010123131

Software Development Practice

Handout #6

<rawat.s@eng.kmutnb.ac.th>

Last Update: 2024-07-22

C/C++ Software Development for Linux Platforms

Expected Learning Outcomes

- The students are expected to be able to:
  - build and automate the process of building Linux programs from source code using the make tool;
  - compile C/C++ source code using command line tools;
  - use VS Code for C/C++ software development using Ubuntu VM or WSL2;
  - develop C/C++ software on a remote machine using the VS Code IDE and VS Code Server;
  - debug C/C++ code during runtime execution.

C/C++ IDE of Choice

- **Option 1)** Using **Geany Code Editor**
  - Support various programming languages (C/C++, Python,...)
  - Installed by default for Raspbian OS / Raspberry Pi SBC
- **Option 2) Microsoft VS Code + C/C++ extension**
  - More professional and popular than Geany
  - Support both local and remote software development
  - Note: The C/C++ extension does not include a C/C++ compiler.

Ref.: - https://www.geany.org/
- https://code.visualstudio.com/docs/languages/cpp

Open Source C/C++ Development Tools

- **Compiler Toolchain:**
  - The GNU Project C/C++ Compiler Collection: `gcc` / `g++`
  - The Clang / LLVM Project: `clang/clang++`
- **Debuggers:**
  - GNU debugger: `gdb`
  - LLVM debugger: `lldb`
- **Build Tools:**
  - GNU Make / Makefile (https://www.gnu.org/software/make/)
  - CMake (https://cmake.org/)

Practical Activities (1)

- Build and install Geany code editor and its plug-ins from source code, targeting the Linux Ubuntu platform.
  - Learn to write a Bash script to automate the build and installation process.
- Use the Geany editor running on Linux Desktop
  - Edit and compile C/C++ code.
  - Run or debug the compiled binary file.
  - Set / unset breakpoints or watch values of variables.

Practical Activities (2)

- Install Microsoft Visual Code for C/C++ developments on a Windows machine, including some extensions such as
  - C/C++ Extension Pack
    - C/C++ language support, IntelliSense and debugging
    - CMake support
  - Remote Development Extension Pack

Ref.: - https://code.visualstudio.com/docs/languages/cpp
- https://code.visualstudio.com/docs/cpp/cpp-debug

Practical Activities (3)

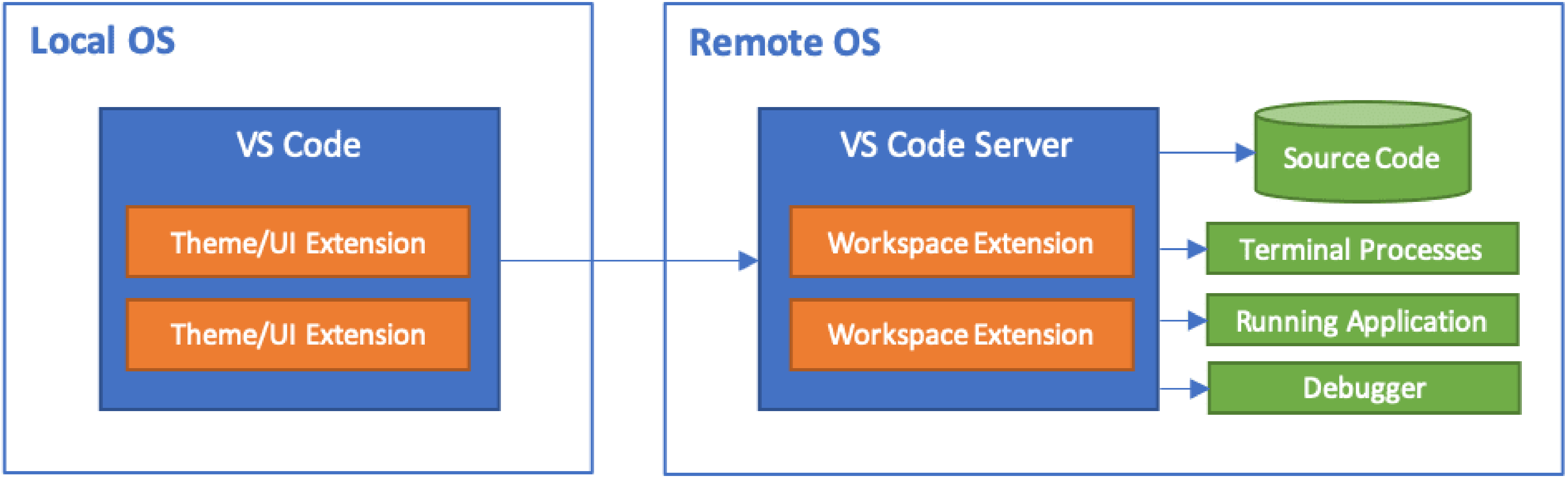
- Use the VS Code IDE on a local machine to access the VS Code Server on a remote headless Linux machine via SSH to build a C/C++ project.
  - Edit and compile source code.
  - Build and debug the project’s program.

VS Code Remote Development

- This **VS Code extension pack** includes three extensions:
  - **Remote – SSH**: Work with source code in any location by opening folders on a remote machine / VM using SSH.
  - **Remote – Containers**: Work with a separate toolchain or container-based application by opening any folder mounted into or inside a Docker container.
  - **Remote – WSL**: Work the Windows Subsystem for Linux.

Ref.: - https://code.visualstudio.com/docs/remote/remote-overview

VS Code Remote Development

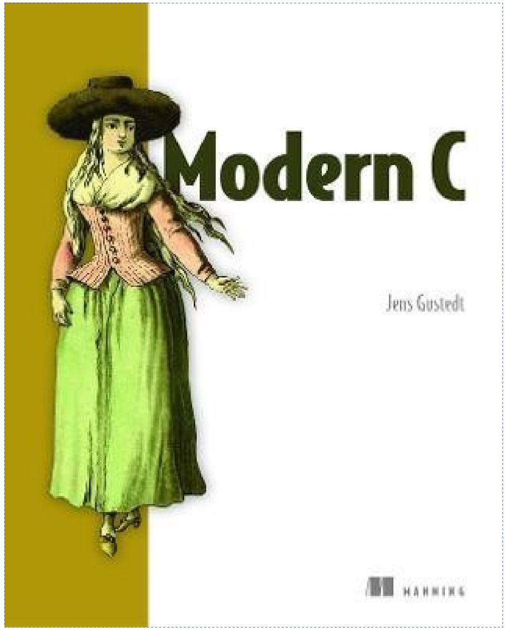


Two options for practical training:

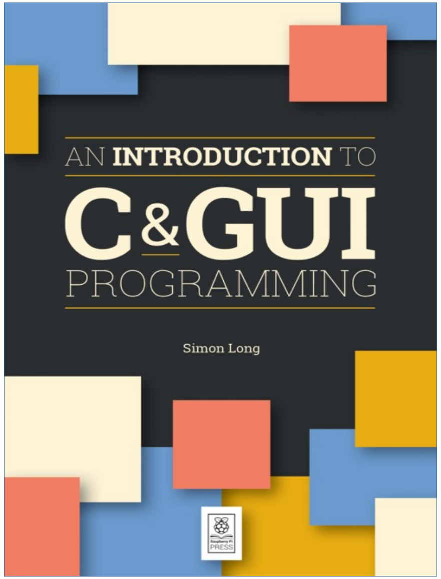
- use **Windows** as a local host and access **Ubuntu VM** as a remote host on the same machine.
- use **Windows** as a local host and access **Ubuntu – WSL2** a remote host on the same machine.

E-Book

- **Modern C** by Jens Gustedt, 2020.
  - Jens Gustedt has released the manuscript of this work under a Creative Commons license for non-commercial use (CC-BY-NC).
- Publisher: Manning Publications Co.
- URLs:
  - https://archive.org/details/modern-c
  - https://archive.org/download/modern-c/Modern%20C.pdf



E-Book



- **An Introduction to C & GUI Programming** by Simon Long, 2019.
  - This book is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)
- Publisher: Raspberry Pi Trading Ltd.
- URLs:
  - https://magpi.raspberrypi.com/books/c-gui-programming
  - https://magpi.raspberrypi.com/books/c-gui-programming/pdf/download

Simple C Demo Code

```
// File: hello.c
// Compile: gcc -std=c99 -g -Wall -o ./hello hello.c
#include <stdio.h>

int main(int argc, char **argv) {
    printf( "Hello World!\n" );
    return 0;
}
```

Compilation options: Invoking the gcc command from a Linux terminal.

`-std=c99`	Use the C99 standard for compilation.
`-Wall`	Show all warnings during the compilation process.
`-g`	Add debug information that can be used by the GDB debugger.
`-o` <output-file>	Save the compiler output in a binary file.

Simple C++ Demo Code

```cpp
// File: hello.cpp
// Compile: g++ -std=c++11 -g -Wall -o ./hello hello.cpp
#include <iostream>

int main() {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

The `build-essential` package

- The "build-essential" package for a Ubuntu or Debian-based Linux Distro is a meta-package necessary for compiling software written in C/C++.
- It includes the GNU compiler collection, debugger, and other development libraries and tools required for compiling software.
- The command installs a lot of packages, including `gcc`, `g++`, `libc`, `make`, etc.

What is Clang ?

- The Clang project (https://clang.llvm.org/) provides an open-source language front-end for the LLVM compiler providing a tooling infrastructure for languages in the C language family (such as C/C++, Objective C/C++, OpenCL, CUDA, ...)
- Its goal is to offer a replacement to the GCC.
- Clang implements all of the ISO C++ 1998, 11 and 14 standards and also provides most of the support of C++17.
- Clang version 14 is the latest major version of Clang as of March 2022.

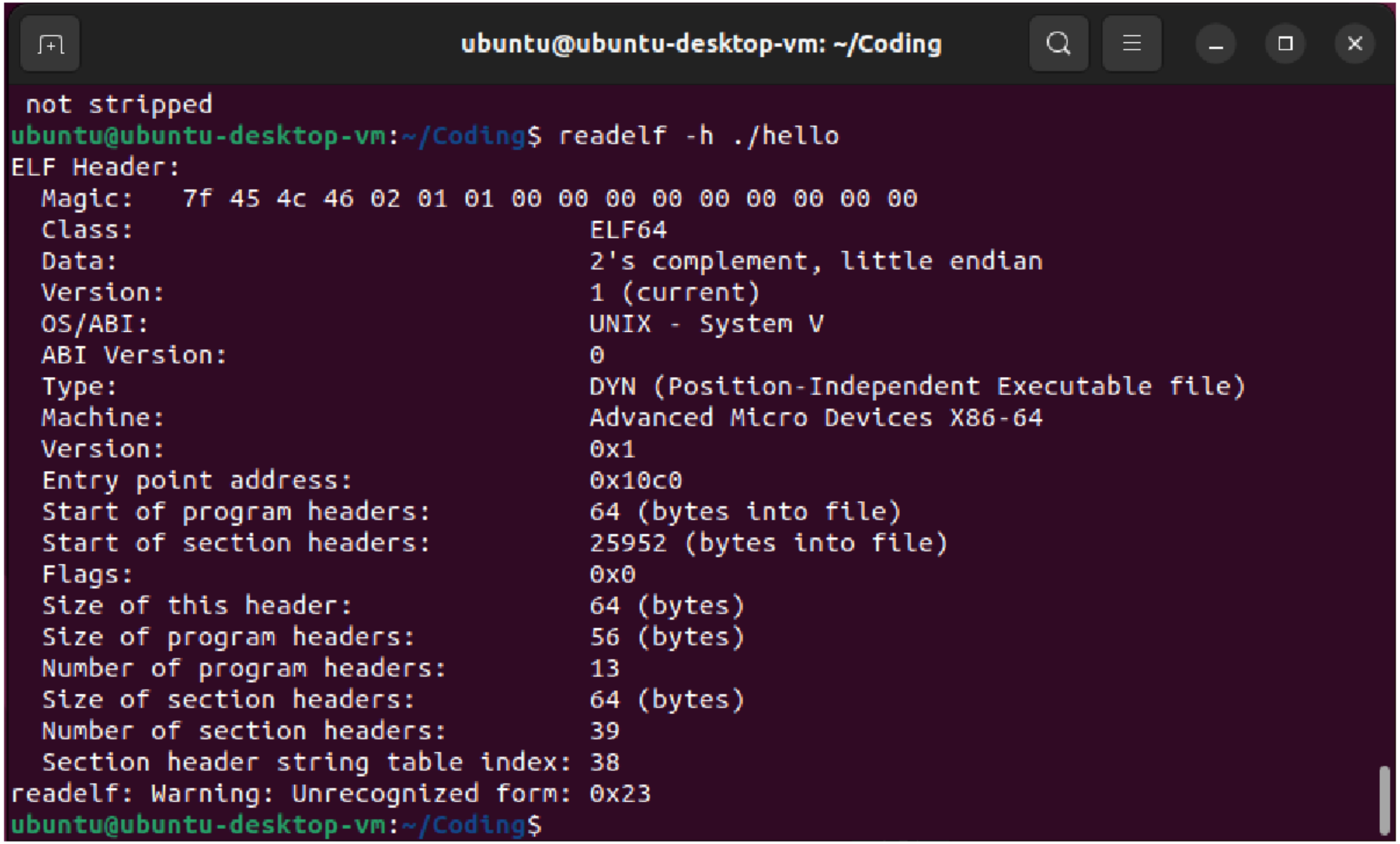
Installation of Clang

```
# install the 'clang' package
$ sudo apt install clang -y

# check the version of clang
$ clang --version | head -n 1
Ubuntu clang version 14.0.0-1ubuntu1

# compile the C++ source code file with clang
$ clang++ -std=c++11 -g -Wall hello.cpp -o hello
```

Check the output file

```
$ file ./hello | tr ',' '\n'

./hello: ELF 64-bit LSB pie executable
 x86-64
 version 1 (SYSV)
 dynamically linked
 interpreter /lib64/ld-linux-x86-64.so.2
 BuildID[sha1]=b2d401c737fb579919481a9281e2991085d256f7
 for GNU/Linux 3.2.0
 with debug_info
 not stripped
```

To display the ELF file header



```
 not stripped
ubuntu@ubuntu-desktop-vm:~/Coding$ readelf -h ./hello
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x10c0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          25952 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         39
  Section header string table index: 38
readelf: Warning: Unrecognized form: 0x23
ubuntu@ubuntu-desktop-vm:~/Coding$
```

Geany Code Editor

- Geany (https://www.geany.org/) is a **small and lightweight** integrated development environment (IDE).
- It was developed to provide a small and fast IDE, which has only a few dependencies from other packages.
- It is using only the GTK3 toolkit, which therefore requires only the GTK3 runtime libraries to run Geany.

Geany Code Editor

- [Manual Steps] Build and install **Geany code editor** and the `geany-plugin-debugger` on Ubuntu 22.04.

