Saturday, August 31, 2024

How to optimize your VSCode Workflow/Setup ?


Samuel SIKATI KENMOGNE
@KenmogneSikati

Overview

- Customizing the User Interface
  - Themes
  - Icons
  - Fonts
- Integrating Powerful Tools

    - Git Integration
    - Debugging Tools
    - Testing Tools
    - Language Support
    - Linters
    - Formatters
- Automating Repetitive Tasks
    - Custom Snippets
    - Tasks
    - Keybindings and Shortcuts
- Manage your settings using Profiles
    - Creating a Profile
    - Switching Between Profiles
    - Importing and Exporting Profiles
- Conclusion
- Further Reading
- Sources

Visual Studio Code (VSCode) is a popular code editor that offers a wide range of features and customization options to enhance your coding experience. By leveraging the power of extensions, settings, and configurations, you can tailor VSCode to suit your specific needs and preferences. In this post, we'll explore advanced techniques for optimizing your VSCode development environment, covering everything from customizing the user interface to integrating powerful tools and automating repetitive tasks. Whether you're a seasoned developer looking to boost your productivity or a newcomer eager to learn more about VSCode, this guide has something for everyone

Customizing the User Interface

One of the key benefits of VSCode is its flexibility when it comes to customizing the user interface. By tweaking the layout, themes, and fonts, you can create a personalized coding environment that suits your style. Here are some tips for customizing the user interface in VSCode:

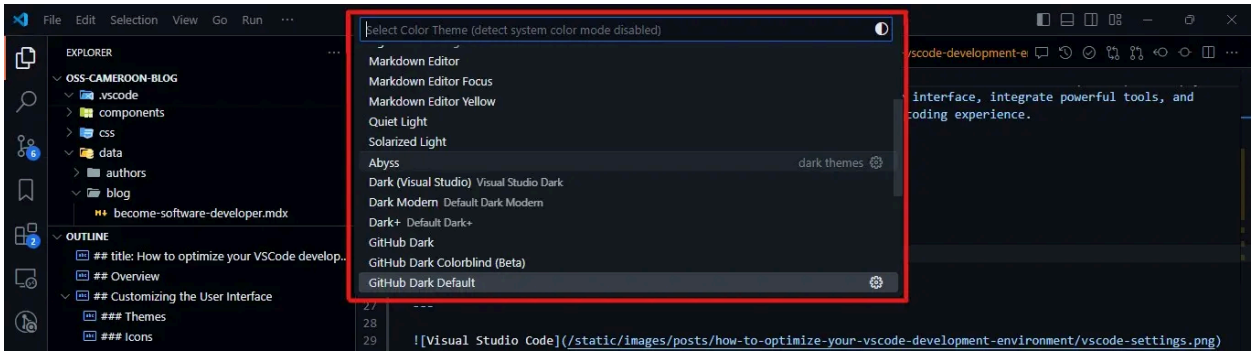
Themes

VSCode offers a wide range of themes that allow you to change the appearance of the editor. Whether you prefer a light theme for better readability or a dark theme for reduced eye strain, there's a theme to suit every taste. You can browse and install themes directly from the VSCode marketplace or create your own custom theme using the built-in theme editor.

To browse and switch themes inside Visual Studio Code:

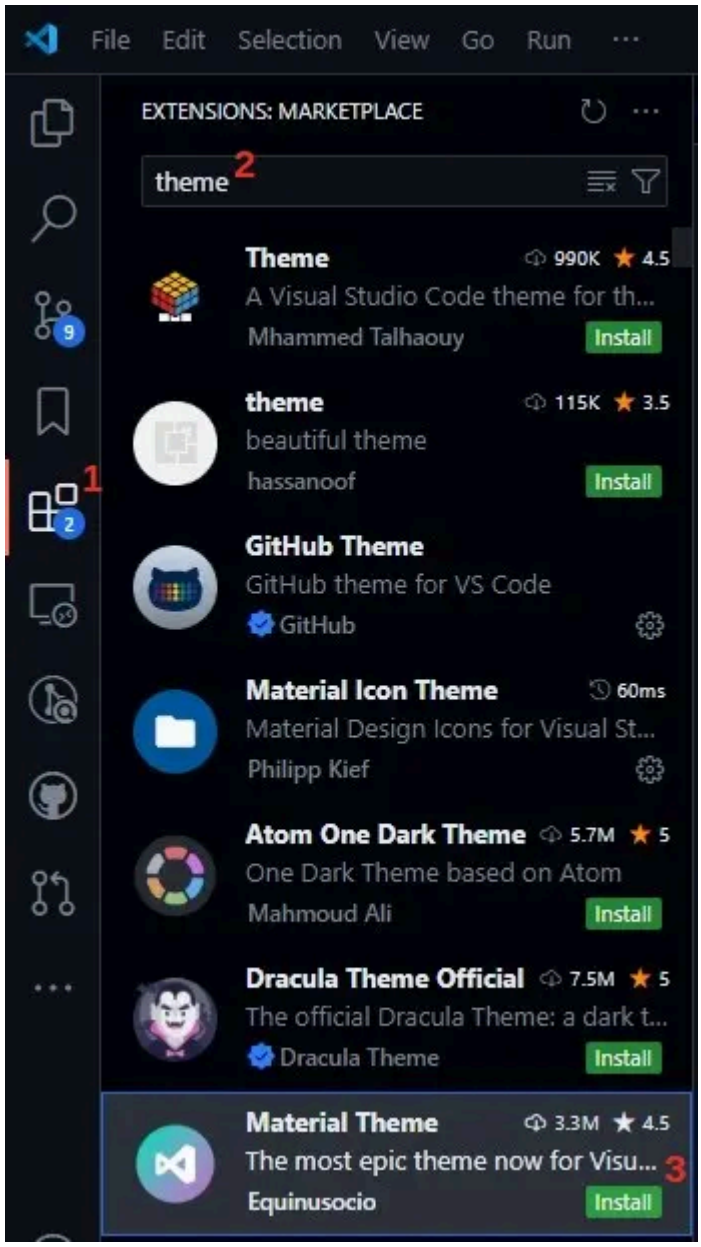
1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Preferences: Color Theme` and press `Enter`.
3. Use the arrow keys to browse through the available themes and press `Enter` to select one.

