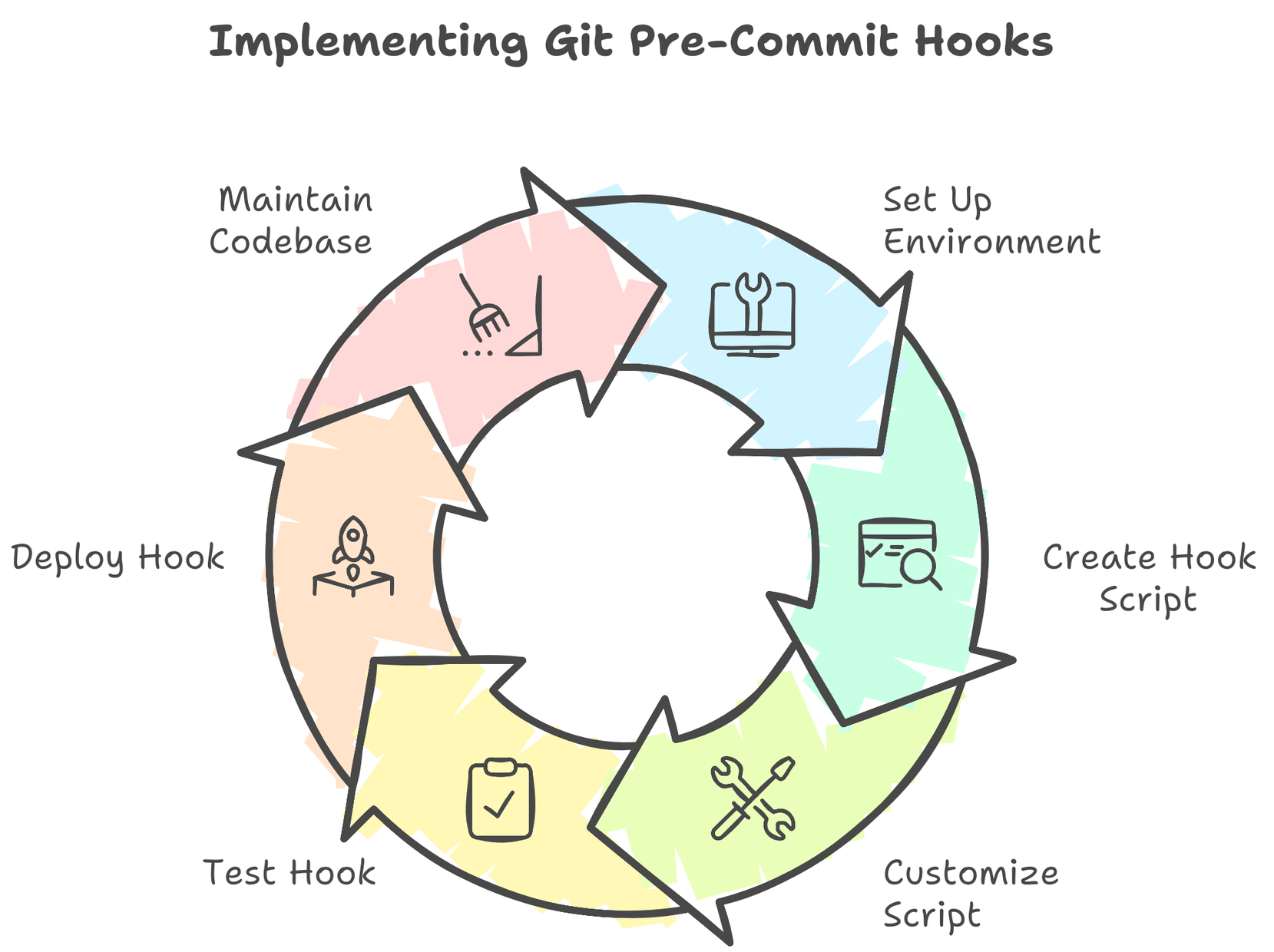
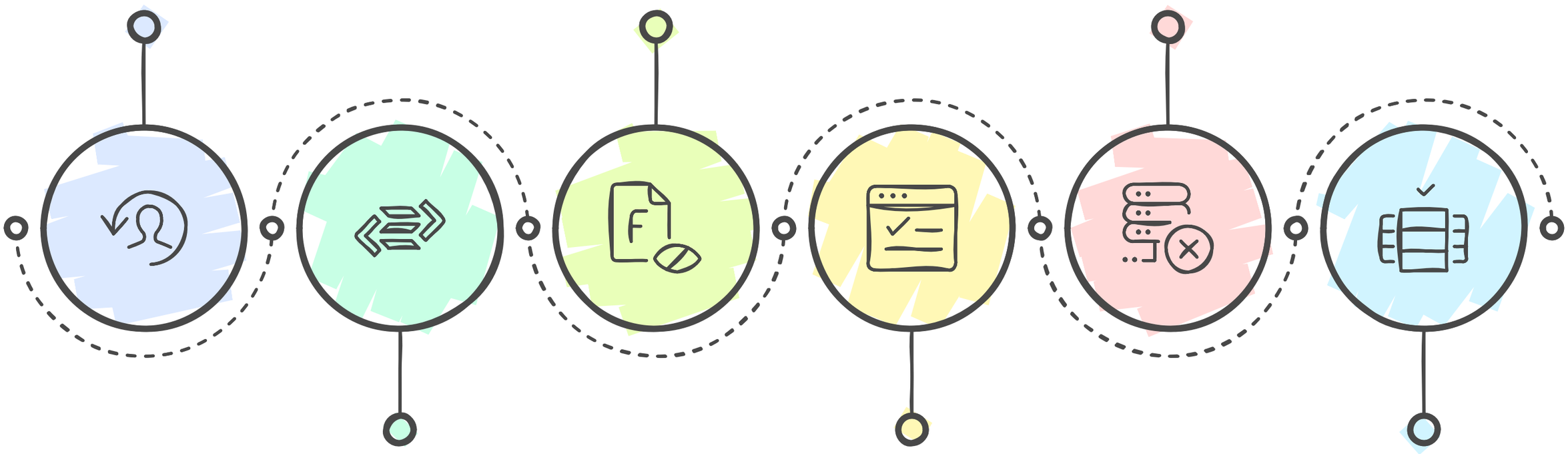
Using Git Pre-Commit Hooks to Prevent Unused Files from Being Committed