```
# remove pre-installed or existing Geany packages.
$ sudo apt autoremove geany-common geany-plugins-common

# install necessary packages to build Geany’s source code:
$ sudo apt-get install -y build-essential gdb \
  libgtk-3-dev autoconf automake autopoint gettext \
  libvte-2.91-dev intltool

# install cppcheck (a static code analysis tool).
$ sudo apt install -y cppcheck
```

Steps to Building the Geany editor

```
# set the version of Geany
$ GEANY_VERSION=1.38
$ ARCHIVE_FILE=geany-${GEANY_VERSION}.tar.gz
# download the archive file of Geany source code (.tar.gz)
$ wget https://download.geany.org/${ARCHIVE_FILE} \
  -O ${ARCHIVE_FILE}
# extract the compressed archive file
$ tar xvfz ${ARCHIVE_FILE}
# change the working directory
$ cd geany-${GEANY_VERSION}/
# configure and build the source code
$ ./configure && make -j $(nproc)
# install the binary file of Geany
$ sudo make install
```

Building Geany’s plug-ins

```
$ GEANY_VERSION=1.38
$ ARCHIVE_FILE=geany-plugins-${GEANY_VERSION}.tar.gz
$ wget https://plugins.geany.org/geany-plugins/${ARCHIVE_FILE} \
  -O ${ARCHIVE_FILE}
$ tar xvfz ${ARCHIVE_FILE}
$ cd ./geany-plugins-${GEANY_VERSION}
$ ./configure --enable-debugger
$ make -j $(nproc) && sudo make install
$ sudo ldconfig -v
```

See: https://iot-kmutnb.github.io/blogs/training/geany_editor/

To run Geany from a command line

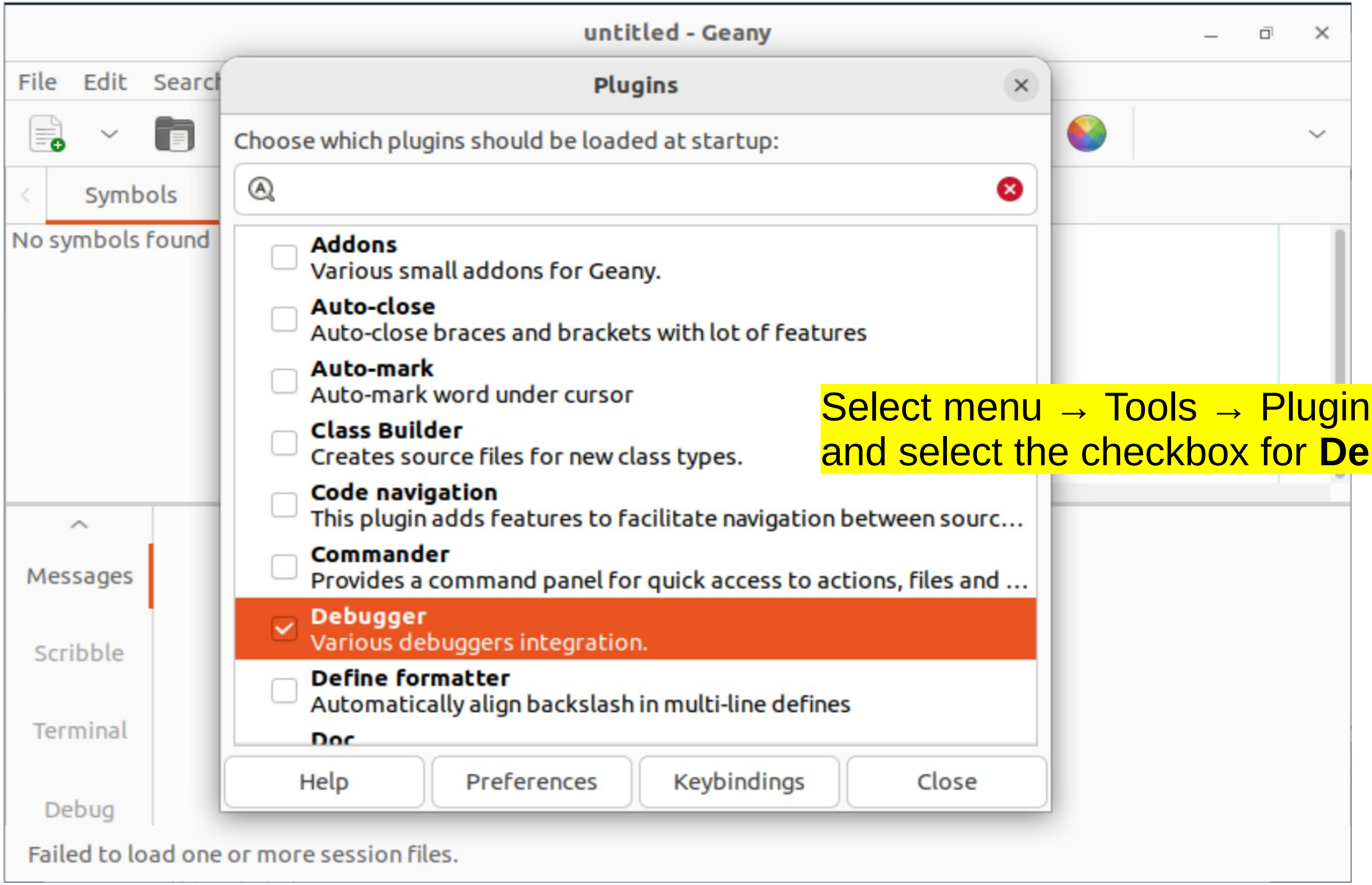
```
# check the version of geany
$ `which geany` --version
geany 1.38 (built on 2022-08-14 with GTK 3.24.33, GLib 2.72.1)

# Run/call the geany program in background mode
$ geany 2> /dev/null &
```

Redirects the output to 'stderr' to '/dev/null'

```
$ geany >/dev/null 2>&1 &
```

Redirects the output to 'stdout' and ‘stderr’ to '/dev/null'

Use the Geany code editor on a Ubuntu Desktop VM (not headless).
Note: If using WSL2 + Ubuntu Desktop, then either GWSL or Remote Desktop is required.



No symbols found

Choose which plugins should be loaded at startup:

- ☐ **Addons**
  Various small addons for Geany.
- ☐ **Auto-close**
  Auto-close braces and brackets with lot of features
- ☐ **Auto-mark**
  Auto-mark word under cursor
- ☐ **Class Builder**
  Creates source files for new class types.
- ☐ **Code navigation**
  This plugin adds features to facilitate navigation between sourc…
- ☐ **Commander**
  Provides a command panel for quick access to actions, files and …
- ☑ **Debugger**
  Various debuggers integration.
- ☐ **Define formatter**
  Automatically align backslash in multi-line defines
- **Doc**

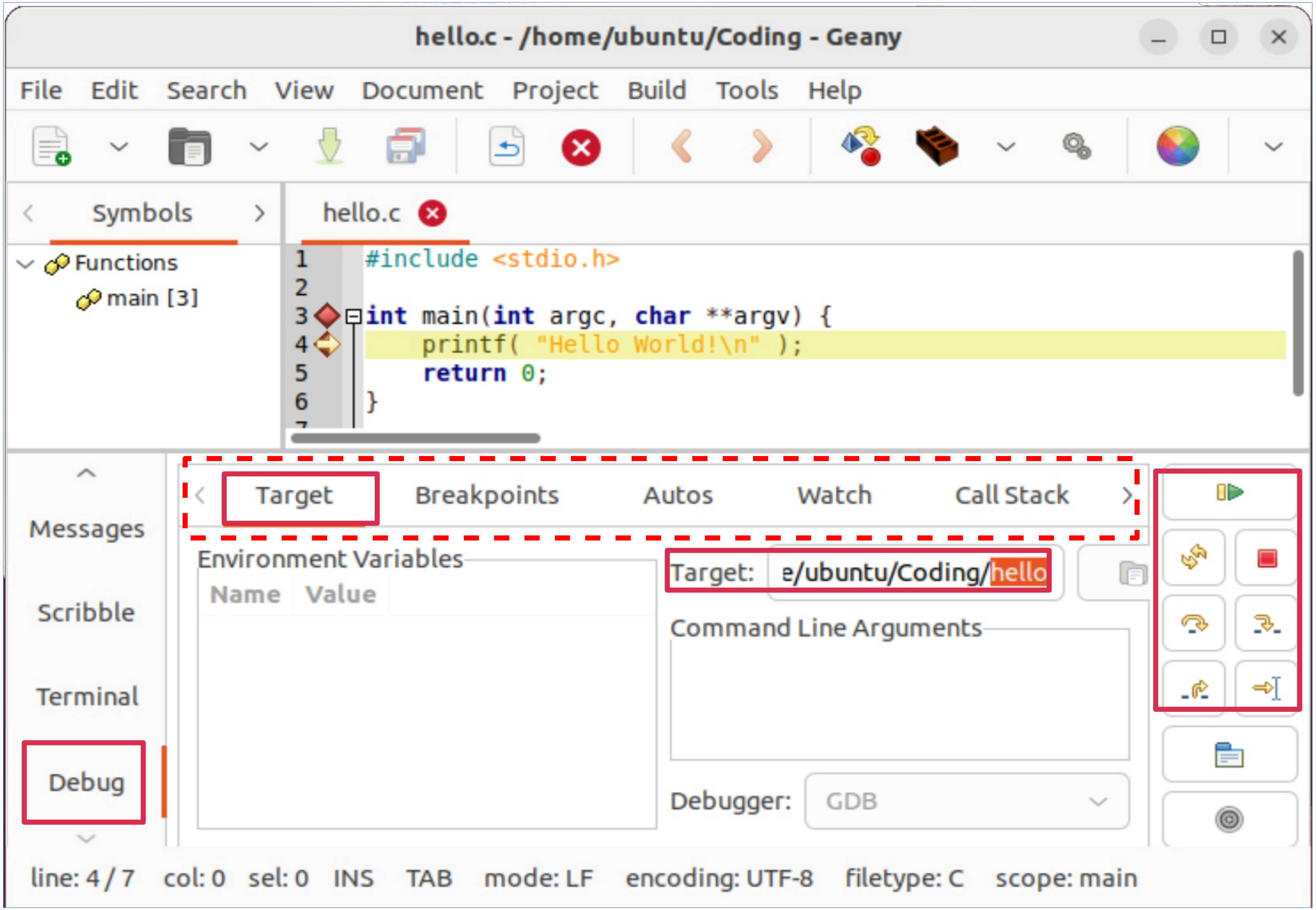
Failed to load one or more session files.

Select menu → Tools → Plugin Manager and select the checkbox for **Debugger**.

Build Command Settings

In the menu, select **Build → Build Commands (for C code)**

- Compile command: `gcc -g -Wall -c "%f"`
- Build command: `gcc -g -Wall -o "%e" "%f"`
- Lint command: `cppcheck --language=c --enable=warning,style --template=gcc "%f"`



```
#include <stdio.h>

int main(int argc, char **argv) {
    printf( "Hello World!\n" );
    return 0;
}
```

Target: e/ubuntu/Coding/hello

Debugger: GDB

line: 4 / 7 col: 0 sel: 0 INS TAB mode: LF encoding: UTF-8 filetype: C scope: main

Verilator – Verilog Simulator

- Verilator is a free and open-source software tool which converts Verilog (a hardware description language) to a cycle-accurate behavioral model in C++.
- It outputs single- or multi-threaded .cpp and .h files, the "Verilated" code.
- The Verilated C++ files are then compiled by a C++ compiler.

https://www.veripool.org/verilator/

Installation of Verilator

Option 1)

```
# Install verilator and gtkwave on Ubuntu.
$ sudo apt install verilator gtkwave
# Show the version of verilator installed locally.
$ verilator --version
Verilator 4.038 2020-07-11 rev v4.036-114-g0cd4a57ad
```

Option 2)

```
# Run verilator in a Docker container (for Ubuntu).
$ docker run -it verilator/verilator:latest --version
Verilator 4.211 devel rev v4.210-59-g3ec3c2c2
```

```
$ docker run -it -v ${PWD}:/work \
  --user $(id -u):$(id -g) verilator/verilator:latest \
    -Wall --cc counter.v --exe --trace
```

Installation of GTKWave

```
# Install verilator and gtkwave on Ubuntu.
$ sudo apt install gtkwave \
  libcanberra-gtk-module libcanberra-gtk3-module
```

Verilog Code Demo:

counter.v

```
module counter #(
    parameter NUM_LEDS  = 6, // set the number of LEDs
    parameter BIT_WIDTH = 8  // set the bit width of the register
) (
    input wire clk,    // Clock input
    input wire nrst,   // Active-low reset input
    input wire en,     // Clock enable input
    output wire [NUM_LEDS-1:0] leds // LED array output
);


  localparam CNT_MSB = BIT_WIDTH-1;
  reg [BIT_WIDTH-1:0] cnt_reg;


  always @(posedge clk or negedge nrst)
  begin
    if (!nrst)
      cnt_reg <= 0;
    else if (en)
      cnt_reg <= cnt_reg + 1;
  end
  assign leds = cnt_reg[CNT_MSB:CNT_MSB-(NUM_LEDS-1)];
endmodule
```

C++ Testbench Example

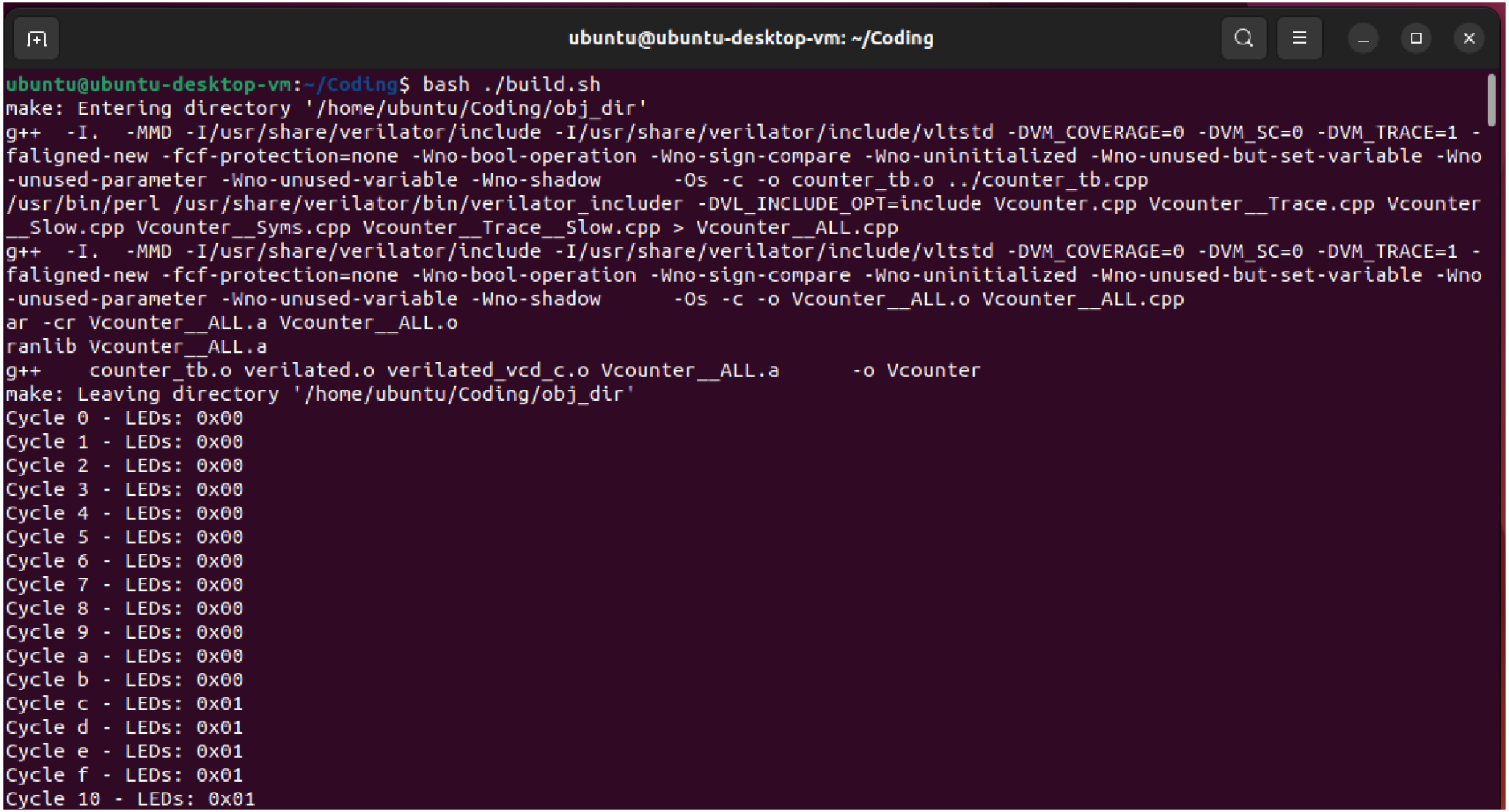
counter_tb.cpp

