使用 Visual Studio Code 进行实验

为什么使用 VSC

本课程的实验需要在 Ubuntu 下完成，如果你的主力操作系统并非 Ubuntu，那可能需要在虚拟机或 WSL 中进行实验。如果你恰好又不熟悉这类环境，那你的工作效率可能会大受影响。Visual Studio Code 是一个跨平台的代码编辑器，能够远程接入实验环境，并在本地提供接近 IDE 的开发体验。为了提升同学们的编码和调试体验，我们建议在本机上安装 VSC，然后通过远程连接的方式完成实验。

安装和配置

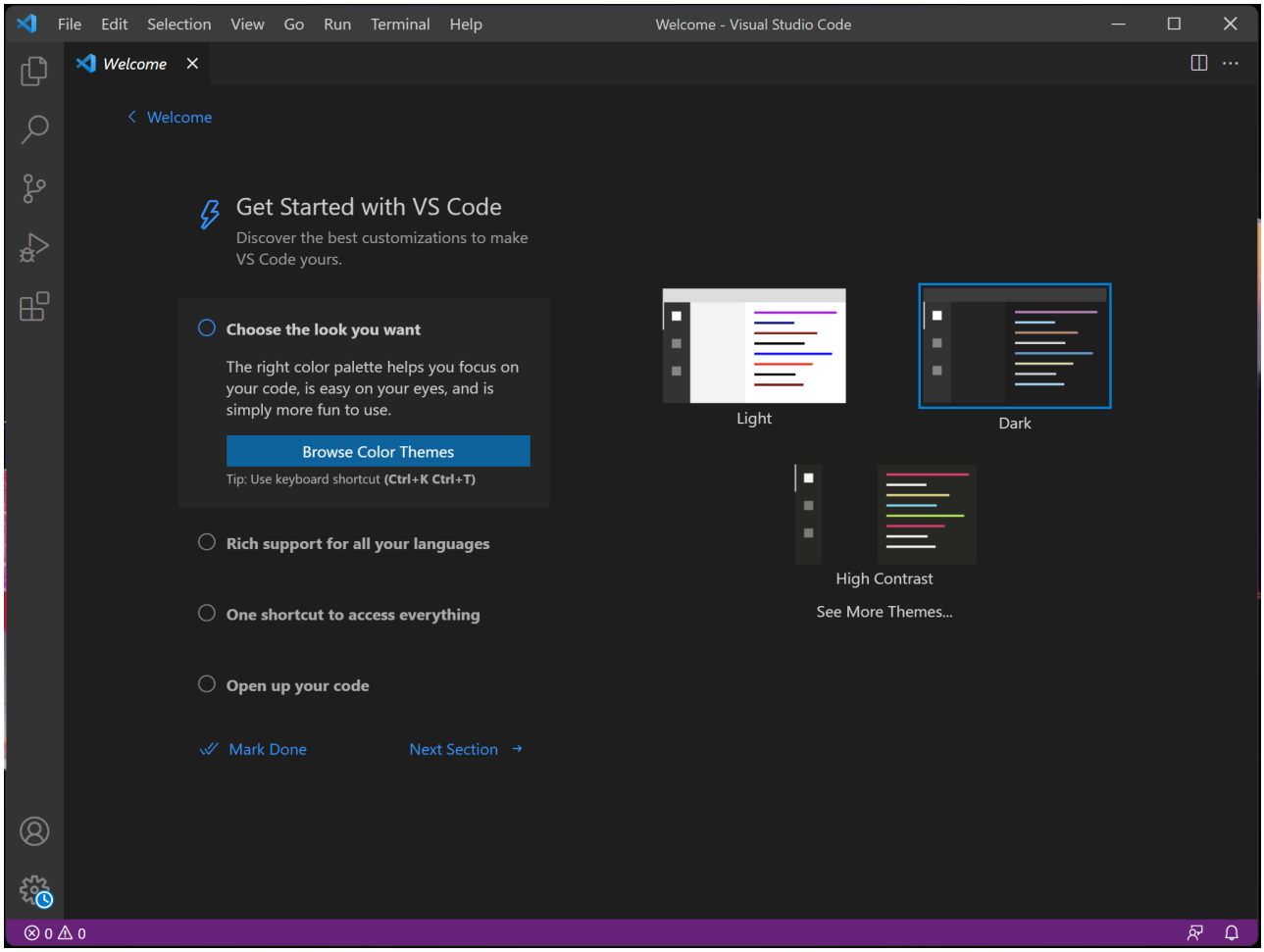
1. 这里假设同学们已经安装好虚拟机（或 WSL），并且能够在本地通过命令行登录。如果想要验证安装的正确性，可以采用以下方式：

   - 如果你使用的是虚拟机，那么在命令行中输入 `ssh <user_name>@<vm_ip>`，然后输入密码；
   - 如果你使用的是 WSL，那么在命令行中输入 `wsl` 或 `bash`。