In this document, we will explore how to utilize Git pre-commit hooks to prevent the inclusion of unused files in your commits. This guide will cover the setup process for Visual Studio IDE, Visual Studio Code, and the Command Prompt, including the Visual Studio Developer Command Prompt. By the end of this document, you will have a clear understanding of how to implement and customize pre-commit hooks to maintain a clean and efficient codebase.



What are Git Pre-Commit Hooks?

Git pre-commit hooks are scripts that run automatically before a commit is finalized. They allow you to perform checks on your code, such as verifying file types, checking for unused files, or running tests. If the script exits with a non-zero status, the commit will be aborted, preventing unwanted changes from being added to the repository.



Setting Up a Pre-Commit Hook

Step 1: Navigate to Your Git Repository

Open your terminal or command prompt and navigate to the root directory of your Git repository:

```
cd path/to/your/repo
```

Step 2: Create the Pre-Commit Hook

In your repository, navigate to the **.git/hooks** directory. Here, you will find sample hook scripts. Create a new file named **pre-commit** (without any extension):

```
touch .git/hooks/pre-commit
```

Step 3: Make the Hook Executable

Ensure that the pre-commit hook is executable. You can do this by running the following command:

```
chmod +x .git/hooks/pre-commit
```

Step 4: Write the Hook Script

Open the **pre-commit** file in your preferred text editor (**Visual Studio, Visual Studio Code,** etc.) and add the following script to check for unused files:

```
#!/bin/bash

# Redirect all output to stderr
exec 1>&2

echo "🔍  Checking for unused files before commit..."

# Get all newly added files (staged)
new_files=$(git diff --cached --name-only --diff-filter=A)

unused_files=()

for file in $new_files; do
    # Get just the filename without the path
    filename=$(basename "$file")

    # Remove extension (e.g., .cs, .js) to get the base name
    base_name="${filename%.*}"

    # Skip empty filenames or special cases
    if [ -z "$base_name" ]; then
        continue
    fi

    # Check if base name is used anywhere in the repo (excluding
the file itself and .git folder)
    if ! git grep -q "$base_name" -- ':(exclude)'"$file"
':!*.git*'; then
        unused_files+=("$file")
    fi
done

if [ ${#unused_files[@]} -gt 0 ]; then
    echo "❌  The following files are added but never used in the
project:"
    for uf in "${unused_files[@]}"; do
        echo "   - $uf"
    done
    echo "❌  Please remove or reference these files before
committing!"
    exit 1
fi
```

Step 5: Customize the Script

Modify this logic as per your needs to specify only required file extensions or any to prevent from being committed. You can add or remove entries as needed.

Step 6: Test the Pre-Commit Hook

Try to commit changes that include one of the unused files you specified. The commit should be aborted with an error message indicating the unused file.

```
git add .
git commit -m "Test commit with unused file"
```

Step 7: Using with Visual Studio and Visual Studio Code

Both **Visual Studio** and **Visual Studio Code** will respect the **pre-commit** hook you created. When you attempt to commit changes through the IDE, the pre-commit hook will execute, and if any unused files are detected, the commit will be blocked.

Step 8: Using with Command Prompt and Visual Studio Developer Command Prompt

The **pre-commit** hook will also work seamlessly when using the Command Prompt or Visual Studio Developer Command Prompt. Simply follow the same steps to commit your changes, and the hook will enforce the rules you set.

Conclusion

By following the steps outlined in this document, you can effectively use Git pre-commit hooks to prevent unused files from being committed to your repository. This practice helps maintain a clean codebase and ensures that only relevant files are included in your version control history. Customize the script as needed to fit your project's requirements, and enjoy a more organized development workflow.

Thank you and never stop learning :)