```cpp
#include <iostream>
#include <iomanip> // Include for std::hex manipulator
#include <verilated.h>
#include "Vcounter.h"
#include <verilated_vcd_c.h> // Include the VCD header

int main(int argc, char** argv) {
    Verilated::commandArgs(argc, argv);
    // Create an instance of the module
    Vcounter* top = new Vcounter;
    // Create a VCD trace
    Verilated::traceEverOn(true);
    VerilatedVcdC* vcdTrace = new VerilatedVcdC;
    top->trace(vcdTrace, 2); // 2 is the trace level
    vcdTrace->open("waveform.vcd"); // Open the VCD file
    // Insert the code block on the next page...
    vcdTrace->close(); // Close VCD file
    top->final(); // Clean up
    delete top;
    return 0;
}
```

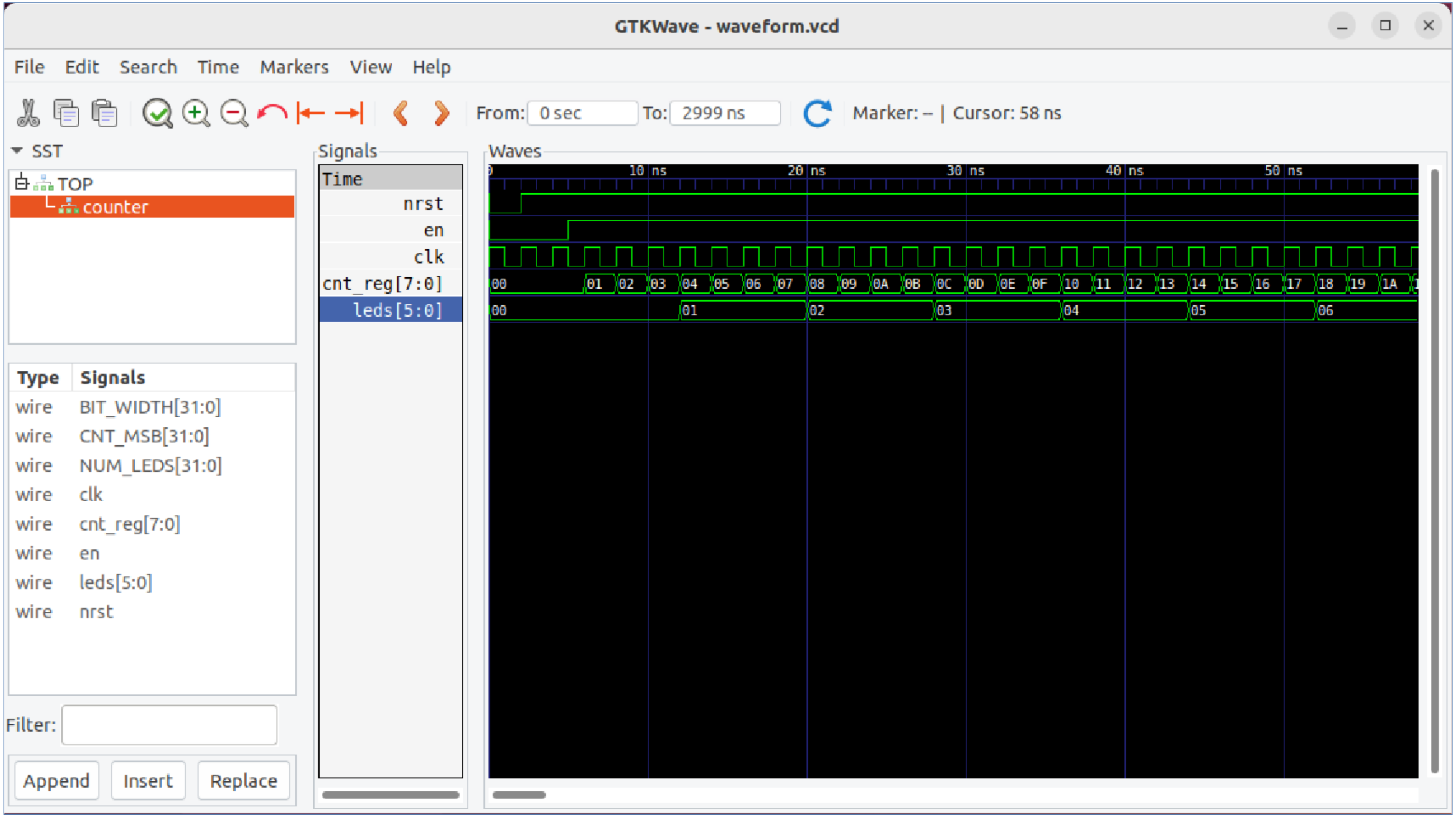
```
// Initialize inputs
top->clk  = 0;
top->nrst = 0;
top->en   = 0;
// Simulate for a 3000 clock cycles
for (int i = 0; i < 3000; ++i) {
    top->clk = !top->clk; // Toggle the clock
    if (i == 2) { top->nrst = 1; }
    if (i == 5) { top->en = 1;   }
    top->eval();
    // Print the LED values in hex string
    std::cout << "Cycle " << i << " - LEDs: 0x"
              << std::hex << std::setw(2)
              << std::setfill('0')
              << (int)top->leds << std::endl;
    vcdTrace->dump(i); // Dump signal values to VCD
    if (Verilated::gotFinish())
        break;
}
```

build.sh

```bash
#!/bin/env bash

# Remove the output object directory and the VCD file.
rm -fr obj_dirs *.vcd
# Compile Verilog source code.
verilator -Wall --cc counter.v --exe --trace
# Compile Verilog source code and C++ testbench.
verilator -Wall --trace -cc counter.v \
  --exe counter_tb.cpp --timescale 1ns/1ns
# Build the executable file for the simulator.
make -C ./obj_dir -f Vcounter.mk Vcounter
# Run the simulator
./obj_dir/Vcounter
```



```
ubuntu@ubuntu-desktop-vm:~/Coding$ bash ./build.sh
make: Entering directory '/home/ubuntu/Coding/obj_dir'
g++  -I.  -MMD -I/usr/share/verilator/include -I/usr/share/verilator/include/vltstd -DVM_COVERAGE=0 -DVM_SC=0 -DVM_TRACE=1 -
faligned-new -fcf-protection=none -Wno-bool-operation -Wno-sign-compare -Wno-uninitialized -Wno-unused-but-set-variable -Wno
-unused-parameter -Wno-unused-variable -Wno-shadow      -Os -c -o counter_tb.o ../counter_tb.cpp
/usr/bin/perl /usr/share/verilator/bin/verilator_includer -DVL_INCLUDE_OPT=include Vcounter.cpp Vcounter__Trace.cpp Vcounter
__Slow.cpp Vcounter__Syms.cpp Vcounter__Trace__Slow.cpp > Vcounter__ALL.cpp
g++  -I.  -MMD -I/usr/share/verilator/include -I/usr/share/verilator/include/vltstd -DVM_COVERAGE=0 -DVM_SC=0 -DVM_TRACE=1 -
faligned-new -fcf-protection=none -Wno-bool-operation -Wno-sign-compare -Wno-uninitialized -Wno-unused-but-set-variable -Wno
-unused-parameter -Wno-unused-variable -Wno-shadow      -Os -c -o Vcounter__ALL.o Vcounter__ALL.cpp
ar -cr Vcounter__ALL.a Vcounter__ALL.o
ranlib Vcounter__ALL.a
g++    counter_tb.o verilated.o verilated_vcd_c.o Vcounter__ALL.a      -o Vcounter
make: Leaving directory '/home/ubuntu/Coding/obj_dir'
Cycle 0 - LEDs: 0x00
Cycle 1 - LEDs: 0x00
Cycle 2 - LEDs: 0x00
Cycle 3 - LEDs: 0x00
Cycle 4 - LEDs: 0x00
Cycle 5 - LEDs: 0x00
Cycle 6 - LEDs: 0x00
Cycle 7 - LEDs: 0x00
Cycle 8 - LEDs: 0x00
Cycle 9 - LEDs: 0x00
Cycle a - LEDs: 0x00
Cycle b - LEDs: 0x00
Cycle c - LEDs: 0x01
Cycle d - LEDs: 0x01
Cycle e - LEDs: 0x01
Cycle f - LEDs: 0x01
Cycle 10 - LEDs: 0x01
```



From: 0 sec To: 2999 ns

Marker: – | Cursor: 58 ns

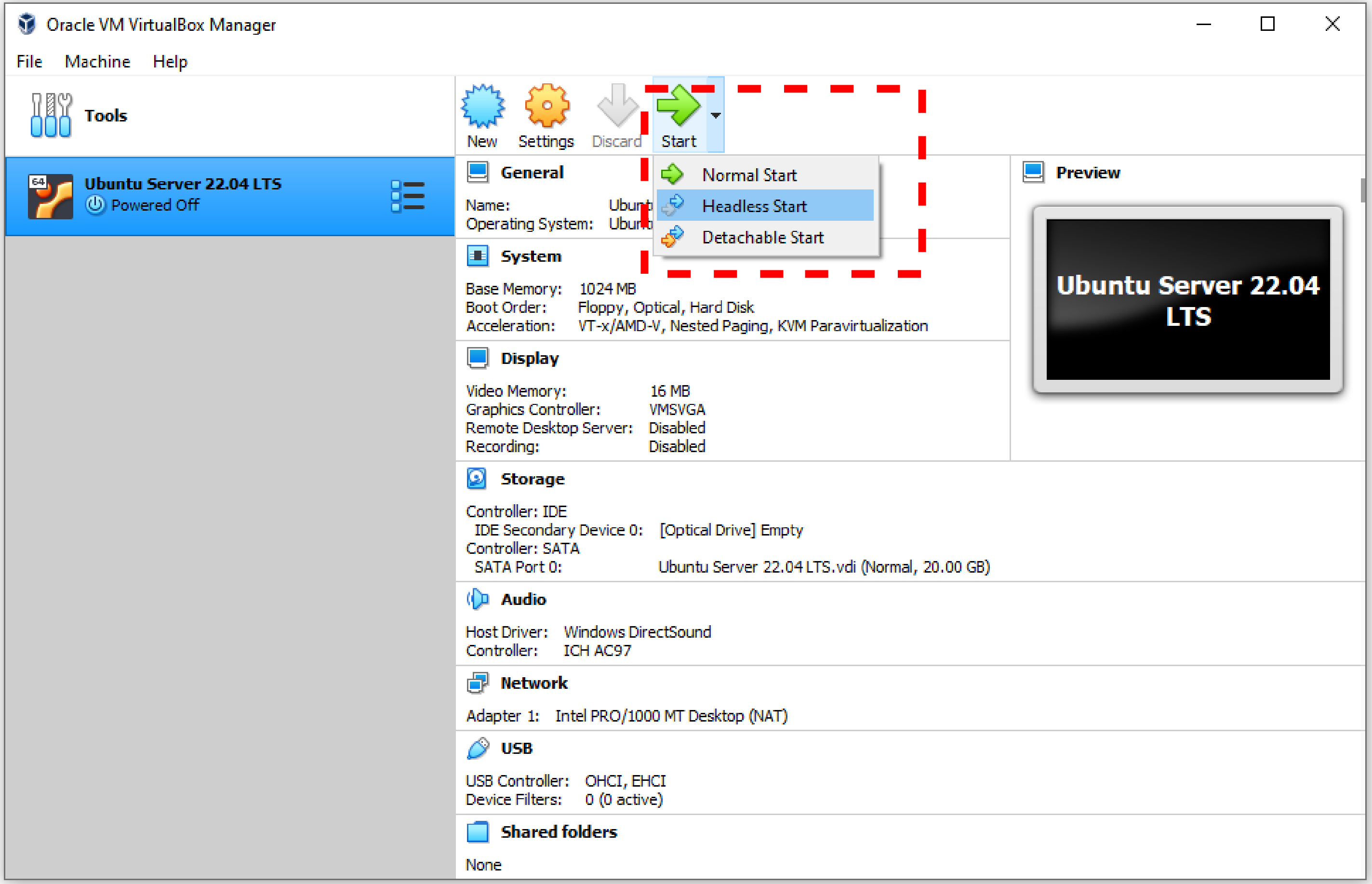
Type	Signals
wire	BIT_WIDTH[31:0]
wire	CNT_MSB[31:0]
wire	NUM_LEDS[31:0]
wire	clk
wire	cnt_reg[7:0]
wire	en
wire	leds[5:0]
wire	nrst