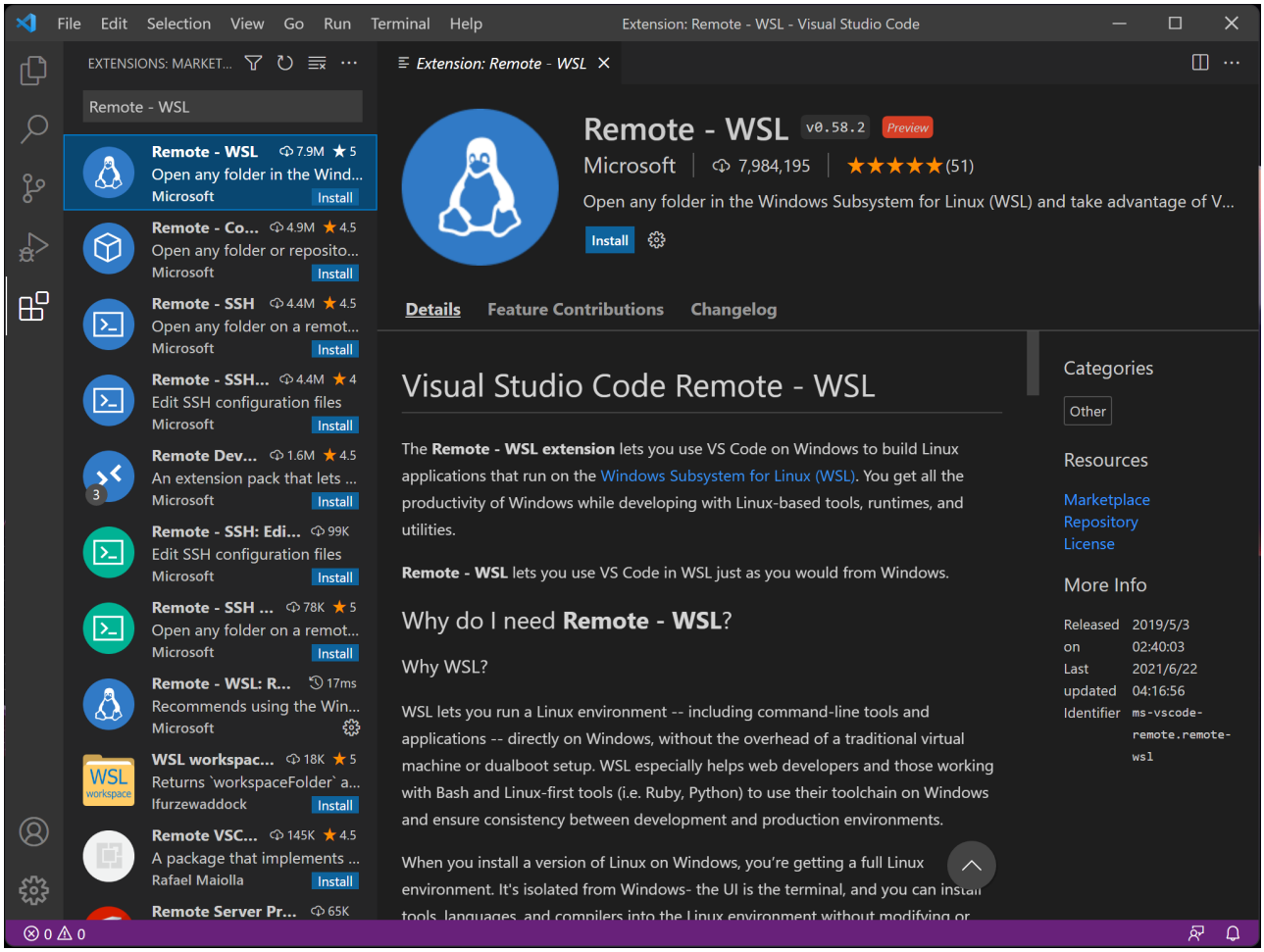
   如果安装正确，此时你应该成功进入 Ubuntu 环境，并看到类似下面的输出：

