

Microsoft Visual Studio Code

A small tutorial.

Made by:

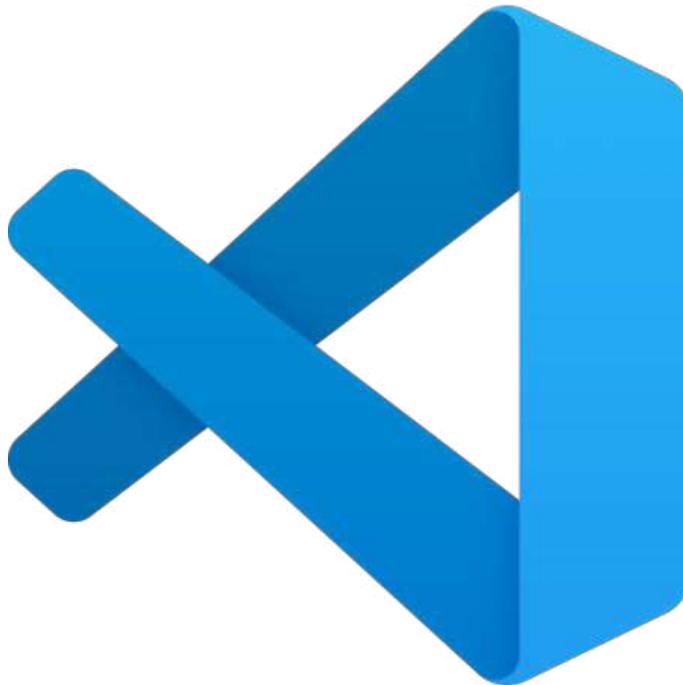
- Alien Embarec Riadi {alu0101035406@ull.edu.es}
- Felipe Andrés Álvarez Avaria {alu0100969535@ull.edu.es}

Index

- 1.) [What is VS Code?](#)
- 2.) [Installation of VSC in Linux and Windows OS](#)
- 3.) [Basic Use of VSC](#)
- 4.) [Extensions](#)
- 5.) [JSDoc with VSC](#)
- 6.) [JS Linting in VSC](#)
- 7.) [JS Debugging in VSC](#)
- 8.) [Remote editing in IaaS VM](#)
- 9.) [Source Control](#)
- 10.) [Keyboard Shortcuts in VSC](#)
- 11.) [Bibliography](#)

Note: These slides and an extended explanation in Markdown format are available at
https://github.com/Alien-97/vsc_presentation

1. What is Visual Studio Code?



Visual Studio Code is a source-code IDE developed by Microsoft for Windows, Linux and macOS.

It includes support for debugging, embedded Git control, syntax highlighting, and other important features.

1.a ¿What is an IDE?

An Integrated Development Environment(IDE), is a software application that provides comprehensive facilities to computer programmers for software development.

1.b VSC History

On November 18, 2015, Visual Studio Code was released under the MIT License and its [source code](#) posted to GitHub.

Visual Studio Code <https://code.visualstudio.com>

editor electron visual-studio-code typescript microsoft

⌚ 60,650 commits

🍴 384 branches

📦 0 packages

🏷️ 145 releases

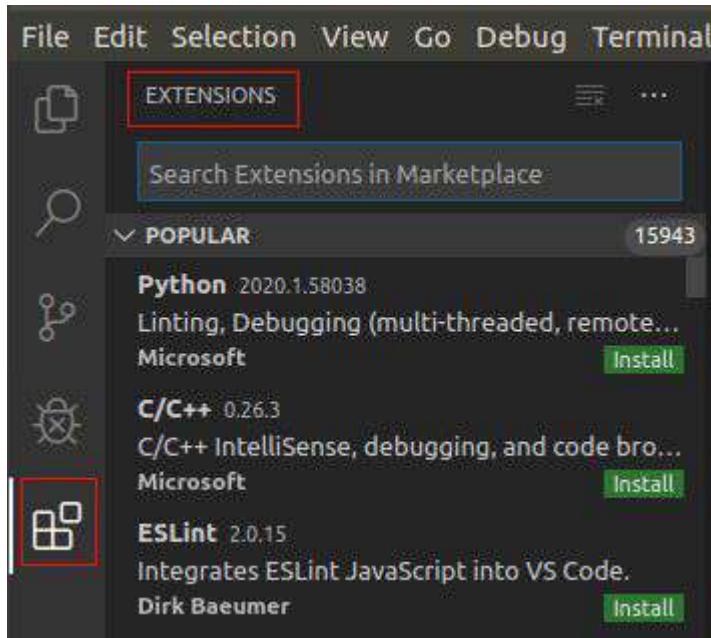
● TypeScript 93.6%

● JavaScript 3.7%

● CSS 1.7%

● Inno Setup 0.6%

1.c VSC Language Support



Visual Studio Code is a source code editor that can be used with a variety of programming languages.

VSC provides default support for JS,CSS,HTML and TypeScript. Other languages support can be added via extensions.

1.d VS Versions

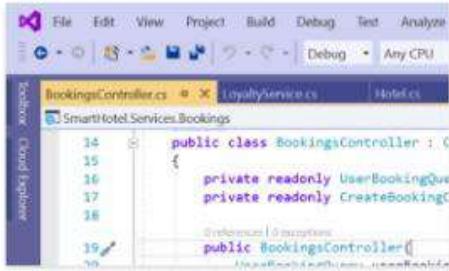
There are several versions of microsoft visual studio availables. We're going to work with **Visual Studio Code**, a code writing and debugging oriented version.

Although later we'll see that you can add features to the IDE with the extensions.

1.d VS Versions



Visual Studio



Full-featured IDE to code, debug, test, and deploy to any platform

[Download Visual Studio](#)

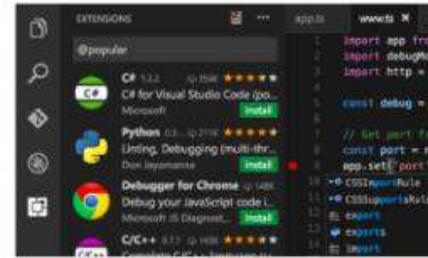
Community 2019

Professional 2019

Enterprise 2019



Visual Studio Code



Editing and debugging on any OS

(By using Visual Studio Code you agree to its [license](#) and [privacy statement](#))

[Download Visual Studio Code](#)

Windows x64User Installer

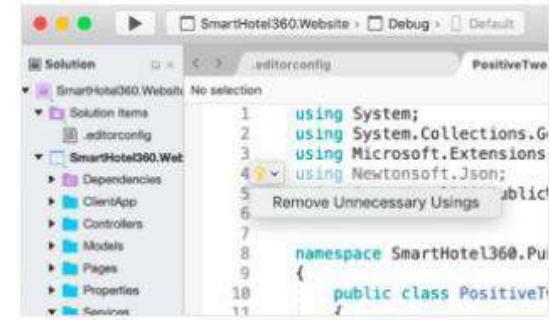
macOSPackage

Linux x64.deb

Linux x64.rpm



Visual Studio for Mac



Develop apps and games for iOS, Android, and web using .NET

[Read more about activating your license](#)

[Download Visual Studio for Mac](#)

[Learn more >](#)

2. Installation of VSC in Linux and Windows OS

		Stable	Insiders
macOS	Package	↓	↓
Windows x64	User Installer	↓	↓
Linux x64	.deb .rpm	↓ ↓	↓ ↓
Other downloads			

Visual Studio Code is available for three main Operating Systems: Linux, Windows and macOS.

Two versions: stable releases and for Beta-testers(Insiders).

2.a How to Install in Linux

- Make sure you are logged in Linux as a user with sudo privileges.
- First, we go to the official Visual Studio Code [page](#) and install the executable file.
- Depending on your Linux Operating System distribution, you will have to install the package for Debian or for RPM.
- See this VSC official guide for [installation in Linux](#).

2.b How to Install VSC in Windows.



Download the executable from the VSC page. Then you have to start the installation wizard for Windows.

3. Basic use of VSC

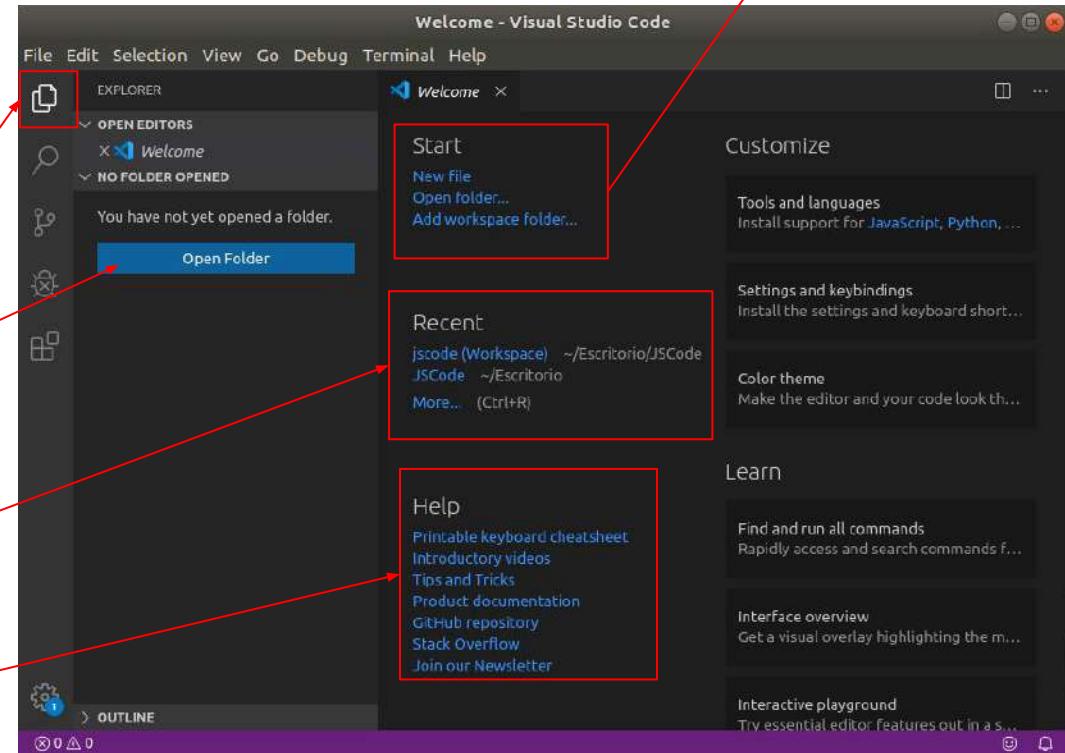
3.a Main Interface

- Welcome Window when code-editor is opened for first time.

File Explorer:
navigate across your
projects and its files

Quick
Access to
Yet
Created
Projects

Help and
Doc area



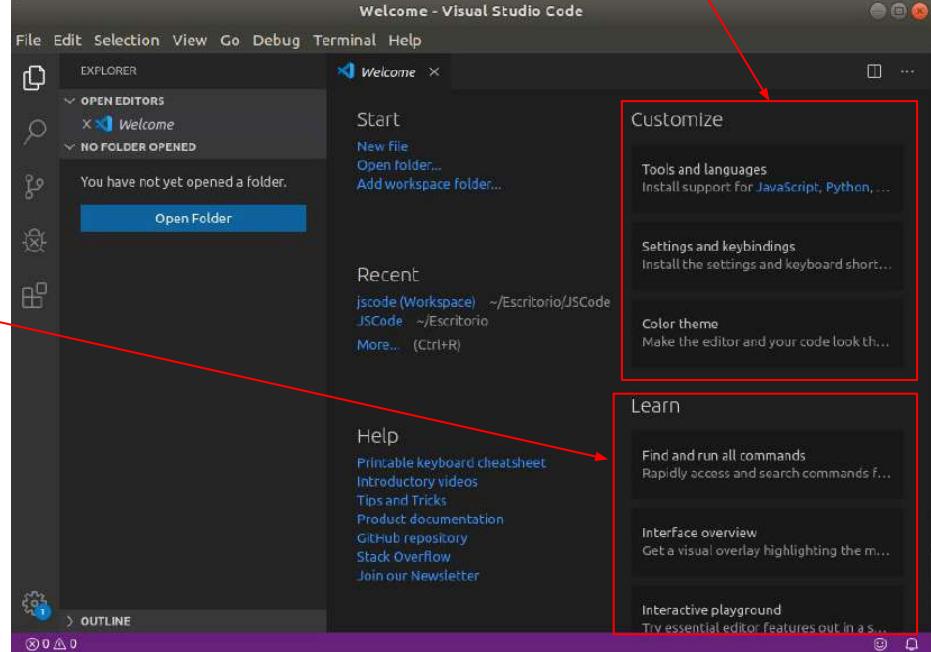
Create a
project

3. Basic use of VSC

3.a Main Interface

Command
Palette: a tools
search
engine, you can
run it with Ctrl +
Shift + P or
clicking here

