Basic Editing

Visual Studio Code is an editor first and foremost, and includes the features you need for highly productive source code editing. This topic takes you through the basics of the editor and helps you get moving with your code.

Keyboard shortcuts

Being able to keep your hands on the keyboard when writing code is crucial for high productivity. VS Code has a rich set of default keyboard shortcuts as well as allowing you to customize them.

- Keyboard Shortcuts Reference (/docs/getstarted/keybindings#_keyboard-shortcuts-reference) - Learn the most commonly used and popular keyboard shortcuts by downloading the reference sheet.
- Install a Keymap extension (/docs/getstarted/keybindings#_keymap-extensions) - Use the keyboard shortcuts of your old editor (such as Sublime Text, Atom, and Vim) in VS Code by installing a Keymap extension.
- Customize Keyboard Shortcuts (/docs/getstarted/keybindings#_keyboard-shortcuts-editor) - Change the default keyboard shortcuts to fit your style.

Multiple selections (multi-cursor)

VS Code supports multiple cursors for fast simultaneous edits. You can add secondary cursors (rendered thinner) with `Alt+Click` . Each cursor operates independently based on the context it sits in. A common way to add more cursors is with `Ctrl+Alt+Down` or `Ctrl+Alt+Up` that insert cursors below or above.

> **Note:** Your graphics card driver (for example NVIDIA) might overwrite these default shortcuts.

`Ctrl+D` selects the word at the cursor, or the next occurrence of the current selection.

> **Tip:** You can also add more cursors with `Ctrl+Shift+L` , which will add a selection at each occurrence of the current selected text.

Multi-cursor modifier

If you'd like to change the modifier key for applying multiple cursors to `Cmd+Click` on macOS and `Ctrl+Click` on Windows and Linux, you can do so with the `editor.multiCursorModifier` setting (/docs/getstarted/settings). This lets users coming from other editors such as Sublime Text or Atom continue to use the keyboard modifier they are familiar with.

The setting can be set to:

- `ctrlCmd` - Maps to `Ctrl` on Windows and `Cmd` on macOS.
- `alt` - The existing default `Alt` .

There's also a menu item **Use Ctrl+Click for Multi-Cursor** in the **Selection** menu to quickly toggle this setting.

The **Go to Definition** and **Open Link** gestures will also respect this setting and adapt such that they do not conflict. For example, when the setting is `ctrlCmd` , multiple cursors can be added with `Ctrl/Cmd+Click` , and opening links or going to definition can be invoked with `Alt+Click` .

Shrink/expand selection

Quickly shrink or expand the current selection. Trigger it with `Shift+Alt+Left` and `Shift+Alt+Right` .

Here's an example of expanding the selection with `Shift+Alt+Right` :

Column (box) selection

Place the cursor in one corner and then hold `Shift+Alt` while dragging to the opposite corner:

Note: This changes to `Shift+Ctrl/Cmd` when using `Ctrl/Cmd` as multi-cursor modifier.

There are also default key bindings for column selection on macOS and Windows, but not on Linux.

Key	**Command**	**Command ID**
`Ctrl+Shift+Alt+Down`	Column Select Down	`cursorColumnSelectDown`

Key	Command	Command ID
`Ctrl+Shift+Alt+Up`	Column Select Up	`cursorColumnSelectUp`
`Ctrl+Shift+Alt+Left`	Column Select Left	`cursorColumnSelectLeft`
`Ctrl+Shift+Alt+Right`	Column Select Right	`cursorColumnSelectRight`
`Ctrl+Shift+Alt+PageDown`	Column Select Page Down	`cursorColumnSelectPageDown`
`Ctrl+Shift+Alt+PageUp`	Column Select Page Up	`cursorColumnSelectPageUp`

You can edit (/docs/getstarted/keybindings) your `keybindings.json` to bind them to something more familiar if you want.

Column Selection mode

The user setting **Editor: Column Selection** controls this feature. Once this mode is entered, as indicated in the Status bar, the mouse gestures and the arrow keys will create a column selection by default. This global toggle is also accessible via the **Selection** > **Column Selection Mode** menu item. In addition, one can also disable Column Selection mode from the Status bar.

Save / Auto Save

By default, VS Code requires an explicit action to save your changes to disk, `Ctrl+S` .

However, it's easy to turn on `Auto Save` , which will save your changes after a configured delay or when focus leaves the editor. With this option turned on, there is no need to explicitly save the file. The easiest way to turn on `Auto Save` is with the **File** > **Auto Save** toggle that turns on and off save after a delay.

For more control over `Auto Save` , open User or Workspace settings (/docs/getstarted/settings) and find the associated settings:

- `files.autoSave` : Can have the values:
  - `off` - to disable auto save.
  - `afterDelay` - to save files after a configured delay (default 1000 ms).
  - `onFocusChange` - to save files when focus moves out of the editor of the dirty file.
  - `onWindowChange` - to save files when the focus moves out of the VS Code window.
- `files.autoSaveDelay` : Configures the delay in milliseconds when `files.autoSave` is configured to `afterDelay` . The default is 1000 ms.

Hot Exit

VS Code will remember unsaved changes to files when you exit by default. Hot exit is triggered when the application is closed via **File** > **Exit** (**Code** > **Quit** on macOS) or when the last window is closed.

You can configure hot exit by setting `files.hotExit` to the following values:

- `"off"` : Disable hot exit.
- `"onExit"` : Hot exit will be triggered when the application is closed, that is when the last window is closed on Windows/Linux or when the `workbench.action.quit` command is triggered (from the **Command Palette**, keyboard shortcut or menu). All windows without folders opened will be restored upon next launch.
- `"onExitAndWindowClose"` : Hot exit will be triggered when the application is closed, that is when the last window is closed on Windows/Linux or when the `workbench.action.quit` command is triggered (from the **Command Palette**, keyboard shortcut or menu), and also for any window with a folder opened regardless of whether it is the last window. All windows without folders opened will be restored upon next launch. To restore folder windows as they were before shutdown, set `window.restoreWindows` to `all` .