   ```
   qh2333@QH-Workstation:~$
   ```

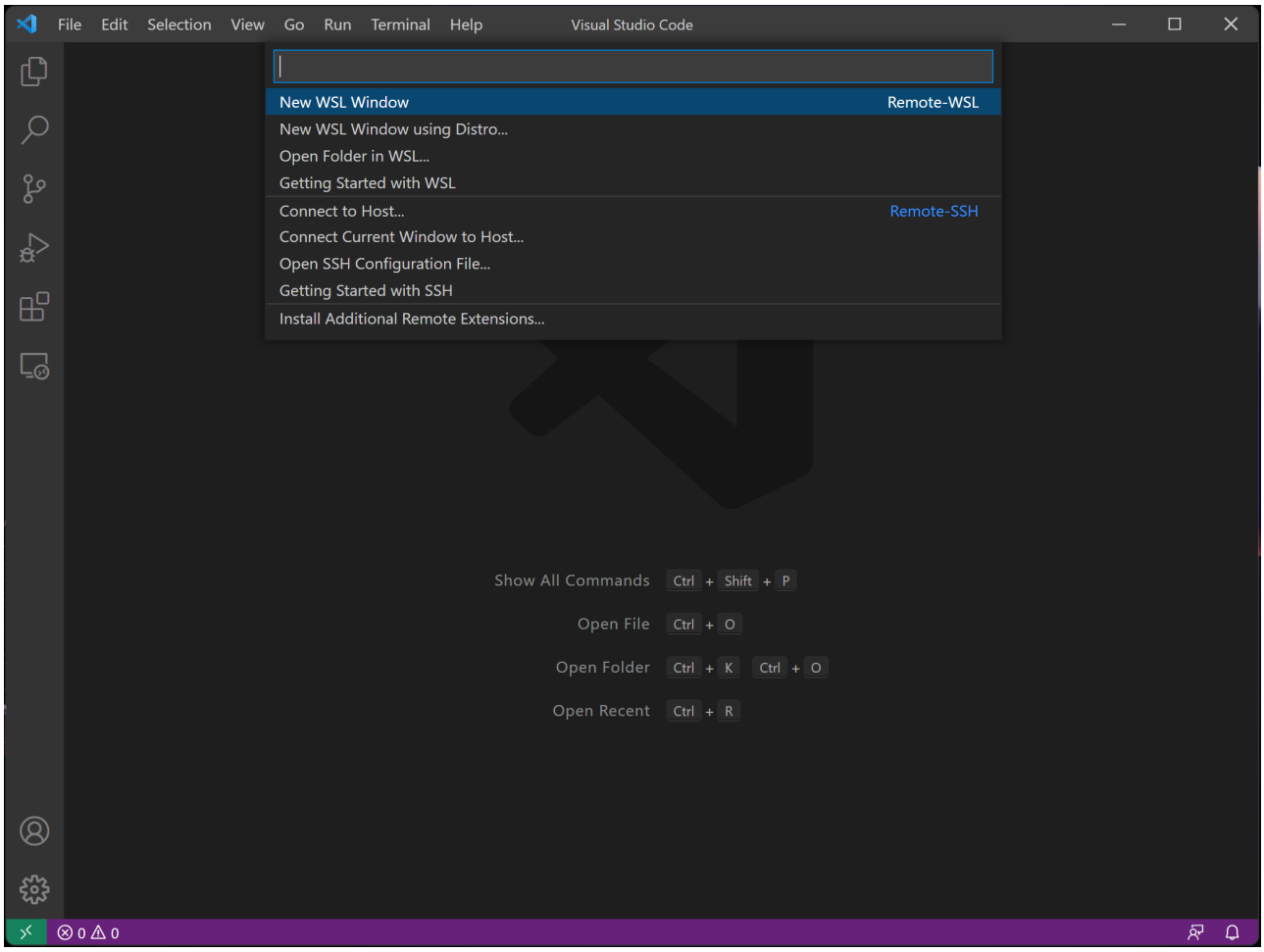
2. 在官方网站下载最新版本的 VSC，完成安装并打开，你应该能看到以下界面：

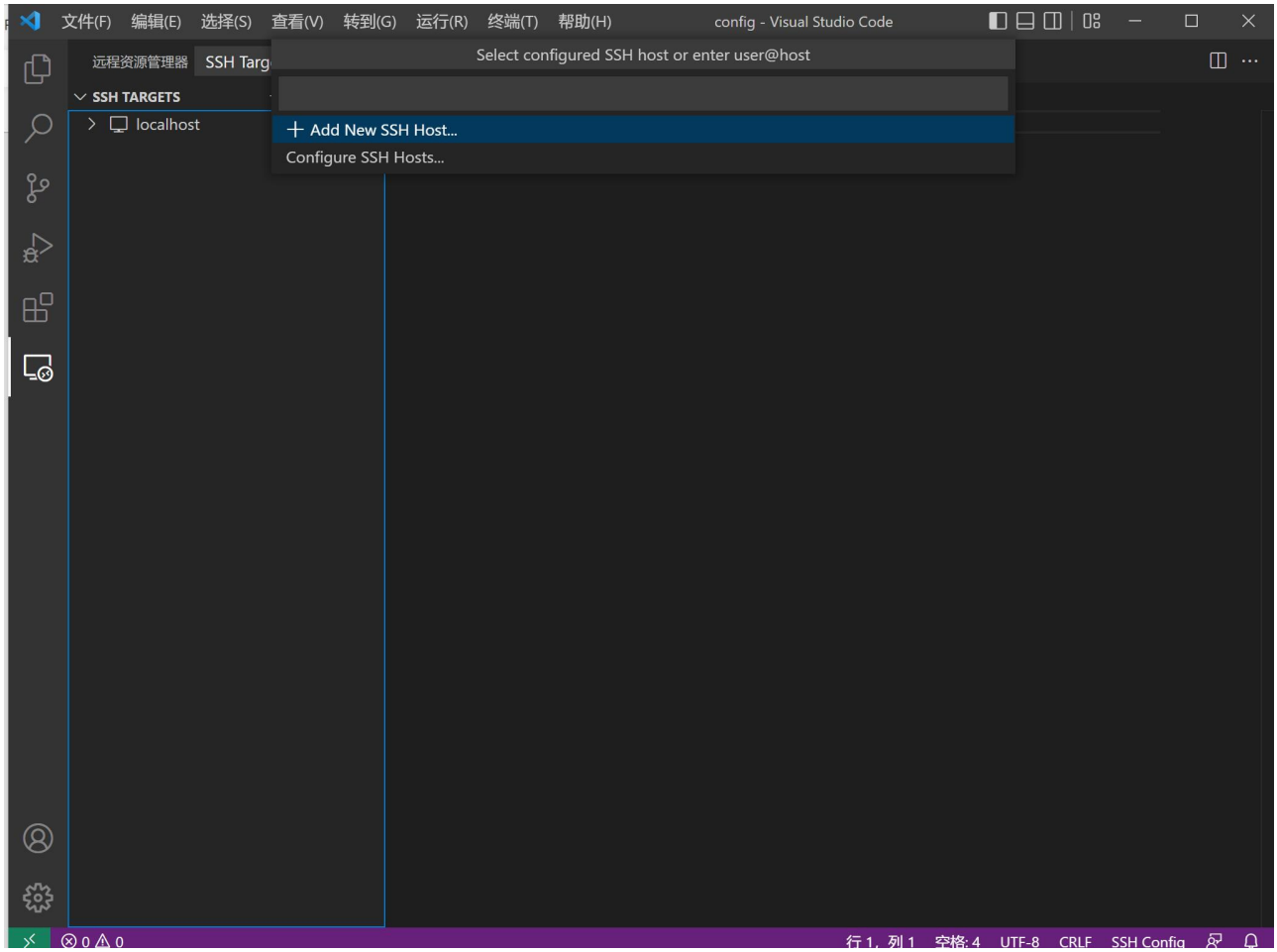


3. 点击左侧第五个图标，或者按快捷键 `Ctrl+Shift+X` 打开扩展面板。你可以在面板上侧的输入框里按名称搜索扩展，然后点击 `Install` 进行安装。



4. 首先根据你选用的实验环境，安装 `Remote - SSH` 或 `Remote - WSL` 拓展。安装完成后， `VSC` 的左下角会出现绿色的 `Open a remote window` 按钮。以虚拟机环境为例，选择 `Remote - SSH` 的 `Connect to Host`，点击之后，会出现如下第二张图的界面，点击`add new SSH Host`,按命令行连接 `ssh` 的语法输入虚拟机的用户名和地址（例：ssh -p 2222 cs144@localhost），然后按提示操作即可（如果要求选择平台，选`Linux`；如果询问是否信任远程主机，选择信任；最后还需要**输入一次密码**）。如果你使用的是我们提供的镜像并按教程配置，虚拟机的用户名/地址应该填写 `cs144@localhost`。