Links to Extensions
support and Color Theme
Setup

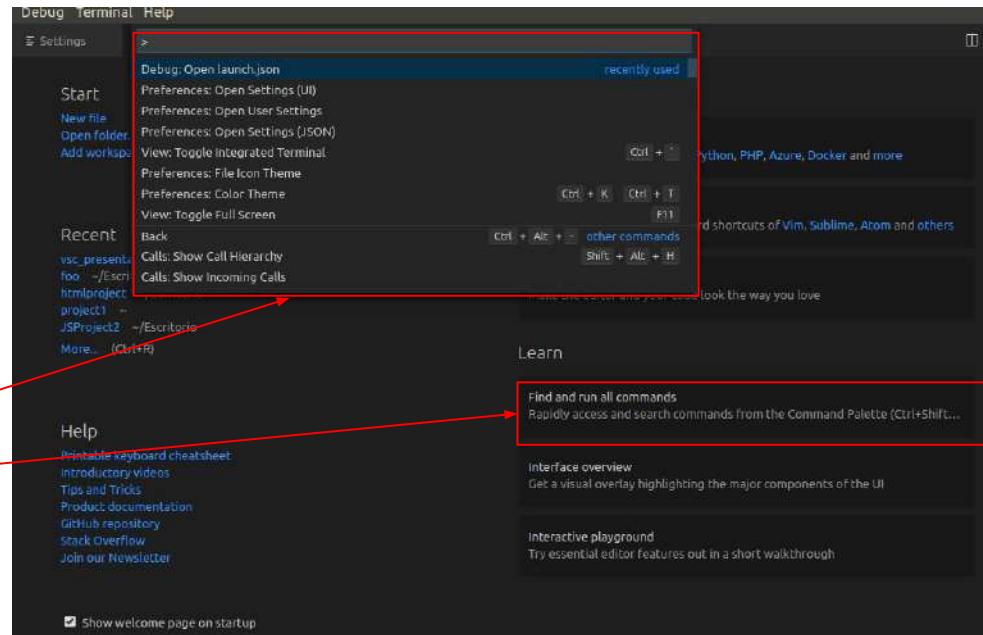


3. Basic use of VSC

3.a Main Interface

Command Palette

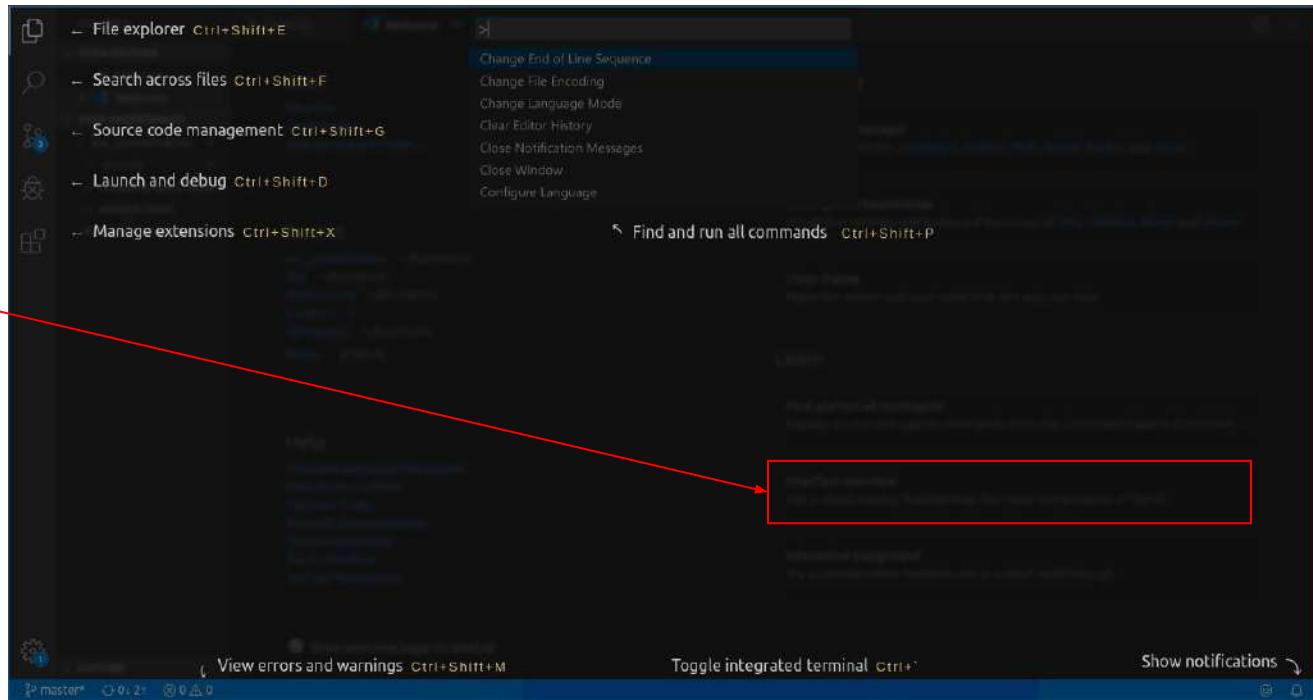
Ctrl +Shift + P



3. Basic use of VSC

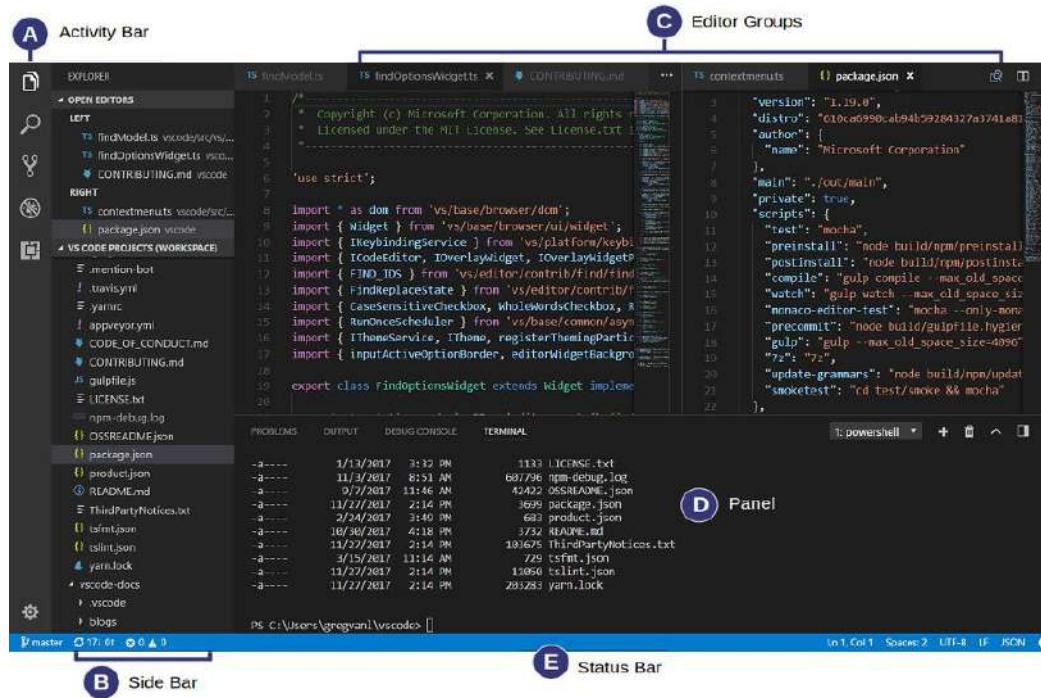
3.a Main Interface

Clicking on
Interface
Overview on
Welcome
Window



3. Basic use of VSC

3.a Main Interface

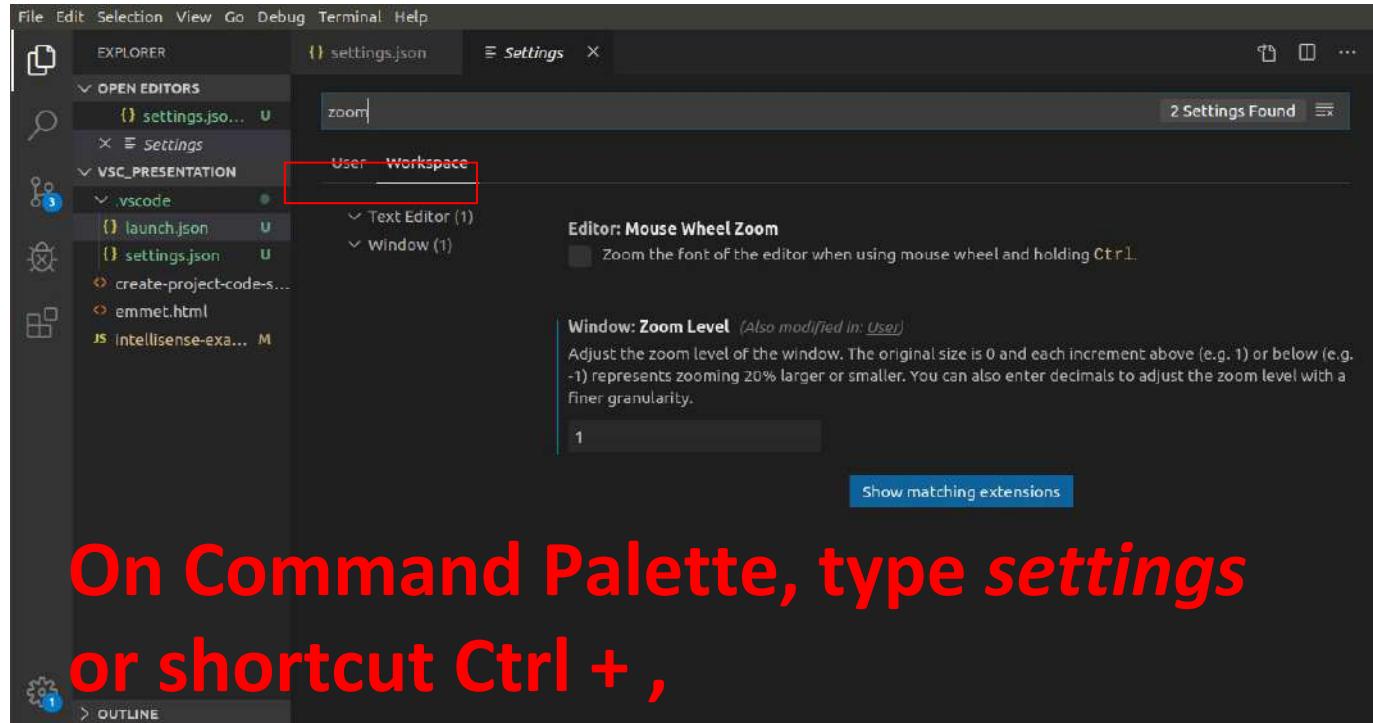


3. Basic use of VSC

3.a Main Interface

User settings:
apply to each vsc instance opened.

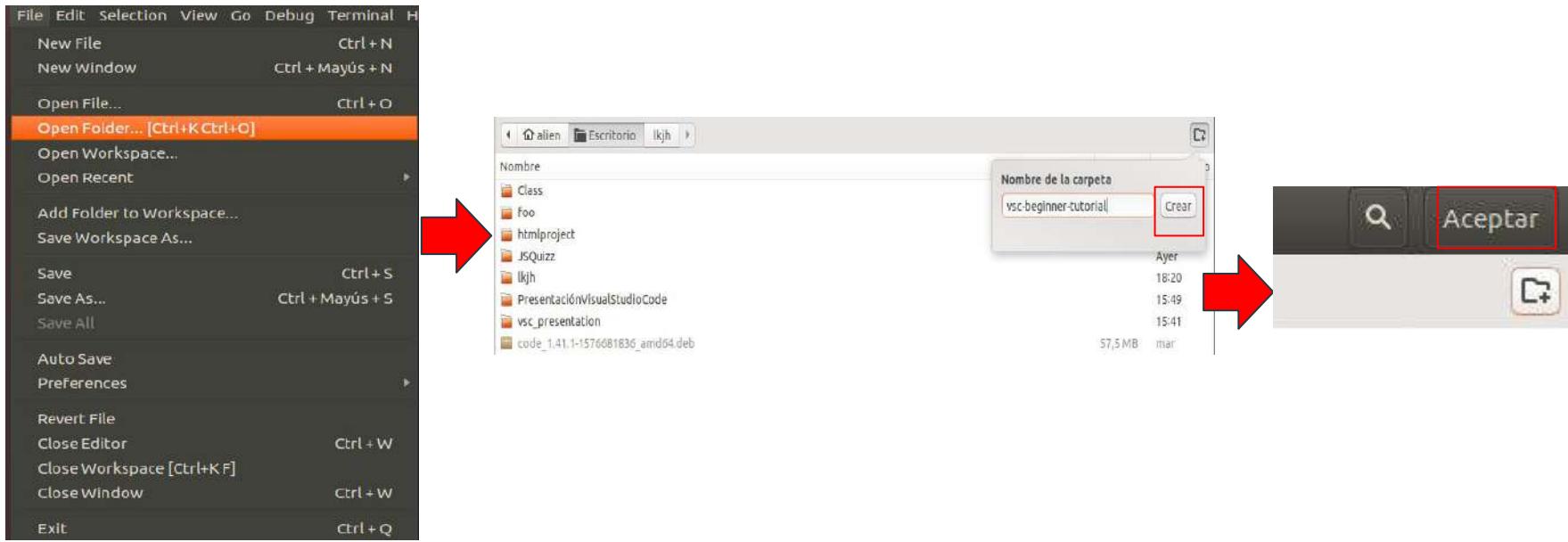
Workspace settings: apply only to one or more projects



3. Basic use of VSC

3.b How to create a project

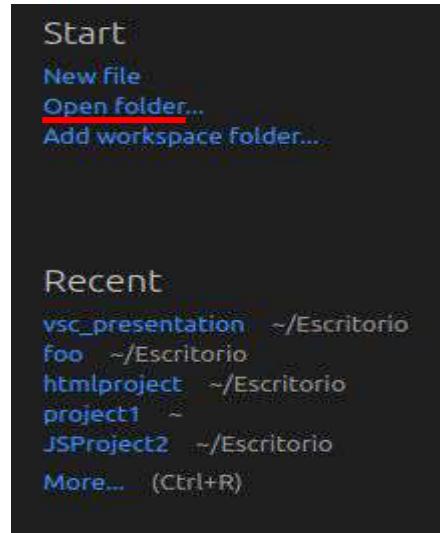
For create a project go to **File > Open Folder > Open existing folder o create a new one .**



3. Basic use of VSC

3.b How to create a project

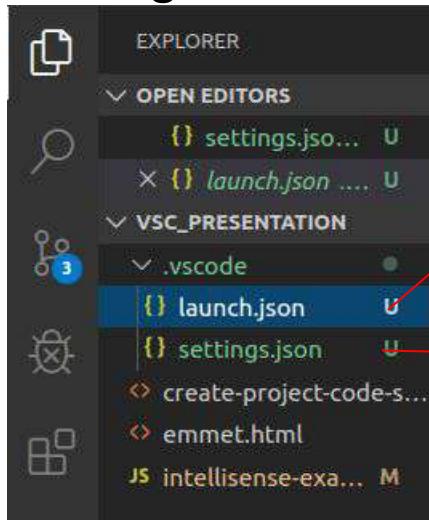
These step can also be done on Welcome Page on **Start > Open folder.**



3. Basic use of VSC

3.b How to create a project

Each project has a workspace, where the specific settings of project are configured.



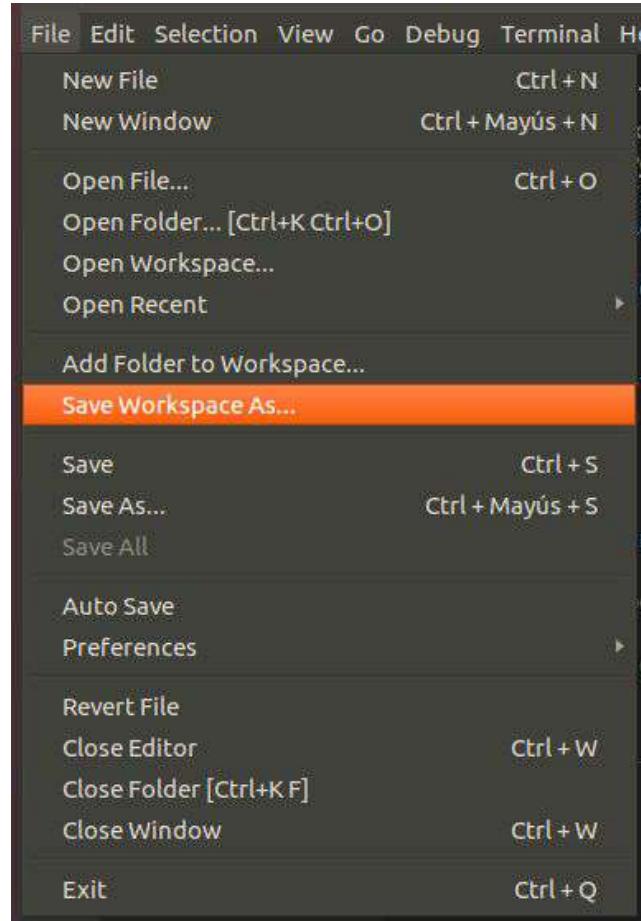
debugging configurations

**settings that should be
applied when the project is
open**

3. Basic use of VSC

3.b How to create a project

For save workspace, save your changes with **Ctrl+S**, then go to **File > Save Workspace As** and choose the location of these file. As it is associated to the project, it should be saved in the project folder.

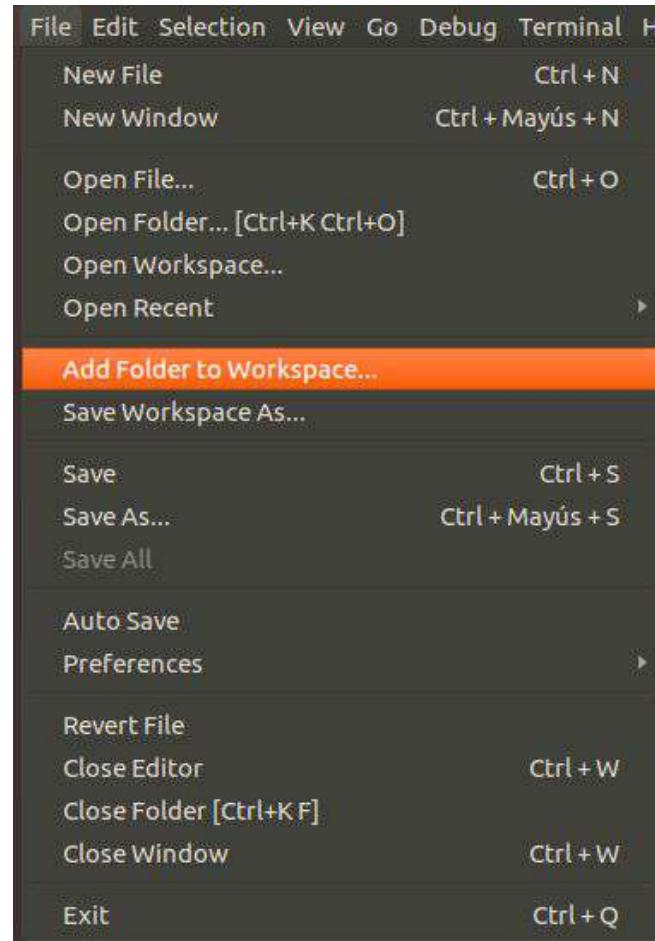


3. Basic use of VSC

3.b How to create a project

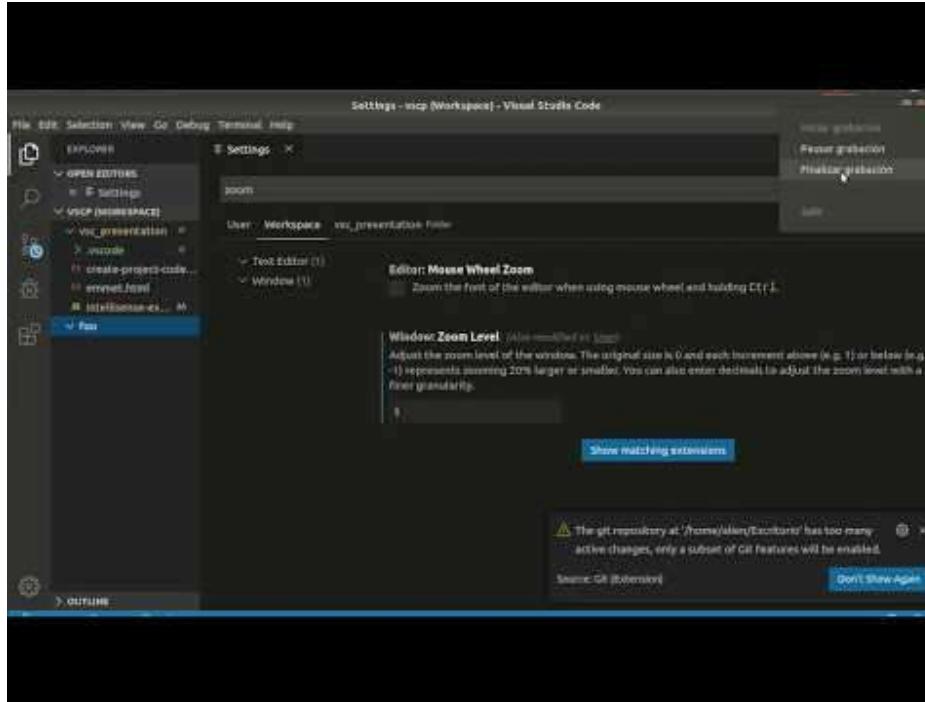
VSC has a feature called **multi-root workspaces**, you can join two or more projects under a single workspace settings.

