หน่วยสมรรถนะ

หน่วยสมรรถนะ

ควบคุมการเดินเครื่องโรงไฟฟ้าพลังความร้อน

สาขาวิชาชีพพลังงานและพลังงานทดแทน

รายละเอียดหน่วยสมรรถนะ

1. รหัสหน่วยสมรรถนะ	GPW-EGS-5-058ZB
2. ชื่อหน่วยสมรรถนะ	ควบคุมการเดินเครื่องโรงไฟฟ้าพลังความร้อน
3. ทบทวนครั้งที่	- / -

4. สร้างใหม่ **ปรับปรุง**

5. สำหรับชื่ออาชีพและรหัสอาชีพ (Occupational Classification)

อาชีพผู้ควบคุมการเดินเครื่องโรงไฟฟ้าพลังความ
ะ ระดับ 5

ISCO-08 3131 เจ้าหน้าที่/ช่างเทคน
เครื่องกังหันผลิตไฟฟ้า (Board Operator)

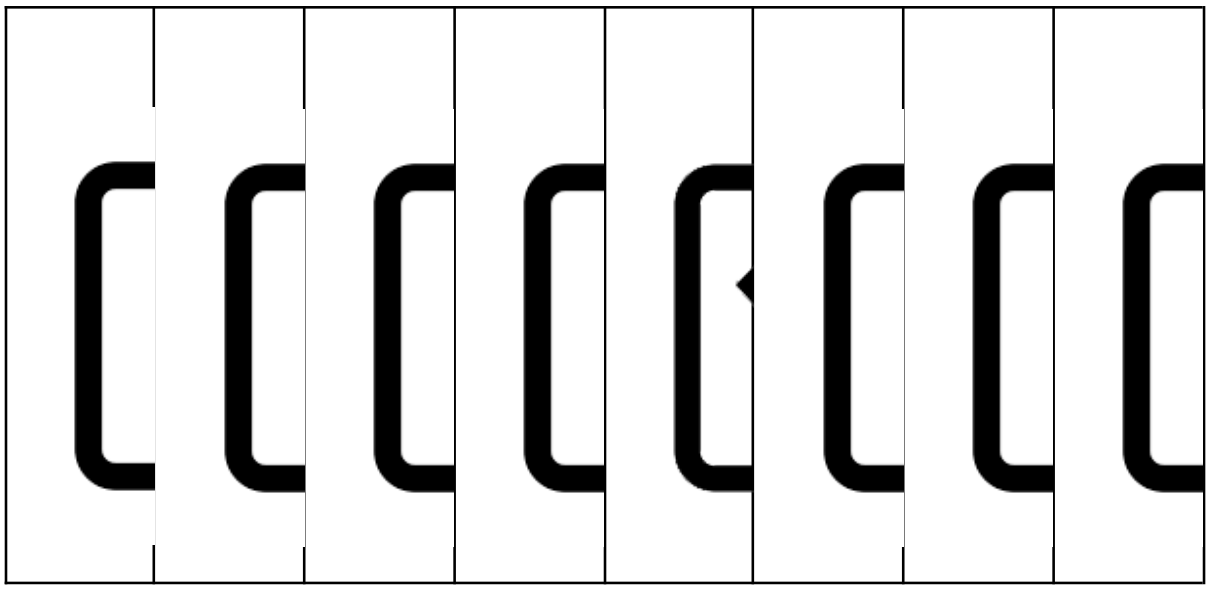
3131 เจ้าหน้าที่/ช่างเทคนิค
องจ่ายกระแสไฟฟ้า (Board Operator)

6. คำอธิบายหน่วยสมรรถนะ (Description of Unit of Competency)

ผู้ที่ผ่านหน่วยสมรรถนะนี้ สามารถควบคุมการเดินเครื่องต่างๆ โรงไฟฟ้าพลังความร้อน ประกอบด้วยหม้อไอน้ำและอุปกรณ์ประกอบ กังหันไอน้ำและอุปกรณ์ประกอบ ระบบกำจัดก๊าซซัลเฟอร์ไดออกไซด์ ระบบไฟฟ้า ระบบสายพานลำเลียง ระบบผลิตและบำบัดน้ำในโรงไฟฟ้า และระบบสนับสนุนระบบผลิตไฟฟ้า โดยสามารถควบคุมการเดินระบบในสภาวะปกติ ควบคุมการเตรียมความพร้อมอุปกรณ์ในการ Start-up และ Shutdown ควบคุมการเตรียมความพร้อมอุปกรณ์ก่อนและระหว่างงานบำรุงรักษา วิเคราะห์และแก้ไขปัญหาระบบในสภาวะฉุกเฉิน ควบคุมประสิทธิภาพการผลิตไฟฟ้า และปฏิบัติตามคำสั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ

7. สำหรับระดับคุณวุฒิ

1	2	3	4	5	6	7	8



8. กลุ่มอาชีพ (Sector)

พลังงานและพลังงานทดแทน สาขางานระบบผลิตไฟฟ้า

9. ชื่ออาชีพและรหัสอาชีพอื่นที่หน่วยสมรรถนะนี้สามารถใช้ได้ (ถ้ามี)

N/A

10. ข้อกำหนดหรือกฎระเบียบที่เกี่ยวข้อง (Licensing or Regulation Related) (ถ้ามี)

N/A

11. สมรรถนะย่อยและเกณฑ์การปฏิบัติงาน (Elements and Performance Criteria)

หน่วยสมรรถนะย่อย (EOC)	เกณฑ์ในการปฏิบัติงาน (Performance Criteria)	รหัส PC (ตามเล่มมาตรฐาน)	รหัส PC (จากระบบ)
PGS-OC02-5-001-01 ควบคุมการเดินระบบในสภาวะปกติ (Normal Operation)	1. อธิบายหลักการทำงานของระบบผลิตไฟฟ้า	PGS-OC02-5-001-01.01	133144
	2. ควบคุมการเดินเครื่องผ่านระบบ Distributed Control System (DCS)	PGS-OC02-5-001-01.02	133145
	3. วินิจฉัยสาเหตุความผิดปกติของระบบผลิตไฟฟ้าจากระบบ Distributed Control System (DCS)	PGS-OC02-5-001-01.03	133146
	4. วิเคราะห์ความเสี่ยงของการทำงานในพื้นที่โรงไฟฟ้า	PGS-OC02-5-001-01.04	133147
	5. ตัดสินใจและประเมินความพร้อมสำหรับการทดสอบอุปกรณ์หลังงานบำรุงรักษาอุปกรณ์ที่มีความผิดปกติ	PGS-OC02-5-001-01.05	133148

	6. ประสานงานกับแผนกบำรุงรักษาเพื่อออกใบแจ้งซ่อม	PGS-OC02-5-001-01.06	133149
	7. จัดทำรายงานประจำวันและบันทึกประวัติการปฏิบัติงาน	PGS-OC02-5-001-01.07	133150
	8. รายงานสถานะของระบบที่รับผิดชอบเพื่อส่งมอบ-กะ	PGS-OC02-5-001-01.08	133151
PGS-OC02-5-001-02 ควบคุมการเตรียมความพร้อมอุปกรณ์ในการ Start-up และ Shut down	1. อธิบายหลักการ Startup โรงไฟฟ้าจากสภาวะต่างๆ	PGS-OC02-5-001-02.01	133152
	2. อธิบายหลักการ Shutdown โรงไฟฟ้าเพื่อวัตถุประสงค์ต่าง ๆ	PGS-OC02-5-001-02.02	133153
	3. ควบคุมการ Startup โรงไฟฟ้าจากสภาวะต่าง ๆ ผ่านระบบ Distributed Control System (DCS)	PGS-OC02-5-001-02.03	133154
	4. ควบคุมการ Shutdown โรงไฟฟ้าเพื่อวัตถุประสงค์ต่าง ๆ ผ่านระบบ Distributed	PGS-OC02-5-001-02.04	133155

	Control System (DCS)		
	5. เฝ้าติดตาม (Monitoring) คุณภาพน้ำในระบบผลิตไฟฟ้าและประสานงานเคมีโรงไฟฟ้า	PGS-OC02-5-001-02.05	133156
	6. แก้ไขความผิดปกติระหว่างการ Startup และ Shutdown	PGS-OC02-5-001-02.06	133157
	7. บันทึกผล/รายงานผลการการ Startup และ Shutdown	PGS-OC02-5-001-02.07	133158
PGS-OC02-5-001-03 ควบคุมการเตรียมความพร้อมอุปกรณ์ก่อนและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ (Planned Outage)	1. อธิบายหลักการเตรียมการทั้งก่อนและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ	PGS-OC02-5-001-03.01	133159
	2. ควบคุมตรวจสอบอุปกรณ์ของระบบทั้งก่อนและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ	PGS-OC02-5-001-03.02	133160
	3. ควบคุมการประสานงานกับหน่วย	PGS-OC02-5-001-03.03	133161