If something happens to go wrong with hot exit, all backups are stored in the following folders for standard install locations:

- **Windows** `%APPDATA%\Code\Backups`
- **macOS** `$HOME/Library/Application Support/Code/Backups`
- **Linux** `$HOME/.config/Code/Backups`

Find and Replace

VS Code allows you to quickly find text and replace in the currently opened file. Press `Ctrl+F` to open the Find Widget in the editor, search results will be highlighted in the editor, overview ruler and minimap.

If there are more than one matched result in the current opened file, you can press `Enter` and `Shift+Enter` to navigate to next or previous result when the find input box is focused.

Seed Search String From Selection

When the Find Widget is opened, it will automatically populate the selected text in the editor into the find input box. If the selection is empty, the word under the cursor will be inserted into the input box instead.

This feature can be turned off by setting `editor.find.seedSearchStringFromSelection` to `false`.

Find In Selection

By default, the find operations are run on the entire file in the editor. It can also be run on selected text. You can turn this feature on by clicking the hamburger icon on the Find Widget.

If you want it to be the default behavior of the Find Widget, you can set `editor.find.autoFindInSelection` to `always`, or to `multiline`, if you want it to be run on selected text only when multiple lines of content are selected.

Advanced find and replace options

In addition to find and replace with plain text, the Find Widget also has three advanced search options:

- Match Case
- Match Whole Word
- Regular Expression

The replace input box support case preserving, you can turn it on by clicking the Preserve Case (**AB**) button.

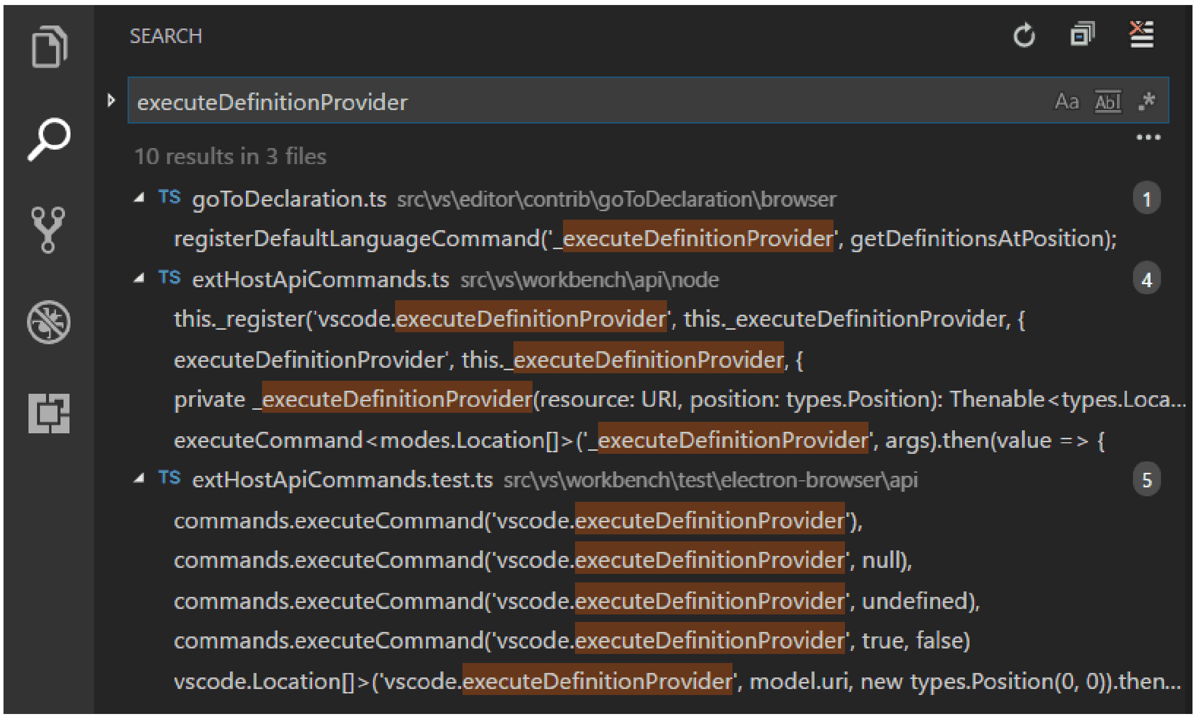
Multiline support and Find Widget resizing

You can search multiple line text by pasting the text into the Find input box and Replace input box. Pressing `Ctrl+Enter` inserts a new line in the input box.

While searching long text, the default size of Find Widget might be too small. You can drag the left sash to enlarge the Find Widget or double click the left sash to maximize it or shrink it to its default size.

Search across files

VS Code allows you to quickly search over all files in the currently opened folder. Press `Ctrl+Shift+F` and enter your search term. Search results are grouped into files containing the search term, with an indication of the hits in each file and its location. Expand a file to see a preview of all of the hits within that file. Then single-click on one of the hits to view it in the editor.