You have to save workspace to a common path to the projects you want to join.



3. Basic use of VSC

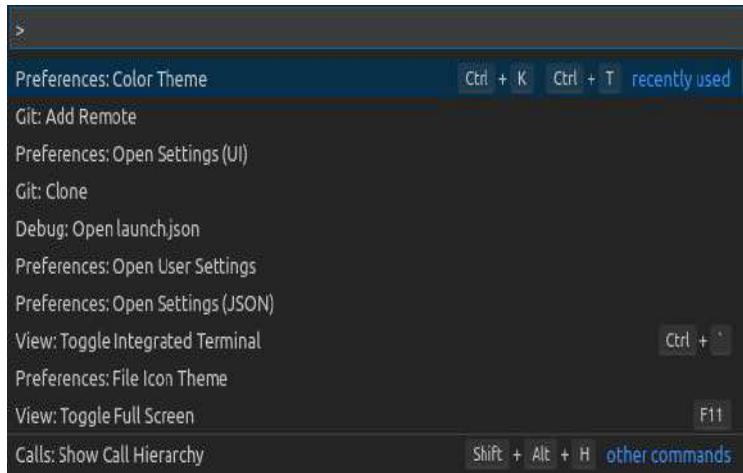
3.b How to create a project. Multi-root workspaces demo



3. Basic use of VSC

3.c Color Theme

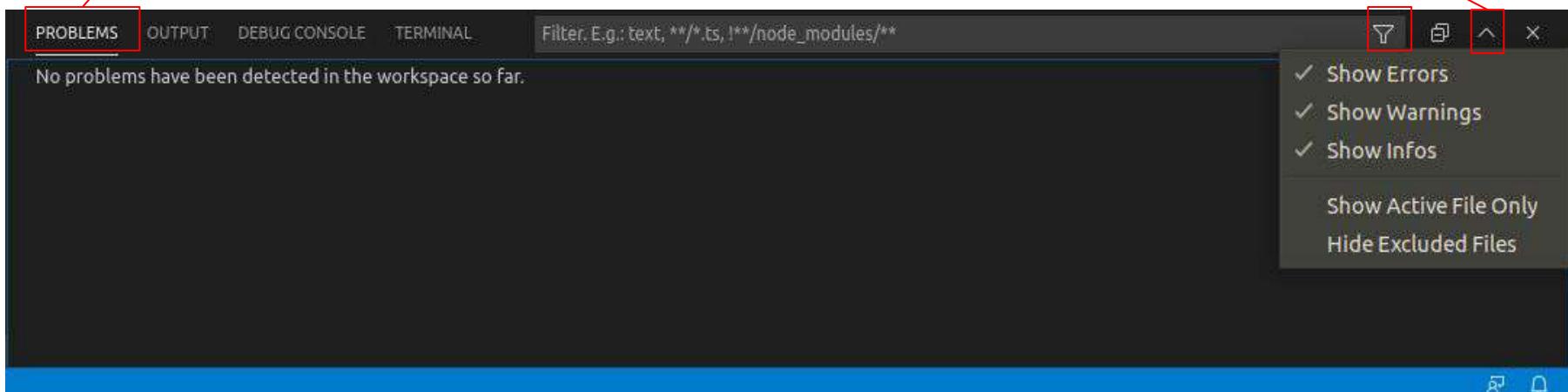
Go to **Command Palette > Color Theme**, also **Ctrl + K Ctrl + T**



3. Basic use of VSC

3.d Panel

Where Run errors are shown



Maximize Panel Size

3. Basic use of VSC

3.d Panel

Additional commands: scroll Up/Down
(Shift + Fn + Up/Down)

New terminal
in new window
(Ctrl + Shift + `)

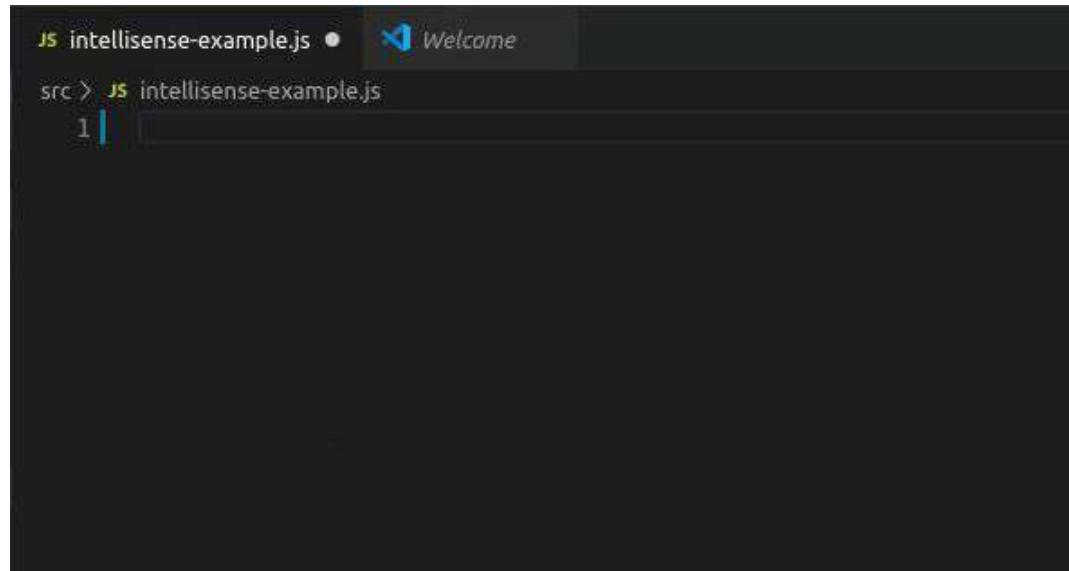
Split terminals
in same
window (Ctrl +
Shift + 5)



3. Basic use of VSC

3.e IntelliSense and Emmet

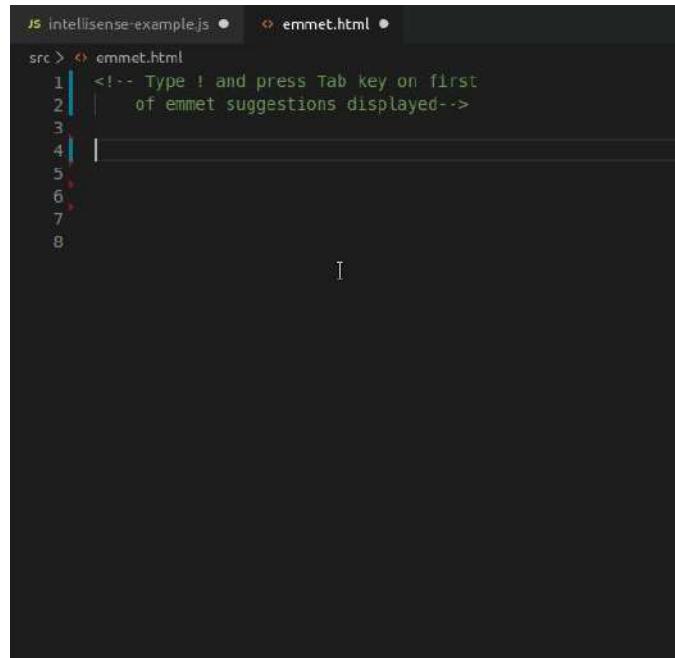
IntelliSense. Type
Inference, JSDoc:
variables and
functions names
and more.



3. Basic use of VSC

3.e IntelliSense and Emmet

Emmet: Snippet
expansion (1).



A screenshot of the Visual Studio Code interface. The title bar shows 'JS intellisense-example.js' and 'emmet.html'. The left sidebar shows 'src > emmet.html'. The main editor area displays the following code:

```
1<!-- Type ! and press Tab key on first
2| of emmet suggestions displayed-->
3|
4|
5|
6|
7|
8|
```

The code consists of a single line starting with '<!--' followed by instructions for Emmet snippet expansion. Lines 1 and 2 contain the text 'Type ! and press Tab key on first' and 'of emmet suggestions displayed-->' respectively. Lines 3 through 8 are empty.

3. Basic use of VSC

3.e IntelliSense and Emmet

Emmet :
Abbreviations
(2).

```
16
17 | <!-- Also you can specify multiple nested html
18 |   tags and emmet put it pressing Tab -->
19 |
20 |
21 |
22 |
```

3. Basic use of VSC

3.f Multi-selector

For example:
rename
multiple
variables.

Select **first occurrence** +
ctrl + d

```
src > JS intellisense-example.js > Person > constructor
1  function Person(firstName, lastName, idCard) {
2    this.FirstName = firstName || "unknown";
3    this.LastName = lastName || "unknown";
4
5    this.IdCard = idCard;
6  }
7
8  Person.prototype.getFullName = function () {
9    return this.FirstName + " " + this.LastName + " " + this.idCard;
10 }
11
12 var psn = new Person("Juan", "Perez", 1111);
13
14 console.log(psn.getFullName());
15
16 // Trying git
```

3. Basic use of VSC

3.f Multi-selector

Other ways:
multiple
selectors, select
first occurrence
and then **left**
click + alt for add
more
occurrences.

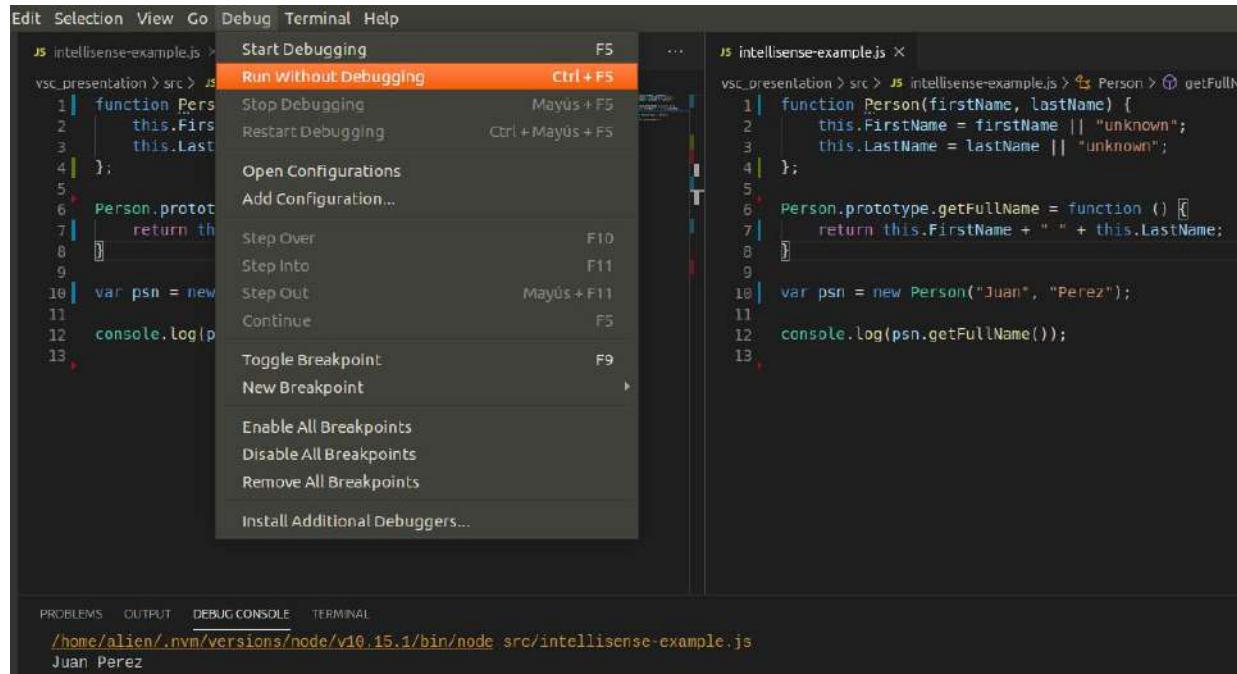
```
src > JS intellisense-example.js > Person > constructor
1  function Person(firstName, lastName, idCard) {
2    this.FirstName = firstName || "unknown";
3    this.LastName = lastName || "unknown";
4
5    this.IdCard = idCard;
6  }
7
8  Person.prototype.getFullName = function () {
9    return this.FirstName + " " + this.LastName + " " + this.idCard;
10 }
11
12 var psn = new Person("Juan", "Perez", 1111);
13
14 console.log(psn.getFullName());
15
16 // Trying git
```

3. Basic use of VSC

3.g How to run code on VS Code

There are many options:

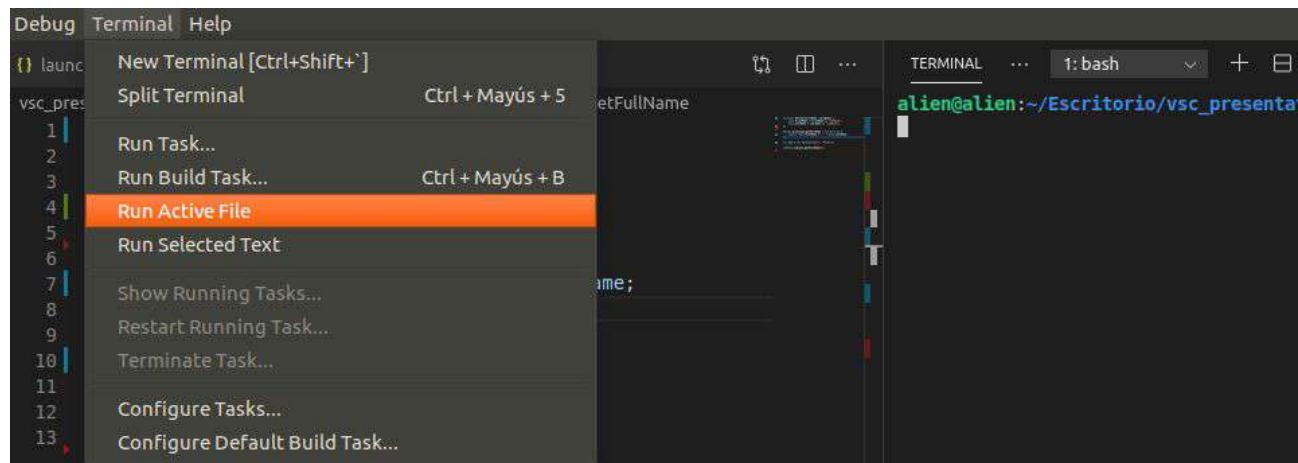
In Debug/Run without Debugging



3. Basic use of VSC

3.g How to run code on VS Code

In Terminal
> Run active
file



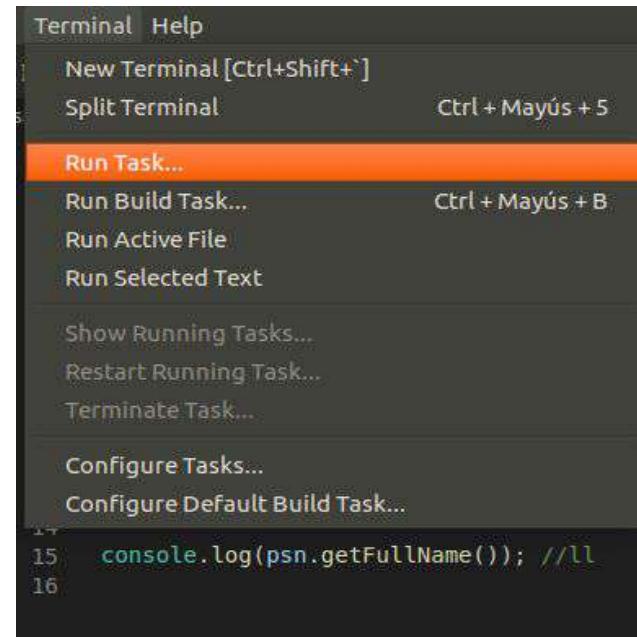
3. Basic use of VSC

3.g How to run code on VS Code

As we will be using Node.js, another way of run code is define a Task going to **package.json**