	งานที่เกี่ยวข้องเพื่อเก็บข้อมูล		
	4. ควบคุมการตรวจสอบความผิดปกติของอุปกรณ์ในระบบทั้งก่อนและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ	PGS-OC02-5-001-03.04	133162
	5. วิเคราะห์ ระบุ และแก้ไขปัญหาเบื้องต้นที่เกิดขึ้นทั้งก่อนและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ	PGS-OC02-5-001-03.05	133163
	6. บันทึกและรายงานผล ทั้งก่อนและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ	PGS-OC02-5-001-03.06	133164
PGS-OC02-5-001-04 วิเคราะห์และแก้ไขปัญหาระบบในสภาวะฉุกเฉิน (Emergency Response)	1. วิเคราะห์สาเหตุและแก้ไขความผิดปกติในสภาวะฉุกเฉิน	PGS-OC02-5-001-04.01	133165
	2. จัดทำรายงานอุบัติเหตุ/อุบัติการณ์	PGS-OC02-5-001-04.02	133166
PGS-OC02-5-001-05 ควบคุม	1. อธิบายปัจจัยที่ส่งผลกับประสิทธิภาพระบบผลิตไฟฟ้า	PGS-OC02-5-001-05.01	133167

ประสิทธิภาพการผลิตไฟฟ้า	2. ควบคุมการเดินเครื่องโรงไฟฟ้าให้ได้ประสิทธิภาพตามเป้าหมาย	PGS-OC02-5-001-05.02	133168
PGS-OC02-5-001-06 ปฏิบัติตามคำสั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ	1. อธิบายเกี่ยวกับข้อกำหนดการเชื่อมต่อโครงข่ายไฟฟ้า (Grid Code) การใช้บริการโครงข่ายไฟฟ้า การปฏิบัติการระบบโครงข่ายไฟฟ้าในสัญญา PPA และค่าสำรองไฟฟ้า	PGS-OC02-5-001-06.01	133169
	2. ติดต่อประสานงานกับศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ	PGS-OC02-5-001-06.02	133170
	3. เดินเครื่องตามคำสั่งของศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ	PGS-OC02-5-001-06.03	133171

12. ความรู้และทักษะก่อนหน้าที่จำเป็น (Pre-requisite Skill & Knowledge)

1.

1. โรงต้นกำลังไฟฟ้า โดยเนื้อหาประกอบด้วย
ต้นกำลังต่าง ๆ ได้แก่ โรงไฟฟ้าพลังน้ำ โรงไฟฟ้า
งความร้อน โรงไฟฟ้าพลังกังหันก๊าซ โรงไฟฟ้าพล
เมร้อนร่วม โรงไฟฟ้าพลังเครื่องยนต์ดีเซล โรงไฟ
งนิวเคลียร์ และยังกล่าวถึงหลักการด้าน
ษฐศาสตร์การผลิตโรงไฟฟ้า

13. ทักษะและความรู้ที่ต้องการ (Required Skills and Knowledge)

ความต้องการด้านทักษะ
ษะในการทำงานด้านเทคนิค (Technical Skills)

1.

ทักษะการปฏิบัติงานควบคุมการเดินเครื่องโรง
ไฟฟ้าพลังความร้อนร่วม

2.

ทักษะการใช้งานระบบ Distributed Control
System (DCS)

3.

ทักษะการวิเคราะห์ในการปฏิบัติงาน

4.

ทักษะการตัดสินใจในการปฏิบัติงาน

5.

ทักษะการใช้งานระบบการจัดการงานบำรุงรักษ
ด้วยคอมพิวเตอร์ (CMMS)

6.

ทักษะการบันทึก จัดทำรายงานและรายงานผล

7.

