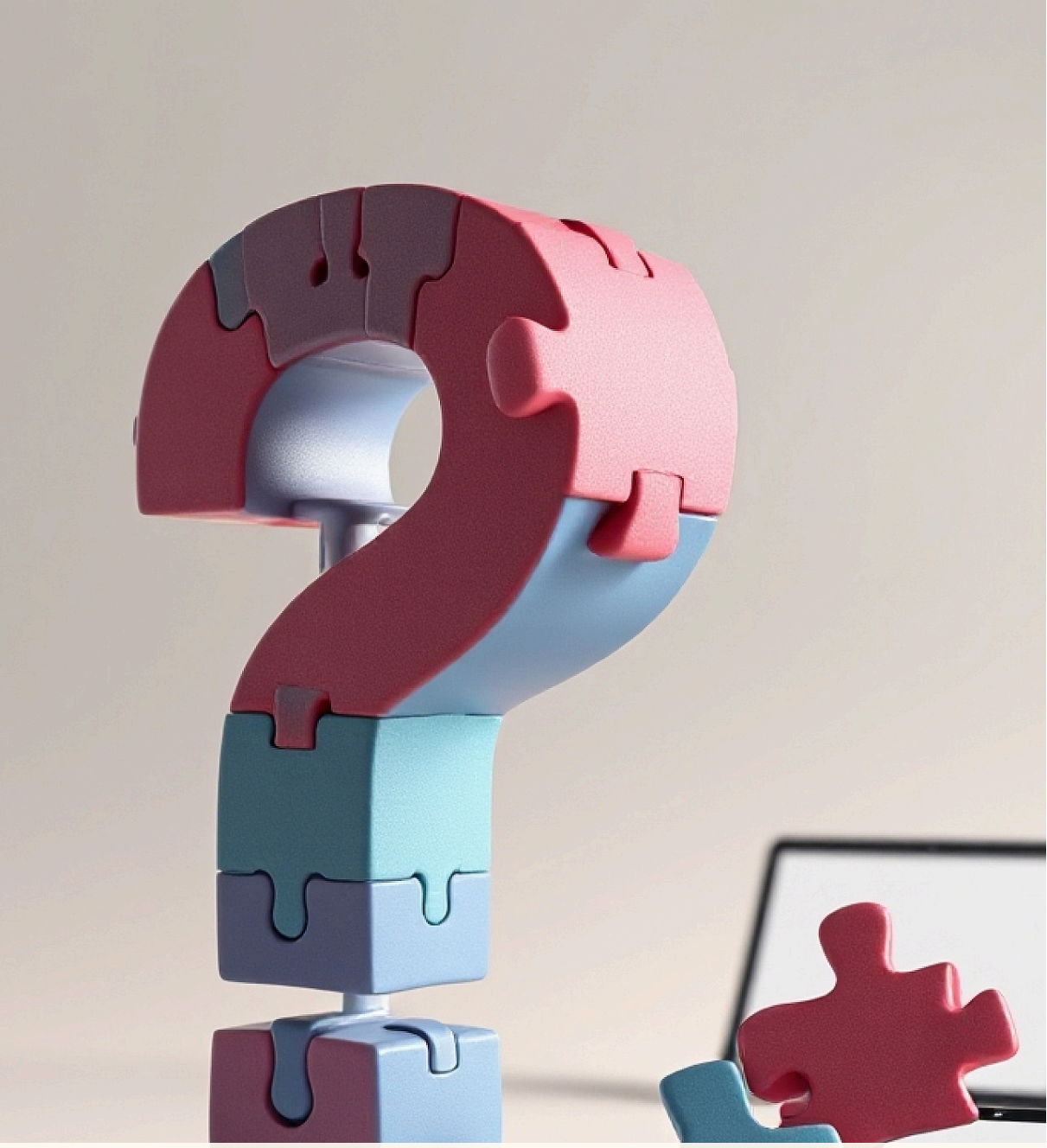

What is the purpose of creating a branch in code?
www.yzokul.com.tr

What is the purpose of creating a branch in code?