In **scripts** hash, write a new task providing a key name and what should be done when the task is applied. For example:

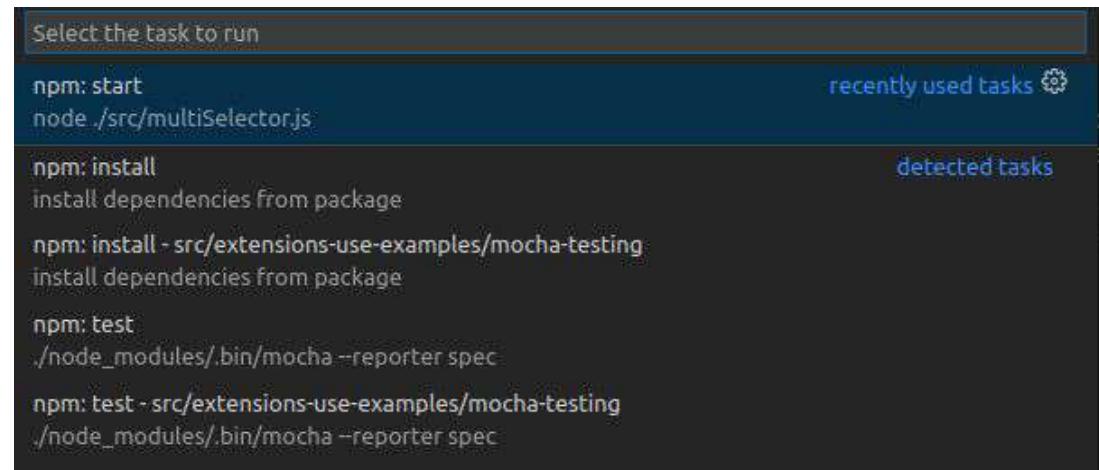
```
"start": "node  
./src/multiSelector.js"
```



3. Basic use of VSC

3.g How to run code on VS Code

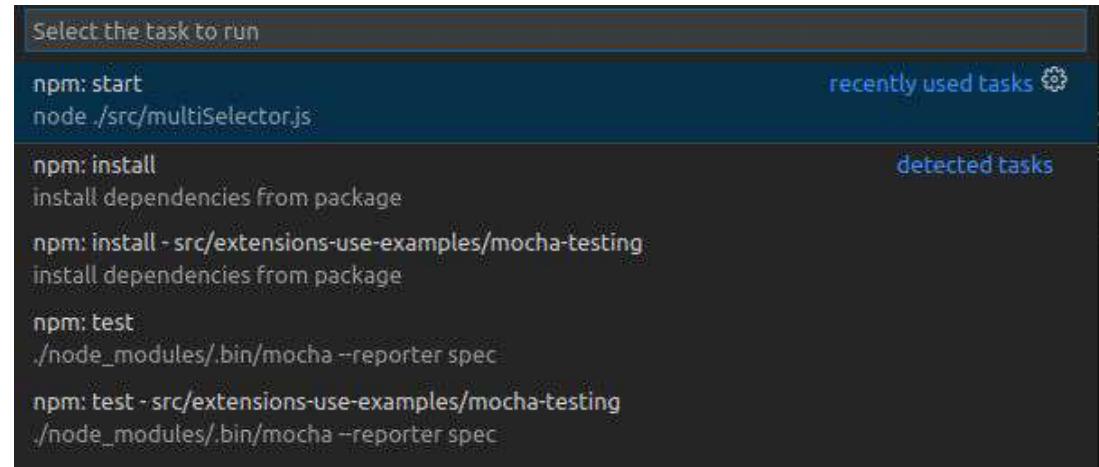
Now, go to the file you want to execute and in status bar, click on **Terminal > Run Task**



3. Basic use of VSC

3.g How to run code on VS Code

Now, go to the file you want to execute and in status bar, click on **Terminal > Run Task**. Choose task by the name you gave it in **package.json**

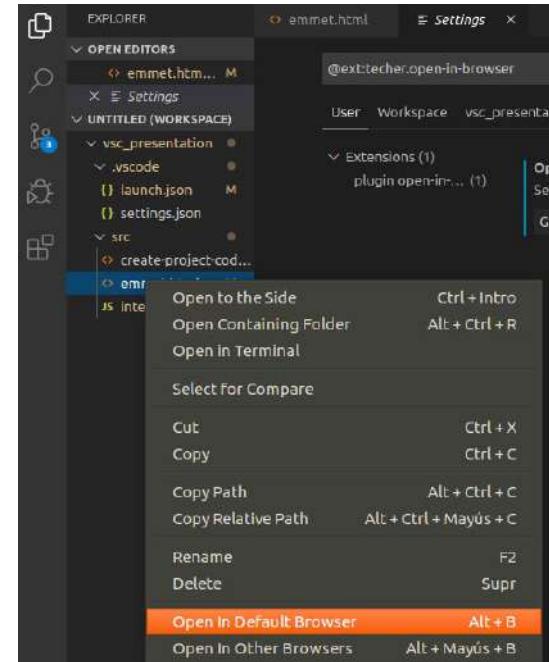


3. Basic use of VSC

3.g How to run code on VS Code

For HTML files, you can use

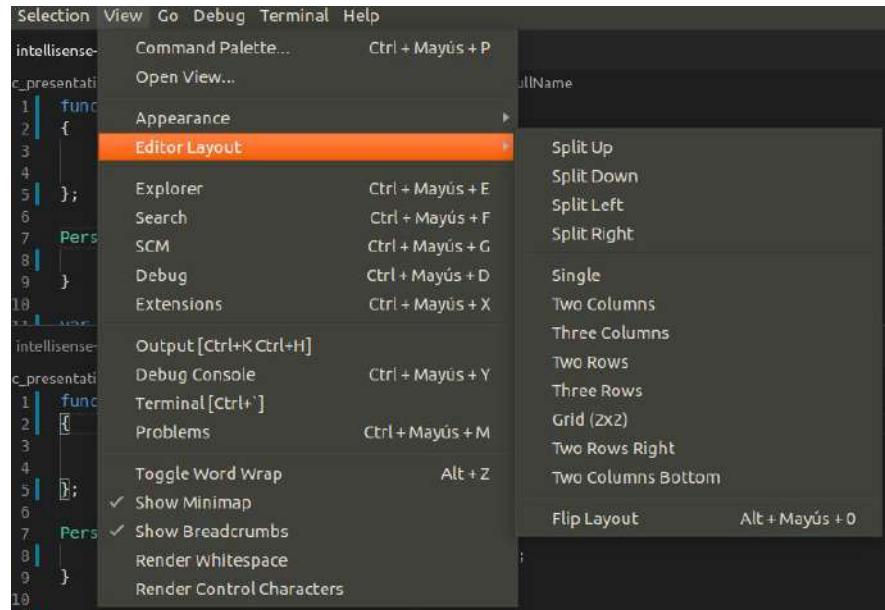
Open in browser extension



3. Basic use of VSC

3.h Editor Layout

Tools bar,
View/Editor Layout



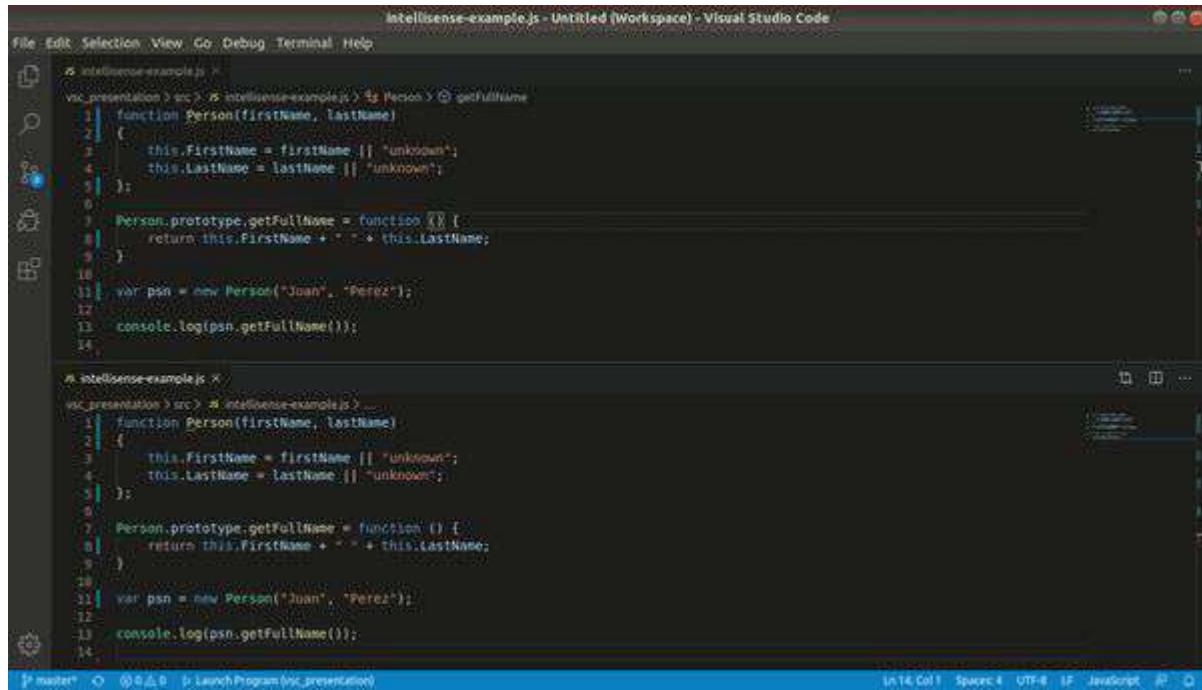
3. Basic use of VSC

3.h Editor Layout

Toggle editor layout,

horizontal/vertical

Shift + Alt + 0



The screenshot shows the Visual Studio Code interface with two code editors open side-by-side. Both editors contain the same JavaScript code:`intellisense-example.js - Untitled (Workspace) - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

1 // intellisense-example.js
2
3 // intellisense-example.js > Person > getFullName
4 function Person(firstName, lastName)
5 {
6 this.firstName = firstName || "unknown";
7 this.lastName = lastName || "unknown";
8 }
9
10 Person.prototype.getFullName = function () {
11 return this.firstName + " " + this.lastName;
12 }
13
14 var psn = new Person("Joan", "Perez");
15
16 console.log(psn.getFullName());
17

intellisense-example.js
1 // intellisense-example.js
2
3 // intellisense-example.js > Person > getFullName
4 function Person(firstName, lastName)
5 {
6 this.firstName = firstName || "unknown";
7 this.lastName = lastName || "unknown";
8 }
9
10 Person.prototype.getFullName = function () {
11 return this.firstName + " " + this.lastName;
12 }
13
14 var psn = new Person("Joan", "Perez");
15
16 console.log(psn.getFullName());
17`

The status bar at the bottom indicates "In 14 Col 1 Spaces: 4 UTF-8 LF Javascript IP O".

3. Basic use of VSC

3.i Code formatting

Format Document,

Ctrl + Shift + I

Format selection,

Ctrl + K Ctrl + F

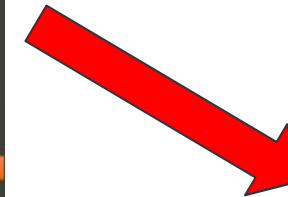
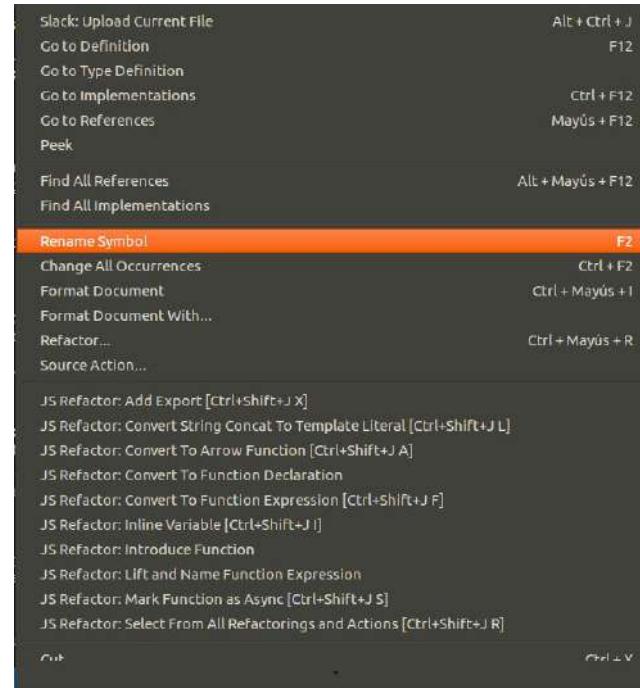
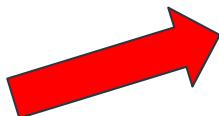
```
4  <!DOCTYPE html>
5  <html lang="en">
6
7  <head>
8      <meta charset="UTF-8">
9      <meta name="viewport" content="width=device-width, initial-scale=1.0">
10     <meta http-equiv="X-UA-Compatible" content="ie=edge">
11     <title>Document</title>
12 </head>
13
14 <body>
15
16 </body>
17
18 </html>
19
20 <!-- Also you can specify multiple nested html
21     tags and emmet put it pressing Tab -->
22
23 <div>
24     <ul>
25         <li>Hello World</li>
26     </ul>
27 </div>
```

3. Basic use of VSC

3.j Code refactoring. Rename Symbol

Rename selected variable and all similar occurrences.

```
9  
10 function function_1() {  
11     //pass;  
12 }
```



```
.js code_refactoring.js ✘  
vscode_presentation > src > python > .js code_refactoring.js > ⚡ function_1  
1 name = "Bob";  
2  
3 age = 20;  
4 console.log(`My name is ${name}. I am ${age} years old.`);  
5  
6 ↴ for (i in name) {  
7     console.log(i);  
8 }  
9  
10 ↴ function function_1() {  
11     //pass: function_1  
12 }  
13             Enter to Rename, Shift+Enter to Preview  
14 ↴ function function.add() {  
15     let extractVariable =;  
16  
17     return extractVariable;  
18 }  
19  
20 function_1();  
21 function add();  
22
```

3. Basic use of VSC

3.j Code refactoring. Extract Variable

Assign temporary variable to Return statement expression.