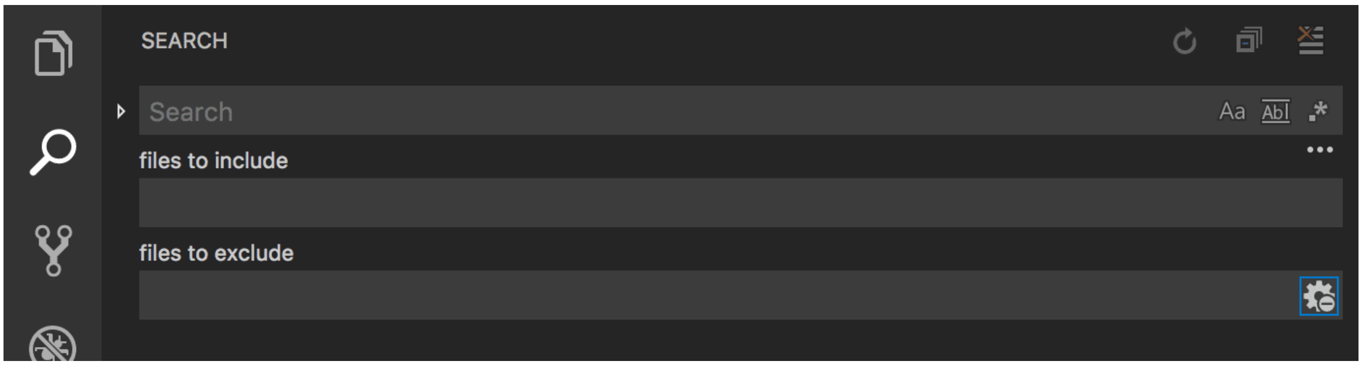


```
executeDefinitionProvider
10 results in 3 files
TS goToDeclaration.ts src\vs\editor\contrib\goToDeclaration\browser   1
    registerDefaultLanguageCommand('_executeDefinitionProvider', getDefinitionsAtPosition);
TS extHostApiCommands.ts src\vs\workbench\api\node   4
    this._register('vscode.executeDefinitionProvider', this._executeDefinitionProvider, {
    executeDefinitionProvider', this._executeDefinitionProvider, {
    private _executeDefinitionProvider(resource: URI, position: types.Position): Thenable<types.Loca...
    executeCommand<modes.Location[]>('_executeDefinitionProvider', args).then(value => {
TS extHostApiCommands.test.ts src\vs\workbench\test\electron-browser\api   5
    commands.executeCommand('vscode.executeDefinitionProvider'),
    commands.executeCommand('vscode.executeDefinitionProvider', null),
    commands.executeCommand('vscode.executeDefinitionProvider', undefined),
    commands.executeCommand('vscode.executeDefinitionProvider', true, false)
    vscode.Location[]>('vscode.executeDefinitionProvider', model.uri, new types.Position(0, 0)).then...
```

> **Tip:** We support regular expression searching in the search box, too.

You can configure advanced search options by clicking the ellipsis (**Toggle Search Details**) below the search box on the right (or press `Ctrl+Shift+J`). This will show additional fields to configure the search.

Advanced search options



In the two input boxes below the search box, you can enter patterns to include or exclude from the search. If you enter `example` , that will match every folder and file named `example` in the workspace. If you enter `./example` , that will match the folder `example/` at the top level of your workspace. Use `,` to separate multiple patterns. Paths must use forward slashes. You can also use glob syntax:

- `*` to match zero or more characters in a path segment
- `?` to match on one character in a path segment
- `**` to match any number of path segments, including none
- `{}` to group conditions (for example `{**/*.html,**/*.txt}` matches all HTML and text files)
- `[]` to **declare** a range of characters to match (`example.[0-9]` to match on `example.0` , `example.1` , …)
- `[!...]` to negate a range of characters to match (`example.[!0-9]` to match on `example.a` , `example.b` , but not `example.0`)

VS Code excludes some folders by default to reduce the number of search results that you are not interested in (for example: `node_modules`). Open settings (/docs/getstarted/settings) to change these rules under the `files.exclude` and `search.exclude` section.

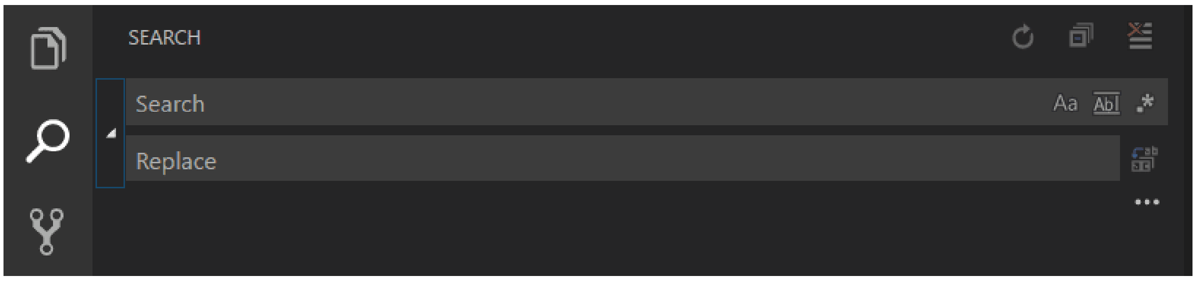
Note that glob patterns in the search view work differently than in settings such as `files.exclude` and `search.exclude` . In the settings, you must use `**/example` to match a folder named `example` in subfolder `folder1/example` in your workspace. In the search view, the `**` prefix is assumed. The glob patterns in these settings are always evaluated relative to the path of the workspace folder.

Also note the **Use Exclude Settings and Ignore Files** toggle button in the **files to exclude** box. The toggle determines whether to exclude files that are ignored by your `.gitignore` files and/or matched by your `files.exclude` and `search.exclude` settings.

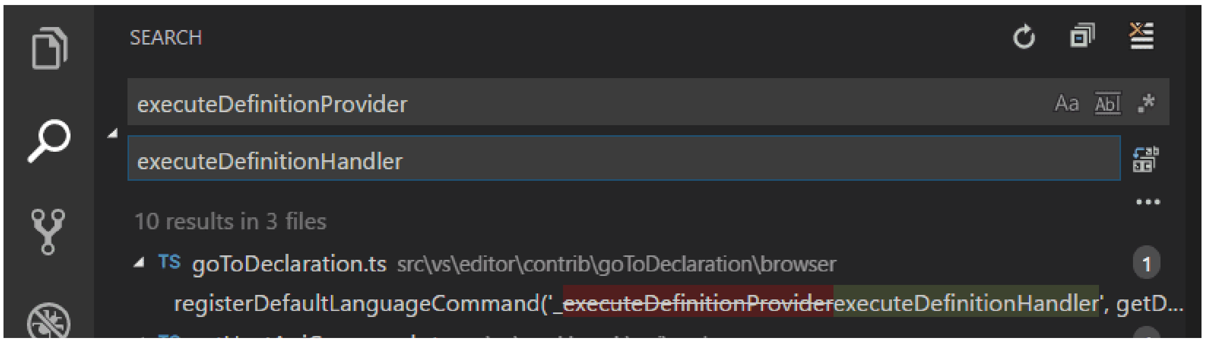
> **Tip:** From the Explorer, you can right-click on a folder and select **Find in Folder** to search inside a folder only.