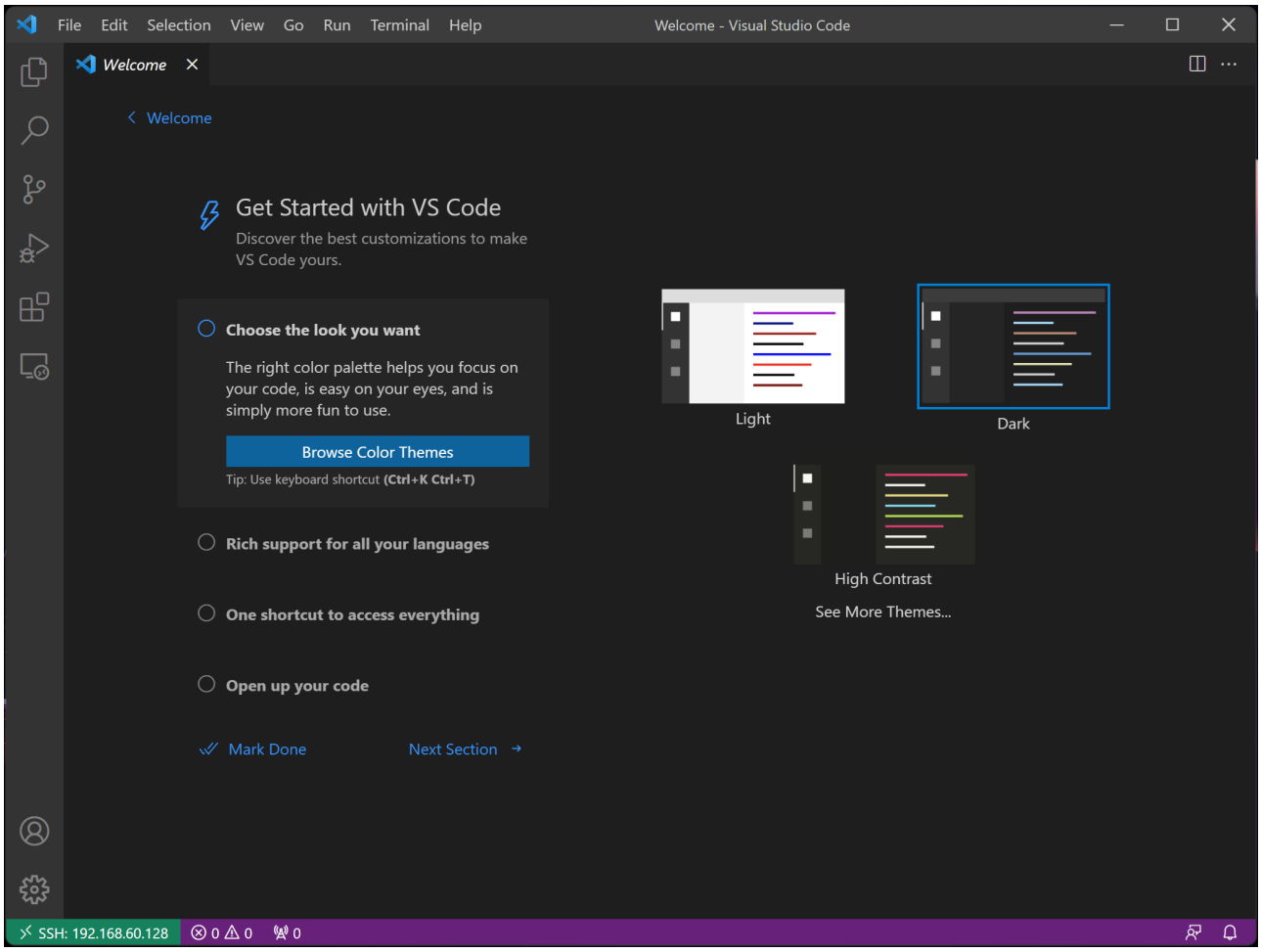




如果还是不行，大家点击VSC 的左下角出现绿色的 `Open a remote window` 按钮，选中并点击`open configuration file`,选中并点击第一行文件路径，会打开一个config文件，清空config文件的内容，然

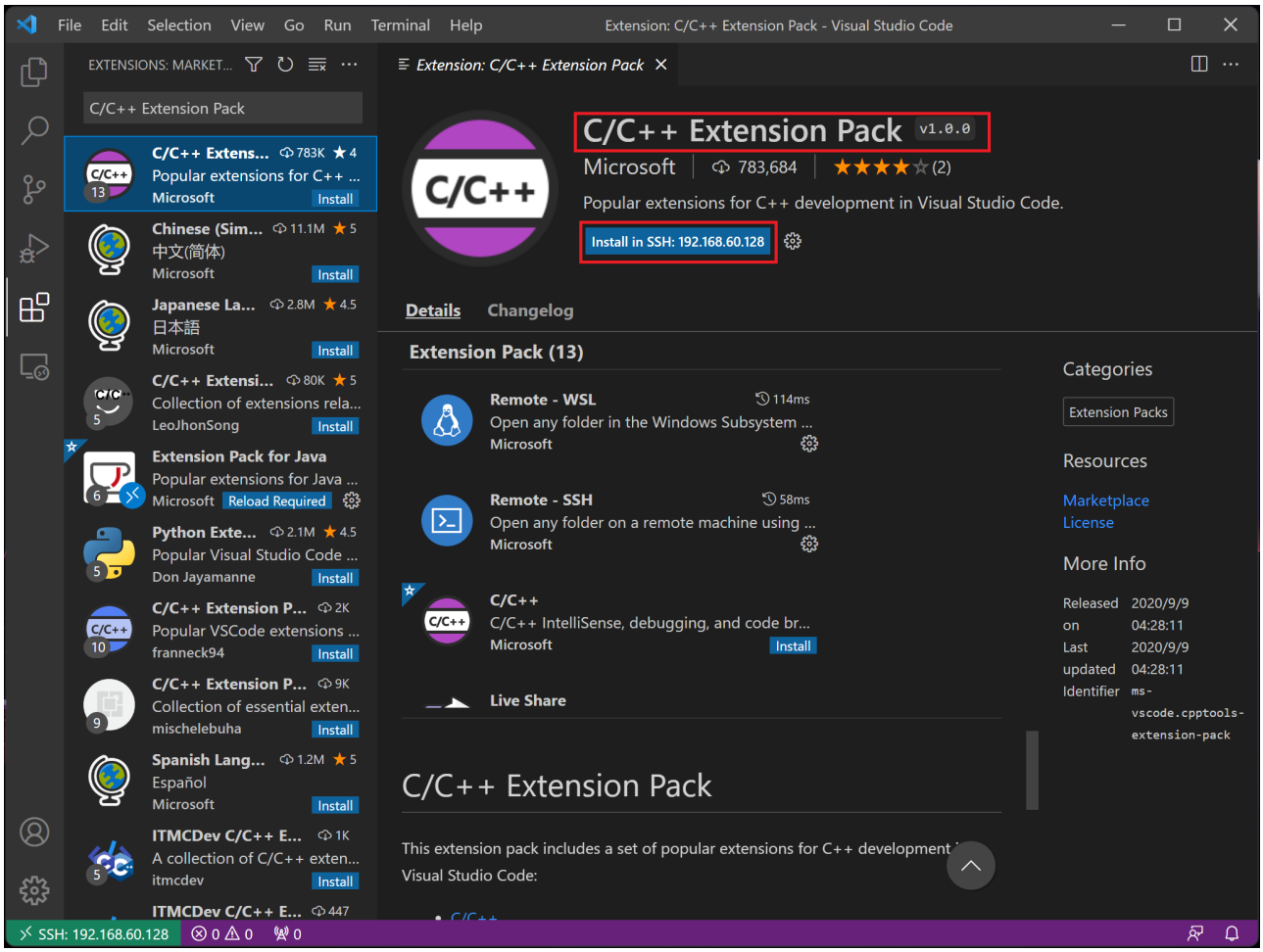
后重新执行第四步（在确保虚拟机打开，ssh服务打开，22端口开放的基础上，理论上这样子就可以了，如果还是不行...联系助教）