The screenshot shows a code editor with the following code:

```
14 function function add() {  
15   return age + 10;  
16 }
```

A context menu is open over the line "return age + 10;". The menu includes the following items:

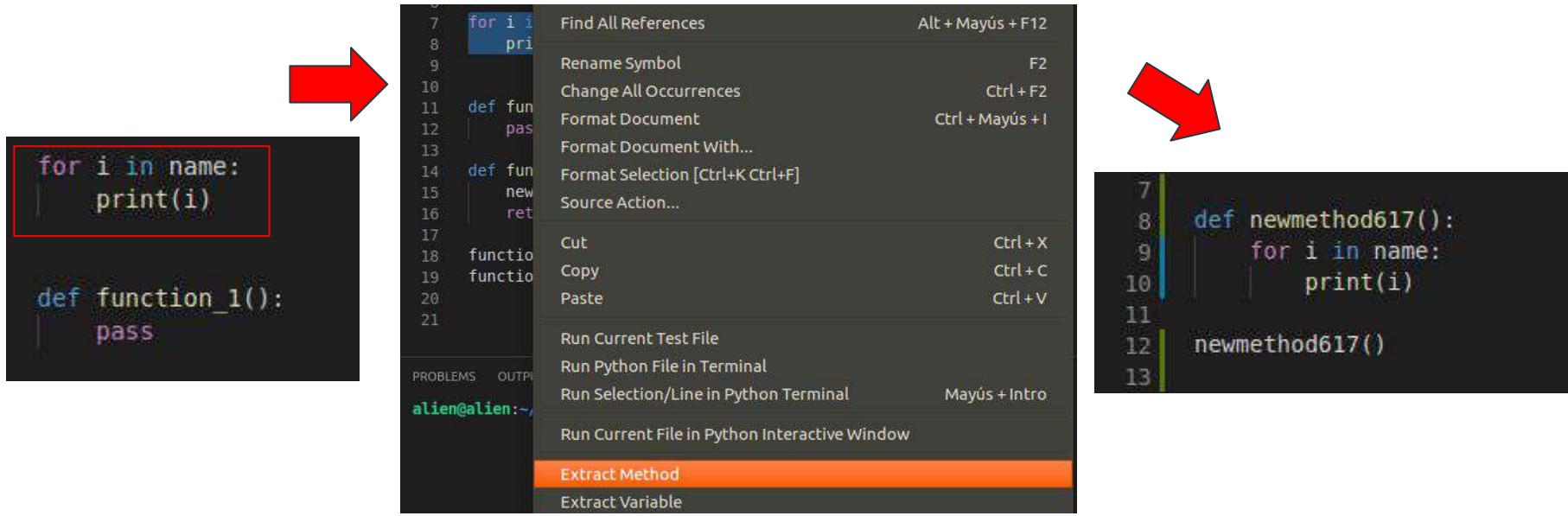
- Find All References
- Find All Implementations
- Rename Symbol
- Change All Occurrences
- Format Document
- Format Document With...
- Format Selection [Ctrl+K Ctrl+F]
- Format Selection With...
- Refactor...
- Source Action...
- JS Refactor: Add Export [Ctrl+Shift+J X]
- JS Refactor: Convert String Concat To Template Literal [Ctrl+Shift+J L]
- JS Refactor: Convert To Arrow Function [Ctrl+Shift+J A]
- JS Refactor: Convert To Function Declaration
- JS Refactor: Convert To Function Expression [Ctrl+Shift+J F]
- JS Refactor: Extract Method [Ctrl+Shift+J M]
- JS Refactor: Extract Variable [Ctrl+Shift+J V]** (highlighted in orange)
- JS Refactor: Inline Variable [Ctrl+Shift+J I]

To the right of the code editor, the resulting code after extracting the variable is shown:

```
14 function function_add() {  
15   let ExtractedVariable = age + 10;  
16  
17   return ExtractedVariable;  
18 }
```

3. Basic use of VSC

3.j Code refactoring.Extract Method



3. Basic use of VSC

3.j Code refactoring. Peek Definition

```
15 console.log(psn.getFullName());  
  
intellisense-example.js ~/Escritorio/vsc_presentation-1/src~ Definitions (2)  
1 function Person(firstName, lastName, idCard) {  
2     this.FirstName = firstName || "unknown";  
3     this.LastName = lastName || "unknown";  
4  
5     this.IdCard = idCard;  
6 }  
7  
8 Person.prototype.getFullName = function () {  
9     return this.FirstName + " " + this.LastName + " " + this.idCard;  
10 }  
11  
12 var psn = new Person("Juan", "Perez", 1111);  
13  
14 console.log(psn.getFullName());  
15
```

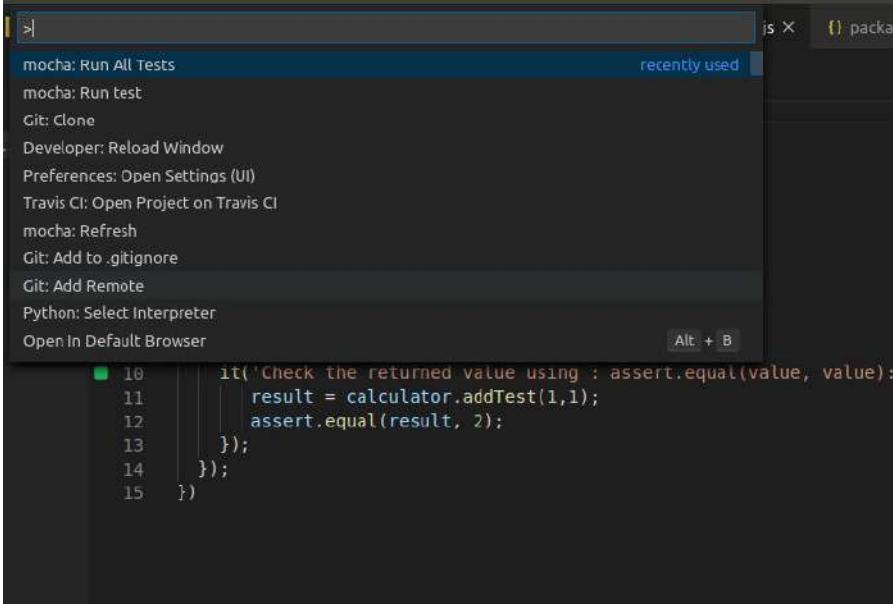
```
prototype.getFullName = function () {  
prototype.getFullName = function () {
```

4. Extensions

a. Mocha-sidebar

Useful
extension for
code testing
with Node.js
in VSC

Write your code and tests, then use extension commands in Command Palette for run your tests on sidebar. Make sure you have installed Mocha and Chai.



```
10    it('Check the returned value using : assert.equal(value, value):', function() {
11      var calculator = new Calculator();
12      var result = calculator.addTest(1,1);
13      assert.equal(result, 2);
14    });
15  })
```

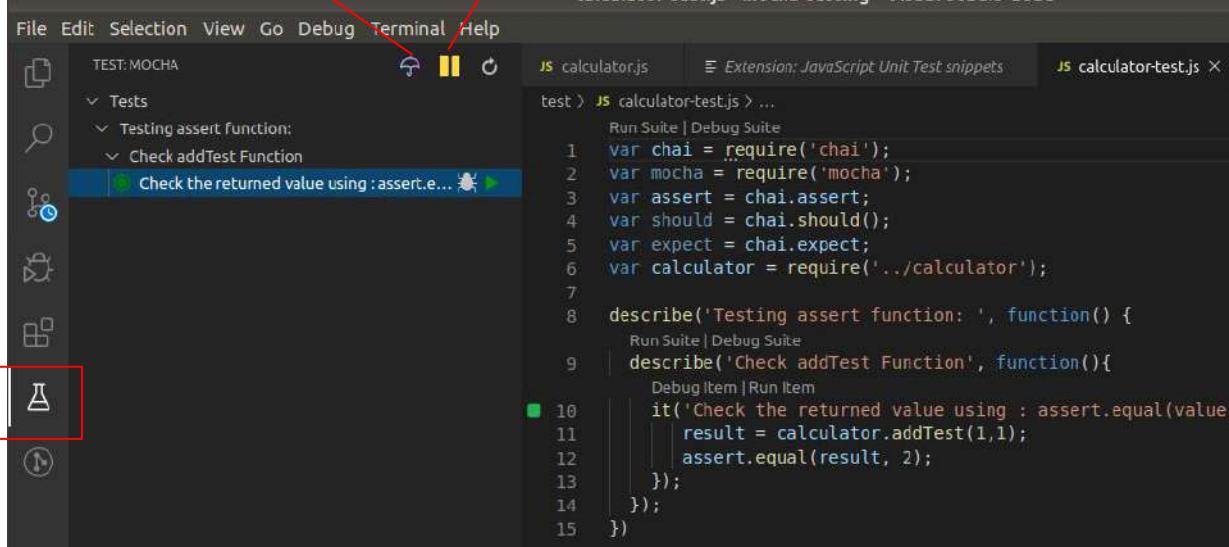
4. Extensions

a. Mocha-sidebar

Note: if you have issues running tests, make sure you have \$NODE_PATH variable set to node package path in your OS.

Code Coverage Statistics

Click for play/pause tests



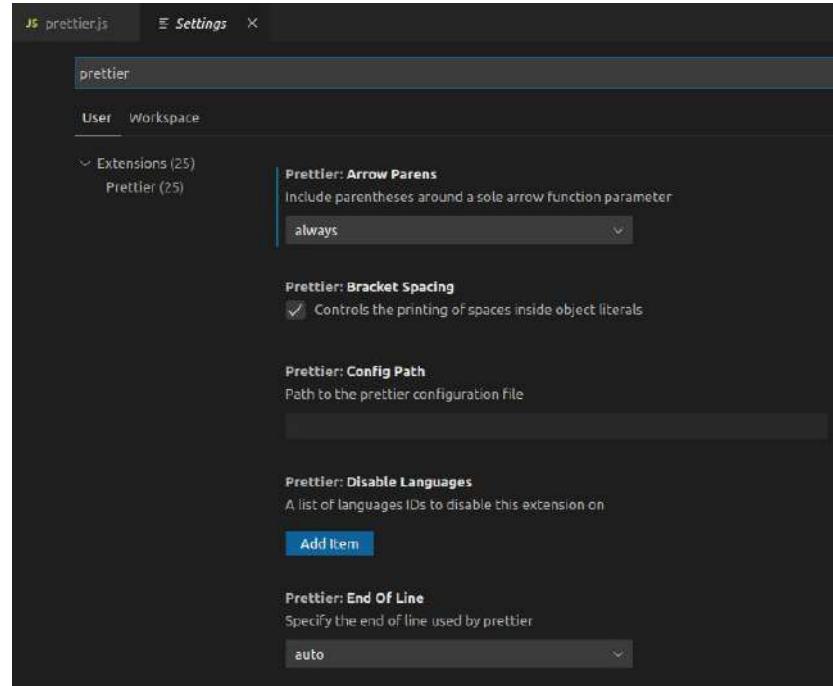
```
File Edit Selection View Go Debug Terminal Help
TEST: MOCHA
File Tests Testing assert function: Check addTest Function
Check the returned value using : assert.equal(result, 2);
JS calculator.js Extension: JavaScript Unit Test snippets JS calculator-test.js ×
test > JS calculator-test.js > ...
Run Suite | Debug Suite
1 var chai = require('chai');
2 var mocha = require('mocha');
3 var assert = chai.assert;
4 var should = chai.should();
5 var expect = chai.expect;
6 var calculator = require('../calculator');
7
8 describe('Testing assert function: ', function() {
9   Run Suite | Debug Suite
10  describe('Check addTest Function', function(){
11    Debug Item | Run Item
12      it('Check the returned value using : assert.equal(result, 2);', function() {
13        result = calculator.addTest(1,1);
14        assert.equal(result, 2);
15      });
16    });
17  });
18});
```

4. Extensions

b.Prettier

Prettier is a code formatter with support for JS and other languages.

It is highly configurable in Settings window



4. Extensions

b.Prettier

```
JS prettier.js > ...
1  const name = "James";
2
3  const person= {
4      first: name};
5
6  console.log(person);
7
8  const sayHelloLinting = (fName) => {
9      console.log(`Hello linting, ${fName}`)
10 };
11
12 sayHelloLinting("James");
13
14 //Some unformatted code
```

Enable Format
On Save Option
on Settings

4. Extensions

c. Git Lens

It helps you to visualize code authorship at a glance via Git blame.

```
src > ts images.ts > [?] images
      Jake Ginnivan, 2 years ago | 1 author (Jake Ginnivan)
1   import preloader from 'spectacle/lib/utils/preloader'
2
3   export const images = [
4     // city: require('../assets/city.jpg'),
5   ]
6
7   preloader(images)
8
```

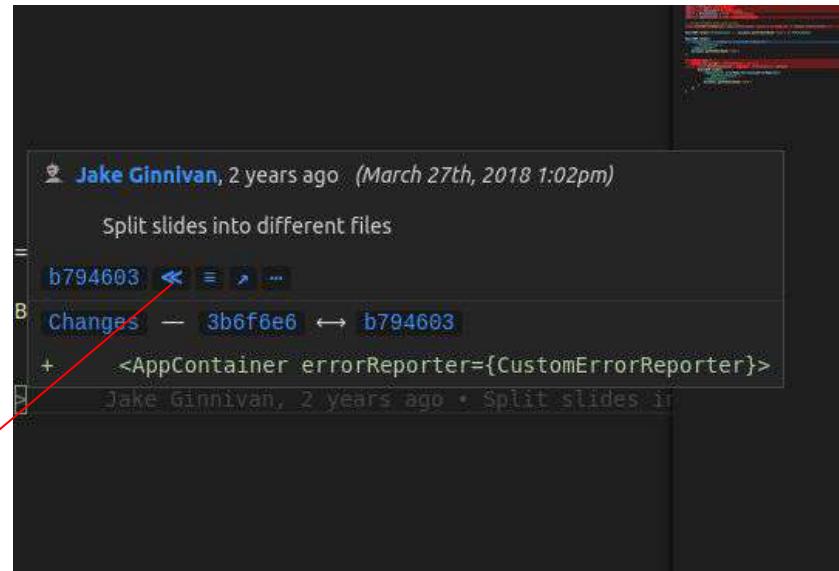
Click on a line and it will show you commit authorship and description

4. Extensions

c. Git Lens

If you hover over description, it will give you more information.

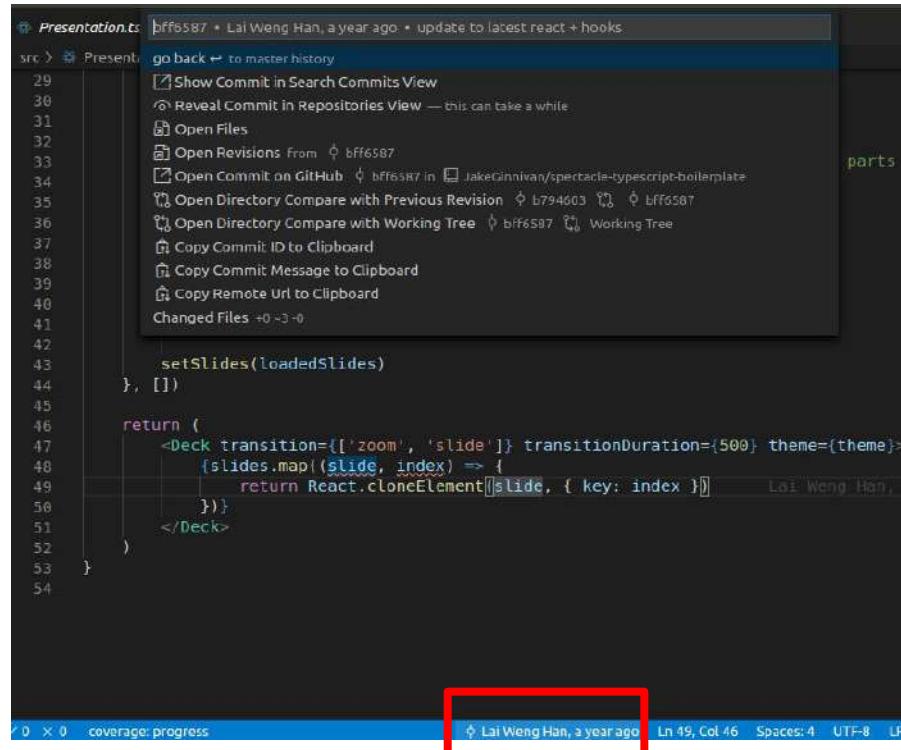
Click this button to see a diff of the whole commit



4. Extensions

c. Git Lens

Another way to interact with the code is to come to the status bar and click commit author section and then comes up a lot of options on Command Palette.



The screenshot shows a code editor with a dark theme. A context menu is open over a line of code. The menu header reads "bff6587 · Lai Weng Han, a year ago · update to latest react + hooks". The menu items include:

- go back ↺ to master history
- Show Commit in Search Commits View
- Reveal Commit in Repositories View — this can take a while
- Open Files
- Open Revisions From bff6587
- Open Commit on GitHub bff6587 in JakeGinnivan/spectacle-typescript-boilerplate
- Open Directory Compare with Previous Revision b794603 bff6587
- Open Directory Compare with Working Tree bff6587 Working Tree
- Copy Commit ID to Clipboard
- Copy Commit Message to Clipboard
- Copy Remote Url to Clipboard

Below the menu, the code editor shows a snippet of React code:

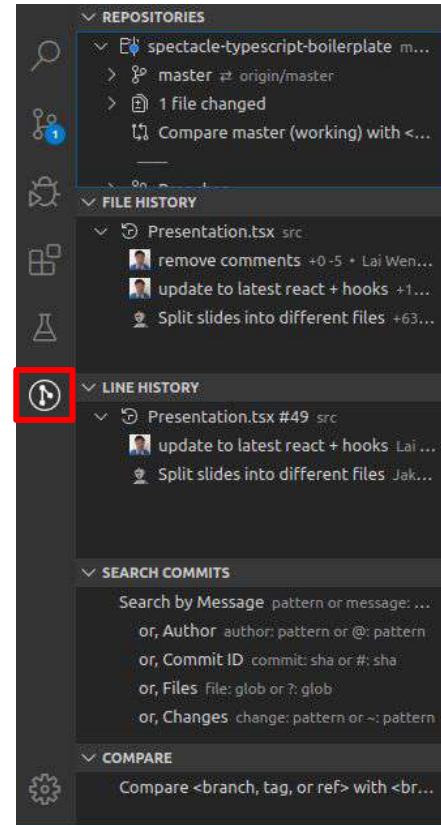
```
src > Present: bff6587 · Lai Weng Han, a year ago · update to latest react + hooks
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
setSlides(loadedSlides)
}, []
)
return (
  <Deck transition={['zoom', 'slide']} transitionDuration={500} theme={theme}>
    {slides.map((slide, index) => {
      return React.cloneElement(slide, { key: index })
    })}
  </Deck>
)
```

The status bar at the bottom of the editor shows the following information: "0 X 0 coverage: progress", "Lai Weng Han, a year ago", "Ln 19, Col 46", "Spaces: 4", "UTF-8", and "LF". A red box highlights the status bar entry "Lai Weng Han, a year ago".