Search and replace

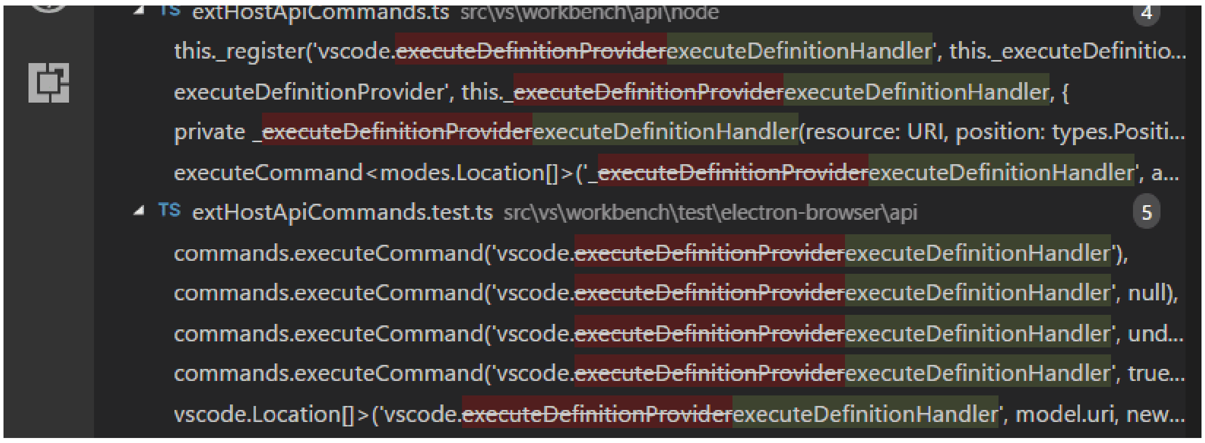
You can also Search and Replace across files. Expand the Search widget to display the Replace text box.



When you type text into the Replace text box, you will see a diff display of the pending changes. You can replace across all files from the Replace text box, replace all in one file or replace a single change.



executeDefinitionProvider
executeDefinitionHandler
10 results in 3 files
TS goToDeclaration.ts src\vs\editor\contrib\goToDeclaration\browser 1
registerDefaultLanguageCommand('_~~executeDefinitionProvider~~executeDefinitionHandler', getD...



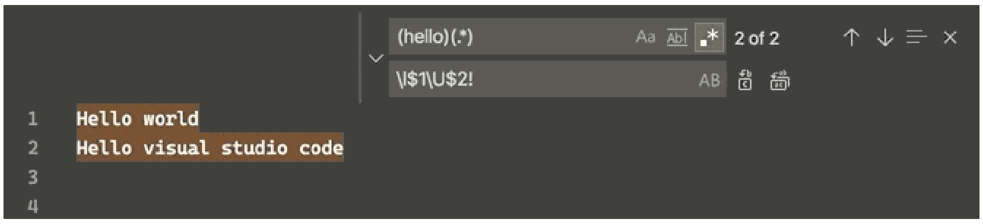
```
extHostApiCommands.ts src\vs\workbench\api\node                                           4
    this._register('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', this._executeDefinitio...
    executeDefinitionProvider', this._~~executeDefinitionProvider~~executeDefinitionHandler, {
    private _~~executeDefinitionProvider~~executeDefinitionHandler(resource: URI, position: types.Positi...
    executeCommand<modes.Location[]>('_~~executeDefinitionProvider~~executeDefinitionHandler', a...
extHostApiCommands.test.ts src\vs\workbench\test\electron-browser\api                     5
    commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler'),
    commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', null),
    commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', und...
    commands.executeCommand('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', true...
    vscode.Location[]>('vscode.~~executeDefinitionProvider~~executeDefinitionHandler', model.uri, new...
```

Tip: You can quickly reuse a previous search term by using `Down` and `Up` to navigate through your search term history.

Case changing in regex replace

VS Code supports changing the case of regex matching groups while doing Search and Replace in the editor or globally. This is done with the modifiers `\u\U\l\L`, where `\u` and `\l` will upper/lowercase a single character, and `\U` and `\L` will upper/lowercase the rest of the matching group.

Example:



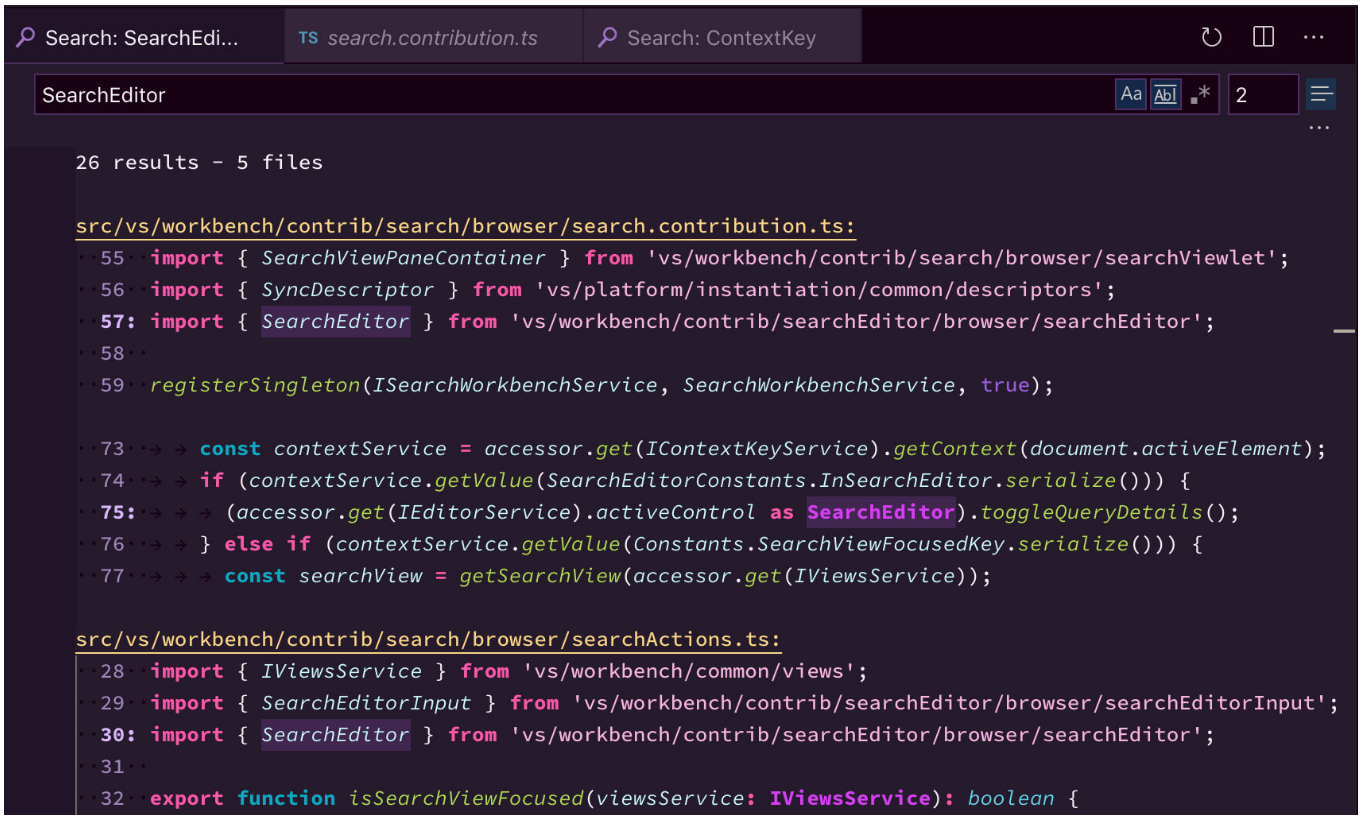
```
(hello)(.*)      Aa  Ab|  .*   2 of 2
\l$1\U$2!        AB

Hello world
Hello visual studio code
```