ทักษะการใช้เทคโนโลยีดิจิทัลในการปฏิบัติงา

ษะในการทำงาน (Soft Skills)

8. ทักษะการติดต่อประสานงาน
9. ทักษะการใช้ภาษาอังกฤษในการปฏิบัติงาน
10. ทักษะการทำงานร่วมกับผู้อื่น (Team Workin
11. ทักษะการแก้ปัญหา
12. ทักษะการควบคุมงาน
13.

ทักษะความเป็นผู้นำ (Leadership)

14.

ทักษะการสอนงานเบื้องต้น

ความต้องการด้านความรู้

1. หลักการทำงานของระบบผลิตไฟฟ้า
2. การใช้งานระบบ Distributed Control Syster (DCS)
3. มีความรู้เรื่องความปลอดภัย อาชีวอนามัย และ สภาพแวดล้อมในการทำงาน และกฎหมายด้า สิ่งแวดล้อมที่เกี่ยวข้อง
4. หลักการ Startup โรงไฟฟ้าจากสภาวะต่าง ๆ
5. หลักการ Shutdown โรงไฟฟ้าเพื่อวัตถุประส ต่าง ๆ
6. ข้อกำหนดการเชื่อมต่อโครงข่ายไฟฟ้า (Grid Code) การใช้บริการโครงข่ายไฟฟ้า การปฏิบัติการระบบโครงข่ายไฟฟ้าในสัญญาซื้อข ไฟฟ้า (Power Purchase Agreement (PPA

7. มีความรู้ด้านสัญญาซื้อขายไฟฟ้า (Power Purchase Agreement (PPA))
8. การควบคุมประสิทธิภาพการผลิตไฟฟ้า
9. ความรู้ความสามารถด้านการใช้โปรแกรมเอกส บนคอมพิวเตอร์
10. ความรู้เกี่ยวกับวิธีการใช้งานระบบการจัดการง บำรุงรักษาด้วยคอมพิวเตอร์ (CMMS)
11. คำศัพท์ภาษาอังกฤษทางเทคนิคในการปฏิบัติ งาน

14. หลักฐานที่ต้องการ (Evidence Guide)

หลักฐานการปฏิบัติงาน (Performance Evidenc ง

1. ใบรับรองการปฏิบัติงานจากสถานประกอบการ (ถ้ามี)
2.

แบบบันทึกผลการสังเกตการปฏิบัติงาน (ถ้ามี)

3.

แบบรวบรวม/แฟ้มสะสมผลงานการปฏิบัติงาน (ถ้ามี)

4.

หลักฐานการอบรมด้านความปลอดภัย อาชีวอ
มัย และสิ่งแวดล้อม (ถ้ามี) โดยไม่ต้องประเมิน
หน่วยสมรรถนะ ด้านความปลอดภัย อาชีวอนา
และสิ่งแวดล้อม

5.

หลักฐานการอบรมหลักสูตรความรู้พื้นฐานโรง
ไฟฟ้า (ถ้ามี) โดยไม่ต้องประเมินในหน่วย
สมรรถนะความรู้พื้นฐานโรงไฟฟ้า

หลักฐานความรู้ (Knowledge Evidence) หรือ

1. หลักฐานการศึกษา
2. ใบรับรองการปฏิบัติงานจากสถานประกอบการ (ถ้ามี)
3. แบบบันทึกผลการสัมภาษณ์ (ถ้ามี)
4. แบบบันทึกผลการสอบข้อเขียน (ถ้ามี)
5. แบบรวบรวม/แฟ้มสะสมผลงาน (Portfolio) ก่ ปฏิบัติงาน (ถ้ามี)

คำแนะนำในการประเมิน

ะเมินเข้ารับการประเมินสามารถนำหลักฐานการปฏิบ
และหลักฐานความรู้มาประกอบในการประเมิน โด
รวมข้อมูลตามรายละเอียดที่แสดงในรายการตรวจ
บ (Check list)

วิธีการประเมิน

พิจารณาหลักฐานความรู้ที่ผู้เข้ารับการประเมินนำม
ดง เช่น ใบรับรองฯ

พิจารณาหลักฐานการปฏิบัติงาน แสดงหลักฐานกา
นการอบรม/ใบรับรองจากสถานประกอบการ (ถ้ามี)

15. ขอบเขต (Range Statement)