```
$ gtkwave waveform.vcd &
```

Remote Code Development

- Open Oracle VM VirtualBox in Host OS (Windows).
- Run Ubuntu VM in headless mode.
- Enable SSH port forwarding to the Ubuntu VM.
- Install / Open VS Code IDE in Host OS.
- Install VS Code Extension Pack for Remote Development.
- Install C/C++ Extension Pack on the remote VS code server.

Ubuntu VM (Headless)



Ubuntu Server 22.04 LTS
Powered Off

General

Name: Ubunt
Operating System: Ubun

System

Base Memory: 1024 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: Ubuntu Server 22.04 LTS.vdi (Normal, 20.00 GB)

Audio

Host Driver: Windows DirectSound
Controller: ICH AC97

Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT)

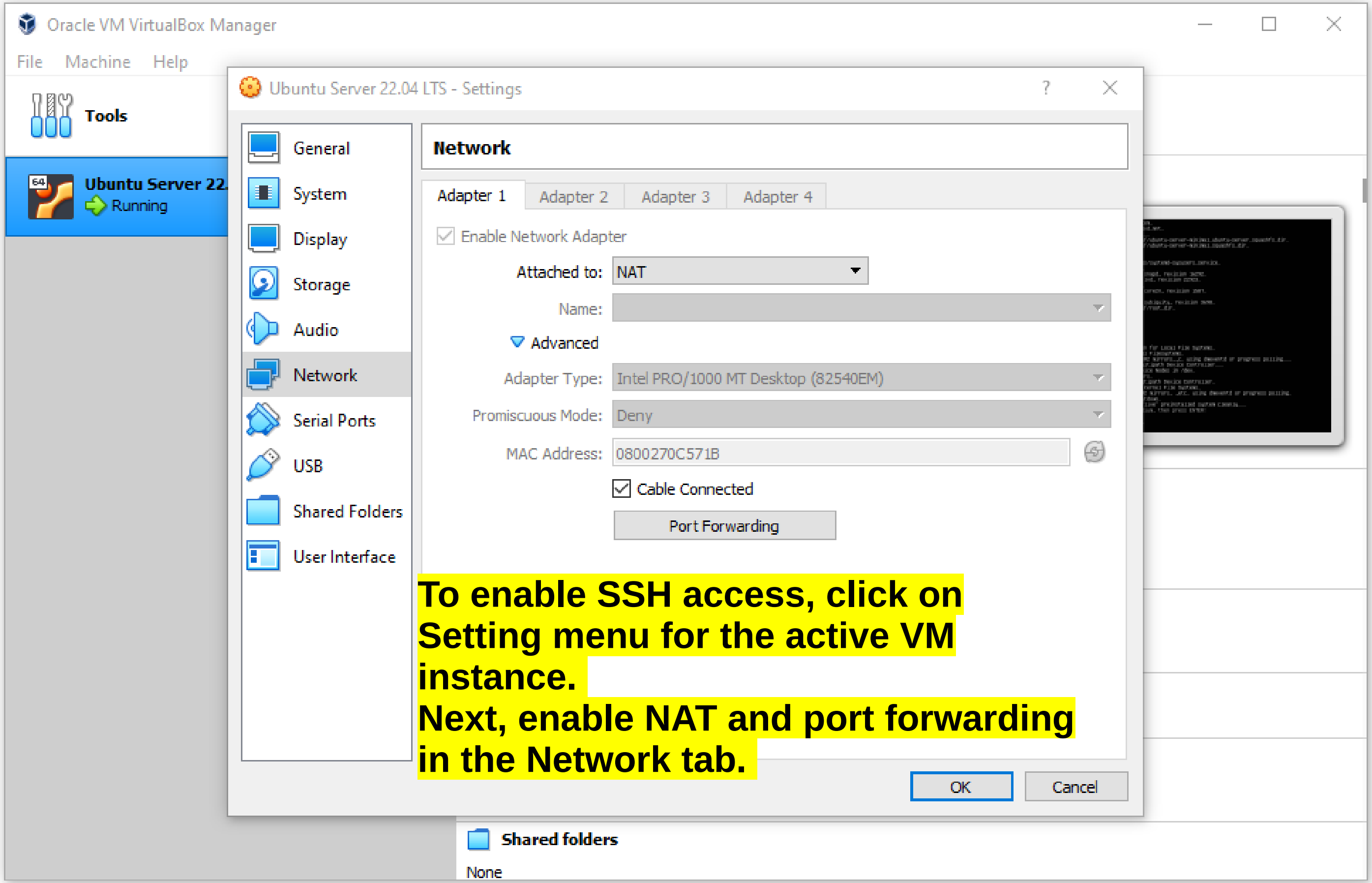
USB

USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)

Shared folders

None

Ubuntu VM (Headless)



Network

☑ Enable Network Adapter

Attached to: NAT

Name:

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

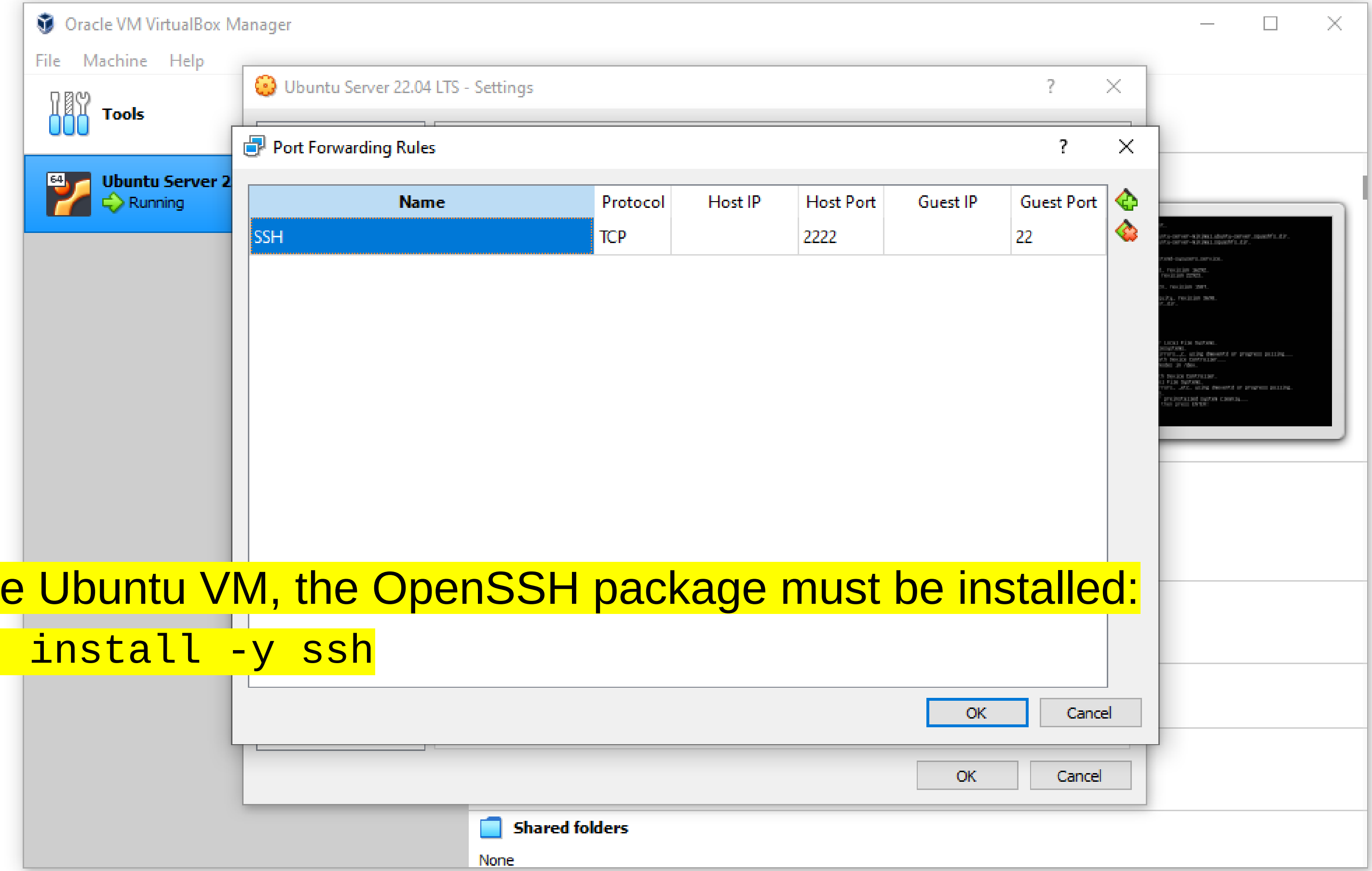
Promiscuous Mode: Deny

MAC Address: 0800270C571B

☑ Cable Connected

To enable SSH access, click on Setting menu for the active VM instance.
Next, enable NAT and port forwarding in the Network tab.

Ubuntu VM (Headless)

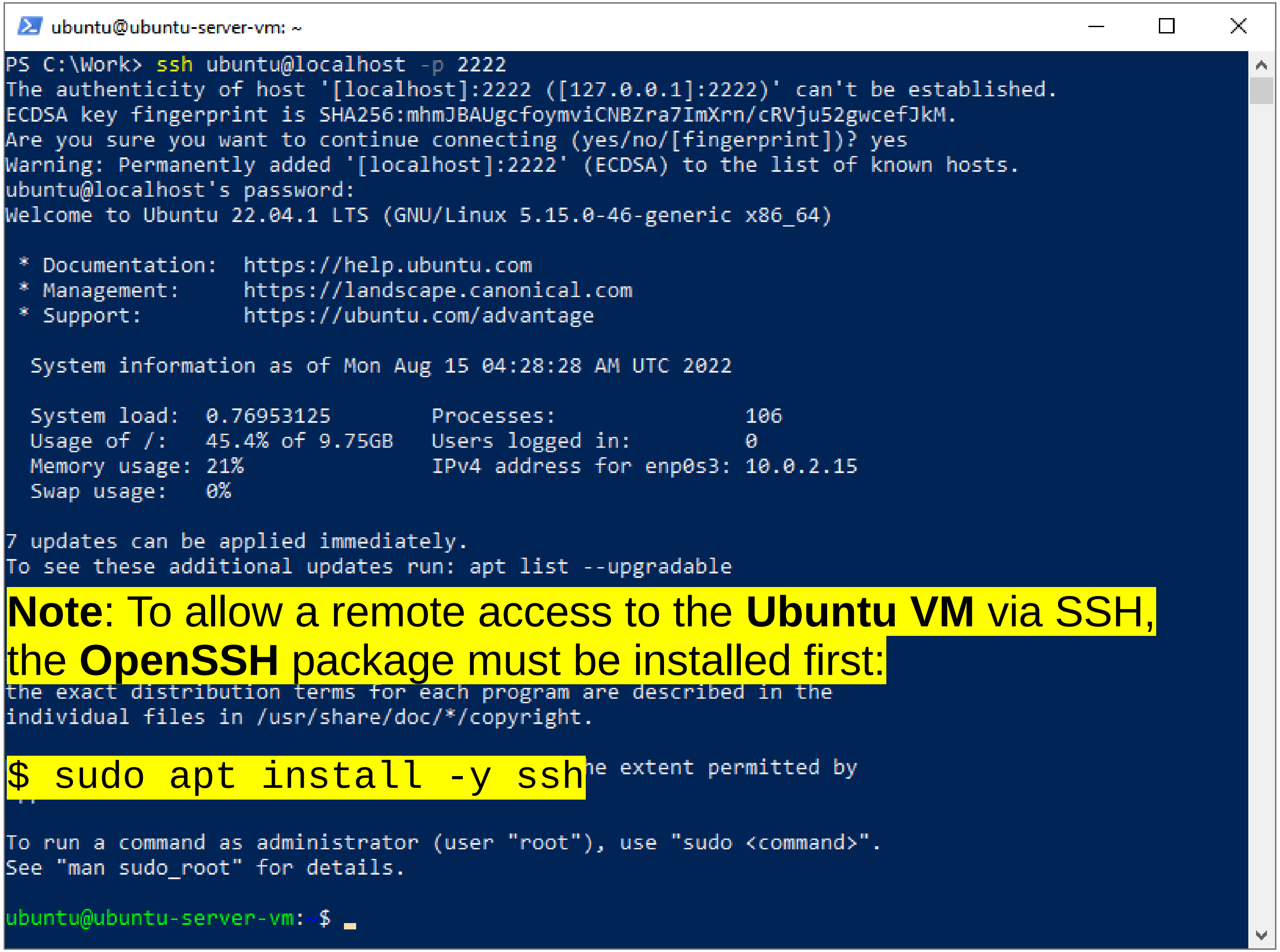


Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
SSH	TCP		2222		22

Note: On the Ubuntu VM, the OpenSSH package must be installed:

```
$ sudo apt install -y ssh
```

Try to use Windows PowerShell to access the Ubuntu VM via SSH.

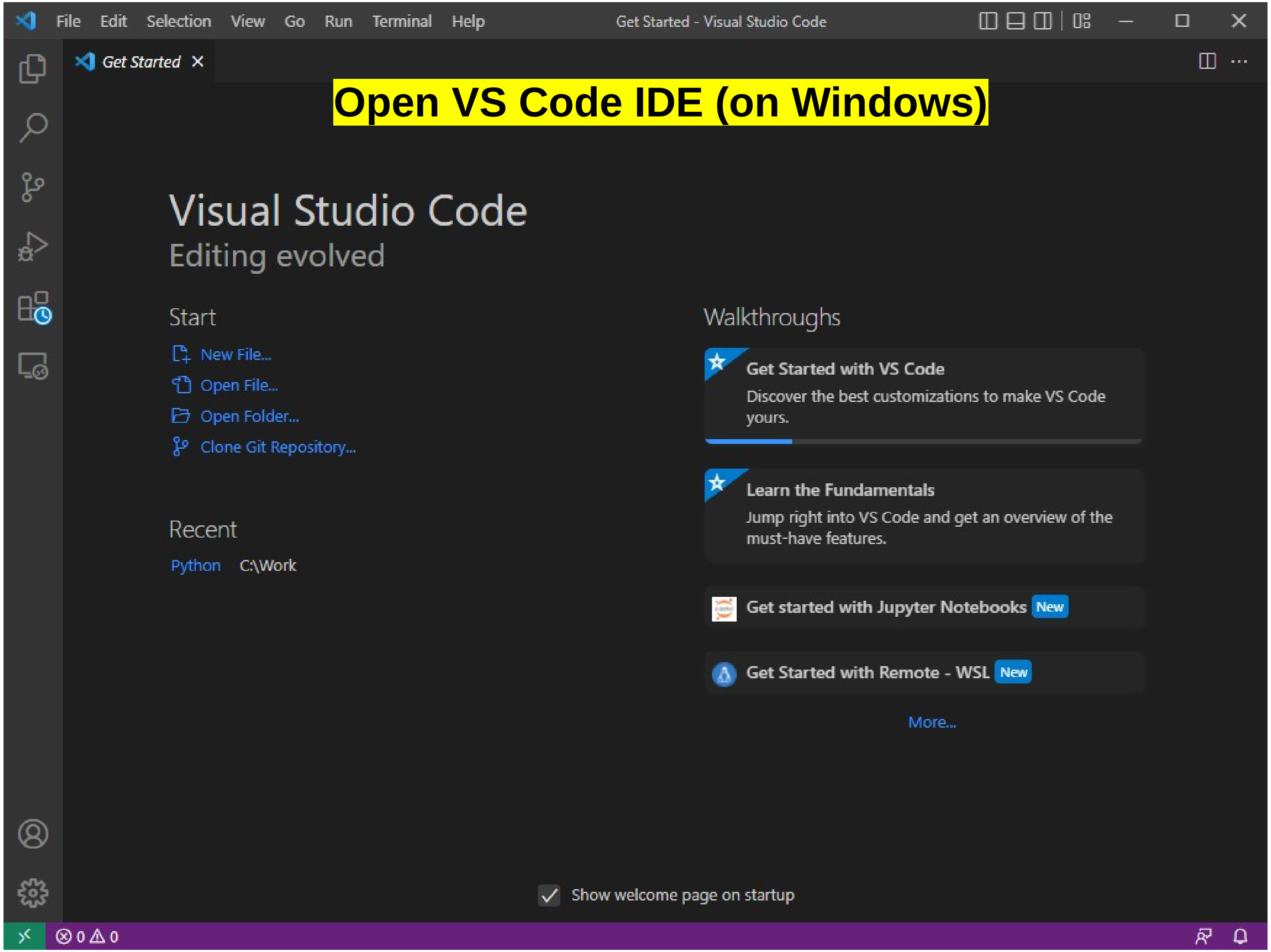


```
PS C:\Work> ssh ubuntu@localhost -p 2222
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
ECDSA key fingerprint is SHA256:mhmJBAUgcfoymviCNBZra7ImXrn/cRVju52gwcefJkM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2222' (ECDSA) to the list of known hosts.
ubuntu@localhost's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-46-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Aug 15 04:28:28 AM UTC 2022

  System load:  0.76953125        Processes:               106
  Usage of /:   45.4% of 9.75GB   Users logged in:         0
  Memory usage: 21%               IPv4 address for enp0s3: 10.0.2.15
  Swap usage:   0%

7 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

he extent permitted by

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ubuntu-server-vm:~$ _
```

Note: To allow a remote access to the **Ubuntu VM** via SSH, the **OpenSSH** package must be installed first:

```
$ sudo apt install -y ssh
```

Open VS Code IDE (on Windows)



Visual Studio Code

Editing evolved

Start

- New File...
- Open File...
- Open Folder...
- Clone Git Repository...

Recent

Python C:\Work

Walkthroughs

Get Started with VS Code
Discover the best customizations to make VS Code yours.

Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.

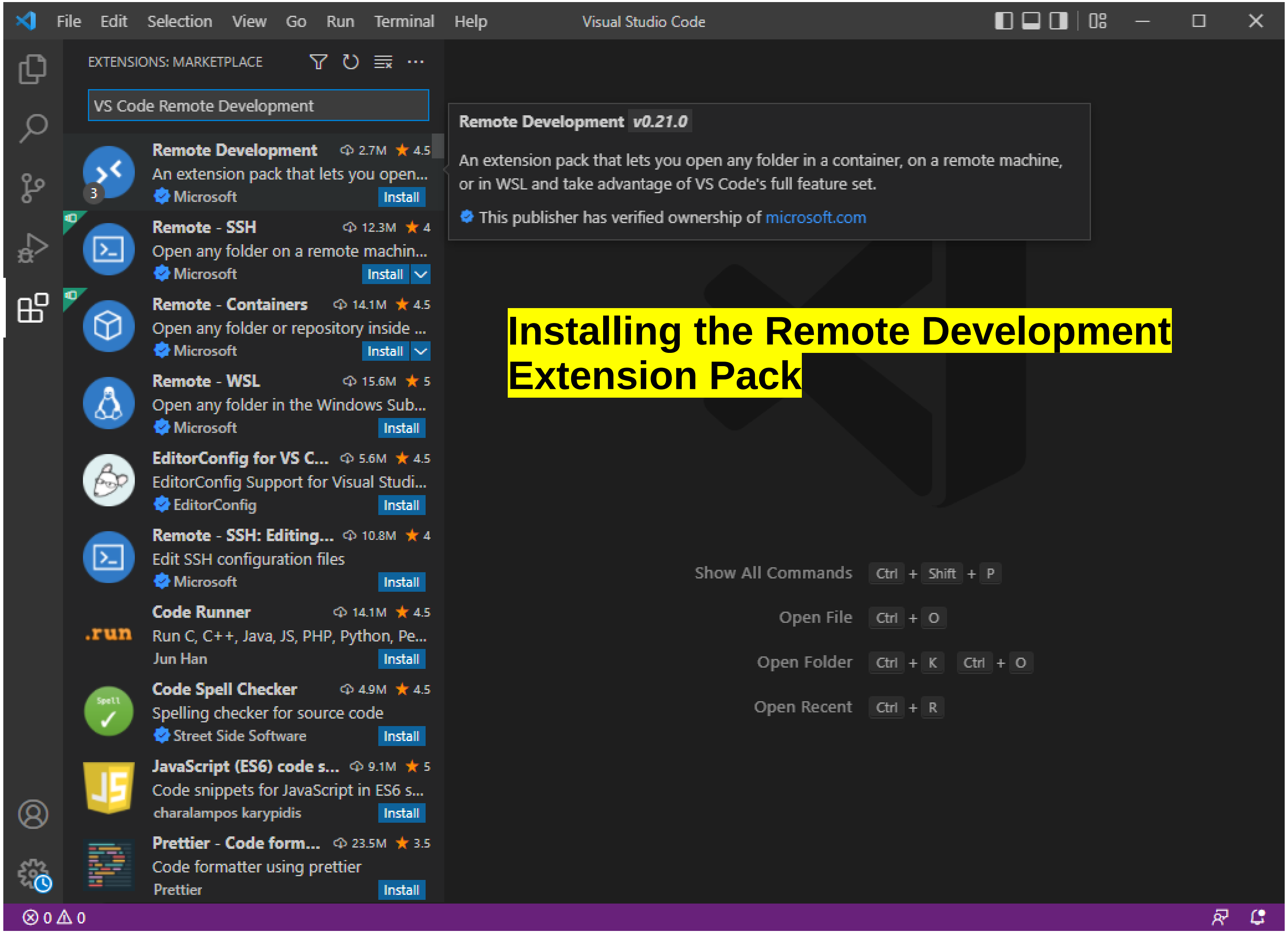
Get started with Jupyter Notebooks New

Get Started with Remote - WSL New

More...

✓ Show welcome page on startup

Installing the Remote Development Extension Pack



VS Code Remote Development

Remote Development 2.7M ★ 4.5
An extension pack that lets you open...
Microsoft Install

Remote - SSH 12.3M ★ 4
Open any folder on a remote machin...
Microsoft Install

Remote - Containers 14.1M ★ 4.5
Open any folder or repository inside ...
Microsoft Install

Remote - WSL 15.6M ★ 5
Open any folder in the Windows Sub...
Microsoft Install

EditorConfig for VS C... 5.6M ★ 4.5
EditorConfig Support for Visual Studi...
EditorConfig Install

Remote - SSH: Editing... 10.8M ★ 4
Edit SSH configuration files
Microsoft Install

Code Runner 14.1M ★ 4.5
Run C, C++, Java, JS, PHP, Python, Pe...
Jun Han Install

Code Spell Checker 4.9M ★ 4.5
Spelling checker for source code
Street Side Software Install

JavaScript (ES6) code s... 9.1M ★ 5
Code snippets for JavaScript in ES6 s...
charalampos karypidis Install

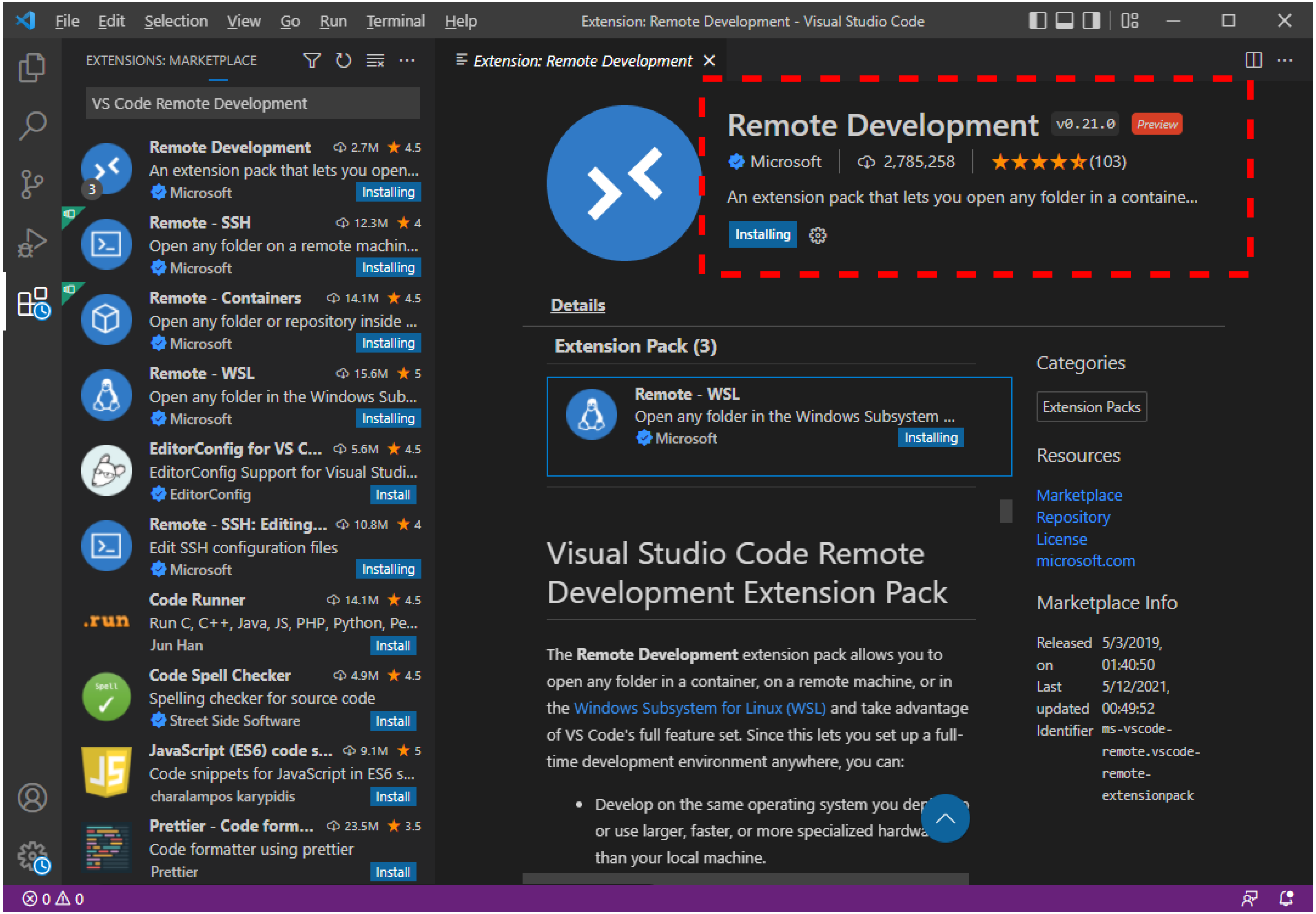
Prettier - Code form... 23.5M ★ 3.5
Code formatter using prettier
Prettier Install

Remote Development ***v0.21.0***

An extension pack that lets you open any folder in a container, on a remote machine, or in WSL and take advantage of VS Code's full feature set.

This publisher has verified ownership of microsoft.com

Show All Commands Ctrl + Shift + P
Open File Ctrl + O
Open Folder Ctrl + K Ctrl + O
Open Recent Ctrl + R
⊗ 0 ⚠ 0


File
Edit
Selection
View
Go
Run
Terminal
Help
Extension: Remote Development - Visual Studio Code
EXTENSIONS: MARKETPLACE
Extension: Remote Development
⊗ 0 ⚠ 0

VS Code Remote Development

Remote Development 2.7M ★ 4.5
An extension pack that lets you open...
Microsoft Installing

Remote - SSH 12.3M ★ 4
Open any folder on a remote machin...
Microsoft Installing

Remote - Containers 14.1M ★ 4.5
Open any folder or repository inside ...
Microsoft Installing

Remote - WSL 15.6M ★ 5
Open any folder in the Windows Sub...
Microsoft Installing

EditorConfig for VS C... 5.6M ★ 4.5
EditorConfig Support for Visual Studi...
EditorConfig Install

Remote - SSH: Editing... 10.8M ★ 4
Edit SSH configuration files
Microsoft Installing

Code Runner 14.1M ★ 4.5
Run C, C++, Java, JS, PHP, Python, Pe...
Jun Han Install

Code Spell Checker 4.9M ★ 4.5
Spelling checker for source code
Street Side Software Install

JavaScript (ES6) code s... 9.1M ★ 5
Code snippets for JavaScript in ES6 s...
charalampos karypidis Install

Prettier - Code form... 23.5M ★ 3.5
Code formatter using prettier
Prettier Install

Remote Development v0.21.0 Preview
Microsoft | 2,785,258 | ★★★★★(103)
An extension pack that lets you open any folder in a containe...
Installing

Details

Extension Pack (3)

Remote - WSL
Open any folder in the Windows Subsystem ...
Microsoft Installing

Visual Studio Code Remote Development Extension Pack

The **Remote Development** extension pack allows you to open any folder in a container, on a remote machine, or in the Windows Subsystem for Linux (WSL) and take advantage of VS Code's full feature set. Since this lets you set up a full-time development environment anywhere, you can:

- Develop on the same operating system you dep[...]o or use larger, faster, or more specialized hardwa[...] than your local machine.

Categories

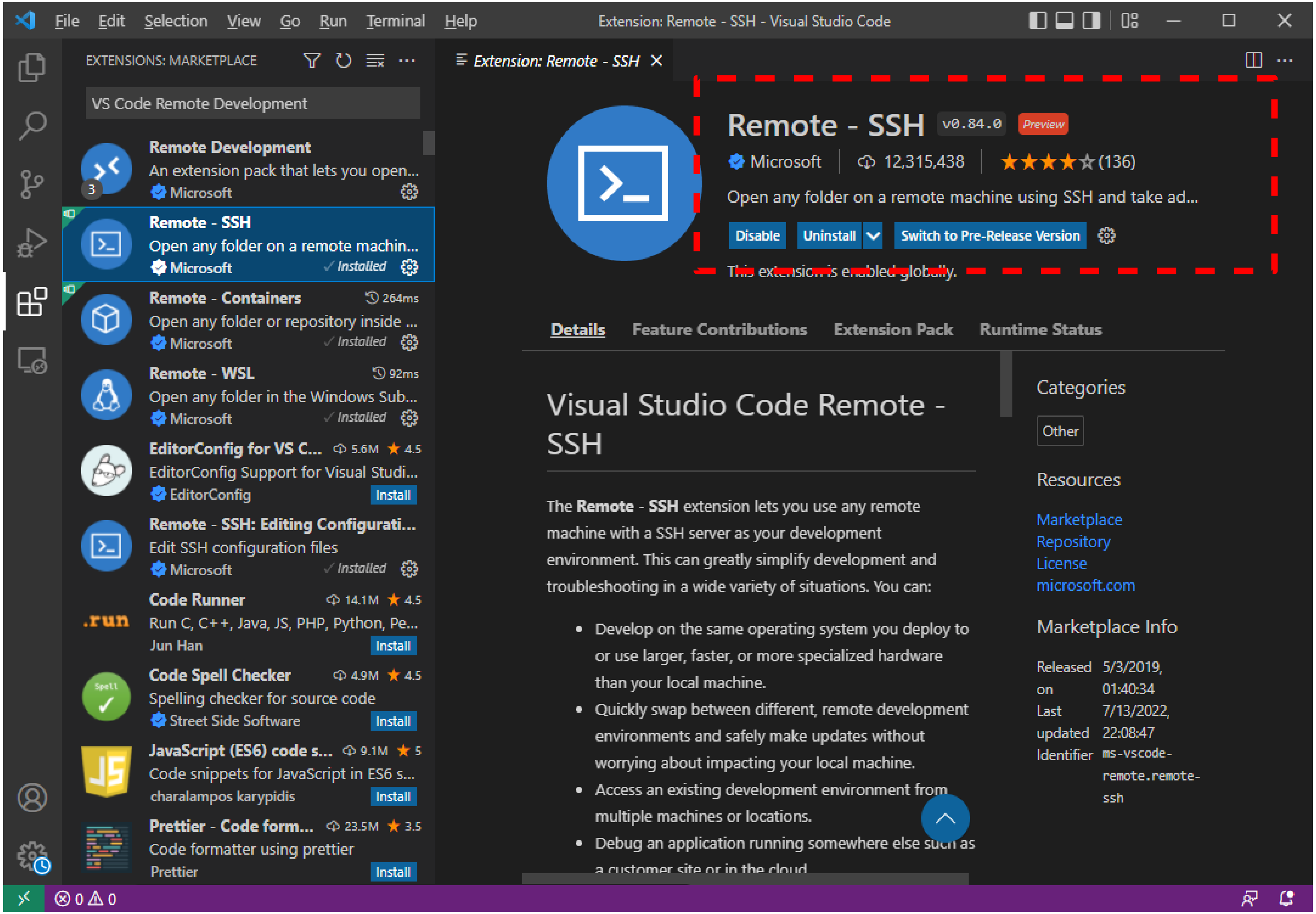
Extension Packs

Resources

Marketplace
Repository
License
microsoft.com

Marketplace Info

Released on	5/3/2019, 01:40:50
Last updated	5/12/2021, 00:49:52
Identifier	ms-vscode-remote.vscode-remote-extensionpack



VS Code Remote Development

Remote Development
An extension pack that lets you open...
Microsoft

Remote - SSH
Open any folder on a remote machin...
Microsoft ✓ Installed

Remote - Containers 264ms
Open any folder or repository inside ...
Microsoft ✓ Installed

Remote - WSL 92ms
Open any folder in the Windows Sub...
Microsoft ✓ Installed

EditorConfig for VS C... 5.6M ★ 4.5
EditorConfig Support for Visual Studi...
EditorConfig Install

Remote - SSH: Editing Configurati...
Edit SSH configuration files
Microsoft ✓ Installed

Code Runner 14.1M ★ 4.5
Run C, C++, Java, JS, PHP, Python, Pe...
Jun Han Install

Code Spell Checker 4.9M ★ 4.5
Spelling checker for source code
Street Side Software Install

JavaScript (ES6) code s... 9.1M ★ 5
Code snippets for JavaScript in ES6 s...
charalampos karypidis Install

Prettier - Code form... 23.5M ★ 3.5
Code formatter using prettier
Prettier Install

Remote - SSH v0.84.0 Preview
Microsoft | 12,315,438 | ★★★★☆ (136)
Open any folder on a remote machine using SSH and take ad...
Disable | Uninstall | Switch to Pre-Release Version
This extension is enabled globally.

Details | Feature Contributions | Extension Pack | Runtime Status

Visual Studio Code Remote - SSH

The **Remote - SSH** extension lets you use any remote machine with a SSH server as your development environment. This can greatly simplify development and troubleshooting in a wide variety of situations. You can:

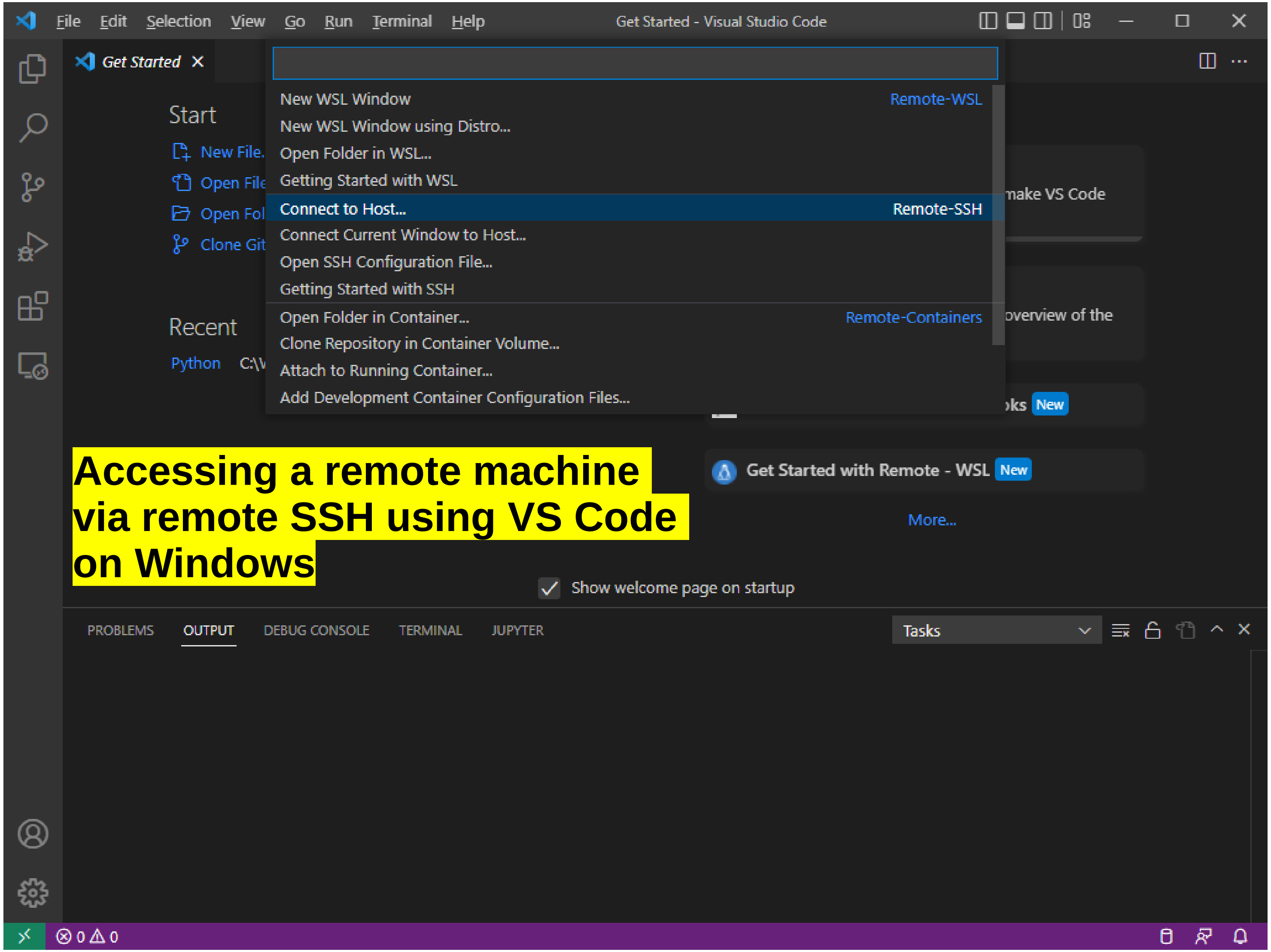
- Develop on the same operating system you deploy to or use larger, faster, or more specialized hardware than your local machine.
- Quickly swap between different, remote development environments and safely make updates without worrying about impacting your local machine.
- Access an existing development environment from multiple machines or locations.
- Debug an application running somewhere else such as a customer site or in the cloud.

Categories
Other

Resources
Marketplace
Repository
License
microsoft.com

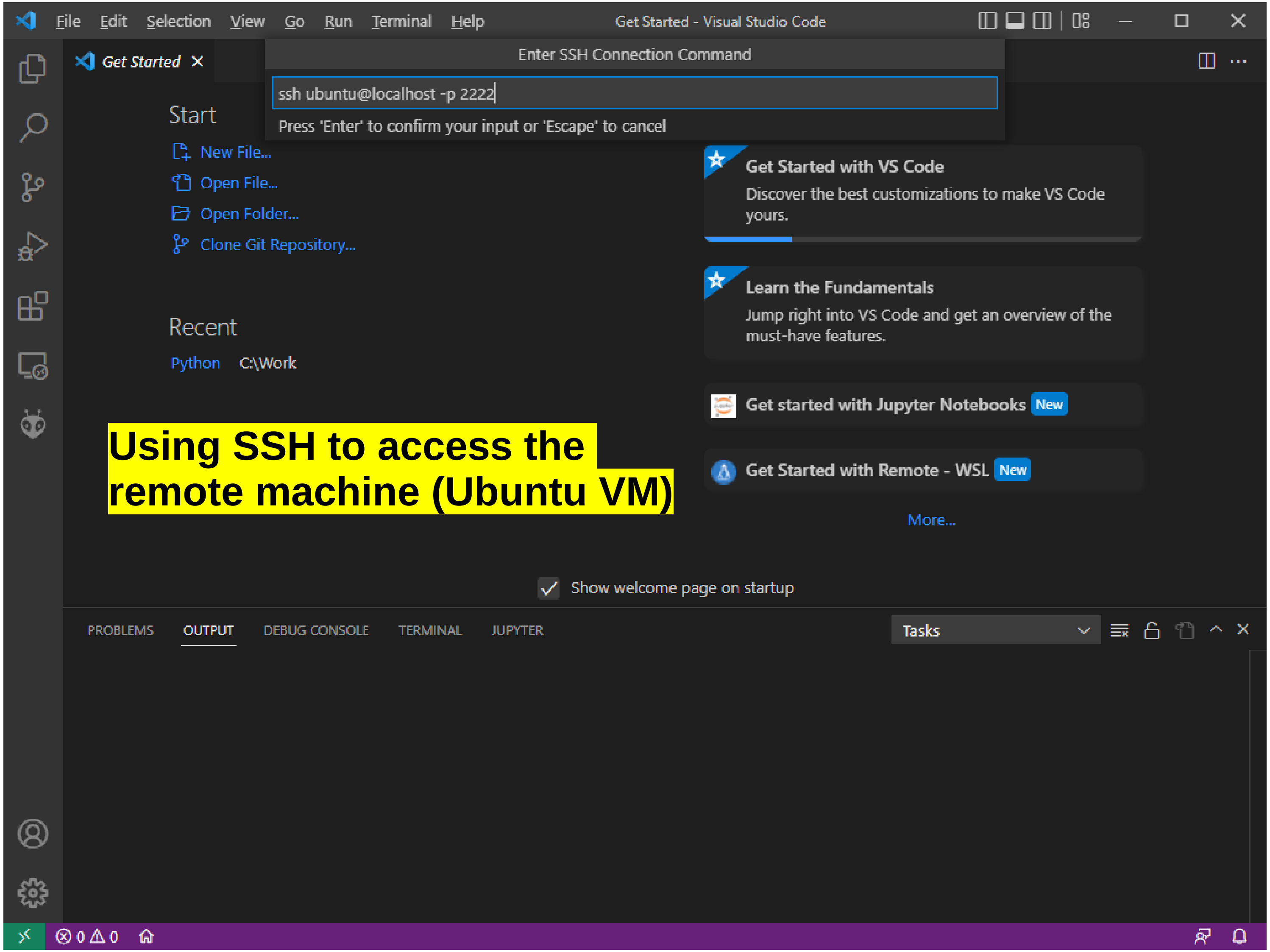
Marketplace Info

Released on	5/3/2019, 01:40:34
Last updated	7/13/2022, 22:08:47
Identifier	ms-vscode-remote.remote-ssh



File
Edit
Selection
View
Go
Run
Terminal
Help
Get Started - Visual Studio Code
Get Started
Start
New File...
Open File
Open Fol
Clone Git
Recent
Python C:\
New WSL Window Remote-WSL
New WSL Window using Distro...
Open Folder in WSL...
Getting Started with WSL
Connect to Host... Remote-SSH
Connect Current Window to Host...
Open SSH Configuration File...
Getting Started with SSH
Open Folder in Container... Remote-Containers
Clone Repository in Container Volume...
Attach to Running Container...
Add Development Container Configuration Files...
make VS Code
overview of the
oks New
Get Started with Remote - WSL New
More...
Show welcome page on startup
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Tasks
0 0

Accessing a remote machine via remote SSH using VS Code on Windows

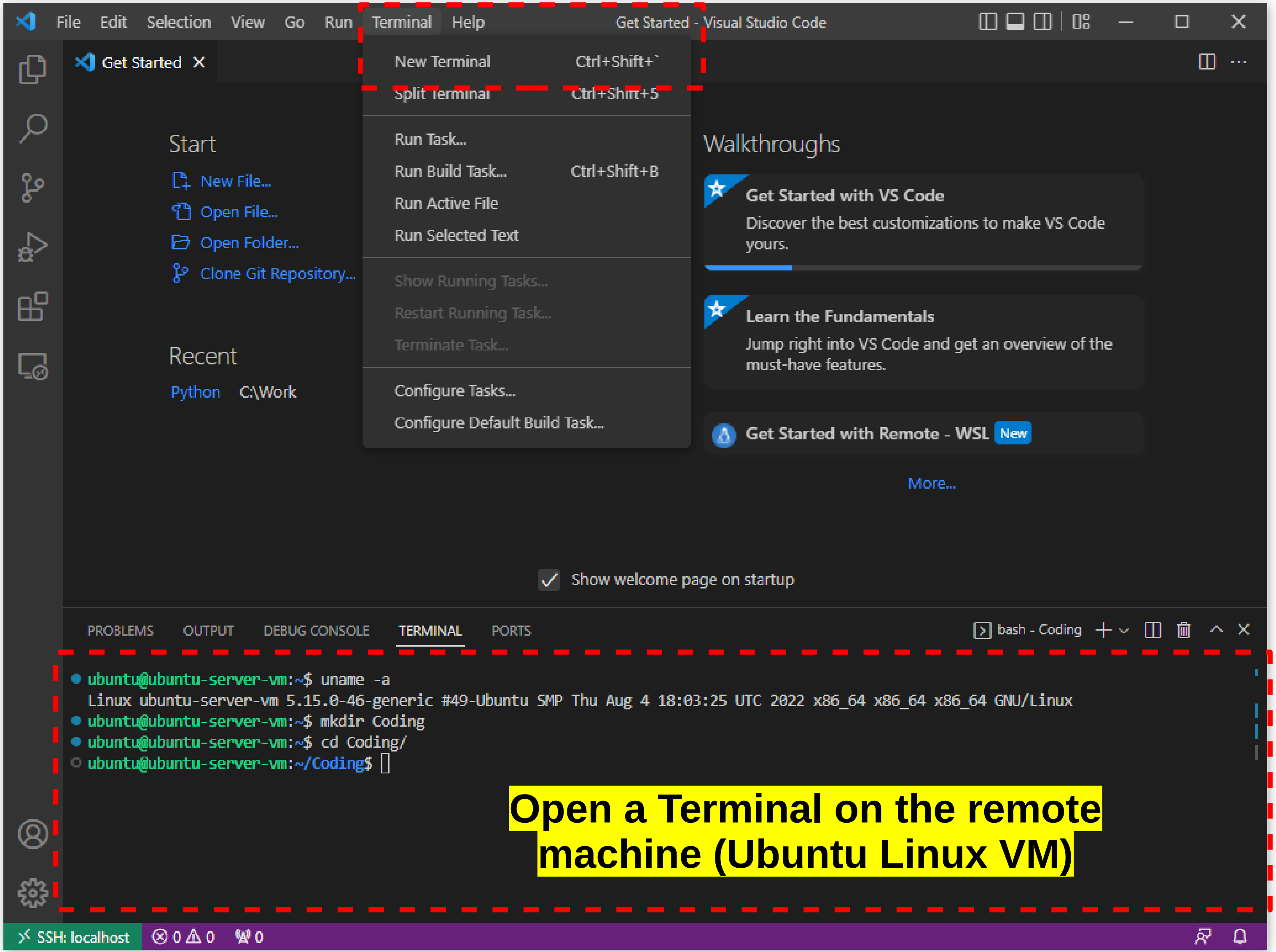

File
Edit
Selection
View
Go
Run
Terminal
Help
Get Started - Visual Studio Code
Get Started
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Recent
Python C:\Work
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
Get started with Jupyter Notebooks New
Get Started with Remote - WSL New
More...
Show welcome page on startup
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Tasks
0
0

Enter SSH Connection Command