4. Extensions

c. Git Lens

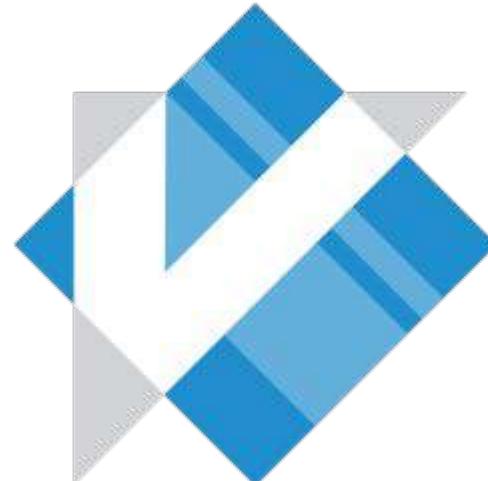
You can find more information clicking on Git Lens Icon on activity sidebar.



4. Extensions

4.d Vim

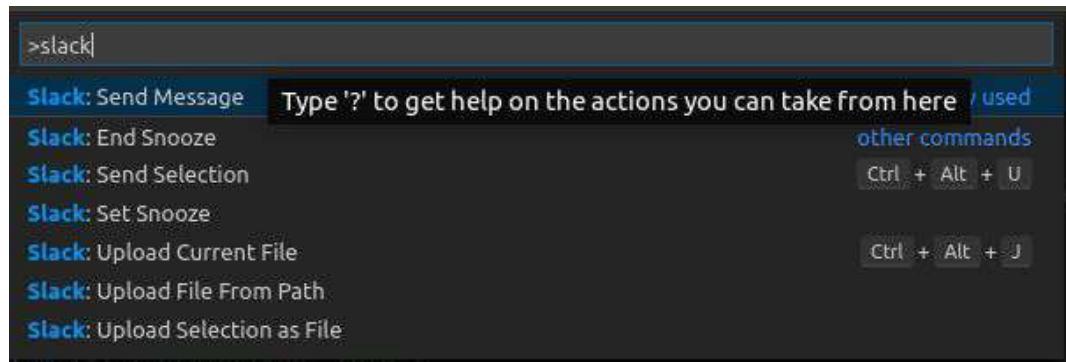
This extension allows the use of most Vim commands on VS Code.



4. Extensiones

4.e Slack

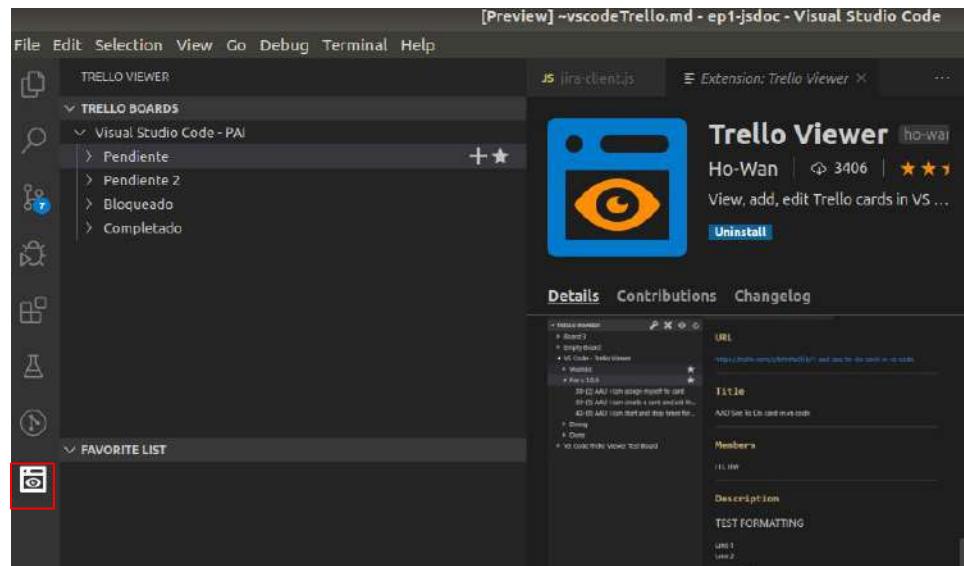
Team messaging integrated in VSC. Important: Once installed, you have to generate an API key on Slack.com that you have to paste on Team Token field in settings.



4. Extensiones

4.f Trello

After follow steps specified [here](#), you can use this extension for project tasks management.



5. JSDoc with VSC

First run on your pc **npm i --save-dev jsdoc** for install jsdoc. It will generate a **jsdoc.json** file.

After that, in node.js project package.json, in **scripts** hash we define a command that will generate documentation. Like this:

```
"docs": "node ./node_modules/.bin/jsdoc -c  
jsdoc.json"
```

Document your code using JSDoc tags and for generate documentation, run: **npm run doc**.

5. JSDoc with VSC

It will outputs a directory with a file-tree like this:

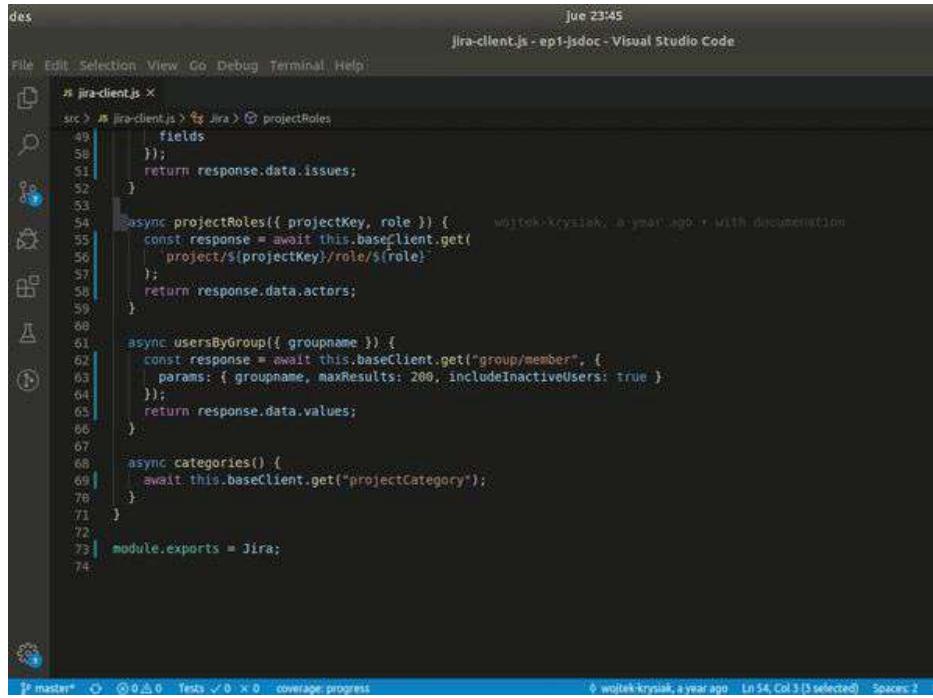
Display the
html in a
browser



```
.   fonts
      ├── OpenSans-BoldItalic-webfont.eot
      ├── OpenSans-BoldItalic-webfont.svg
      ├── OpenSans-BoldItalic-webfont.woff
      ├── OpenSans-Bold-webfont.eot
      ├── OpenSans-Bold-webfont.svg
      ├── OpenSans-Bold-webfont.woff
      ├── OpenSans-Italic-webfont.eot
      ├── OpenSans-Italic-webfont.svg
      ├── OpenSans-Italic-webfont.woff
      ├── OpenSans-LightItalic-webfont.eot
      ├── OpenSans-LightItalic-webfont.svg
      ├── OpenSans-LightItalic-webfont.woff
      ├── OpenSans-Light-webfont.eot
      ├── OpenSans-Light-webfont.svg
      ├── OpenSans-Light-webfont.woff
      ├── OpenSans-Regular-webfont.eot
      ├── OpenSans-Regular-webfont.svg
      └── OpenSans-Regular-webfont.woff
    global.html
    index.html
    jira-client.js.html
    Jira.html
  scripts
    └── linenumber.js
      └── prettify
        ├── Apache-License-2.0.txt
        ├── lang-css.js
        └── prettify.js
  styles
    ├── jsdoc-default.css
    ├── prettify-jsdoc.css
    └── prettify-tomorrow.css
  tutorial-starting-guide.html
4 directories, 30 files
```

5. JSDoc with VSC

You can use a VSC extension called **Document This** that easy up the task of writing JSDoc tags in your code.



The screenshot shows a dark-themed instance of Visual Studio Code. The title bar reads "jira-client.js - ep1-javadoc - Visual Studio Code". The status bar at the bottom indicates "Jue 23:45", "master", "0/0 0/0 Tests", "coverage progress", "wojtek-krysak a year ago", "Ln 54, Col 3 (5 selected)", "Spaces: 2", and "UTF-8". The main editor area contains the following code:

```
des
File Edit Selection View Go Debug Terminal Help
src > jira-client.js > Jira > projectRoles
  fields
);
return response.data.issues;
}

async projectRoles({ projectKey, role }) {
  const response = await this.baseClient.get(
    `project/${projectKey}/role/${role}`
  );
  return response.data.actors;
}

async usersByGroup({ groupname }) {
  const response = await this.baseClient.get("group/member", {
    params: { groupname, maxResults: 200, includeInactiveUsers: true }
  });
  return response.data.values;
}

async categories() {
  await this.baseClient.get("projectCategory");
}

module.exports = Jira;
```

6. JS Linting in VSCode

a. Definition

Linting is an analysis process used to find:

- programming errors
- bugs
- code that doesn't fit certain guidelines

6. JS Linting in VSCode

b. Why using a Linter?

- Without a linter we only know about errors when they occur in execution.
- With a linter we can find errors without running the program in interpreted languages, saving time.

6. JS Linting in VSCode

c. Linting in VSCode

VSCode doesn't come with any linter built-in.

We will use ESLint, as it's the most popular and maintained extension.



6. JS Linting in VSCode

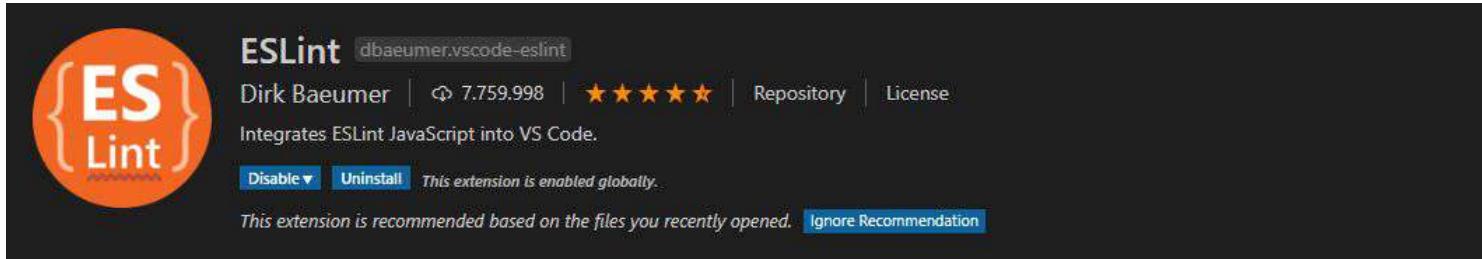
d. ESLint

ESLint is a open source Javascript linting software where developers can use their own linting rules.

6. JS Linting in VSCode

e. Installing ESLint

- First, we need to install its VSCode extension



6. JS Linting in VSCode

e. Installing ESLint

Secondly, we need to install its npm package.

- locally: `$ npm install eslint`
- globally: `$ npm install -g eslint`

6. JS Linting in VSCode

e. Installing ESLint

Now we have to configure ESLint.

- For local: `$./node_modules/.bin/eslint --init`
- For global: `$ eslint --init`

6. JS Linting in VSCode

f. Configuring ESLint

.eslintrc.json

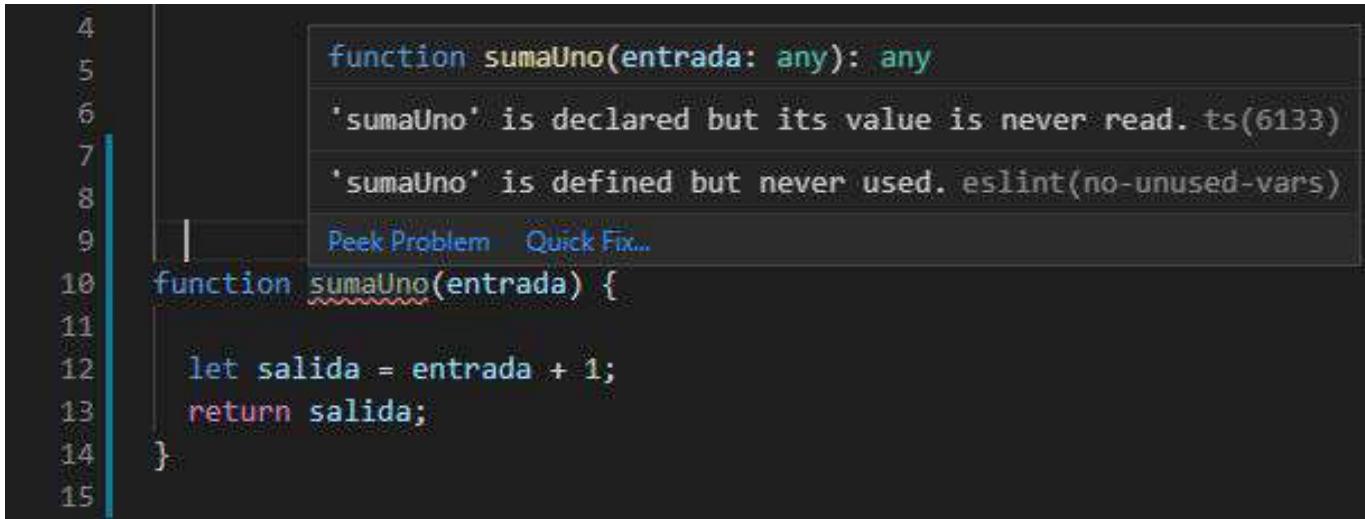
```
① .eslintrc.json > {} globals
1  {
2      "env": {
3          "es6": true,
4          "node": true
5      },
6      "extends": "eslint:recommended",
7      "globals": {
8          "Atomics": "readonly",
9          "SharedArrayBuffer": "readonly"
10     },
11     "parserOptions": {
12         "ecmaVersion": 2018,
13         "sourceType": "module"
14     },
15     "rules": {
16     }
17 }
```

```
> eslint --init

? How would you like to use ESLint? To check syntax and find problems
? What type of modules does your project use? CommonJS (require(exports))
? Which framework does your project use? None of these
? Does your project use TypeScript? No
? Where does your code run? Browser, Node
? What format do you want your config file to be in? JSON
```

6. JS Linting in VSCode

g. Using ESLint



A screenshot of the Visual Studio Code (VSCode) interface demonstrating ESLint integration. The code editor shows a file with the following content:

```
4
5     function sumaUno(entrada: any): any
6
7     'sumaUno' is declared but its value is never read. ts(6133)
8
9     |     'sumaUno' is defined but never used. eslint(no-unused-vars)
10    function sumaUno(entrada) {
11
12        let salida = entrada + 1;
13        return salida;
14    }
15
```

The line 'sumaUno' is highlighted with a red squiggle underlining. A tooltip box appears over the line, containing two ESLint error messages:

- 'sumaUno' is declared but its value is never read. ts(6133)
- 'sumaUno' is defined but never used. eslint(no-unused-vars)

Below the tooltip, there are two buttons: "Peek Problem" and "Quick Fix...".

7. JS Debugging in VSC

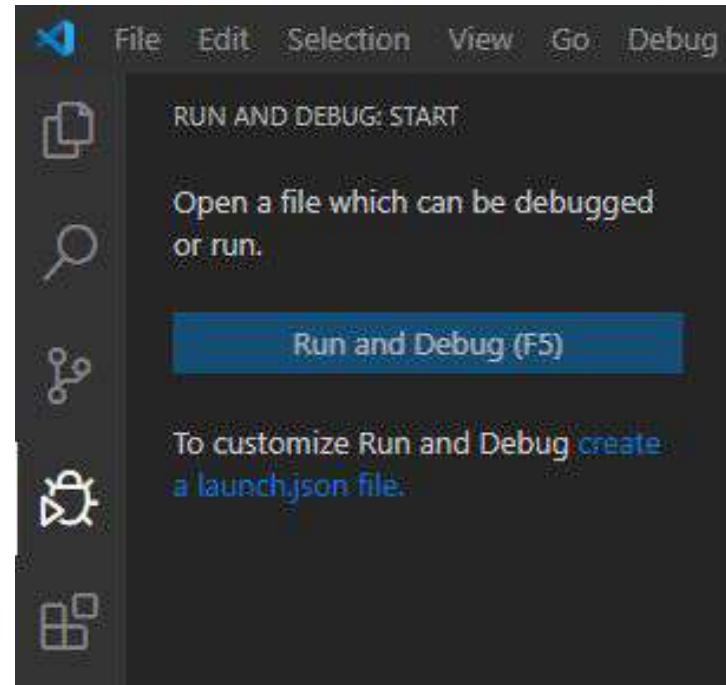
a. Debugger

- VSCode comes with node.js debugger built-in.
- There are debuggers for most languages, which can be installed as an extension.