The modifiers can also be stacked - for example, `\u\u\u$1` will uppercase the first three characters of the group, or `\l\U$1` will lowercase the first character, and uppercase the rest. The capture group is referenced by `$n` in the replacement string, where `n` is the order of the capture group.

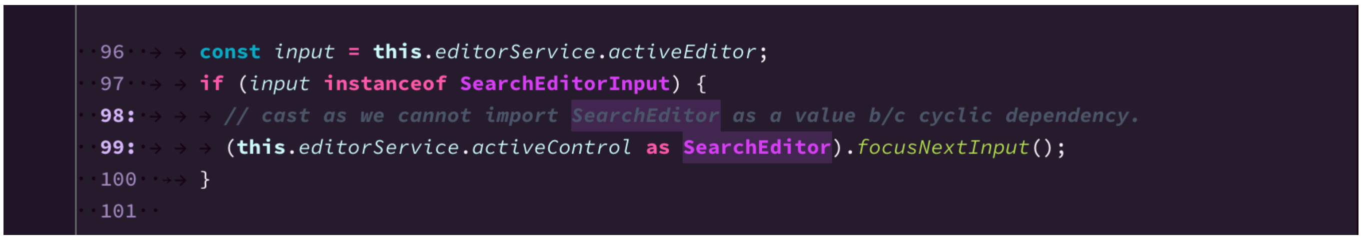
Search Editor

Search Editors let you view workspace search results in a full-sized editor, complete with syntax highlighting and optional lines of surrounding context.

Below is a search for the word 'SearchEditor' with two lines of text before and after the match for context:



```
Search: SearchEdi...   TS search.contribution.ts   Search: ContextKey

SearchEditor                                                   Aa  Ab|  .*   2

26 results - 5 files

src/vs/workbench/contrib/search/browser/search.contribution.ts:
  55  import { SearchViewPaneContainer } from 'vs/workbench/contrib/search/browser/searchViewlet';
  56  import { SyncDescriptor } from 'vs/platform/instantiation/common/descriptors';
  57: import { SearchEditor } from 'vs/workbench/contrib/searchEditor/browser/searchEditor';
  58
  59  registerSingleton(ISearchWorkbenchService, SearchWorkbenchService, true);

  73      const contextService = accessor.get(IContextKeyService).getContext(document.activeElement);
  74      if (contextService.getValue(SearchEditorConstants.InSearchEditor.serialize())) {
  75:         (accessor.get(IEditorService).activeControl as SearchEditor).toggleQueryDetails();
  76      } else if (contextService.getValue(Constants.SearchViewFocusedKey.serialize())) {
  77          const searchView = getSearchView(accessor.get(IViewsService));

src/vs/workbench/contrib/search/browser/searchActions.ts:
  28  import { IViewsService } from 'vs/workbench/common/views';
  29  import { SearchEditorInput } from 'vs/workbench/contrib/searchEditor/browser/searchEditorInput';
  30: import { SearchEditor } from 'vs/workbench/contrib/searchEditor/browser/searchEditor';
  31
  32  export function isSearchViewFocused(viewsService: IViewsService): boolean {
```

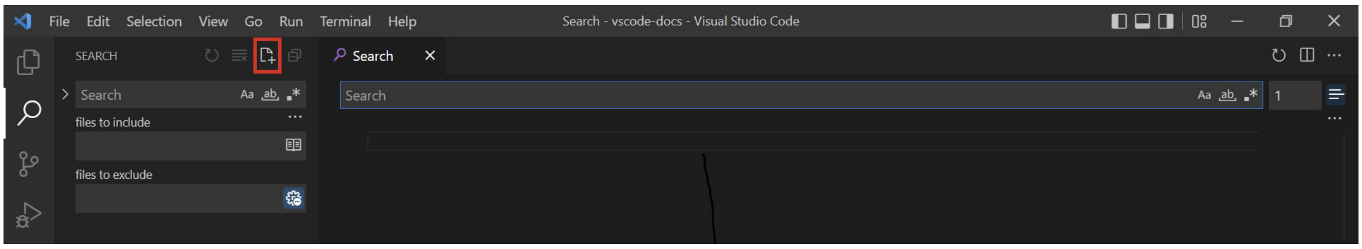


```
        const input = this.editorService.activeEditor;
        if (input instanceof SearchEditorInput) {
            // cast as we cannot import SearchEditor as a value b/c cyclic dependency.
            (this.editorService.activeControl as SearchEditor).focusNextInput();
        }
```

The **Open Search Editor** command opens an existing Search Editor if one exists, or to otherwise create a new one. The **New Search Editor** command will always create a new Search Editor.