```
ssh ubuntu@localhost -p 2222
```

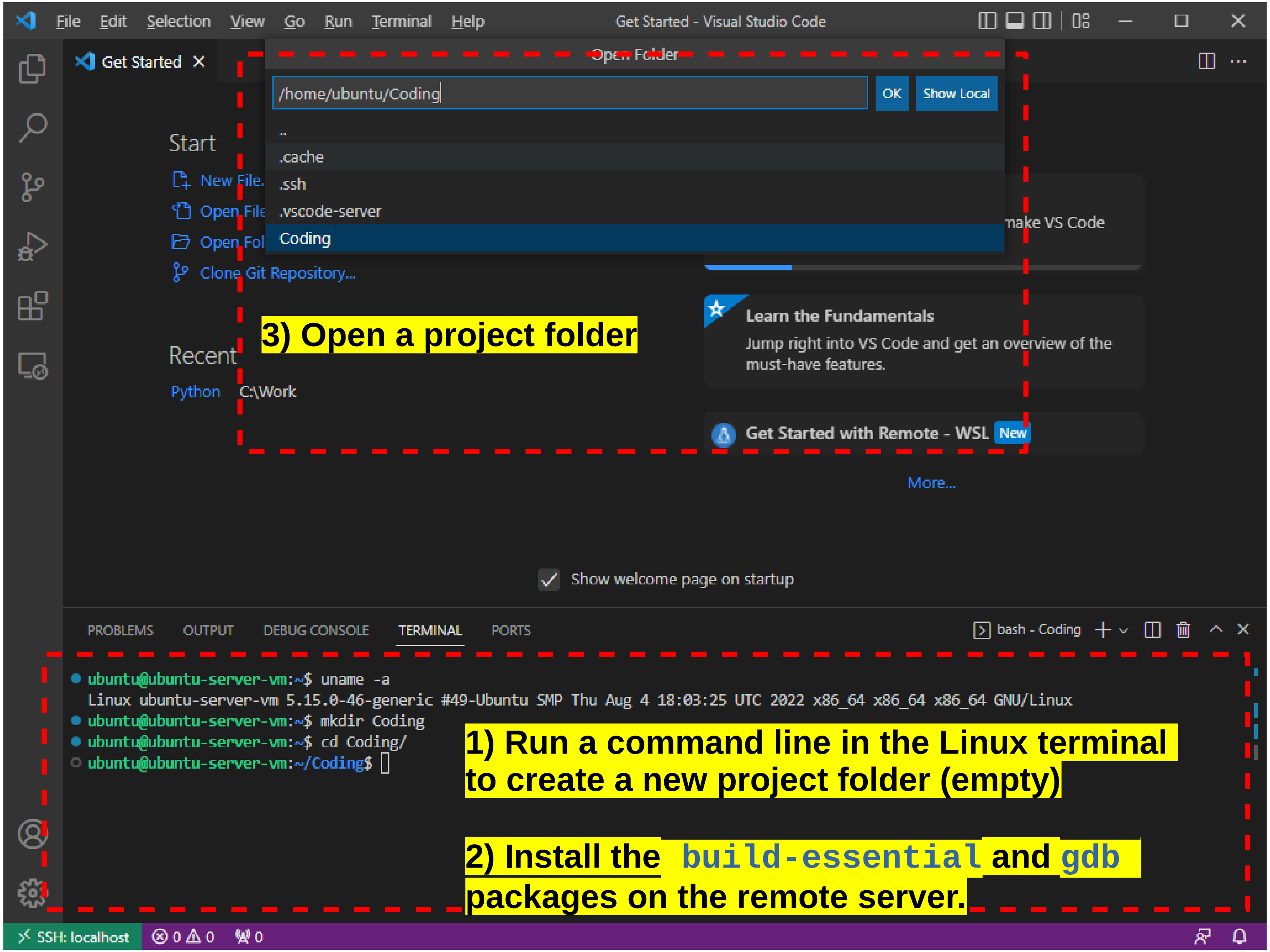
Press 'Enter' to confirm your input or 'Escape' to cancel

Using SSH to access the remote machine (Ubuntu VM)



```
ubuntu@ubuntu-server-vm:~$ uname -a
Linux ubuntu-server-vm 5.15.0-46-generic #49-Ubuntu SMP Thu Aug 4 18:03:25 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
ubuntu@ubuntu-server-vm:~$ mkdir Coding
ubuntu@ubuntu-server-vm:~$ cd Coding/
ubuntu@ubuntu-server-vm:~/Coding$
```

Open a Terminal on the remote machine (Ubuntu Linux VM)



File Edit Selection View Go Run Terminal Help
Get Started - Visual Studio Code
Get Started
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Recent
Python C:\Work
make VS Code
Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.
Get Started with Remote - WSL New
More...
Show welcome page on startup
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - Coding
SSH: localhost
0 0 0

Open Folder

/home/ubuntu/Coding OK Show Local

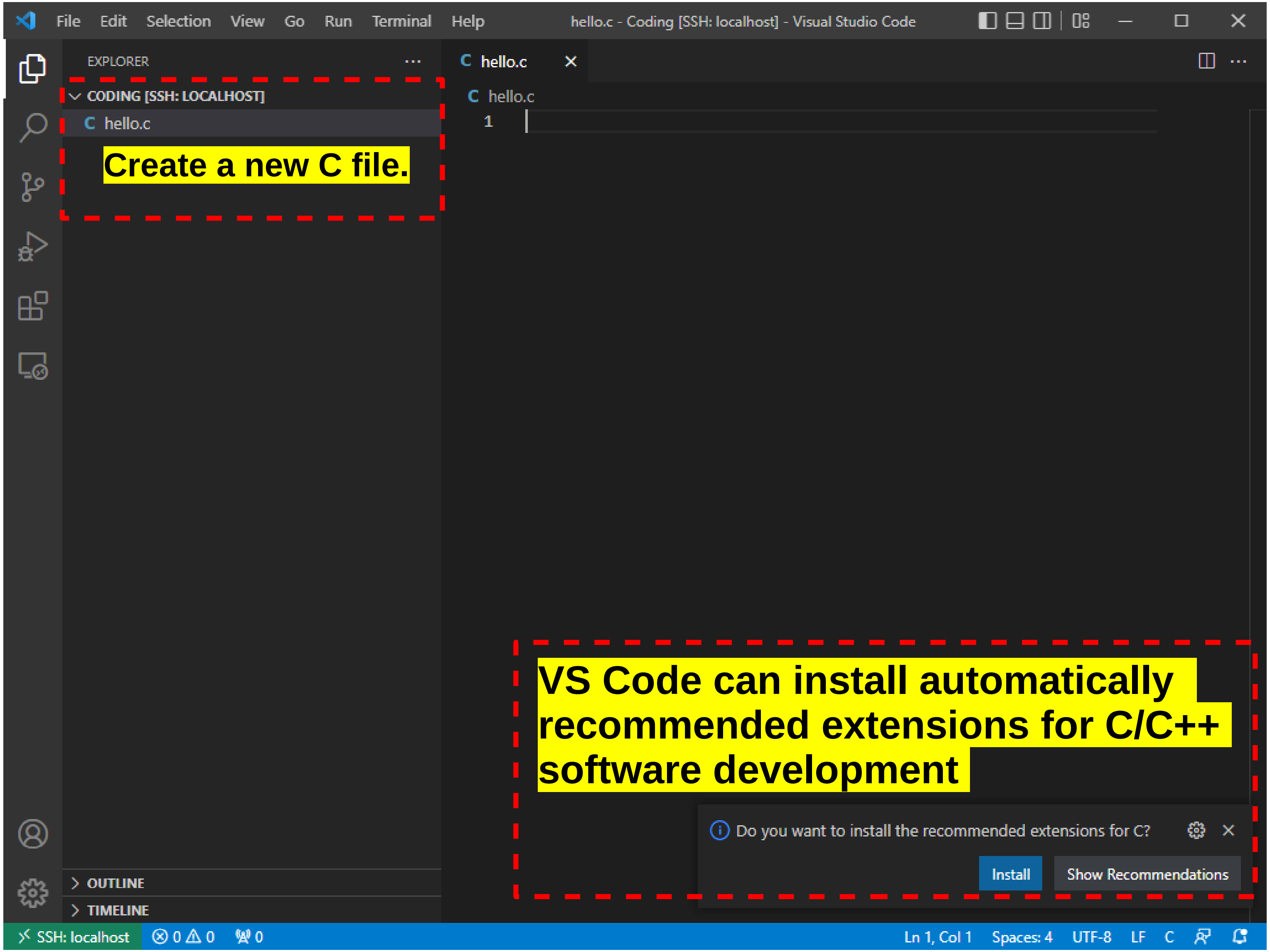
..
.cache
.ssh
.vscode-server
Coding