ขอบเขตของการประเมินสมรรถนะในหน่วย
รรถนะนี้ ผู้เข้ารับการประเมินจะถูกประเมินทักษะใน
ควบคุมการเดินระบบในสภาวะปกติ การควบคุม
เตรียมความพร้อมอุปกรณ์ในการ Startup และ
utdown การควบคุมการเตรียมความพร้อมอุปกรณ์
นและระหว่างงานบำรุงรักษาแบบหยุดตามวาระ กา
ราะห์และแก้ไขปัญหาระบบในสภาวะฉุกเฉิน การ

ุมประสิทธิภาพการผลิตไฟฟ้า และการปฏิบัติตา
ั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ

) คำแนะนำ

ผู้เข้ารับการประเมินจะต้องควบคุมการเดินเครื่อง
ฟ้าพลังความร้อน โดยต้องทราบถึงหลักการควบคุ
เดินเครื่องไฟฟ้าดังกล่าว

) คำอธิบายรายละเอียด

1. ควบคุมการเดินเครื่องในสภาวะปกติ (Normal Operation) หมายถึง การควบคุมกระบวนการผลิตไฟฟ้าในโรงไฟฟ้าให้เป็นไปตามแผนสั่งการเดินเครื่อง และพารามิเตอร์ต่างๆ ของโรงไฟฟ้าอยู่ในเกณฑ์ปกติตามที่ระบุไว้ในคู่มือ
2. Distributed Control System : DCS หมายถึง ระบบควบคุมขนาดใหญ่
สำหรับกระบวนการที่มีความซับซ้อนที่สามารถให้พนักงาน ควบคุมสามารถปรับตั้งค่าควบคุม (Set Point) ได้จากห้องควบคุมส่วนกลาง (Central Control Room)

3. วินิจฉัยสาเหตุความผิดปกติของระบบผลิตไฟ
จากระบบ Distributed Control System (DC
หมายถึง การหาสาเหตุความผิดปกติของโรง
ไฟฟ้าด้วย Logic Control เช่น Bailey Contr
Wiring Diagram
4. วิเคราะห์ความเสี่ยงก่อนเข้าพื้นที่ หมายถึง กา
ตรวจสอบพื้นที่ปฏิบัติงานตามหลักความ
ปลอดภัยก่อนอนุมัติใบอนุญาตทำงาน (Work
Permit)
5. สภาวะฉุกเฉิน (Emergency Response) ของ
โรงไฟฟ้า หมายถึงเหตุการณ์ที่เป็นไปโดย
ปัจจุบันทันด่วน โดยไม่คาดคิดหรือคาดการณ์
ล่วงหน้าและส่งผลกระทบต่อกำลังการผลิต
ทรัพย์สิน ความปลอดภัยและสิ่งแวดล้อม
6. การบำรุงรักษาแบบหยุดตามวาระ (Planned
Outage) หมายถึง การทำงานบำรุงรักษา
เครื่องจักร อุปกรณ์ระหว่างหยุดเดินเครื่องตาม
แผน
7. ประสิทธิภาพการผลิตไฟฟ้า หมายถึง อัตรากา
ใช้ความร้อนเฉลี่ยต่อผลิตพลังงานไฟฟ้า (Hea
Rate) และค่าความพร้อมจ่าย (Equivalent
Available Factor) ความสูญเสียในระบบ (Los
Analysis)

8. ศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ หมายถึ
หน่วยงานที่รับผิดชอบการวางแผนการผลิต แ
เป็นผู้สั่งการให้โรงไฟฟ้าทุกแห่งในระบบ เดิน
เครื่องผลิตไฟฟ้าในเวลาต่าง ๆ รวมทั้งควบคุม
การซื้อขายไฟกับประเทศเพื่อนบ้าน เพื่อให้กา
ผลิตไฟฟ้ามีความสมดุลกับความต้องการใช้ แ
ควบคุมการจ่ายไฟและคุณภาพของไฟฟ้าให้เ
ไปตามมาตรฐานและข้อกำหนดของกระทรวง
พลังงาน
9. คำสั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ เ
Prior Notice, Resumption, Declaration,
Acceptance, Dispatch Instruction, Post
Event, Startup Time เป็นต้น
10. ระบบโครงข่ายไฟฟ้า หมายถึง ระบบส่งไฟฟ้า
ของการไฟฟ้าฝ่ายผลิตแห่งประเทศไทย (กฟ
หรือระบบจำหน่ายไฟฟ้าของการไฟฟ้า
ส่วนภูมิภาค (กฟภ.) หรือระบบจำหน่ายไฟฟ้า
ของการไฟฟ้านครหลวง (กฟน.)