Alternatively, you can use the shortcut `Ctrl+K Ctrl+T` to quickly open the theme selection menu.



To install a new theme:

1. Open the Extensions view by pressing `Ctrl+Shift+X`.
2. In the search bar, type `theme` and browse through the available themes.
3. Click on the theme you want to install and then click the `Install` button.
4. After installation, you can activate the theme by following the steps to browse and switch themes.



Icons

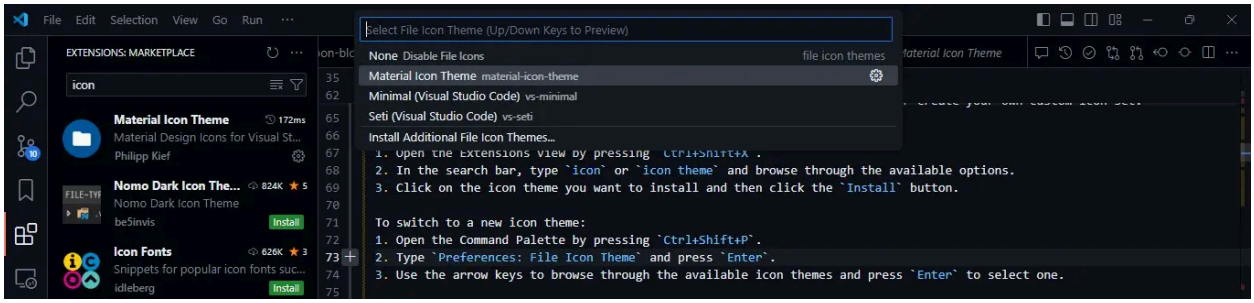
Icons play an important role in helping you quickly identify different file types and elements in your project. VSCode supports icon themes that replace the default icons with custom-designed icons for a more visually appealing experience. You can choose from a variety of icon themes available in the marketplace or create your own custom icon set.

To install a new icon theme:

1. Open the Extensions view by pressing `Ctrl+Shift+X`.
2. In the search bar, type `icon` or `icon theme` and browse through the available options.
3. Click on the icon theme you want to install and then click the `Install` button.

To switch to a new icon theme:

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Preferences: File Icon Theme` and press `Enter`.
3. Use the arrow keys to browse through the available icon themes and press `Enter` to select one.



Here are some popular icon theme extensions you might find useful:

- VSCode Icons
- Material Icon Theme
- Simple Icons
- Seti Icons

Fonts

Choosing the right font can make a big difference in your coding experience. VSCode allows you to customize the font family, size, and weight to suit your preferences. You can install custom fonts or choose from the built-in font options to find the perfect balance between readability and aesthetics.

To install a new font:

1. Open the Extensions view by pressing `Ctrl+Shift+X`.
2. In the search bar, type `font` and browse through the available options.
3. Click on the font extension you want to install and then click the `Install` button.

To customize your font settings:

1. Open the Command Palette by pressing `Ctrl+Shift+P`.

2. Type `Preferences: Open Settings (JSON)` and press `Enter`.
3. Add or modify the following settings to customize your font:

```
"editor.fontFamily": "Cascadia Code",
"editor.fontSize": 14,
"editor.fontWeight": "normal"
```

Here are some popular font extensions you might find useful:

- Fira Code
- Cascadia Code
- JetBrains Mono
- Hack
- IBM Plex

Integrating Powerful Tools

VSCode comes with a rich ecosystem of extensions that provide additional functionality and integration with external tools. By installing the right extensions, you can enhance your coding workflow and streamline common tasks. Here are some powerful tools you can integrate with VSCode:

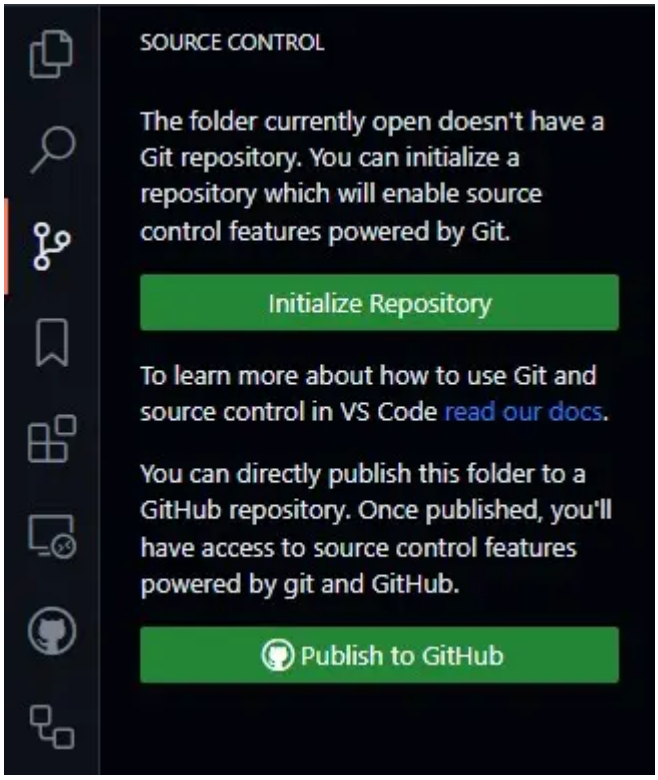
Git Integration

Git is a popular version control system that allows you to track changes in your codebase and collaborate with other developers. VSCode comes with built-in Git integration that provides a seamless interface for managing your Git repositories. You can view changes, commit code, and resolve merge conflicts directly from the editor.

To use Git integration in VSCode:

1. Open the Source Control view by clicking on the Source Control icon in the Activity Bar or pressing `Ctrl+Shift+G G`.
2. Initialize a new Git repository or clone an existing one.

3. Stage changes, commit code, and push/pull changes to/from your remote repository.



Useful Git/GitHub Extensions

In addition to the built-in Git integration, there are several extensions that can enhance your Git and GitHub workflow in VSCode. Here are some popular extensions:

- GitLens: GitLens supercharges the built-in Git capabilities of VSCode. It helps you visualize code authorship at a glance via Git blame annotations and provides powerful comparison commands, history exploration, and more.
- GitHub Pull Requests and Issues: This extension allows you to manage GitHub pull requests and issues directly within VSCode. You can review, comment, and merge pull requests, as well as create and track issues.
- Git Graph: Git Graph provides a visual representation of your repository's commit history. It allows you to explore the history, view details of commits, and perform Git actions directly from the graph.
- Git History: This extension allows you to view the history of files and branches in a Git repository. You can see the commit history, compare branches, and view file changes over time.
- GitHub Actions: This extension helps you manage GitHub Actions workflows directly from VSCode. You can create, edit, and monitor workflows, as well as

view logs and results.

Debugging Tools

VSCode offers robust debugging capabilities that allow you to inspect and troubleshoot your code with ease. By installing debugging extensions for your programming language or framework, you can set breakpoints, step through code, and analyze variables in real-time. This can help you identify and fix bugs more efficiently.

To set up debugging in VSCode:

1. Open the Run and Debug view by clicking on the Run icon in the Activity Bar or pressing `Ctrl+Shift+D`.
2. Click on the gear icon to create a `launch.json` file with your debugging configuration.
3. Set breakpoints in your code by clicking in the gutter next to the line numbers.
4. Start debugging by clicking the green play button or pressing `F5`.

Sample `launch.json` for a Node.js Project

Here is an example of a `launch.json` configuration for a Node.js project:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Next.js App",
      "runtimeExecutable": "yarn",
      "runtimeArgs": ["run", "dev"],
      "skipFiles": ["<node_internals>/**"],
      "cwd": "${workspaceFolder}",
      "outFiles": ["${workspaceFolder}/**/*.js"]
    },
    {
      "type": "node",
      "request": "attach",
      "name": "Attach to Process",
```