```
ubuntu@ubuntu-server-vm:~$ uname -a
Linux ubuntu-server-vm 5.15.0-46-generic #49-Ubuntu SMP Thu Aug 4 18:03:25 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
ubuntu@ubuntu-server-vm:~$ mkdir Coding
ubuntu@ubuntu-server-vm:~$ cd Coding/
ubuntu@ubuntu-server-vm:~/Coding$
```

3) Open a project folder

1) Run a command line in the Linux terminal to create a new project folder (empty)

2) Install the `build-essential` and `gdb` packages on the remote server.



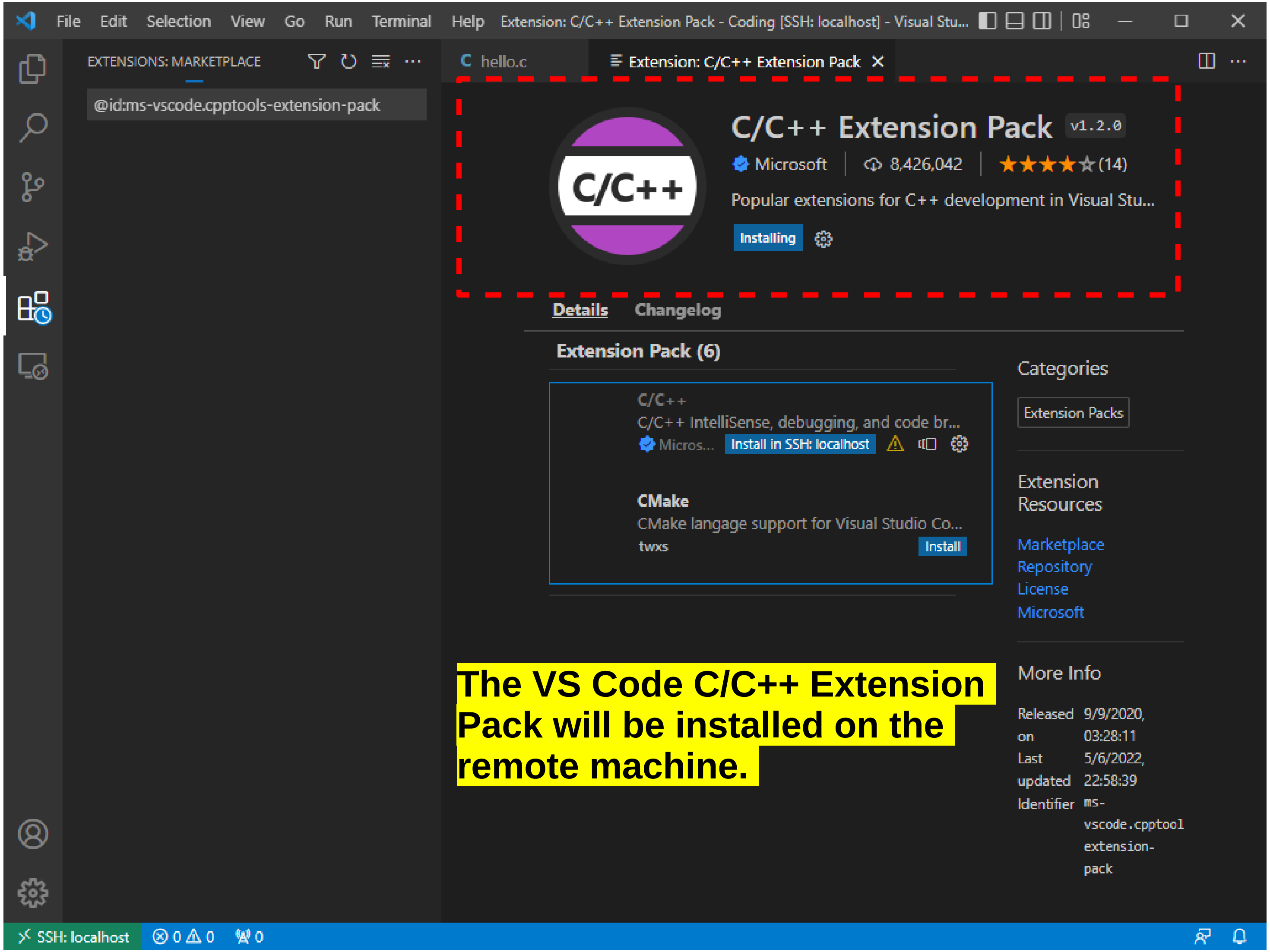
Create a new C file.

VS Code can install automatically recommended extensions for C/C++ software development

Do you want to install the recommended extensions for C?

Install

Show Recommendations



@id:ms-vscode.cpptools-extension-pack

C/C++ Extension Pack v1.2.0

Microsoft | 8,426,042 | ★★★★☆ (14)

Popular extensions for C++ development in Visual Stu...

Extension Pack (6)

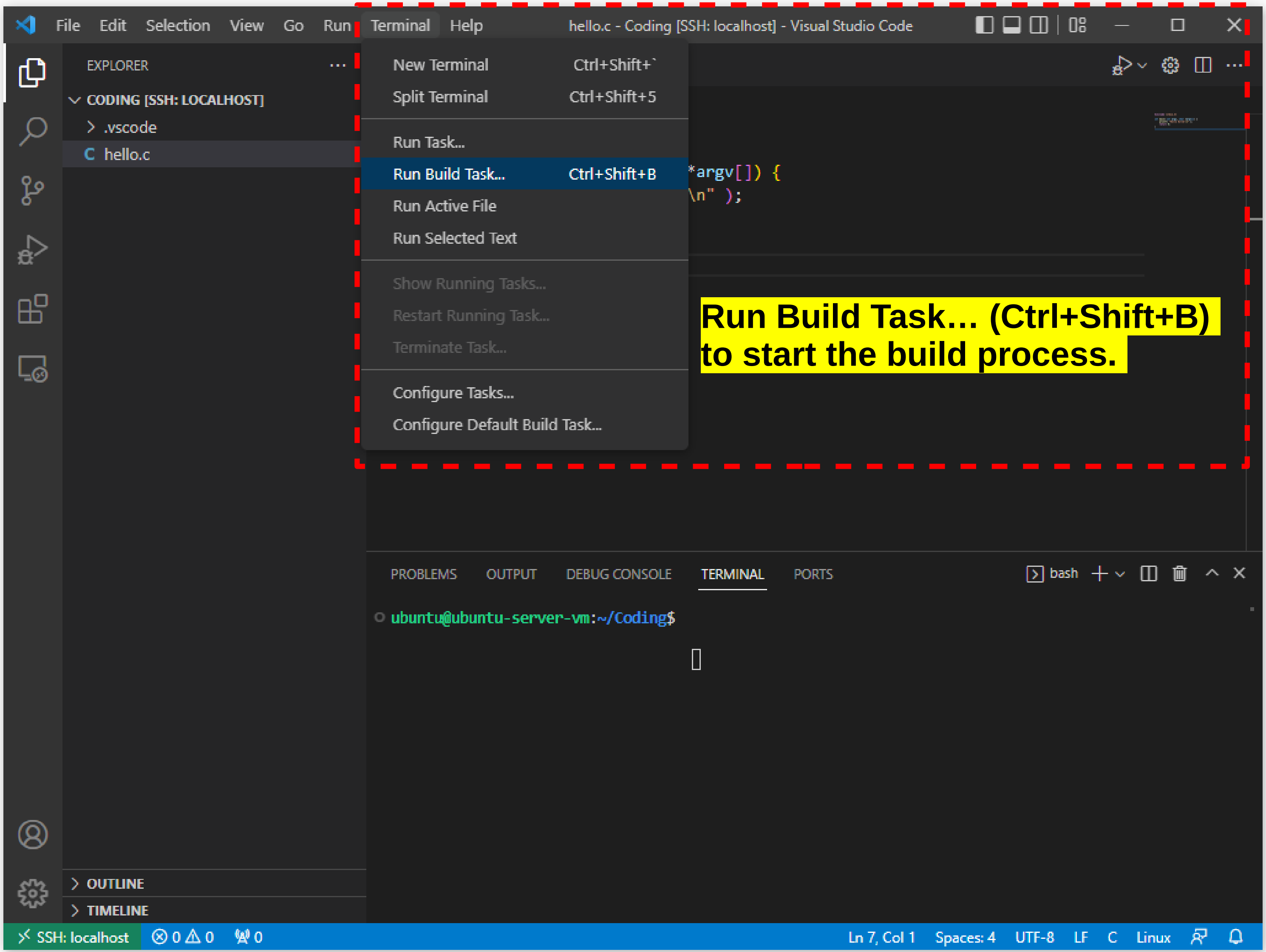
C/C++
C/C++ IntelliSense, debugging, and code br...
Micros...

CMake
CMake langage support for Visual Studio Co...
twxs

More Info

Released on	9/9/2020, 03:28:11
Last updated	5/6/2022, 22:58:39
Identifier	ms-vscode.cpptool extension-pack

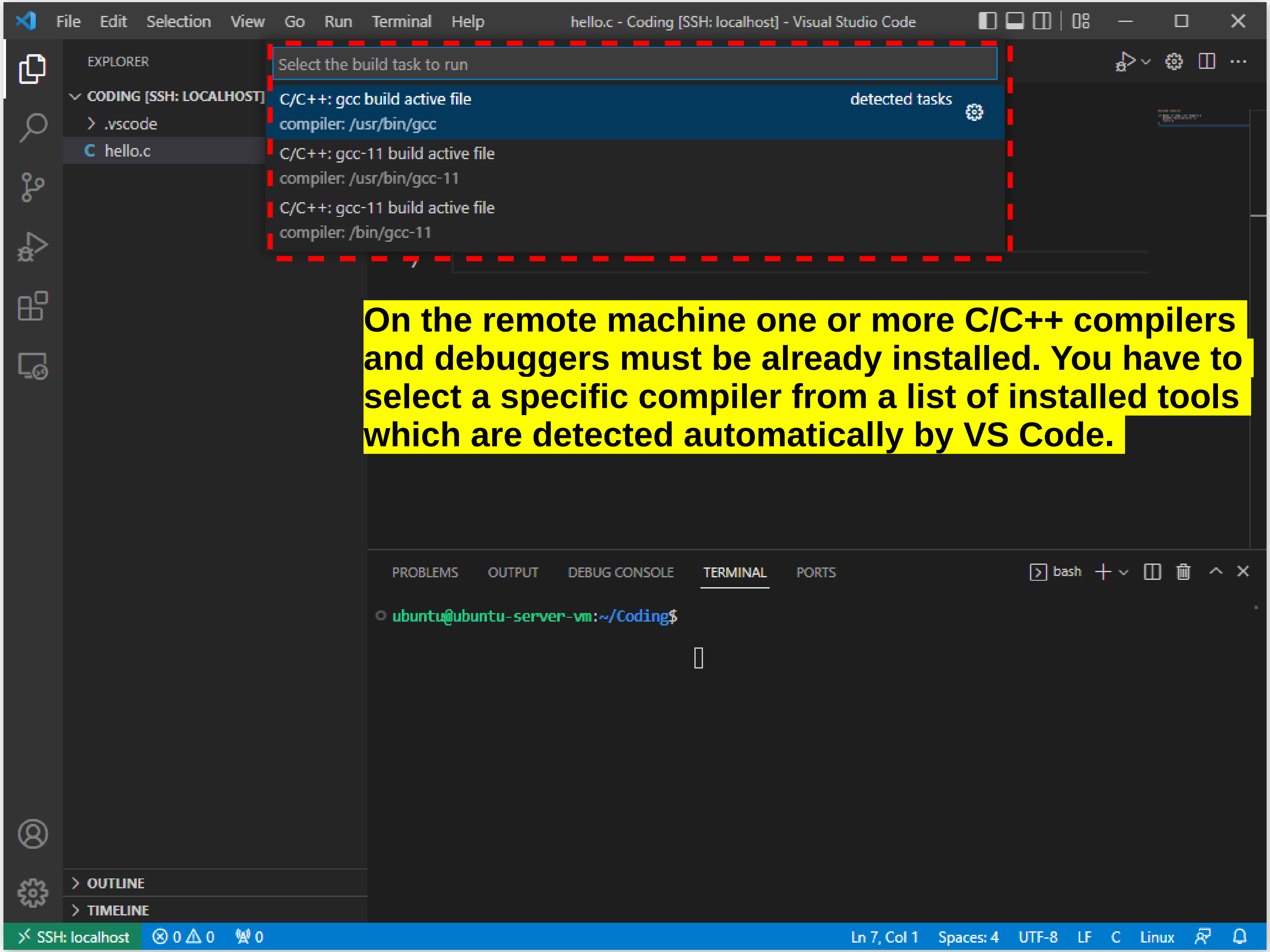
The VS Code C/C++ Extension Pack will be installed on the remote machine.



```
*argv[]) {
\n" );
```

```
ubuntu@ubuntu-server-vm:~/Coding$
```

Run Build Task… (Ctrl+Shift+B) to start the build process.



Select the build task to run

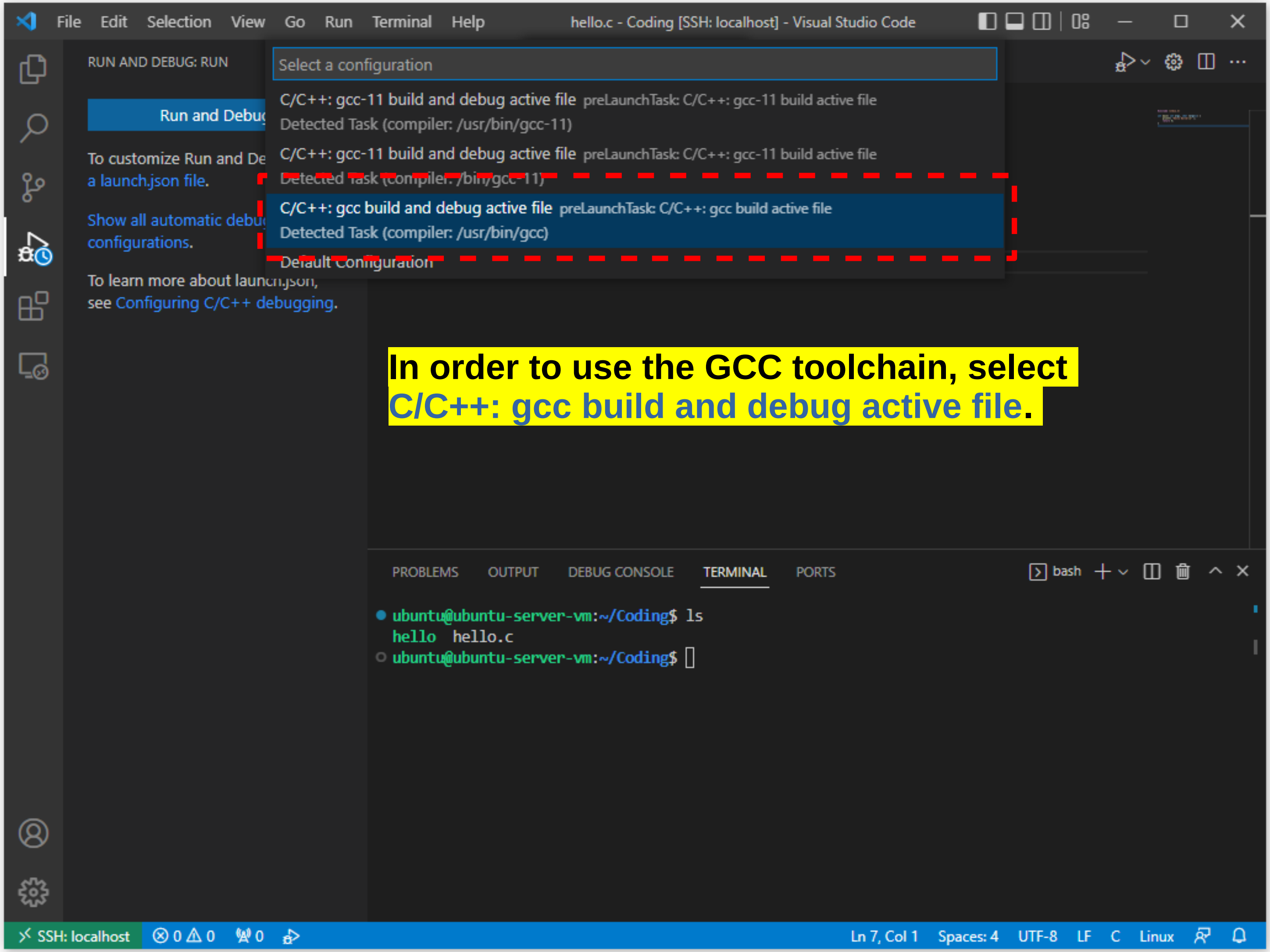
C/C++: gcc build active file — detected tasks
compiler: /usr/bin/gcc

C/C++: gcc-11 build active file
compiler: /usr/bin/gcc-11

C/C++: gcc-11 build active file
compiler: /bin/gcc-11