16. หน่วยสมรรถนะร่วม (ถ้ามี)

N/A

17. อุตสาหกรรมร่วม/กลุ่มอาชีพร่วม (ถ้ามี)

N/A

18. รายละเอียดกระบวนการและวิธีการประเมิน (Assessment Description and Procedure)

1.

1 เครื่องมือประเมิน ควบคุมการเดินระบบในสภาวะ
ติ (Normal Operation)

1. ข้อเขียนแบบปรนัย เช่น การควบคุมการเดินระ
ในสภาวะปกติ (Normal Operation)
2. ข้อเขียนแบบอัตนัย เช่น การควบคุมการเดิน
ระบบในสภาวะปกติ (Normal Operation)
3. การสัมภาษณ์ เช่น การสอบสัมภาษณ์เกี่ยวกับ
การควบคุมการเดินระบบในสภาวะปกติ (Norm
Operation)

2 เครื่องมือประเมิน ควบคุมการเตรียมความพร้อม
กรณ์ในการ Startup และ Shutdown

1. ข้อเขียนแบบปรนัย เช่น การควบคุมการเตรียม ความพร้อมอุปกรณ์ในการ Start-up และ Shu down
2. ข้อเขียนแบบอัตนัย เช่น การควบคุมการเตรียม ความพร้อมอุปกรณ์ในการ Startup และ Shutdown
3. การสัมภาษณ์ เช่น การสอบสัมภาษณ์เกี่ยวกับ การควบคุมการเตรียมความพร้อมอุปกรณ์ในก Startup และ Shutdown

3 เครื่องมือประเมิน ควบคุมการเตรียมความพร้อม กรณ์ก่อนและระหว่างงานบำรุงรักษาแบบหยุดตาม ะ (Planned Outage)

1. ข้อเขียนแบบปรนัย เช่น การควบคุมการเตรียม ความพร้อมอุปกรณ์ก่อนและระหว่างงานบำรุง รักษาแบบหยุดตามวาระ (Planned Outage)
2. ข้อเขียนแบบอัตนัย เช่น การควบคุมการเตรียม ความพร้อมอุปกรณ์ก่อนและระหว่างงานบำรุง รักษาแบบหยุดตามวาระ (Planned Outage)
3. การสัมภาษณ์ เช่น การสอบสัมภาษณ์เกี่ยวกับ

การควบคุมการเตรียมความพร้อมอุปกรณ์ก่อน และระหว่างงานบำรุงรักษาแบบหยุดตามวาระ (Planned Outage)

4 เครื่องมือประเมิน วิเคราะห์และแก้ไขปัญหาระบ
สภาวะฉุกเฉิน (Emergency Response)

1. ข้อเขียนแบบปรนัย เช่น การวิเคราะห์และแก้ไ ปัญหาระบบในสภาวะฉุกเฉิน (Emergency Response)
2. ข้อเขียนแบบอัตนัย เช่น การวิเคราะห์และแก้ไ ปัญหาระบบในสภาวะฉุกเฉิน (Emergency Response)
3. การสัมภาษณ์ เช่น การสอบสัมภาษณ์เกี่ยวกับ การวิเคราะห์และแก้ไขปัญหาระบบในสภาวะ ฉุกเฉิน (Emergency Response)

5 เครื่องมือประเมิน ควบคุมประสิทธิภาพการผลิต ฟ้า

1. ข้อเขียนแบบปรนัย เช่น การควบคุมประสิทธิภาการผลิตไฟฟ้า
2. ข้อเขียนแบบอัตนัย เช่น การควบคุมประสิทธิภาพการผลิตไฟฟ้า
3. การสัมภาษณ์ เช่น การสอบสัมภาษณ์เกี่ยวกับการควบคุมประสิทธิภาพการผลิตไฟฟ้า