In the Search Editor, results can be navigated to using **Go to Definition** actions, such as `F12` to open the source location in the current editor group, or `Ctrl+K F12` to open the location in an editor to the side. Additionally, double-clicking can optionally open the source location, configurable with the `search.searchEditor.doubleClickBehaviour` setting.

You can also use the **Open New Search Editor** button at the top of the Search view, and can copy your existing results from a Search view over to a Search Editor with the **Open in editor** link at the top of the results tree, or the **Search Editor: Open Results in Editor** command.


File Edit Selection View Go Run Terminal Help
Search - vscode-docs - Visual Studio Code
SEARCH
Search
files to include
files to exclude
Search
Search
1

The Search Editor above was opened by selecting the **Open New Search Editor** button (third button) on the top of the Search view.

Search Editor commands and arguments

- `search.action.openNewEditor` - Opens the Search Editor in a new tab.
- `search.action.openInEditor` - Copy the current Search results into a new Search Editor.
- `search.action.openNewEditorToSide` - Opens the Search Editor in a new window next to the window you currently have opened.

There are two arguments that you can pass to the Search Editor commands (`search.action.openNewEditor` , `search.action.openNewEditorToSide`) to allow keybindings to configure how a new Search Editor should behave:

- `triggerSearch` - Whether a search be automatically run when a Search Editor is opened. Default is true.
- `focusResults` - Whether to put focus in the results of a search or the query input. Default is true.

For example, the following keybinding runs the search when the Search Editor is opened but leaves the focus in the search query control.

```
{
  "key": "ctrl+o",
  "command": "search.action.openNewEditor",
  "args": { "query": "VS Code", "triggerSearch": true, "focusResults": false }
}
```

Search Editor context default

The `search.searchEditor.defaultNumberOfContextLines` setting has a default value of 1, meaning one context line will be shown before and after each result line in the Search Editor.

Reuse last Search Editor configuration

The `search.searchEditor.reusePriorSearchConfiguration` setting (default is `false`) lets you reuse the last active Search Editor's configuration when creating a new Search Editor.

IntelliSense

We'll always offer word completion, but for the rich languages (/docs/languages/overview), such as JavaScript, JSON, HTML, CSS, SCSS, Less, C# and TypeScript, we offer a true IntelliSense experience. If a language service knows possible completions, the IntelliSense suggestions will pop up as you type. You can always manually trigger it with `Ctrl+Space` . By default, `Tab` or `Enter` are the accept keyboard triggers but you can also customize these key bindings (/docs/getstarted/keybindings).

Tip: The suggestions filtering supports CamelCase so you can type the letters which are upper cased in a method name to limit the suggestions. For example, "cra" will quickly bring up "createApplication".

Tip: IntelliSense suggestions can be configured via the `editor.quickSuggestions` and `editor.suggestOnTriggerCharacters` settings (/docs/getstarted/settings).

JavaScript and TypeScript developers can take advantage of the npmjs (https://www.npmjs.com) type declaration (typings) file repository to get IntelliSense for common JavaScript libraries (Node.js, React, Angular). You can find a good explanation on using type declaration files in the JavaScript language (/docs/languages/javascript#_intellisense) topic and the Node.js (/docs/nodejs/nodejs-tutorial) tutorial.

Learn more in the IntelliSense document (/docs/editor/intellisense).

Formatting

VS Code has great support for source code formatting. The editor has two explicit format actions:

- **Format Document** (`Shift+Alt+F`) - Format the entire active file.
- **Format Selection** (`Ctrl+K Ctrl+F`) - Format the selected text.

You can invoke these from the **Command Palette** (`Ctrl+Shift+P`) or the editor context menu.

VS Code has default formatters for JavaScript, TypeScript, JSON, HTML, and CSS. Each language has specific formatting options (for example, `html.format.indentInnerHtml`) which you can tune to your preference in your user or workspace settings (/docs/getstarted/settings). You can also disable the default language formatter if you have another extension installed that provides formatting for the same language.

```
"html.format.enable": false
```

Along with manually invoking code formatting, you can also trigger formatting based on user gestures such as typing, saving or pasting. These are off by default but you can enable these behaviors through the following settings (/docs/getstarted/settings):

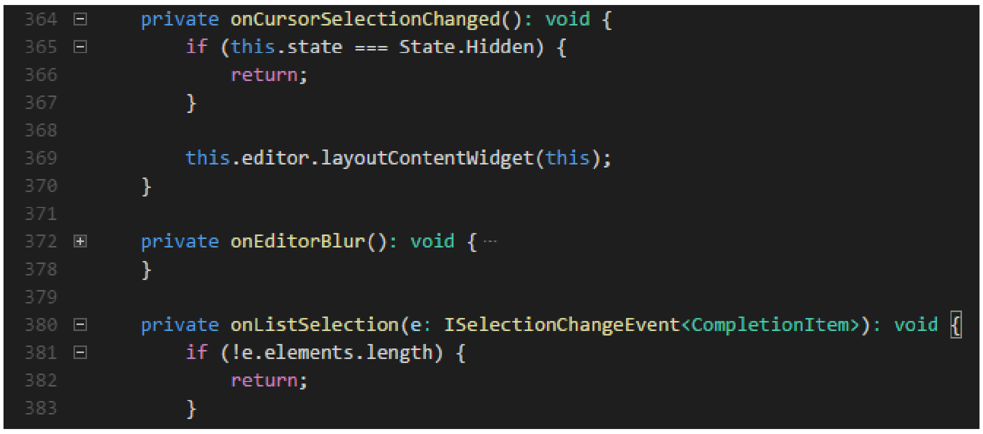
- `editor.formatOnType` - Format the line after typing.
- `editor.formatOnSave` - Format a file on save.
- `editor.formatOnPaste` - Format the pasted content.