```
ubuntu@ubuntu-server-vm:~/Coding$
```

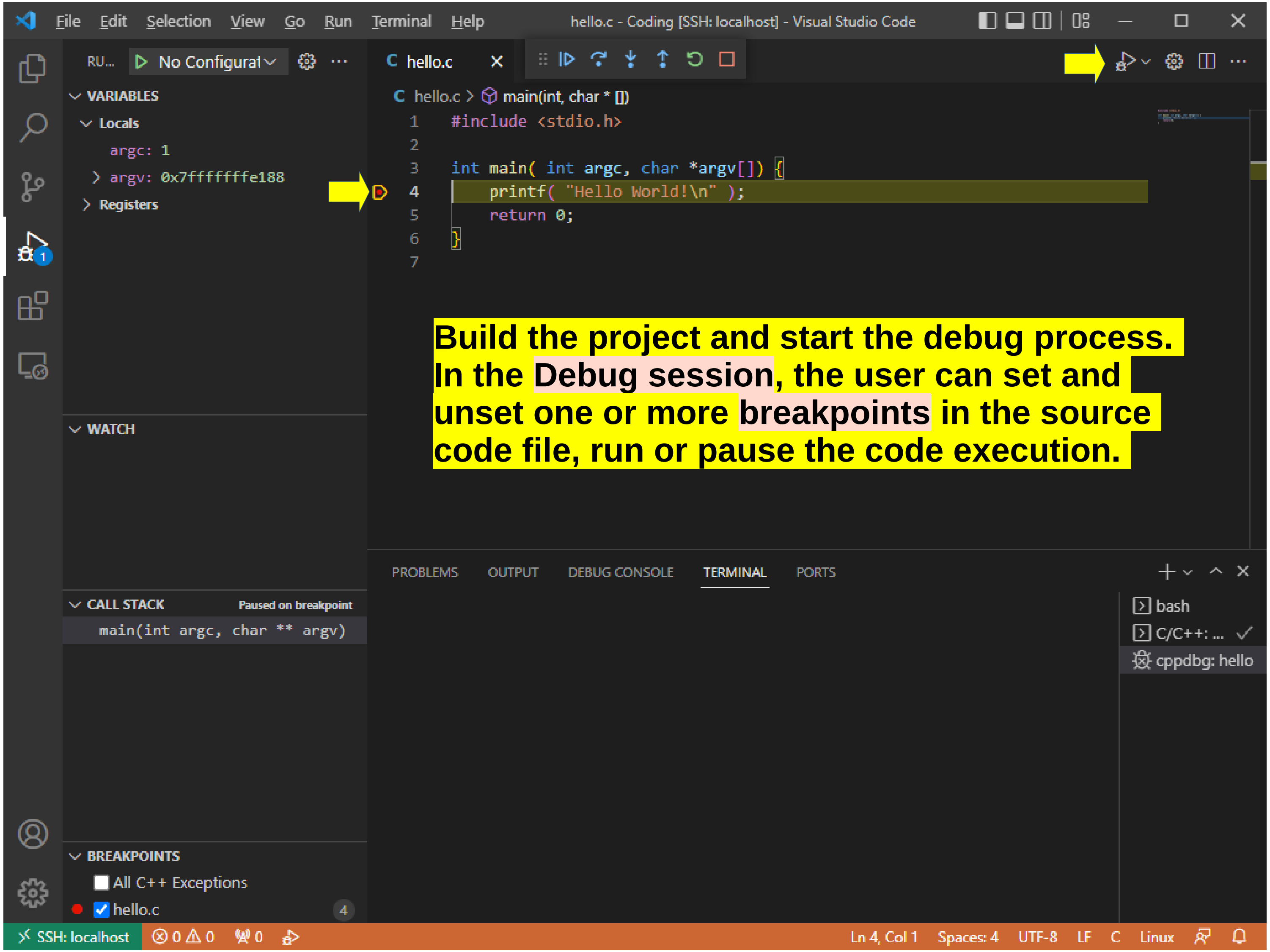
On the remote machine one or more C/C++ compilers and debuggers must be already installed. You have to select a specific compiler from a list of installed tools which are detected automatically by VS Code.



File Edit Selection View Go Run Terminal Help
hello.c - Coding [SSH: localhost] - Visual Studio Code
RUN AND DEBUG: RUN
Run and Debug
To customize Run and De
a launch.json file.
Show all automatic debu
configurations.
To learn more about launch.json,
see Configuring C/C++ debugging.
Select a configuration
C/C++: gcc-11 build and debug active file preLaunchTask: C/C++: gcc-11 build active file
Detected Task (compiler: /usr/bin/gcc-11)
C/C++: gcc-11 build and debug active file preLaunchTask: C/C++: gcc-11 build active file
Detected Task (compiler: /usr/bin/gcc-11)
C/C++: gcc build and debug active file preLaunchTask: C/C++: gcc build active file
Detected Task (compiler: /usr/bin/gcc)
Default Configuration
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
bash
SSH: localhost
Ln 7, Col 1
Spaces: 4
UTF-8
LF
C
Linux

```
ubuntu@ubuntu-server-vm:~/Coding$ ls
hello  hello.c
ubuntu@ubuntu-server-vm:~/Coding$ 
```

In order to use the GCC toolchain, select **C/C++: gcc build and debug active file**.



```
argc: 1
argv: 0x7fffffffe188
```

```
main(int argc, char ** argv)
```

```c
#include <stdio.h>

int main( int argc, char *argv[]) {
    printf( "Hello World!\n" );
    return 0;
}

```

Build the project and start the debug process. In the Debug session, the user can set and unset one or more breakpoints in the source code file, run or pause the code execution.

GNU Make

- A **Makefile** consists of a set of **rules** in a file called **Makefile**.
  - Each **rule** starting with its name and a colon symbol (**:**) specifies one or more **targets** (i.e., file names) in the same line.
  - Each **rule** may have some prerequisites (also called **dependencies**), which are also file names, separated by spaces, and need to exist before the **commands** for the target are run.
  - Commands represent a series of steps typically used to make the target(s).
  - Note that each command per line starts with a **Tab** character, not spaces.
  - A line **comment** is a text that follows a **sharp** symbol (**#**).

Note that there are a number of popular **C/C++ build systems** such as **GNU Make**, **Ninja** and **CMake**.

File: Makefile

```
# use the GCC C compiler
CC=gcc
# enable compilation warning and turn on debug info
CFLAGS=-std=gnu99 -Wall -g3

all: main
	@echo "done..."
main: main.o
	@echo "Link the object file."
	$(CC) $(CFLAGS) main.o -o main
main.o: main.c
	@echo "Compile the main.c file."
	$(CC) $(CFLAGS) -c main.c
clean:
	@echo "Remove the object file and the binary file."
	rm -f main.o main
```

```
$ make --version | head -n2
GNU Make 4.3
Built for x86_64-pc-linux-gnu
$ make clean all -f Makefile
```

GNU Make

- The `make` command updates a target if it depends on the prerequisite files that have been modified since the target was last modified, or if the target does not exist.
- If make is executed without parameters it updates the first target listed in the `Makefile`.
- The @ symbol can be used to suppress echoing a command line to the standard output.
- Like a bash script, variables can be used in the `Makefile`.
  - Variables can be defined by using the = operator.
  - Variables can be accessed by using the @ symbol followed by the variable name enclosed with parentheses (…) or curly brackets {...}.

GNU Makefile

- There are some **Automatic Variables** such as:
  - $@ the target filename without the file extension.
  - $< the first prerequisite filename.
  - $^ the filenames of all the prerequisites, separated by spaces, discard duplicates.
  - $? the names of all prerequisites that are newer than the target, separated by spaces.

File: Makefile (revised)

```
# use the GCC C compiler
CC=gcc
# enable compilation warning and turn on debug info
CFLAGS=-std=gnu99 -Wall -g3
# define Phony targets (which are not file names)
.PHONY: all clean
all: main
    @echo "done..."
main: main.o
    @echo "Link the object file."
    $(CC) $(CFLAGS) $^ -o $@
main.o: main.c
    @echo "Compile the $< file."
    $(CC) $(CFLAGS) -c $<
clean:
    @echo "Remove the object file and the binary file."
    rm -f *.o main
```

Estimation of Pi

- Demo: Estimating the value of Pi using Monte Carlo simulation method.
  - The idea is to generate a large number of uniformly distributed random points in a 2D plane with domain as a 1×1 square.
  - Then, the estimated value of Pi is defined as the ratio of number points that lied inside the circle and total number of generated points, multiplied by 4.
  - Note that the ratio of these two areas is pi/4.

Estimation of Pi

File: estimate_pi.c

```
#include "estimate_pi.h"
#include <stdlib.h>
#include <stdint.h>

double estimate_pi( uint64_t num_iters ) {
  double x,y;
  uint64_t count = 0;
  for( uint64_t i=0; i < num_iters; i++ ) {
    x = ((double)rand()) / RAND_MAX;
    y = ((double)rand()) / RAND_MAX;
    if (x*x + y*y <= 1.0) {
        count++; // increment the counter
    }
  }
  return (4.0*count)/num_iters;
}
```

File: estimate_pi.h

```
#ifndef __ESTIAMTE_PI_H
#define __ESTIMATE_PI_H

#include <stdint.h>

double estimate_pi(
         uint64_t num_iters );

#endif
```

Estimation of Pi

File: main.c

```c
#include <stdio.h>        // for printf()
#include <time.h>         // for time()
#include <stdlib.h>       // for srand()
#include <stdint.h>       // for uint64_t
#include "estimate_pi.h"  // for estimate_pi()

int main( int argc, char *argv[] ) {
  // initialize the pseudo-random number generator
  srand( time(NULL) );
  uint64_t n = 1000000L;
  for ( int i=0; i < 10; i++ ) {
     printf( "%2d) Estimation of Pi = %lf\n",
             (i+1), estimate_pi(n) );
  }
  return 0;
}
```

Estimation of Pi

```
$ gcc ./estimate_pi.c main.c -Wall -I./ -o estimate_pi
$ ./estimate_pi
 1) Estimation of Pi = 3.141948
 2) Estimation of Pi = 3.141832
 3) Estimation of Pi = 3.141008
 4) Estimation of Pi = 3.139972
 5) Estimation of Pi = 3.139968
 6) Estimation of Pi = 3.143260
 7) Estimation of Pi = 3.142628
 8) Estimation of Pi = 3.141756
 9) Estimation of Pi = 3.140880
10) Estimation of Pi = 3.141488
```

Makefile for Multiple Source Files

```
# use the GCC C compiler
CC=gcc
# enable compilation warning and turn on debug info
CFLAGS +=-std=gnu99 \
    -Wall -Og -g3
# define object files
OBJ_FILES = main.o estimate_pi.o
# define Phony targets
.PHONY: all clean
all: main
    @echo "done..."
main: $(OBJ_FILES)
    $(CC) $(CFLAGS) $^ -o $@
%.o: %.c # use pattern rules
    $(CC) $(CFLAGS) -c $<
clean:
    rm -f *.o main
```

Questions

Q1) Why do we need to include the C header file in the following code?

```
#include <stdio.h>

int main( int argc, char **argv ) {
  unsigned int n=0;
  printf( "Please enter a positive number: " );
  scanf( "%u", &n );
  if ( n > 10 ) { n = 10; }
  for ( int i=1; i <= n; i++ ) {
    printf( "%d) Hello world!\n", i );
  }
  return 0;
}
```

Questions

Q2) Explain the difference between the following three code snippets.
Are they syntactically correct code in the C programming language?

```
#include <stdio.h>

int main(int argc, char **argv) {
  printf( "Hello world!\n" );
  return 0;
}
```

```
#include <stdio.h>

void main(void) {
  printf( "Hello world!\n" );
}
```

```
#include <stdio.h>

int main()
{
  printf( "Hello world!\n" );
  return 0;
}
```

Questions

Q3) Rewrite the for loop statement in the following C code using a while loop statement.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(void) {
  srand( time(NULL) );
  int n = 1 + rand() % 10;
  printf( "n = %d\n", n );
  for (int i=n; i >= 0; i--) {
    printf( "Count down %d\n", i );
  }
  return 0;
}
```

Questions

Q4) What is wrong with the C code given below? Debug this code with breakpoints.

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

typedef unsigned char byte;

int main(void) {
  srand( time(NULL) );
  byte n = 1 + rand() % 10;
  printf( "n = %d\n", n );
  for ( byte i=n; i >= 0; i-- ) {
    printf( "Count down %d\n", i );
  }
  return 0;
}
```

Questions

Q5) Rewrite the nested if-else statement in the following C code using a switch statement.

```c
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] ) {
  if ( argc != 2 ) {
     printf( "Please specify an integer!\n" );
     return -1;
  }
  int n = atoi( argv[1] );
  char *str;
  if ( n==0 ) { str = "Zero"; }
  else if ( n==1 || n==-1 ) { str = "Plus or minus one"; }
  else {  str = "Others"; }
  printf( "%s\n", str );
  return 0;
}
```

Questions

Q6) Modify the `main()` function so that it produces a hex string of random data of n bytes.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int get_random_data( int *rng ) {
  int fd = open( "/dev/random",
                 O_RDONLY );
  if (fd) {
    read( fd, rng, sizeof(int) );
    close( fd );
    return 0; // ok
  }
  return -1; // error
}
```

```
int main( void ) {
   int x;
   if ( !get_random_data( &x ) ) {
     printf( "0x%08x (%d)\n",x,x );
   } else {
     printf( "error!!!\n" );
   }
   return 0;
}
```

Questions

Q7) Consider the C code given below. Explain what happens when executing this code.

```
#include <stdio.h>

int get_random_data( size_t n, int *buf )
{
  FILE *fd = fopen("/dev/urandom","rb");
  if ( fd ) {
    for ( size_t i=0; i < n; i++ ) {
      fread( &buf[i],sizeof(int),1,fd );
    }
    fclose( fd );
    return 0;
  }
  return -1;
}
```

```
int main( void ) {
  int data[8];
  size_t n = sizeof(data)/sizeof(int);
  if ( !get_random_data(n, data) ) {
    for ( int i=0; i < n; i++ ) {
      printf( "%02X", data[i] );
    }
    printf("\n");
  } else {
    printf( "error!!!\n" );
  }
  return 0;
}
```

Questions

Q8) Write a C program that is functionally equivalent to the Python script given below:

```
#!/usr/bin/env python3

# convert an integer to a hex string
def to_hex( value ):
    HEX_DIGITS = '0123456789abcdef'
    s = ''
    if (value < 0):
        value += (1 << 32)  # note for a 32-bit value
    while True:
        d = HEX_DIGITS[ value & 0xf ]
        s = d + s
        value >>= 4
        if value == 0:
            break
    s = '0x' + s
    return s
```

Code continues on the next page..

Questions

Code continues on the previous page..

```
if __name__ == "__main__":
    import sys
    if  len(sys.argv) > 1:
        for s in sys.argv[1:]:
            try:
                if s.lower().startswith('0x'):
                    x = int(s,16)
                elif s.lower().startswith('0b'):
                    x = int(s,2)
                else:
                    x = int(s)
            except ValueError:
                print( 'Value error' )
                continue
            print( s, to_hex(x) )
    else:
        print( 'Please specify an integer number.' )
```