6 เครื่องมือประเมินปฏิบัติตามคำสั่งศูนย์ควบคุมระงังไฟฟ้าแห่งชาติ

1. ข้อเขียนแบบปรนัย เช่น การปฏิบัติตามคำสั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ
2. ข้อเขียนแบบอัตนัย เช่น การปฏิบัติตามคำสั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ
3. การสัมภาษณ์ เช่น การสอบสัมภาษณ์เกี่ยวกับการปฏิบัติตามคำสั่งศูนย์ควบคุมระบบกำลังไฟฟ้าแห่งชาติ

วาล์วควบคุมถูกใช้ใน 4 ระบบหลักของโรงไฟฟ้า

By วิลล์ ดอน
March 1, 2021

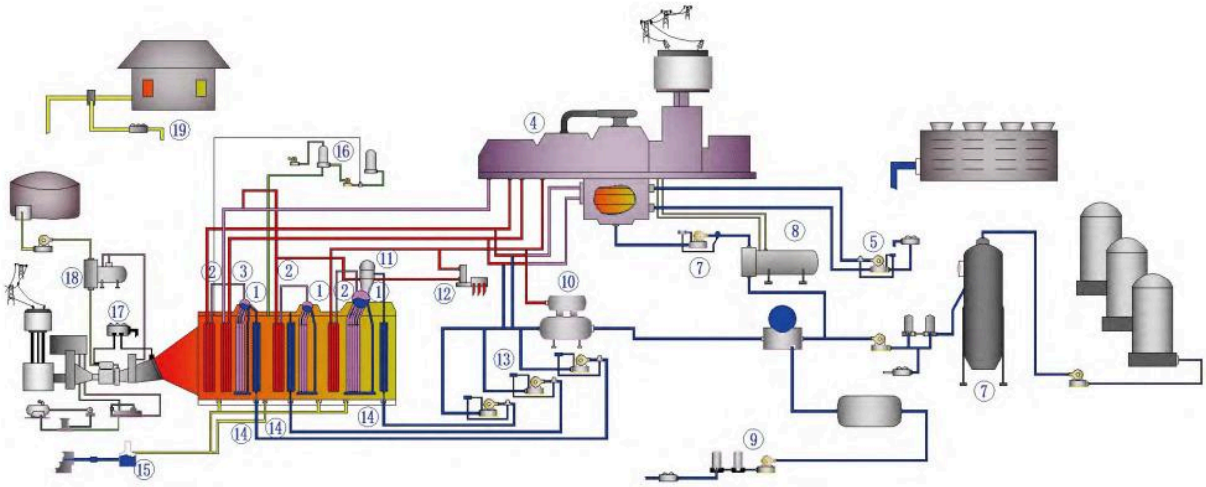
การขอ วาล์วควบคุม เป็นอุปกรณ์ควบคุมท่อส่งของไหล ทำหน้าที่เชื่อมต่อ ปิด และสั่งจ่ายของไหล ควบคุมแรงดันปานกลางและการไหล และป้องกันท่อและอุปกรณ์จากการทำงานที่ไม่เหมาะสม มีบทบาทสำคัญในระบบควบคุมของโรงไฟฟ้า

โรงไฟฟ้าเป็นสถานีที่ครอบคลุมพลังงานดั้งเดิมเป็นพลังงานไฟฟ้าสำหรับสิ่งอำนวยความสะดวกหรือการขนส่งคงที่ ตัวอย่างเช่น สำหรับไฟ น้ำ ลม เซลล์แสงอาทิตย์ ความร้อนใต้พิภพ น้ำขึ้นน้ำลง ชีวภาพ เชื้อเพลิง หรือโรงไฟฟ้านิวเคลียร์ โดยเฉพาะอย่างยิ่งกำลังการผลิตติดตั้งของโรงไฟฟ้าเชื้อเพลิงฟอสซิลคิดเป็น 70% ของทั้งหมดทั่วโลก โดยมีการผลิตไฟฟ้า 80% ของโรงไฟฟ้าทั้งหมด โรงไฟฟ้าเชื้อเพลิงฟอสซิล ซึ่งเป็นส่วนสำคัญในการผลิตพลังงานไฟฟ้า กำลังมีบทบาทพื้นฐานในการเติบโตทางเศรษฐกิจและการปรับปรุงมาตรฐานการครองชีพ เป็นที่รู้จักจากการเผาไหม้ถ่านหิน น้ำมันปิโตรเลียม หรือ ก๊าซธรรมชาติโดยจำแนกเป็นถ่านหิน 1 เชื้อเพลิง 1 โรงไฟฟ้าก๊าซธรรมชาติ และโรงไฟฟ้าพลังความร้อนร่วม