7. JS Debugging in VSCode

b. Debug View

- In this section we can debug our program.
- We need to create a config file for custom debugging.



7. JS Debugging in VSCode

c. Entering debug mode

We can enter debug mode:

- Using Debug View, Debug Button (or F5)
- Using **\$ node --inspect-brk app.js**

7. JS Debugging in VSCode

d. Breakpoints

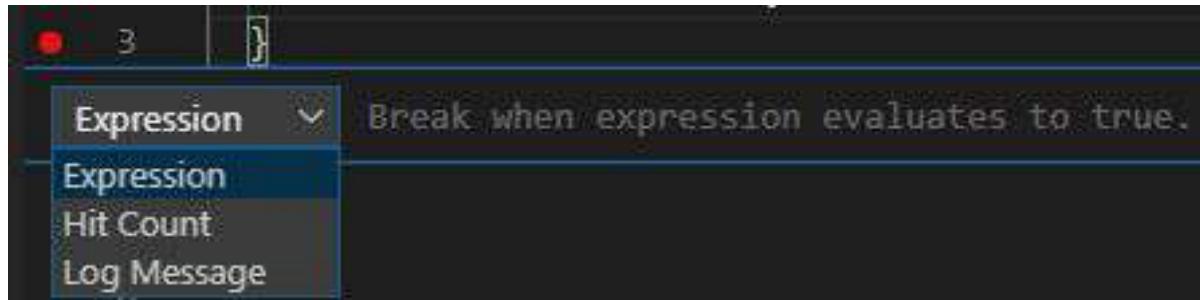
- Breakpoints work as usual, clicking right side of line numbering does activate them.
- Right-click for managing a breakpoint.



7. JS Debugging in VSCode

d. Breakpoints

We can create a conditional breakpoint if we edit it and add an expression.



8. Remote editing in IaaS VM

a. Introduction

- It's possible to edit files that are directly in a remote VM.
- This way it's easy to code from different platforms, while keeping source code and its environment in one place.

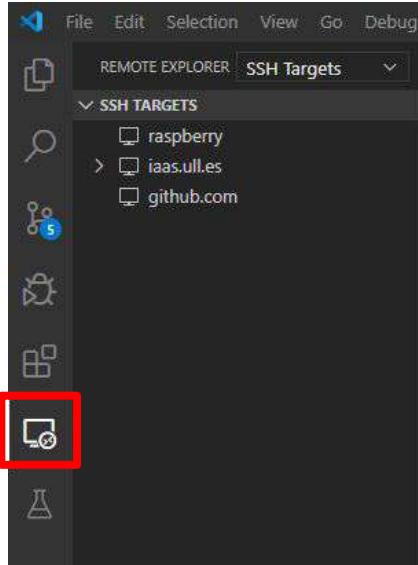
8. Remote editing in IaaS VM

b. Setup requirements

- SSH Client in local computer
- VSCode in local computer
- SSH Extension in VSCode
- SSH Server in remote VM (already installed)

8. Remote editing in IaaS VM

c. Setup



ssh_config example:

```
Host github.com
User git
Port 22
HostName github.com
IdentityFile ~/.ssh/alu
TCPKeepAlive yes
IdentitiesOnly yes
```

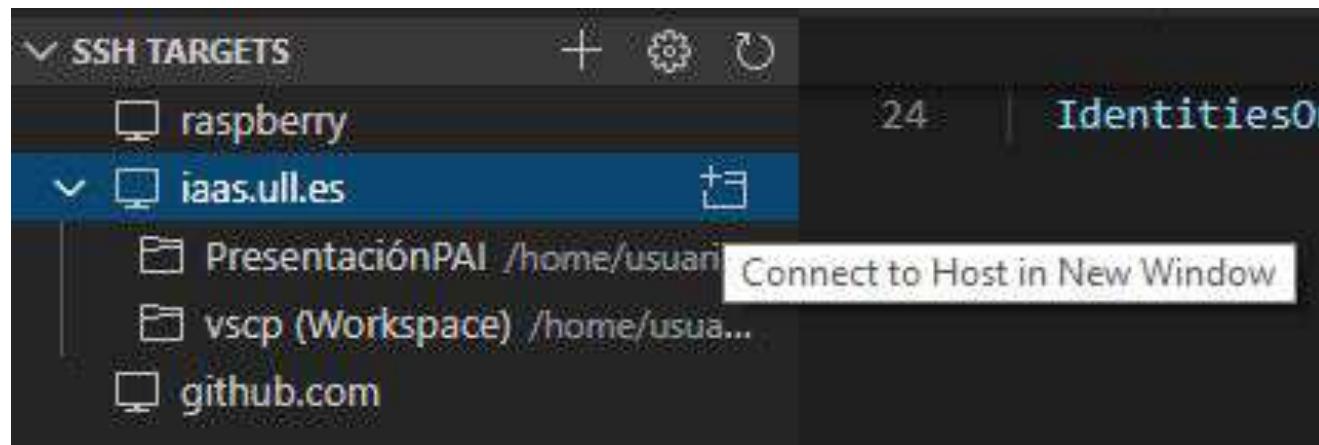
8. Remote editing in IaaS VM

d. Key pair based authentication

- Generate private and public keys in local computer
- Register public key in authorized_keys in VM.
- Use appropriate config in ssh_config in local computer

8. Remote editing in IaaS VM

e. Login



8. Remote editing in IaaS VM

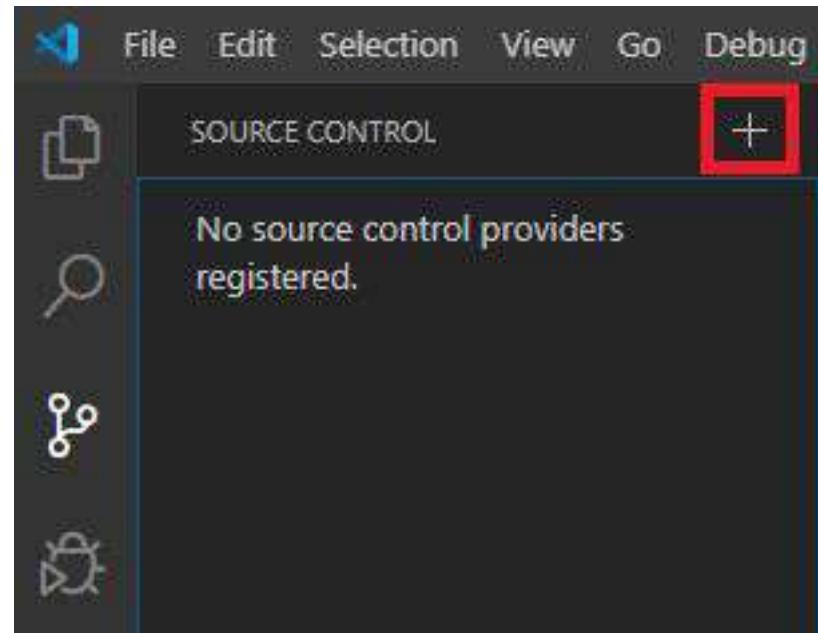
f. Considerations

- On first login, VSCode will try to download and install a little program used to make this connection work.
- Extensions need to be installed on remote machine and local machine at the same time to be used.

9. Source Control in VSC

a. git integration

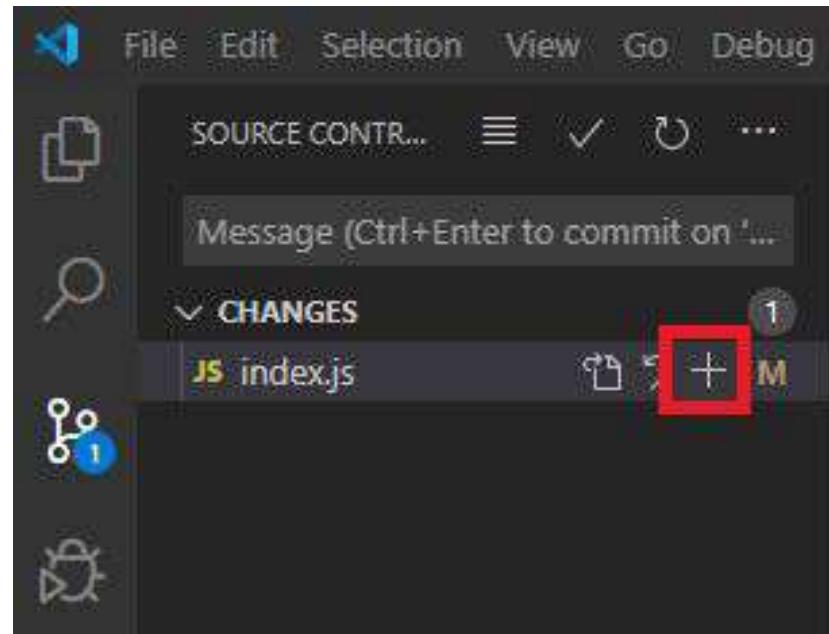
We can open a git project or create a new one.



9. Source Control in VSCode

a. git integration

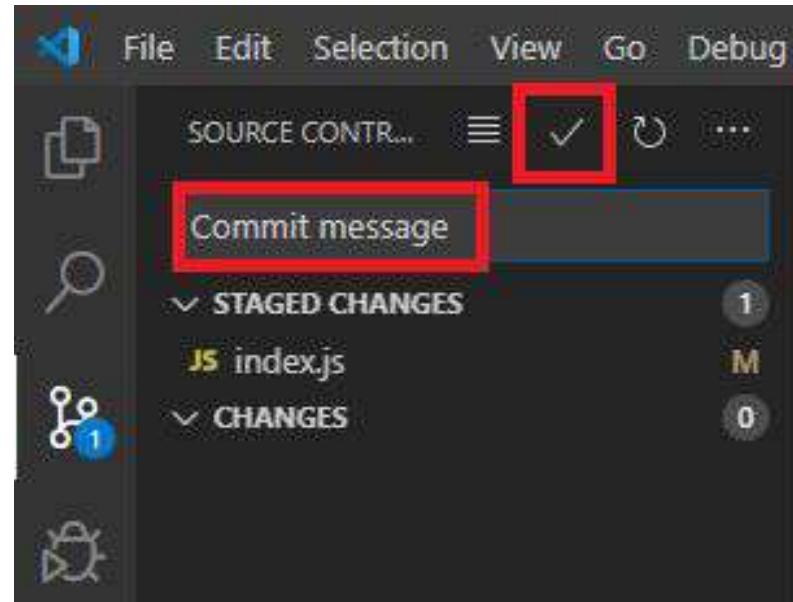
Once we have changes, we can stage them.



9. Source Control in VSCode

a. git integration

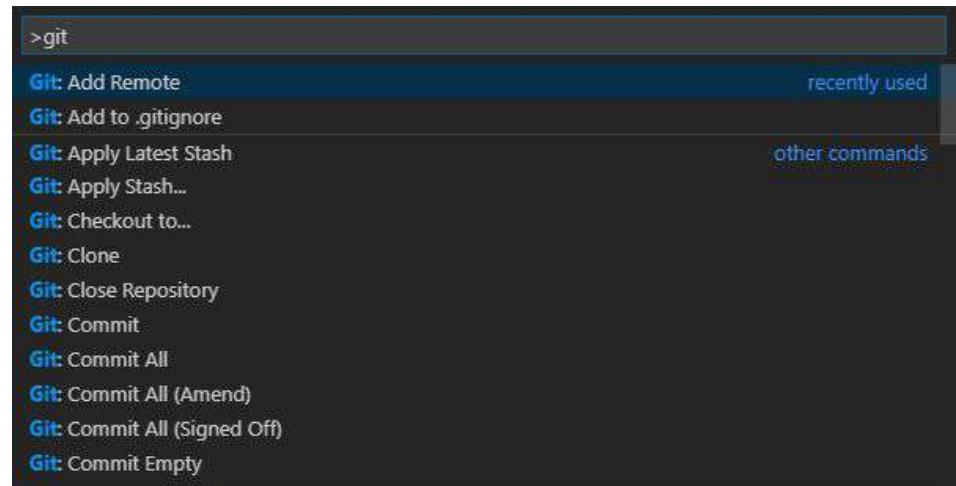
Finally we can make a commit



9. Source Control in VSCode

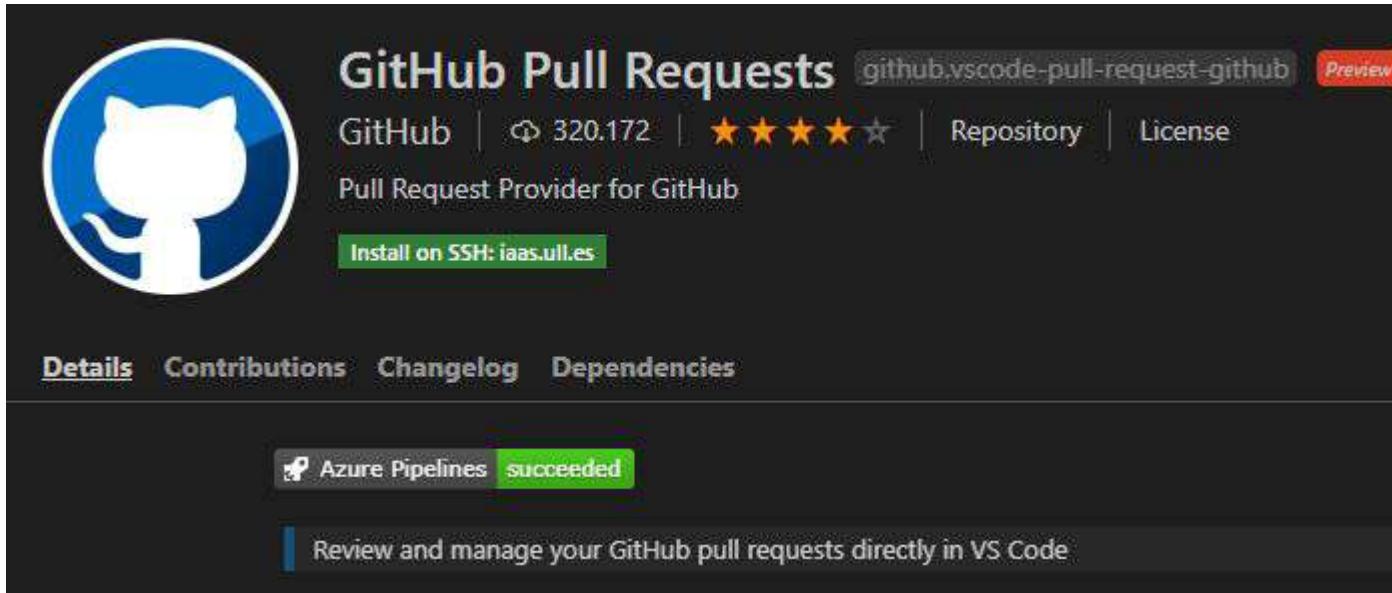
b. Adding a remote

- You can use Command Palette to use git commands as clone and add remote.