5. 如果一切顺利，`VSC` 会在**新**窗口里建立起到实验环境的连接，此时左下角的绿色按钮上会显示类似 `SSH: 192.168.x.x` 或是 `WSL: Ubuntu-xx.xx` 的内容。

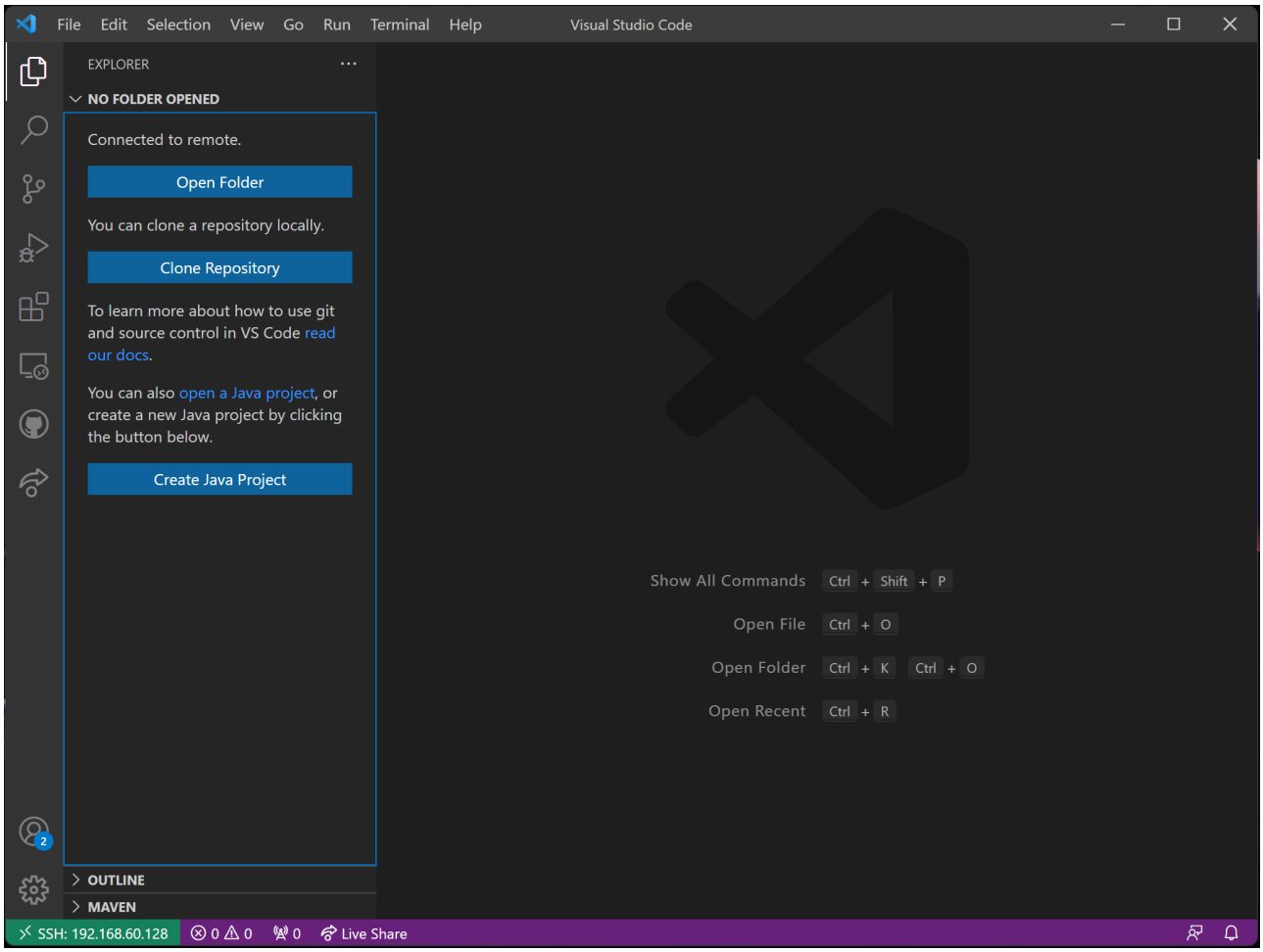


6. 转到**新**窗口里，我们需要再次打开扩展面板（点击左侧第五个图标，或者按快捷键 `Ctrl+Shift+X` ），并在**实验环境中**安装更多扩展。这里推荐安装 `C/C++ Extension Pack` **这一扩展包**（当然你也可以自己选择其他你熟悉的拓展），它包括了本课程中可能会用到的几个拓展：