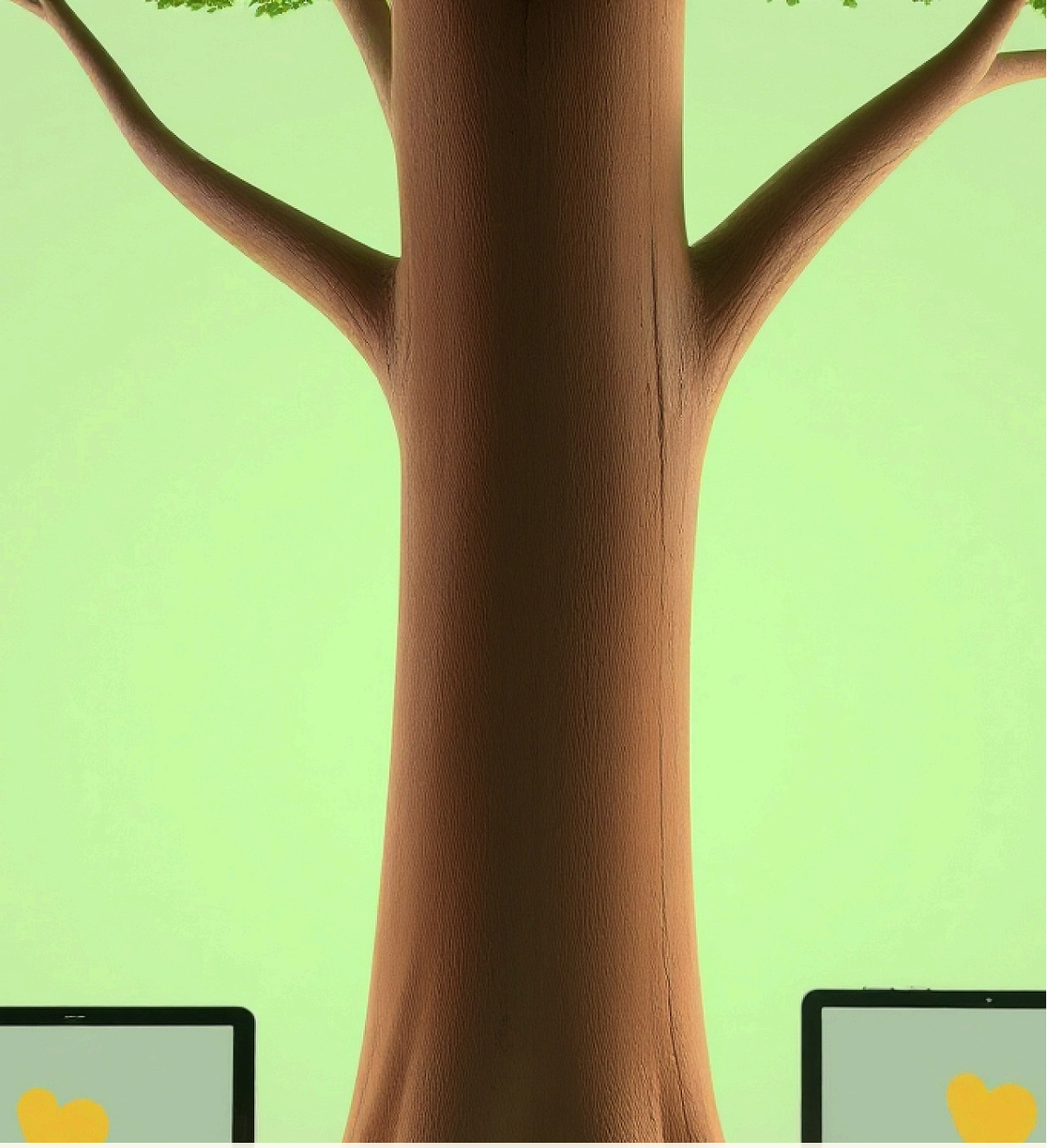
You have probably seen the question: what is the purpose of creating a branch in code? The short answer is that a branch lets you modify a project along a separate line of development while the main version stays usable. This means you can try new things without disturbing what already works. Branches are a core idea in version control, which is a system that tracks changes to files over time.