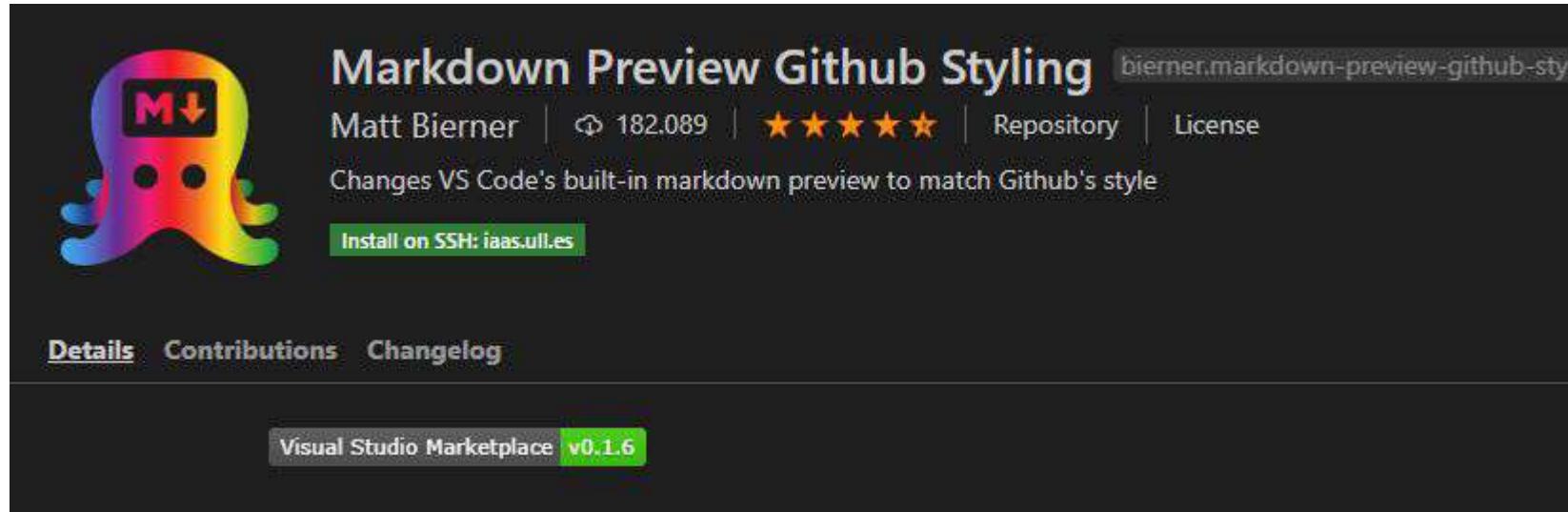
9. Source Control in VSCode

b. GitHub integration



9. Source Control in VSCode

b. GitHub integration



The screenshot shows the details page for the "Markdown Preview Github Styling" extension in the Visual Studio Marketplace. The extension icon is a colorful, stylized octopus-like creature with a central "M" logo. The title is "Markdown Preview Github Styling" by Matt Bierner, with 182.089 installs and a 5-star rating. It includes links to the repository and license. A description states it changes VS Code's built-in markdown preview to match Github's style. A green button at the bottom says "Install on SSH: iaas.ull.es". Below the main section are links for "Details", "Contributions", and "Changelog". At the bottom, it says "Visual Studio Marketplace v0.1.6".

Markdown Preview Github Styling bierner.markdown-preview-github-styl

Matt Bierner | 182.089 | ★★★★★ | Repository | License

Changes VS Code's built-in markdown preview to match Github's style

Install on SSH: iaas.ull.es

Details Contributions Changelog

Visual Studio Marketplace v0.1.6

10. Keyboard Shortcuts

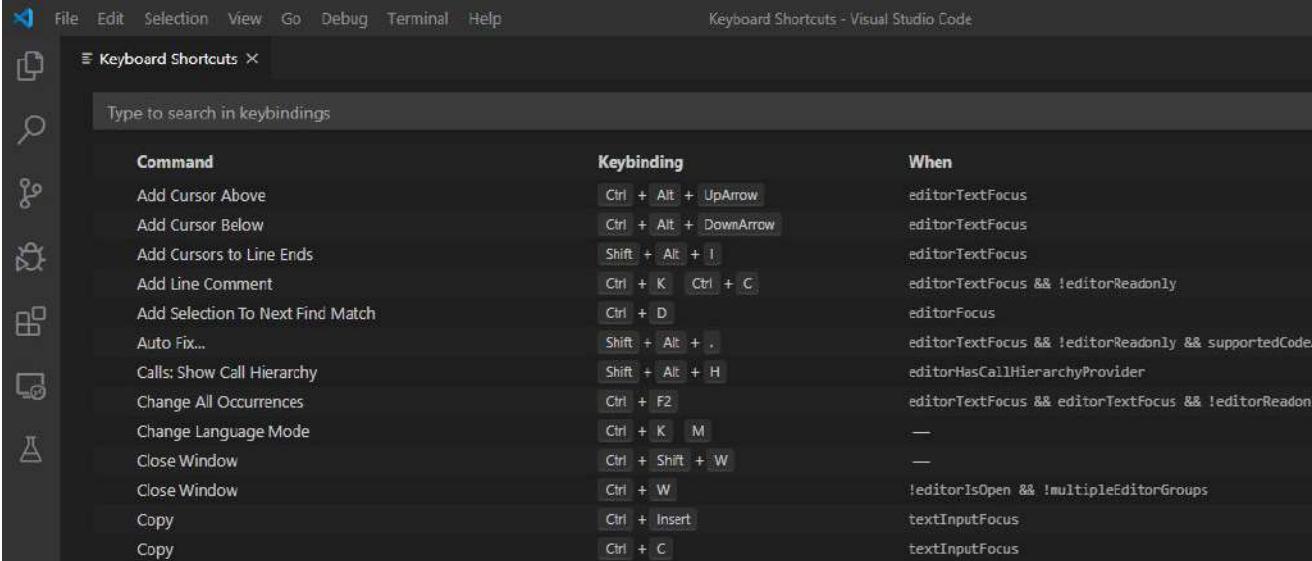
a. Introduction

- Most tasks in VSCode can be done using keyboard alone.
- You can edit all keyboard shortcuts and add custom ones too.

10. Keyboard Shortcuts

b. Editing keyboard shortcuts

Keyboard
shortcut
editor



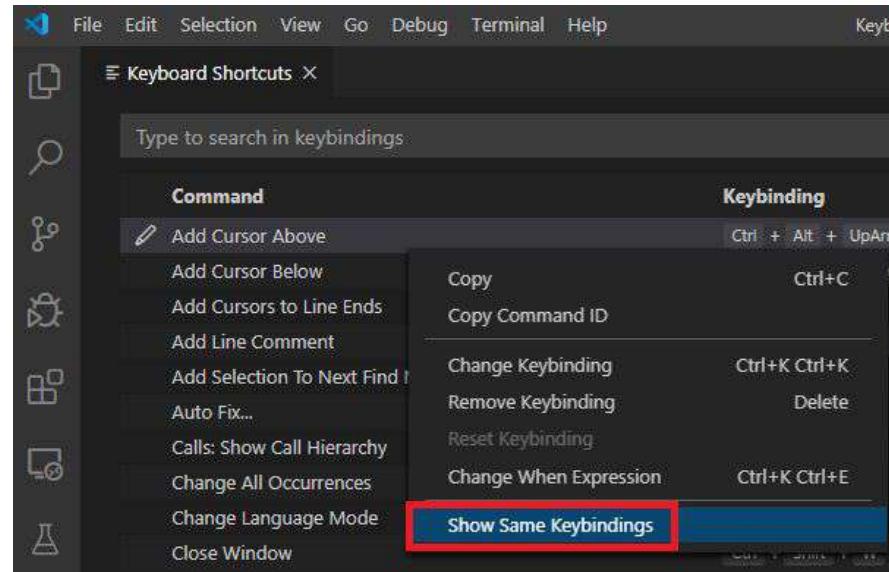
The screenshot shows the 'Keyboard Shortcuts' settings page in Visual Studio Code. The title bar reads 'Keyboard Shortcuts - Visual Studio Code'. The main area has a dark background with white text. On the left is a vertical toolbar with icons for file operations like Open, Save, Find, and Copy/Paste. The central panel has a search bar at the top labeled 'Type to search in keybindings'. Below it is a table with three columns: 'Command', 'Keybinding', and 'When'. The 'Command' column lists various editor actions. The 'Keybinding' column shows the corresponding keyboard shortcuts. The 'When' column specifies the context where each shortcut is active.

Command	Keybinding	When
Add Cursor Above	Ctrl + Alt + UpArrow	editorTextFocus
Add Cursor Below	Ctrl + Alt + DownArrow	editorTextFocus
Add Cursors to Line Ends	Shift + Alt + I	editorTextFocus
Add Line Comment	Ctrl + K Ctrl + C	editorTextFocus && !editorReadOnly
Add Selection To Next Find Match	Ctrl + D	editorFocus
Auto Fix...	Shift + Alt + .	editorTextFocus && !editorReadOnly && supportedCodeActions
Calls: Show Call Hierarchy	Shift + Alt + H	editorHasCallHierarchyProvider
Change All Occurrences	Ctrl + F2	editorTextFocus && editorTextFocus && !editorReadOnly
Change Language Mode	Ctrl + K M	—
Close Window	Ctrl + Shift + W	—
Close Window	Ctrl + W	!editorIsOpen && !multipleEditorGroups
Copy	Ctrl + Insert	textInputFocus
Copy	Ctrl + C	textInputFocus

10. Keyboard Shortcuts

b. Editing keyboard shortcuts

Check for conflicting
keybindings



10. Keyboard Shortcuts

c. Custom keyboard shortcuts

- You can create custom keyboard shortcuts by declaring them in `keybindings.json`
- Each key binding only needs 3 values:
“key”, “command” and “when”

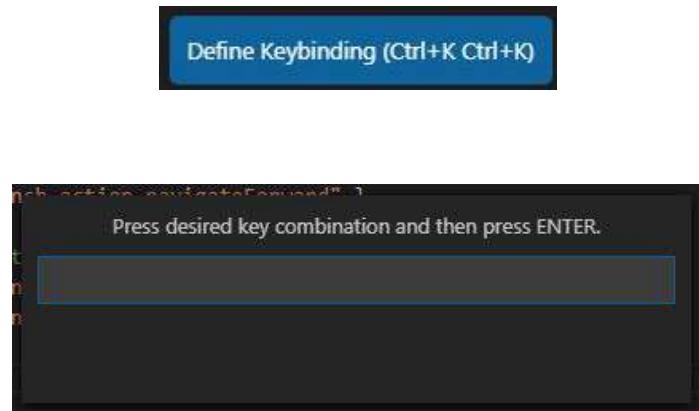
10. Keyboard Shortcuts

```
1 // Place your key bindings in this file to override the defaults
2 [
3     // Keybindings that are active when the focus is in the editor
4     { "key": "home",             "command": "cursorHome",           "when": "editorTextFocus" },
5     { "key": "shift+home",       "command": "cursorHomeSelect",    "when": "editorTextFocus" },
6
7     // Keybindings that are complementary
8     { "key": "f5",              "command": "workbench.action.debug.continue", "when": "inDebugMode" },
9     { "key": "f5",              "command": "workbench.action.debug.start",      "when": "!inDebugMode" },
10
11    // Global keybindings
12    { "key": "ctrl+f",          "command": "actions.find" },
13    { "key": "alt+left",         "command": "workbench.action.navigateBack" },
14    { "key": "alt+right",        "command": "workbench.action.navigateForward" },
15
16    // Global keybindings using chords (two separate keypress actions)
17    { "key": "ctrl+k enter",    "command": "workbench.action.keepEditor" },
18    { "key": "ctrl+k ctrl+w",   "command": "workbench.action.closeAllEditors" },
19
20 ]
```

10. Keyboard Shortcuts

c. Custom keyboard shortcuts

- While editing keybinding.json we can create custom keyboard shortcuts
- This automatically inserts a new object in keybinding.json, so you only need to edit.



10. Keyboard Shortcuts

c. Custom keyboard shortcuts

```
{  
  "key": "ctrl+shift+b",  
  "command": "commandId",  
  "when": "editorTextFocus"  
}
```

10. Keyboard Shortcuts

c. Custom keyboard shortcuts

- To remove a key binding, we can make an entry on keybindings.json with a “negative” command.

```
// To remove the second rule, for example, add in keybindings.json:  
{ "key": "tab", "command": "-jumpToNextSnippetPlaceholder" }  
]
```

10. Keyboard Shortcuts

d. Cheatsheets

- There are cheatsheets for Windows, Linux and Mac default keyboard shortcuts.

Visual Studio Code

Keyboard shortcuts for Linux

General

Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window-instance
Ctrl+W	Close window-instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ / -	Move line down/up
Ctrl+Shift+K	Delete line
Ctrl+Enter /	Insert line below/ above
Ctrl+Shift+Enter	
Ctrl+Shift+\`	Jump to matching bracket
Ctrl+] / Ctrl+[Indent/Outdent line
Home / End	Go to beginning/end of line
Ctrl+ Home / End	Go to beginning/end of file
Ctrl+ / -	Scroll line up/down
Alt+ PgUp / PgDn	Scroll page up/down
Ctrl+Shift+ [/]	Fold/unfold region
Ctrl+K Ctrl+ [/]	Fold/unfold all subregions
Ctrl+K Ctrl+0 /	Fold/Unfold all regions
Ctrl+K Ctrl+J	
Ctrl+K Ctrl+C	Add line comment
Ctrl+K Ctrl+U	Remove line comment
Ctrl+/	Toggle line comment
Ctrl+Shift+A.	Toggle block comment
Alt+Z	Toggle word wrap

Rich languages editing

Ctrl+Space	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Ctrl+Shift+I	Format document
Ctrl+K Ctrl+F	Format selection
F12	Go to Definition
Ctrl+Shift+F10	Peek Definition
Ctrl+K F12	Open Definition to the side
Ctrl+.	Quick Fix
Shift+F12	Show References
F2	Rename Symbol
Ctrl+K Ctrl+X	Trim trailing whitespace
Ctrl+K M	Change file language

Multi-cursor and selection

Alt+Click	Insert cursor*
Shift+Alt+ / -	Insert cursor above/below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt+ --	Expand selection
Shift+Alt+ --	Shrink selection
Shift+Alt+ drag mouse	Column (box) selection

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+ + / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in file
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+C	Open new command prompt/terminal
Ctrl+K Ctrl+H	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K Ctrl+V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Search and replace

Ctrl+F	Find
Ctrl+H	Replace
F3 / Shift+F3	Find next/previous
Alt+Enter	Select all occurrences of Find match
Ctrl+D	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match

Navigation

Ctrl+T	Show all Symbols
Ctrl+G	Go to Line...
Ctrl+P	Go to File...
Ctrl+Shift+O	Go to Symbol...
Ctrl+Shift+M	Show Problems panel
F8	Go to next error or warning
Shift+F8	Go to previous error or warning
Ctrl+Shift+Tab	Navigate editor group history
Ctrl+Alt+-	Go back
Ctrl+Shift+-	Go forward
Ctrl+M	Toggle Tab moves focus

Editor management

Ctrl+W	Close editor
Ctrl+K F	Close folder
Ctrl+L	Split editor
Ctrl+ / 2 / 3	Focus into 1 st , 2 nd , 3 rd editor group
Ctrl+K Ctrl+ --	Focus into previous editor group
Ctrl+K Ctrl+ --	Focus into next editor group
Ctrl+Shift+PgUp	Move editor left
Ctrl+Shift+PgDn	Move editor right
Ctrl+K --	Move active editor group left/up
Ctrl+K --	Move active editor group right/down

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+W	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window-instance

Debug

F9	Toggle breakpoint
F5	Start / Continue
F11 / Shift+F11	Step into/out
F10	Step over
Shift+F5	Stop
Ctrl+K Ctrl+I	Show hover

Integrated terminal

Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+Shift+C	Copy selection
Ctrl+Shift+V	Paste into active terminal
Ctrl+Shift+I / -	Scroll up/down
Shift+PgUp / PgDn	Scroll page up/down
Shift+ Home / End	Scroll to top/bottom

* The Alt+Click gesture may not work on some Linux distributions. You can change the modifier key for the Insert cursor command to Ctrl+Click with the "editor.multiCursorModifier" setting.

10. Keyboard Shortcuts

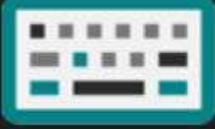
e. Keymaps

- With a keymap you can get the same keyboard shortcuts as other popular text editors like SublimeText, Atom and XCode.



10. Keyboard Shortcuts

e. Keymaps

 Atom Keymap ms-vscode.atom-keybindings Preview

Microsoft | ⚡ 465.906 | ★★★★★ | Repository | License

Popular Atom keybindings for Visual Studio Code

[Install](#)

This extension is recommended because you have Atom installed. [Ignore Recommendation](#)

[Details](#) [Contributions](#) [Changelog](#)

Atom Keymap for VS Code

This extension ports popular Atom keyboard shortcuts to Visual Studio Code. After installing the extension and restarting VS Code your favorite keyboard shortcuts from Atom are now available.

11. Bibliography

- [Running VSC in Linux](#)
- [Keyboard Shortcuts](#)
- [Basic Editing in VSC](#)
- [Getting Started with VSC](#)
- [Documentatios of VSC features](#)
- [What is a Workspace in VSC](#)
- [Debugging in VSC](#)
- [Key bindings management in VSC](#)
- [Setup and Customizing VSC](#)
- [Quick tour of JS tools in VSC](#)

11. Bibliography

- [Working with JS in VSC](#)
- [Node.js Tools for VSC](#)
- [Using Version Control in VSC](#)
- [10 Essential VS Code Extensions for JavaScript Developers in 2019](#)
- [Mocha-sidebar documentation](#)
- [Git Repository has too many active changes error solution](#)
- [Tips of using IntelliSense for JavaScript](#)

Finish. Thank you
for your attention,
feel free to ask us
any questions you
have about these
presentation.