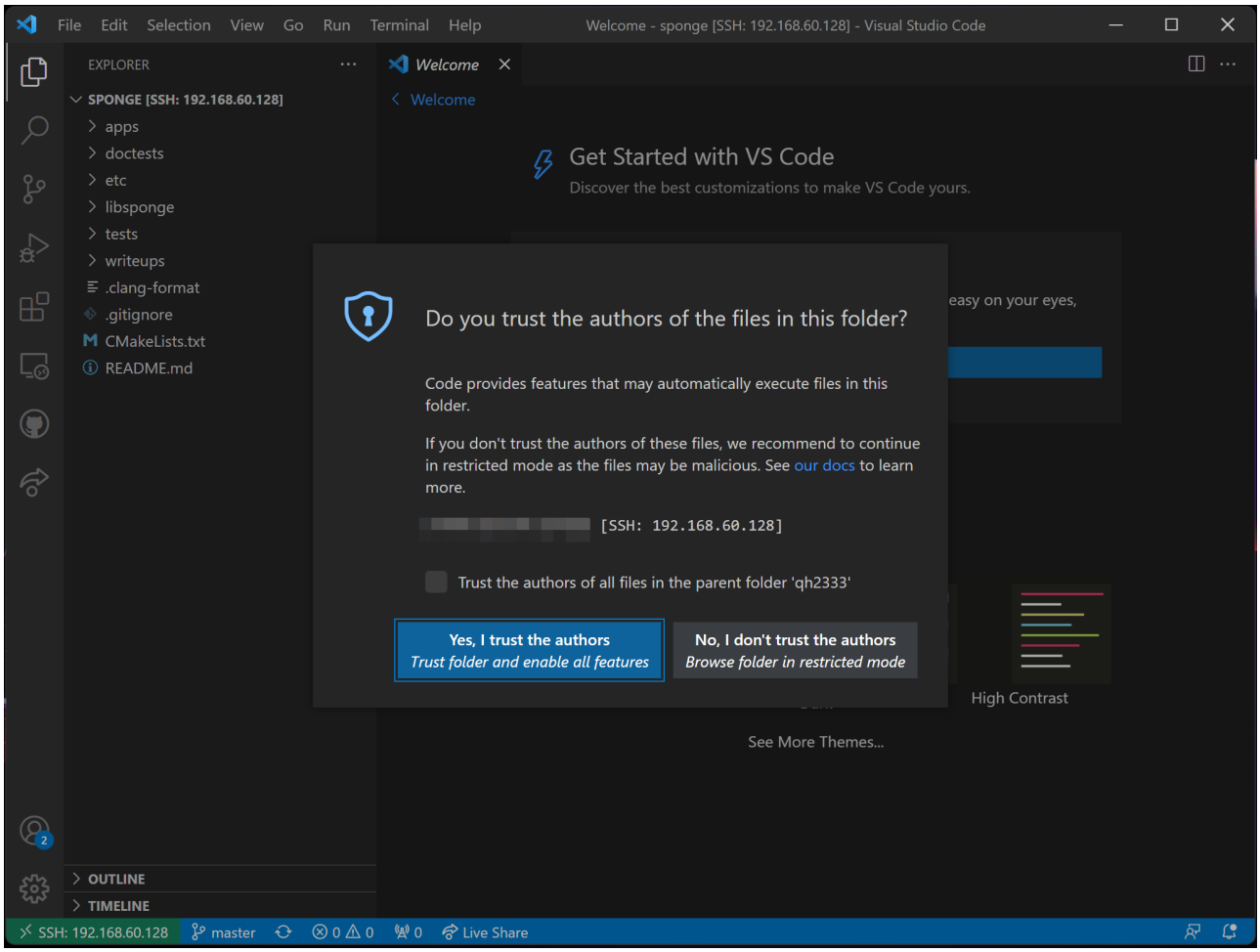
   - **C/C++：** C/C++的语法高亮、调试功能
   - **CMake：** CMake的语法高亮
   - **CMake Tools：** CMake工具，可以让你直接在VSC里配置CMake项目，还能方便地编译和调试



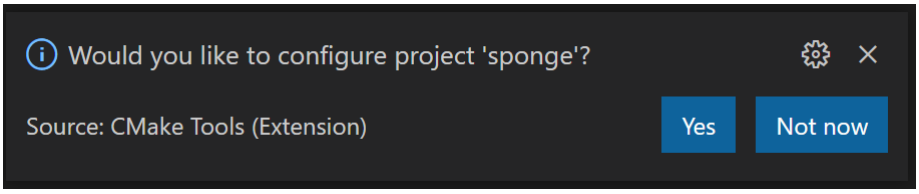
7. 假设你之前已经把实验文件夹下载到了远程环境中，那这时候你就可以打开它了。点击左侧第一个图标或按快捷键 `Ctrl+Shift+E` 打开 `Explorer` 面板，点击 `Open Folder`，找到实验文件夹并打开。