```
      "processId": "${command:PickProcess}",
      "skipFiles": ["<node_internals>/**"]
    }
  ]
}
```

- **Launch Program**: This configuration launches your Node.js application. The `program` attribute specifies the entry point of your application (e.g., `app.js`).
- **Attach to Process**: This configuration allows you to attach the debugger to an already running Node.js process. The `processId` attribute uses the `PickProcess` command to let you select the process to attach to.

Steps to Use the Sample `launch.json`

1. **Create the `launch.json` File**:
   - Open the Run and Debug view by clicking on the Run icon in the Activity Bar or pressing `Ctrl+Shift+D`.
   - Click on the gear icon to create a `launch.json` file.
   - Select **Node.js** from the list of environments.
2. **Add the Configuration**:
   - Replace the content of the launch.json file with the sample configuration provided above.
3. **Set Breakpoints**:
   - Open your JavaScript/Node.js file (e.g., `app.js`).
   - Click in the gutter next to the line numbers to set breakpoints.
4. **Start Debugging**:
   - Select the desired configuration (e.g., "Launch Program") from the dropdown in the Run and Debug view.
   - Click the green play button or press `F5` to start debugging. Debugging in VSCode

By following these steps, you can set up and use the debugging tools in VSCode to inspect and troubleshoot your code more effectively. The sample launch.json configuration provides a starting point for debugging a Node.js project, but you can customize it to fit the specific needs of your project.

Testing Tools

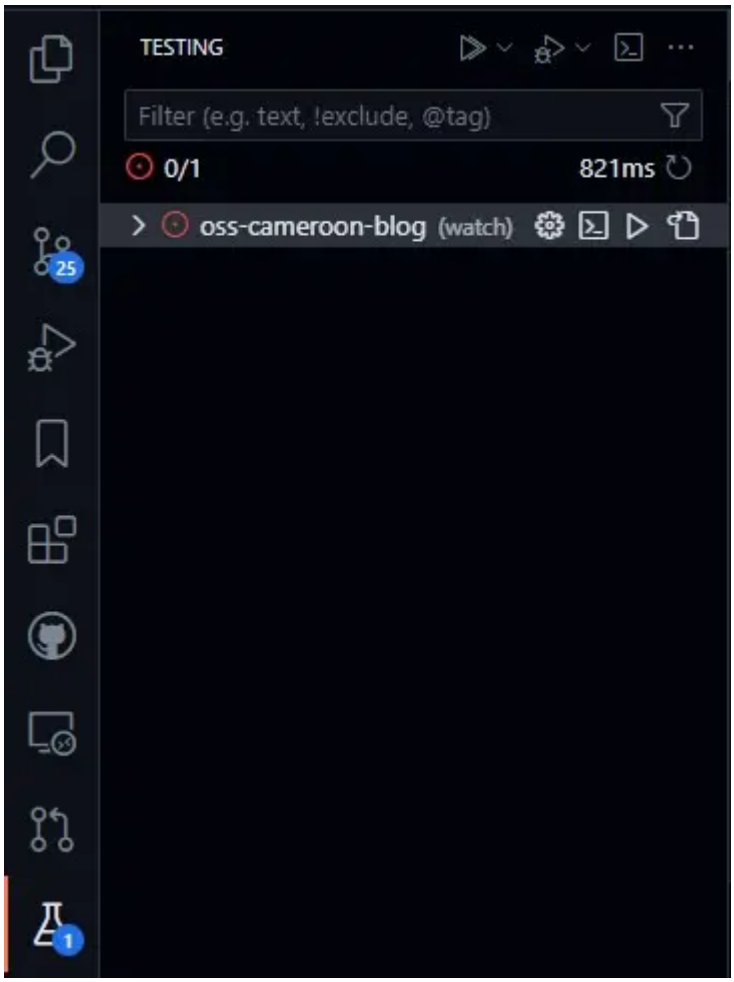
Testing is a crucial part of the development process. VSCode supports various testing frameworks through extensions, allowing you to run and debug tests directly from the editor. Popular testing extensions include:

- **Jest**: For JavaScript and TypeScript testing.
- **PyTest**: For Python testing.
- **JUnit**: For Java testing.

These extensions provide features like test discovery, running tests, and viewing test results within the editor.

To set up testing in VSCode:

1. Install the appropriate testing extension for your language or framework from the Extensions view (`Ctrl+Shift+X`).
2. Configure your testing framework in your project settings or configuration files.
3. Open the Testing view by clicking on the beaker icon in the Activity Bar.
4. Run and debug tests directly from the Testing view.



Language Support

VSCode supports a wide range of programming languages and frameworks out of the box. However, you can enhance the editor's language support by installing language-specific extensions. These extensions provide syntax highlighting, code completion, and other language-specific features to improve your coding experience.

To enhance language support in VSCode:

1. Open the Extensions view by pressing `Ctrl+Shift+X`.
2. In the search bar, type the name of the language or framework you want to support (e.g., `Python`, `JavaScript`, `C#`).
3. Click on the extension you want to install and then click the `Install` button.

Linters

Linters help you maintain code quality by analyzing your code for potential errors, coding standard violations, and other issues. VSCode supports a wide range of

linters through extensions. Some popular linters include:

- **ESLint**: For JavaScript and TypeScript.
- **Pylint**: For Python.
- **Rubocop**: For Ruby.

These linters can be configured to run automatically on file save or manually, helping you catch issues early in the development process.

To set up a linter in VSCode:

1. Install the appropriate linter extension from the Extensions view (`Ctrl+Shift+X`).
2. Configure the linter settings in your project configuration files (e.g., `.eslintrc` , `pylintrc`).
3. The linter will automatically analyze your code and highlight issues in the editor.

Formatters

Code formatters help you maintain a consistent coding style by automatically formatting your code according to predefined rules. VSCode supports various formatters through extensions. Some popular formatters include:

- **Prettier**: For JavaScript, TypeScript, and other languages.
- **Black**: For Python.
- **Beautify**: For HTML, CSS, and JavaScript.

These formatters can be configured to run automatically on file save, ensuring that your codebase remains clean and consistent.

To set up a formatter in VSCode:

1. Install the appropriate formatter extension from the Extensions view (`Ctrl+Shift+X`).
2. Configure the formatter settings in your project configuration files (e.g., `.prettierrc` , `pyproject.toml`).

3. The formatter will automatically format your code on save or when triggered manually.

Automating Repetitive Tasks

Automation is a key aspect of optimizing your development workflow. By automating repetitive tasks, you can save time and focus on more important aspects of your code. VSCode offers several features that allow you to automate common tasks, such as snippets, tasks, and extensions.

Custom Snippets

Snippets are code templates that can be inserted into your code with a simple shortcut. VSCode comes with built-in snippets for common programming constructs, but you can create your own custom snippets to speed up your coding process. By defining custom snippets for frequently used code patterns, you can write code faster and more efficiently.

To create a custom snippet:

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Preferences: Configure User Snippets` and press `Enter`.
3. Select the language for which you want to create a snippet.
4. Add your custom snippet in the JSON file that opens.

Example of a custom snippet for a Markdown/MDX list:

```
{
  "Markdown List": {
    "prefix": "mdlist",
    "body": ["- ${1:Item 1}", "- ${2:Item 2}", "- ${3:Item 3}"],
    "description": "Create a Markdown list"
  }
}
```

To add this snippet to your VSCode configuration:

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Preferences: Configure User Snippets` and press `Enter`.
3. Select `mdx` from the list of languages.
4. Add the snippet to the `mdx.json` file.

Now, you can type `mdlist` in any Markdown file and press `Tab` to insert a pre-formatted list.

Tasks

Tasks allow you to define custom scripts that automate common build tasks, test suites, or deployment processes. By creating task configurations in your `tasks.json` file, you can run scripts with a single command and streamline your development workflow. Tasks can be triggered manually or set to run automatically based on specific events.

To create a custom task:

1. Open the Command Palette by pressing Ctrl+Shift+P.
2. Type Tasks: Configure Task and press Enter.
3. Select Create tasks.json file from template and choose the type of task you want to create.
4. Define your task in the tasks.json file.

Example of a custom task to run a build script:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "build",
      "type": "shell",
      "command": "yarn run build",
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "problemMatcher": []
```

```
    },
    {
      "label": "start",
      "type": "shell",
      "command": "yarn start",
      "group": {
        "kind": "build",
        "isDefault": false
      },
      "problemMatcher": []
    }
  ]
}
```

Running the Custom Task

Once you have defined your custom task in the `tasks.json` file, you can easily run it using VSCode's built-in task runner.

Using the Command Palette

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Tasks: Run Build Task` and press `Enter`.
3. If you have multiple build tasks defined, select the task labeled "`build`" from the list.

Using the Keyboard Shortcut

1. Press `Ctrl+Shift+B` to run the default build task.
2. If you have multiple build tasks defined, select the task labeled "`build`" from the list.

By following these steps, you can quickly and easily run your custom build task in VSCode, streamlining your development workflow and automating repetitive tasks.

Keybindings and Shortcuts

Keybindings and shortcuts are essential for improving productivity in VSCode. By customizing keybindings, you can tailor the editor to your workflow and speed up common tasks. Custom keybindings offer several benefits:

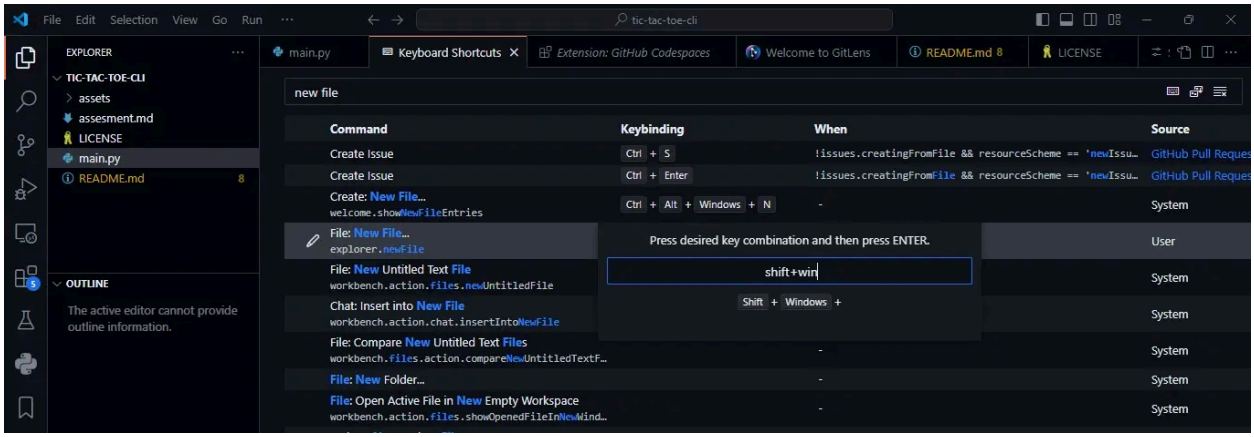
- **Efficiency**: Speed up common tasks by using shortcuts.
- **Customization**: Tailor the editor to your workflow.
- **Consistency**: Maintain a consistent set of keybindings across different projects and environments.

VSCode also allows you to import keybindings from other popular IDEs, making it easier to transition and maintain your productivity. Here's how you can manage keybindings, use shortcuts effectively, and import keybindings from other IDEs:

Customizing Keybindings

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Preferences: Open Keyboard Shortcuts` and press `Enter`.
3. In the Keyboard Shortcuts editor, you can search for commands and assign new keybindings by clicking on the pencil icon next to the command.

Alternatively, you can use the shortcut `Ctrl+K Ctrl+S` to quickly open the Keyboard Shortcuts editor.



You can also setup keybinding without the editor by adding custom user keybinding which will override the default keybinding avialable on the editor

Steps to Custom Keybinding

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Preferences: Open Keyboard Shortcuts (JSON)` and press `Enter`.
3. Edit the keybindings.json File: For Example you can add the following entry to the `keybindings.json` file:

```
[
    {
        "key": "Ctrl+n",
        "command": "explorer.newFile"
    },
    {
        "key": "Shift+F6",
        "command": "workbench.action.tasks.runTask",
        "args": "start"
    }
]
```

4. Save the keybindings.json File:

Now you can press Shift+F6 to run the "start" task, which will execute `yarn start` to start your Next.js application.