Think of a project as a folder full of files, like code files and images. This folder is what you will change when you work on new features or fixes. A branch will let you make those changes in an organized way, separate from the main project. For example, a game project might have files for graphics, sounds, and game logic.



A project keeps a history of its changes over time. This history records what was changed, when, and by whom. Everyone working on the project builds on that same history, so you can see how the project evolved. Branches use this history as a starting point for separate work. This shared record helps teams collaborate without losing track of changes.



A branch is a separate line of development inside the same project. It starts from the current version and lets you make changes without touching the main line. You can create a branch whenever you want to work on something new. The main line, often called the main branch, remains unchanged while you work on your branch. This separation is the key purpose of creating a branch.



While you work on a branch, the main version keeps working as before. People can still use it, and it is not disturbed by your unfinished work. This means you can take your time and experiment without breaking anything for others. For example, if you are adding a new feature, the main version remains stable and usable. This safety is a major reason to use branches.



Picture a game project where someone wants to build a new menu. They create a branch for the menu work while the main version of the game stays playable. On the branch, they can design and code the menu without affecting the game that players are using. Once the menu is ready, they can bring it into the main version. This example shows how branches enable parallel work.



On the branch, you can try things, make mistakes, and change files freely. None of this affects the main version until you decide to bring the changes back. You can test your ideas and refine them without worrying about breaking the main project. This freedom encourages experimentation and learning. For beginners, this is a safe way to practice coding.



A branch is not a completely independent project. It belongs to the same project and shares its files and history. When you create a branch, you are not starting from scratch; you are building on what already exists. This means you can reuse code and assets without duplicating them. Branches are lightweight and easy to create, so you can have many branches for different tasks.



When the new menu is complete and working, the branch has done its job. Now the finished change is ready to go back into the main line. You have tested the menu on the branch and are confident it works. The next step is to combine your work with the main version. This is where merging comes in.



Merging is the process of combining the completed branch changes into the main branch with the appropriate operations. After merging, the main version now includes the new menu. The merge operation takes the changes you made on the branch and applies them to the main branch. This is how your work becomes part of the main project. Merging is a fundamental operation in version control.



Why merge instead of copying files? Merging brings the branch's history of changes along with the files. This keeps one clear record of how the project grew. If you simply copied files, you would lose the history of who changed what and why. Merging preserves the context of your work, making it easier to understand and maintain the project over time.



If the same files were changed on different branches, the merge cannot always combine them automatically. The tool needs help deciding which change to keep. This situation is called a conflict. Conflicts happen when two branches modify the same part of a file in different ways. You will need to resolve the conflict manually.



A conflict is a moment where you compare the two versions of the same file and choose the right result. After that, the merge can finish. You might need to edit the file to combine the changes or pick one version over the other. Resolving conflicts is a normal part of collaborating with branches. It ensures that the final code is correct and consistent.



A common mistake is changing the main version directly for big work, so it can break while unfinished. A branch avoids this by keeping risky work separate. If you work directly on the main branch, you might introduce bugs that affect everyone. By using a branch, you can experiment and test without disrupting the main project. Always create a branch for significant changes.



Another mistake is leaving finished work on a branch forever, so the main version never gets it. Completed changes should be merged back. If you forget to merge, your work is not integrated into the main project, and others cannot benefit from it. Make merging a regular part of your workflow. Once a feature is done and tested, merge it promptly.



To review, a branch lets you develop separately while the main version stays usable, and finished work is merged back, with conflicts resolved if needed. Remember: branch for separate work, merge when done. This approach keeps your project stable and your changes organized. By following these practices, you can collaborate effectively and maintain a healthy codebase.