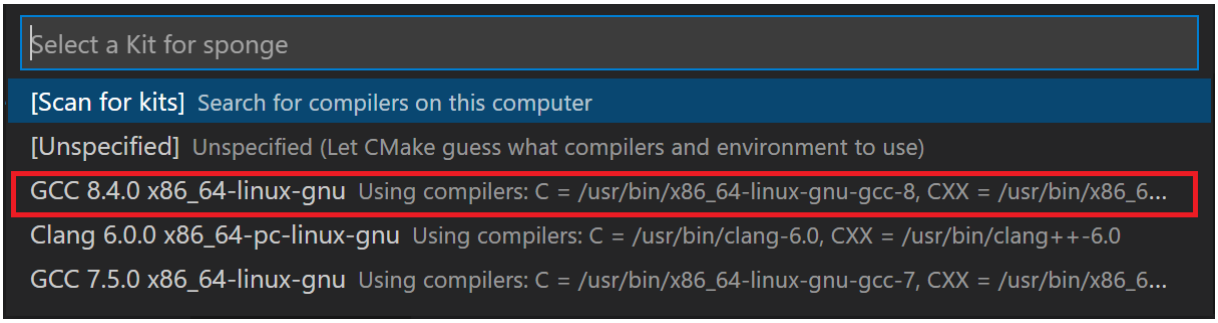
8. 打开文件夹后， VSC可能会询问你是否信任文件夹，这里选择 Yes。



9. 在右下角， `CMake Tools` 会询问是否需要配置项目。这里可以选择 `Yes` 以跳过手动生成配置的几条指令。

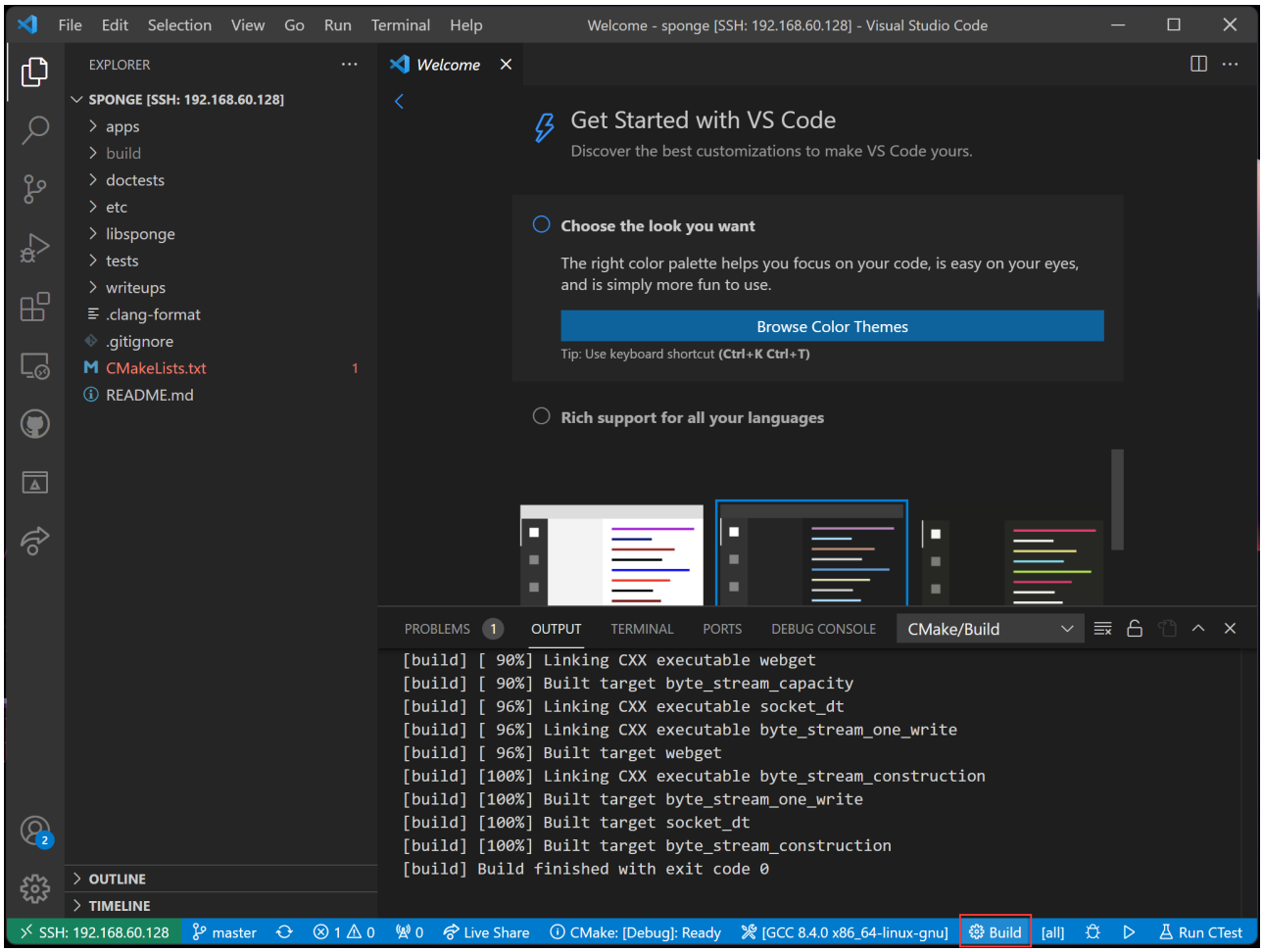


编译器建议选择 `GCC8` 或者更高版本。



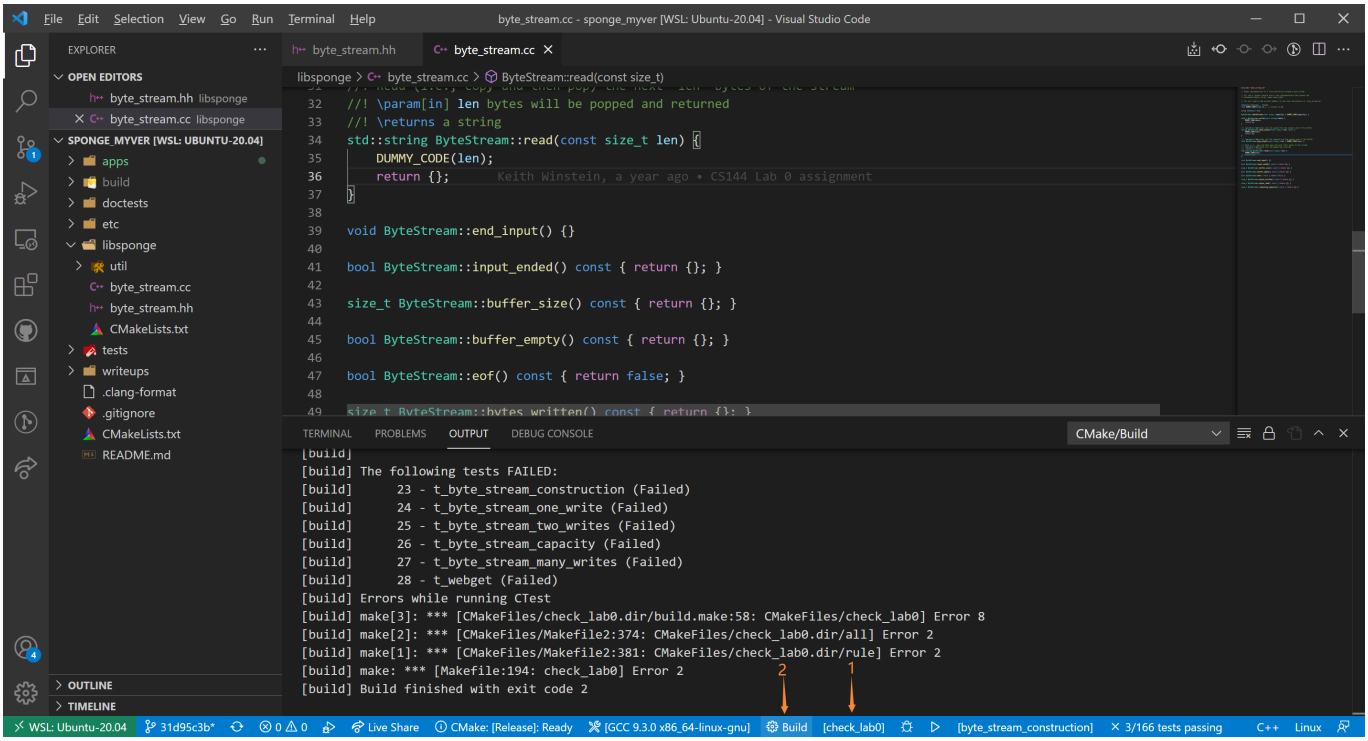
注意，从这一步开始，图形界面中用 `CMake Tools` 完成的操作与实验说明中用命令行 `cmake` 或 `make` 完成的操作完全等价，可以互相代替。

11. 配置完成后，点击下侧的 `Build` 按钮就可以开始编译项目。如果编译能够成功（输出`Build finished with exit code 0`），说明目前为止的配置都是正确的。