Useful Shortcuts

Here are some useful shortcuts to enhance your productivity in VSCode:

Operation	Windows/Linux	Mac
Command Palette	`Ctrl+Shift+P`	`⇧⌘P`
Toggle Terminal	`Ctrl+``	`^``
Toggle Sidebar	`Ctrl+B`	`⌘B`
Open File	`Ctrl+P`	`⌘P`
Find in Files	`Ctrl+Shift+F`	`⌘⇧F`
Go to Definition	`F12`	`F12`
Rename Symbol	`F2`	`F2`
Format Document	`Shift+Alt+F`	`⇧⌥F`
Toggle Focus Mode	`F11`	`⌘^F`

For more detailed keyboard shortcuts, you can refer to the official VSCode keyboard reference cheatsheets:

- Windows Keyboard Shortcuts

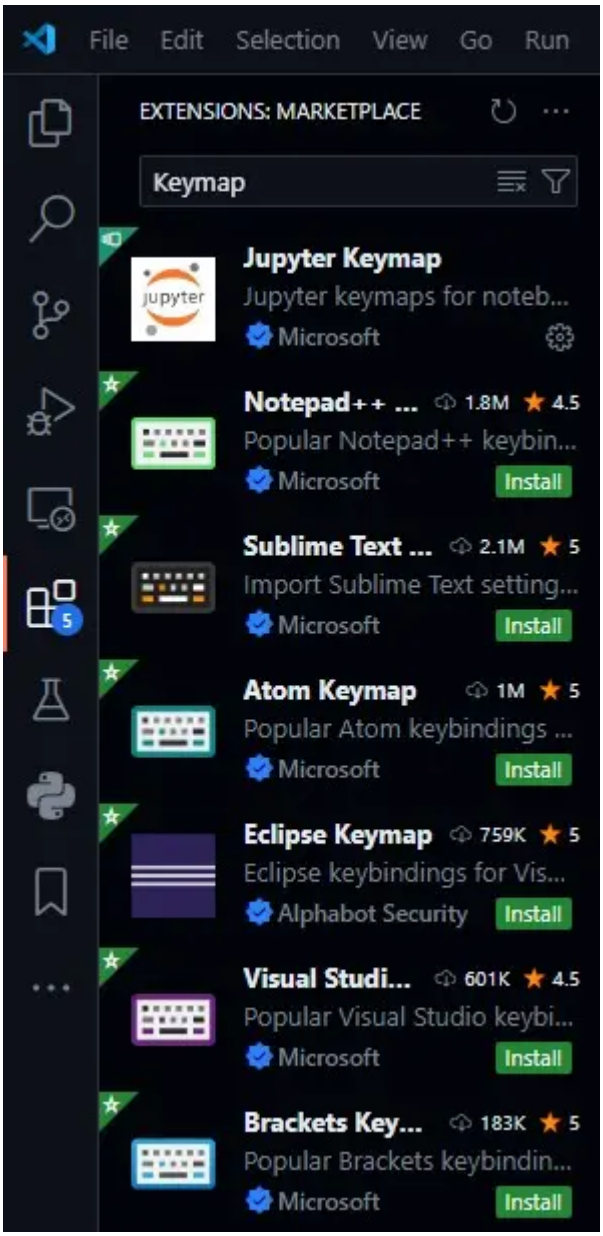
- Mac Keyboard Shortcuts
- Linux Keyboard Shortcuts

These shortcuts and references will help you navigate and use VSCode more efficiently across different operating systems.

Importing Keybindings from Other IDEs

VSCode supports importing keybindings from other popular IDEs, allowing you to use familiar shortcuts. Here's how you can import keybindings:

1. Open the Extensions view by pressing `Ctrl+Shift+X`.
2. In the search bar, type the name of the IDE you want to import keybindings from (e.g., `Visual Studio Keymap`, `Sublime Text Keymap`, `Atom Keymap`).
3. Click on the keymap extension you want to install and then click the `Install` button.
4. After installation, the keybindings from the selected IDE will be applied to VSCode.



By leveraging these keybinding options, you can create a more efficient and personalized coding environment in VSCode.

Manage your settings using Profiles

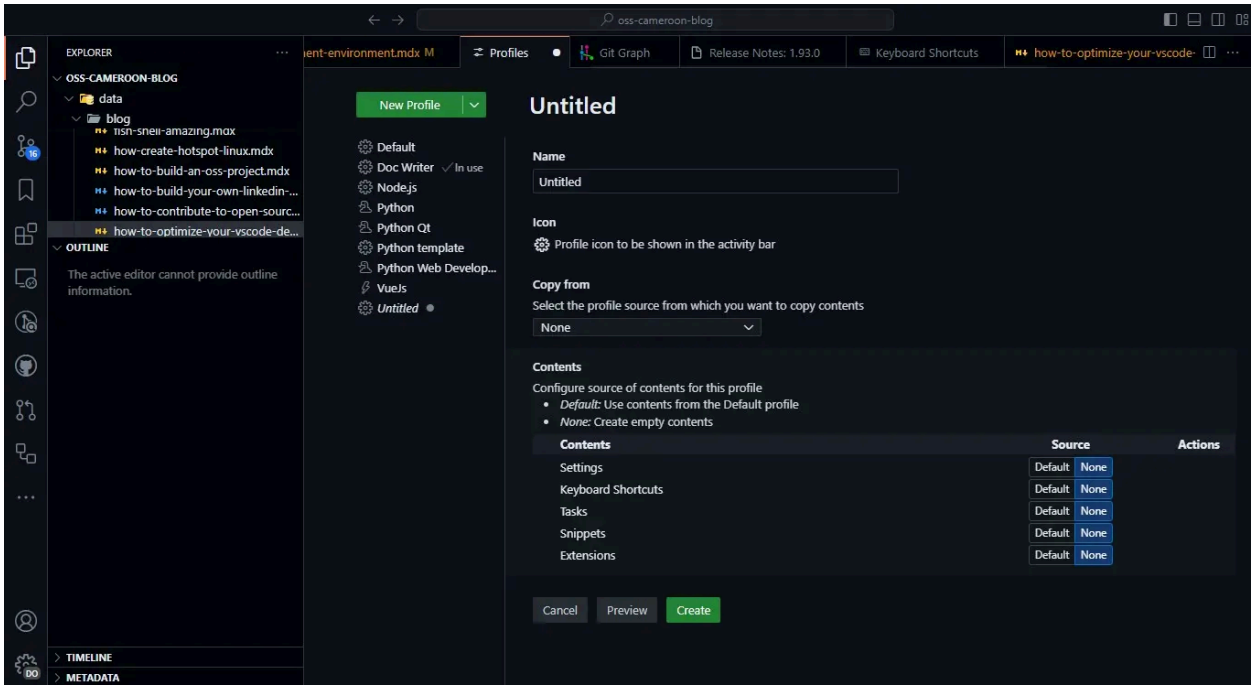
VSCode profiles allow you to create and manage different sets of configurations, extensions, and settings tailored to specific workflows or projects. This feature is particularly useful if you work on multiple projects with different requirements or if you want to separate your work and personal development environments.