Note: Not all formatters support format on paste as to do so they must support formatting a selection or range of text.

In addition to the default formatters, you can find extensions on the Marketplace to support other languages or formatting tools. There is a `Formatters` category so you can easily search and find formatting extensions (https://marketplace.visualstudio.com/search?target=VSCode&category=Formatters&sortBy=Installs). In the **Extensions** view search box, type 'formatters' or 'category:formatters' to see a filtered list of extensions within VS Code.

Folding

You can fold regions of source code using the folding icons on the gutter between line numbers and line start. Move the mouse over the gutter and click to fold and unfold regions. Use `Shift + Click` on the folding icon to fold or unfold the region and all regions inside.



```
    private onCursorSelectionChanged(): void {
        if (this.state === State.Hidden) {
            return;
        }

        this.editor.layoutContentWidget(this);
    }

    private onEditorBlur(): void {⋯
    }

    private onListSelection(e: ISelectionChangeEvent<CompletionItem>): void {
        if (!e.elements.length) {
            return;
        }
```

You can also use the following actions:

- Fold (`Ctrl+Shift+[`) folds the innermost uncollapsed region at the cursor.
- Unfold (`Ctrl+Shift+]`) unfolds the collapsed region at the cursor.

- Toggle Fold (`Ctrl+K Ctrl+L`) folds or unfolds the region at the cursor.
- Fold Recursively (`Ctrl+K Ctrl+[`) folds the innermost uncollapsed region at the cursor and all regions inside that region.
- Unfold Recursively (`Ctrl+K Ctrl+]`) unfolds the region at the cursor and all regions inside that region.
- Fold All (`Ctrl+K Ctrl+0`) folds all regions in the editor.
- Unfold All (`Ctrl+K Ctrl+J`) unfolds all regions in the editor.
- Fold Level X (`Ctrl+K Ctrl+2` for level 2) folds all regions of level X, except the region at the current cursor position.
- Fold All Block Comments (`Ctrl+K Ctrl+/`) folds all regions that start with a block comment token.

Folding regions are by default evaluated based on the indentation of lines. A folding region starts when a line has a smaller indent than one or more following lines, and ends when there is a line with the same or smaller indent.

Folding regions can also be computed based on syntax tokens of the editor's configured language. The following languages already provide syntax aware folding: Markdown, HTML, CSS, LESS, SCSS, and JSON.

If you prefer to switch back to indentation-based folding for one (or all) of the languages above, use:

```
  "[html]": {
    "editor.foldingStrategy": "indentation"
  },
```

Regions can also be defined by markers defined by each language. The following languages currently have markers defined:

Language	Start region	End region
Bat	`::#region` or `REM #region`	`::#endregion` or `REM #endregion`
C#	`#region`	`#endregion`
C/C++	`#pragma region`	`#pragma endregion`
CSS/Less/SCSS	`/*#region*/`	`/*#endregion*/`
Coffeescript	`#region`	`#endregion`
F#	`//#region` or `(#_region)`	`//#endregion` or `(#_endregion)`
Java	`//#region` or `//<editor-fold>`	`// #endregion` or `//</editor-fold>`
Markdown	`<!-- #region -->`	`<!-- #endregion -->`
Perl5	`#region` or `=pod`	`#endregion` or `=cut`
PHP	`#region`	`#endregion`
PowerShell	`#region`	`#endregion`
Python	`#region` or `# region`	`#endregion` or `# endregion`
TypeScript/JavaScript	`//#region`	`//#endregion`
Visual Basic	`#Region`	`#End Region`

To fold and unfold only the regions defined by markers use:

- Fold Marker Regions (`Ctrl+K Ctrl+8`) folds all marker regions.
- Unfold Marker Regions (`Ctrl+K Ctrl+9`) unfolds all marker regions.

Fold selection

The command **Create Manual Folding Ranges from Selection** (`Ctrl+K Ctrl+,`) creates a folding range from the currently selected lines and collapses it. That range is called a **manual** folding range that goes on top of the ranges computed by folding providers.

Manual folding ranges can be removed with the command **Remove Manual Folding Ranges** (`Ctrl+K Ctrl+.`).

Manual folding ranges are especially useful for cases when there isn't programming language support for folding.

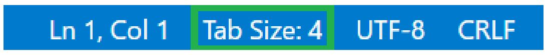
Indentation

VS Code lets you control text indentation and whether you'd like to use spaces or tab stops. By default, VS Code inserts spaces and uses 4 spaces per `Tab` key. If you'd like to use another default, you can modify the `editor.insertSpaces` and `editor.tabSize` settings (/docs/getstarted/settings).

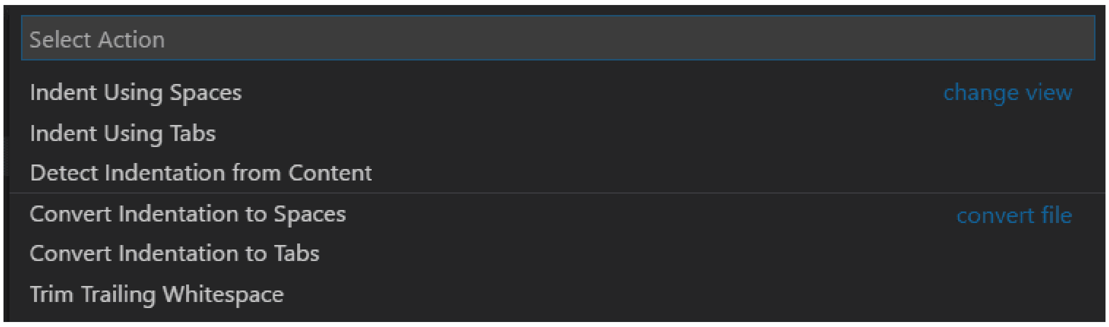
```
    "editor.insertSpaces": true,
    "editor.tabSize": 4,
```

Auto-detection

VS Code analyzes your open file and determines the indentation used in the document. The auto-detected indentation overrides your default indentation settings. The detected setting is displayed on the right side of the Status Bar:


Ln 1, Col 1   Tab Size: 4   UTF-8   CRLF

You can click on the Status Bar indentation display to bring up a dropdown with indentation commands allowing you to change the default settings for the open file or convert between tab stops and spaces.