测试和调试

本实验的测试基于 `CTest`， `CMake Tools` 插件提供了一次性执行**所有**测试的功能，但为了区分每次实验用到的测试用例，我们额外定义了几个专用的 `target` （构建对象），按 `check_labx` 的规律命名。每当完成一部分的功能，你都可以进行一次测试来验证功能的正确与否：首先在箭头1处选择你要执行的测试所对应的 `target`，然后点击箭头2处的 `Build` 按钮。

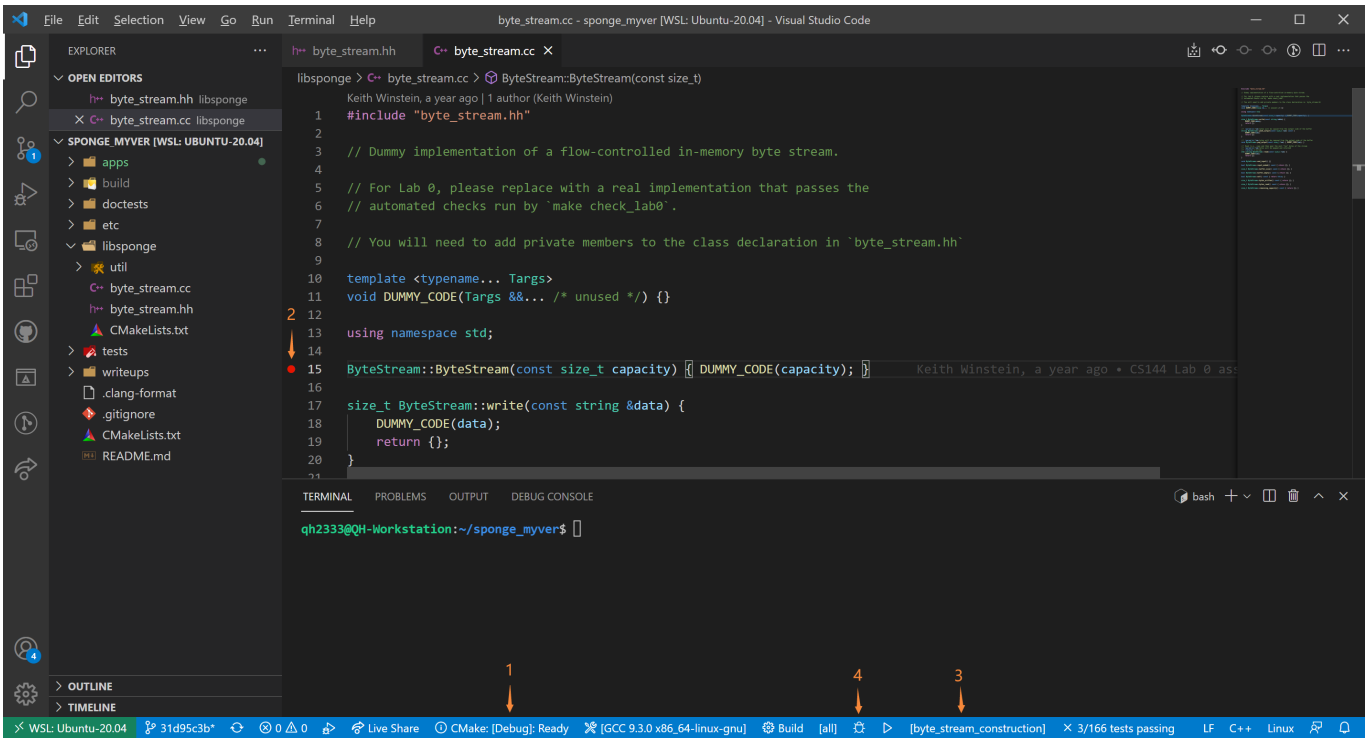


上面的截图中由于并没有编写任何功能代码，所以所有测试都没有通过。如果你的功能正确，你应该可以看到如下输出：

```
[build]     Start 26: t_byte_stream_construction
[build] 1/9 Test #26: t_byte_stream_construction .......   Passed    0.00 sec
[build]     Start 27: t_byte_stream_one_write
[build] 2/9 Test #27: t_byte_stream_one_write ..........   Passed    0.00 sec
[build]     Start 28: t_byte_stream_two_writes
[build] 3/9 Test #28: t_byte_stream_two_writes .........   Passed    0.00 sec
[build]     Start 29: t_byte_stream_capacity
[build] 4/9 Test #29: t_byte_stream_capacity ...........   Passed    0.28 sec
[build]     Start 30: t_byte_stream_many_writes
[build] 5/9 Test #30: t_byte_stream_many_writes ........   Passed    0.00 sec
[build]     Start 31: t_webget
[build] 6/9 Test #31: t_webget .........................   Passed    1.94 sec
[build]     Start 48: t_address_dt
[build] 7/9 Test #48: t_address_dt .....................   Passed    0.21 sec
[build]     Start 49: t_parser_dt
[build] 8/9 Test #49: t_parser_dt ......................   Passed    0.00 sec
[build]     Start 50: t_socket_dt
[build] 9/9 Test #50: t_socket_dt ......................   Passed    0.00 sec
[build]
[build] 100% tests passed, 0 tests failed out of 9
[build]
[build] Total Test time (real) =   2.46 sec
[build] Built target check_lab0
[build] Build finished with exit code 0
```

当功能不正确时，调试器能帮助同学们（相对）快速地定位代码中的 bug。`CMake Tools` 插件可以调用 `Linux` 环境下常见的调试器 `gdb`，允许在图形界面里用类似 IDE 的方式调试代码。

1. 在调试前首先需要确保箭头1处的配置为 Debug，否则可能出现很多奇怪的问题。如果这里修改了配置，那需要将构建对象改为 `all` 以后重新 `Build`。
2. 假设同学们在编写 `ByteStream` 类的代码时，在 `t_byte_stream_construction` 这一测试项中失败了，那么可以首先找到怀疑出错的代码行，并在此处打断点，例如箭头2。
3. 在箭头3处选择执行失败的测试项，然后点击箭头4处的 `Debug` 按钮。



如果一切顺利，`VSC` 会进入如下状态。大家可以在左侧的调试面板里监视变量的值、查看调用栈，也可以在右上角进行单步调试。注意，大部分测试用例在执行出错时都会打印已执行的操作，如果能好好利用这一信息再确定断点位置，调试效率也会大幅增加。