Creating a Profile

Creating a profile in VSCode allows you to set up a unique environment for different projects or workflows. This can include specific extensions, settings, and

configurations that are best suited for the task at hand.

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Profiles: New Profile` and press `Enter`.
3. Enter a name for your new profile and press `Enter`.

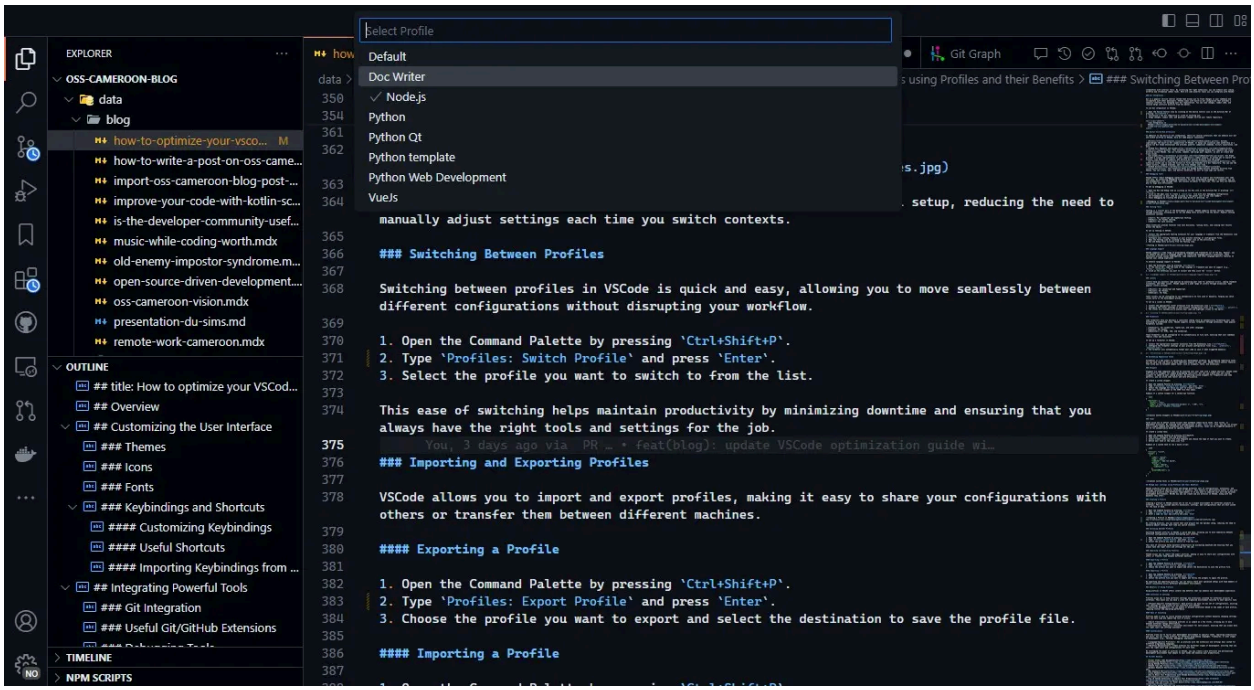


By creating profiles, you can ensure that each project has the optimal setup, reducing the need to manually adjust settings each time you switch contexts.

Switching Between Profiles

Switching between profiles in VSCode is quick and easy, allowing you to move seamlessly between different configurations without disrupting your workflow.

1. Open the Command Palette by pressing `Ctrl+Shift+P`.
2. Type `Profiles: Switch Profile` and press `Enter`.
3. Select the profile you want to switch to from the list.



This ease of switching helps maintain productivity by minimizing downtime and ensuring that you always have the right tools and settings for the job.