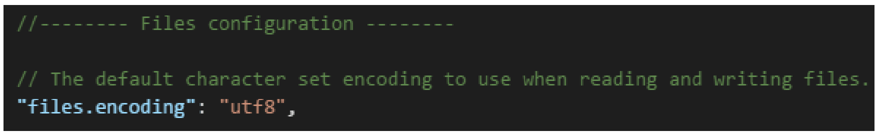


Note: VS Code auto-detection checks for indentations of 2, 4, 6 or 8 spaces. If your file uses a different number of spaces, the indentation may not be correctly detected. For example, if your convention is to indent with 3 spaces, you may want to turn off `editor.detectIndentation` and explicitly set the tab size to 3.

```
    "editor.detectIndentation": false,
    "editor.tabSize": 3,
```

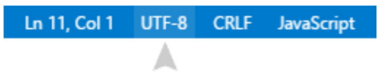
File encoding support

Set the file encoding globally or per workspace by using the `files.encoding` setting in **User Settings** or **Workspace Settings**.

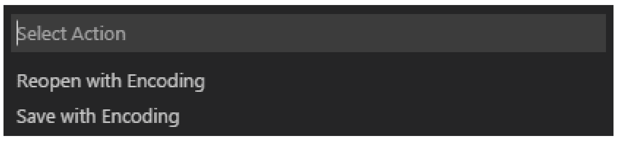


```
//-------- Files configuration --------

// The default character set encoding to use when reading and writing files.
"files.encoding": "utf8",
```

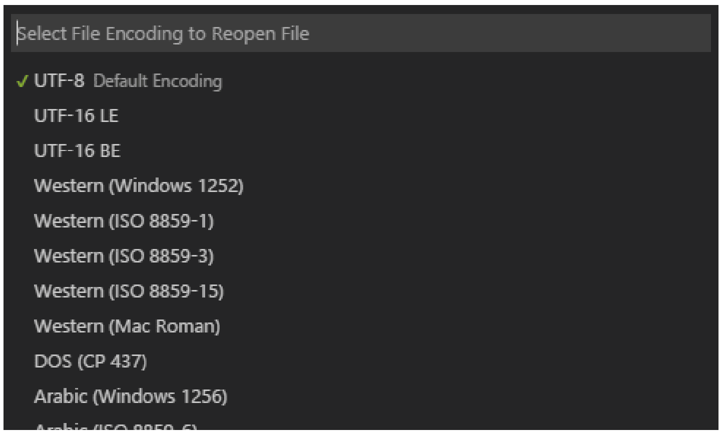
You can view the file encoding in the status bar.


Ln 11, Col 1   UTF-8   CRLF   JavaScript

Click on the encoding button in the status bar to reopen or save the active file with a different encoding.



Then choose an encoding.



Next steps

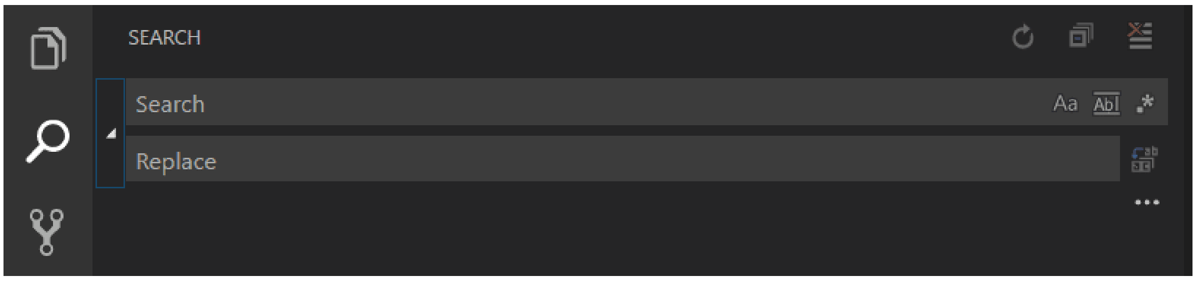
You've covered the basic user interface - there is a lot more to VS Code. Read on to find out about:

- Intro Video - Setup and Basics (/docs/introvideos/basics) - Watch a tutorial on the basics of VS Code.
- User/Workspace Settings (/docs/getstarted/settings) - Learn how to configure VS Code to your preferences through user and workspace settings.
- Code Navigation (/docs/editor/editingevolved) - Peek and Goto Definition, and more.
- Integrated Terminal (/docs/terminal/basics) - Learn about the integrated terminal for quickly performing command-line tasks from within VS Code.
- IntelliSense (/docs/editor/intellisense) - VS Code brings smart code completions.
- Debugging (/docs/editor/debugging) - This is where VS Code really shines.

Common questions

Is it possible to globally search and replace?

Yes, expand the Search view text box to include a replace text field. You can search and replace across all the files in your workspace. Note that if you did not open VS Code on a folder, the search will only run on the currently open files.



How do I turn on word wrap?

You can control word wrap through the `editor.wordWrap` setting (/docs/getstarted/settings). By default, `editor.wordWrap` is `off` but if you set to it to `on`, text will wrap on the editor's viewport width.

```
    "editor.wordWrap": "on"
```

You can toggle word wrap for the VS Code session with `Alt+Z`.

You can also add vertical column rulers to the editor with the `editor.rulers` setting, which takes an array of column character positions where you'd like vertical rulers.

How can I avoid placing extra cursors in word wrapped lines?

If you'd like to ignore line wraps when adding cursors above or below your current selection, you can pass in `{ "logicalLine": true }` to `args` on the keybinding like this:

```
{
  "key": "shift+alt+down",
  "command": "editor.action.insertCursorBelow",
  "when": "textInputFocus",
  "args": { "logicalLine": true },
},
{
  "key": "shift+alt+up",
  "command": "editor.action.insertCursorAbove",
  "when": "textInputFocus",
  "args": { "logicalLine": true },
},
```