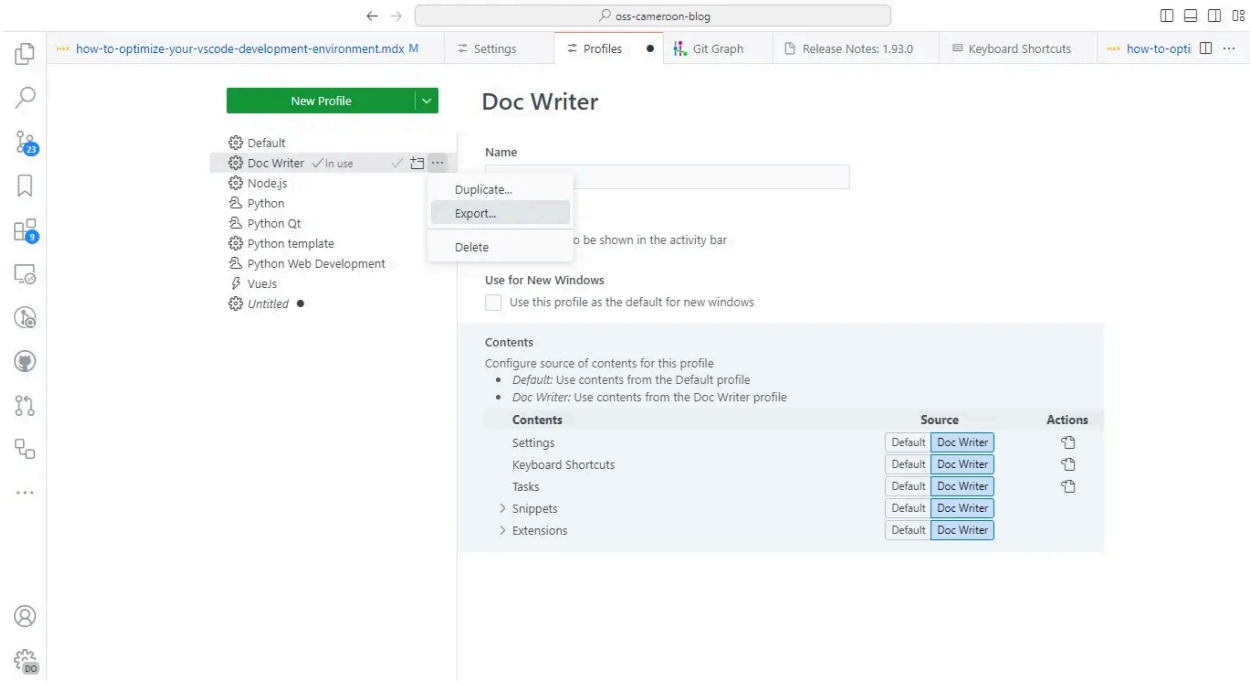
Importing and Exporting Profiles

VSCode allows you to import and export profiles, making it easy to share your configurations with others or transfer them between different machines.

Exporting a Profile

Exporting a profile in Visual Studio Code allows you to save your current settings, extensions, and configurations, making it easy to transfer your setup to another machine or share it with others.

1. **Open the Command Palette**:
    - Press `Ctrl+Shift+P` to open the Command Palette.
2. **Export the Profile**:
    - Type `Profiles: Export Profile` and press `Enter`.
    - Choose the profile you want to export.
    - Select the destination to save the profile file. The profile will be exported in a `.code-profile` format, which is a JSON file containing all your settings, extensions, and configurations.



You can also create new profiles based on an exported profile. This allows you to customize and extend the configurations for different projects or environments.

Conclusion

Optimizing your VSCode development environment is a continuous process that involves experimenting with different settings, extensions, and configurations to find what works best for you. By customizing the user interface, integrating powerful tools, and automating repetitive tasks, you can create a more efficient and personalized coding experience in VSCode. Whether you're a seasoned developer looking to boost your productivity or a newcomer eager to learn more about VSCode, these advanced techniques will help you take your coding skills to the next level. Try out some of the tips and tricks mentioned in this guide and see how they can enhance your coding workflow. Happy coding!

Further Reading

- Visual Studio Code Documentation
- Customizing VSCode
- Using VSCode Profiles
- VSCode Tips and Tricks

- How to Boost Your Productivity with VSCode Extensions
- Top 10 VSCode Extensions to Improve Your Productivity
- VSCode Tips and Tricks for Power Users
- Mastering VSCode Settings and Configurations

Sources

- Visual Studio Code. (n.d.). Retrieved from https://code.visualstudio.com/
- Ngrok. (n.d.). Retrieved from https://ngrok.com/
- Microsoft Docs. (n.d.). Retrieved from https://docs.microsoft.com/
- FreeCodeCamp. (n.d.). Retrieved from https://www.freecodecamp.org/news/boost-productivity-vscode-extensions/
- Duomly. (n.d.). Retrieved from https://dev.to/duomly/top-10-vscode-extensions-to-improve-your-productivity-4f1m
- Smashing Magazine. (n.d.). Retrieved from https://www.smashingmagazine.com/2020/04/vscode-tips-tricks-power-users/
- DigitalOcean. (n.d.). Retrieved from https://www.digitalocean.com/community/tutorials/mastering-vscode-settings-and-configurations

---

Discuss on Twitter • View on GitHub

TAGS
VSCODE CUSTOMIZATION DEVELOPMENT-ENVIRONMENT PRODUCTIVITY EXTENSIONS SETTINGS DEVELOPER-TOOLS

PREVIOUS ARTICLE
OSDD : Open Source Driven Development

← Back to the blog