เราทราบดีว่ามีบริการวาล์วอุตสาหกรรมมากมายสำหรับการผลิตกระแสไฟฟ้า วันนี้เราจะมาพูดถึงระบบของโรงไฟฟ้าที่วาล์วควบคุมของเราใช้งานเป็นหลัก

ระบบผลิตไฟฟ้าพลังความร้อนร่วม

วัฏจักรร่วมของกังหันแก๊ส หรือที่เรียกว่า วงจรรวมไอน้ำของแก๊ส หมายถึง กังหันแก๊สและกังหันไอน้ำที่รวมกันเป็นวิธีการผลิตกระแสไฟฟ้า ซึ่งประกอบด้วยกังหันเผาไหม้ (คอมเพรสเซอร์ เตาเผา กังหัน ระบบควบคุม และระบบเสริม) ความร้อน การกู้คืน หม้อน้ำ และกังหันไอน้ำ

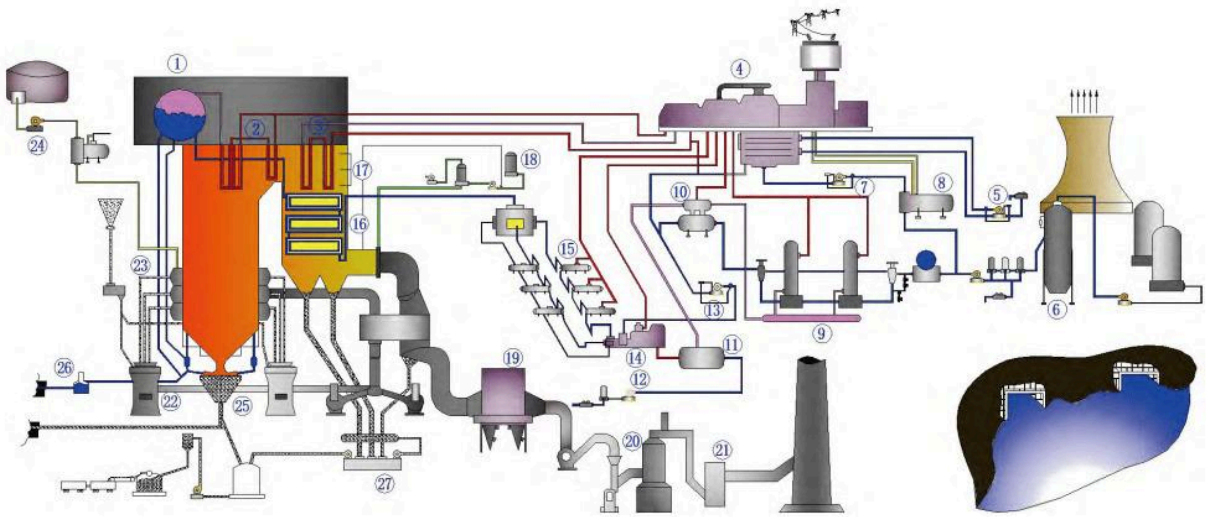


กระบวนการผลิตไฟฟ้าพลังความร้อนร่วม

ระบบผลิตไฟฟ้าจากถ่านหินทั่วไป

หน่วยผลิตไฟฟ้าจากถ่านหินส่วนใหญ่ประกอบด้วยระบบการเผาไหม้ (โดยมีหม้อไอน้ำเป็นแกนหลัก) ระบบไอน้ำ-น้ำ (รวมถึงปั๊มต่างๆ เครื่องทำน้ำร้อนป้อน คอนเดนเซอร์ ท่อ ผนังน้ำ ฯลฯ) ระบบไฟฟ้า (ส่วนใหญ่รวมถึง กังหันไอน้ำและหม้อแปลงไฟฟ้าหลัก) และระบบควบคุม เป็นต้น สองเครื่องแรกผลิตไอน้ำที่มีอุณหภูมิและความดันสูง ระบบไฟฟ้านำเสนอการแปลงพลังงานความร้อน

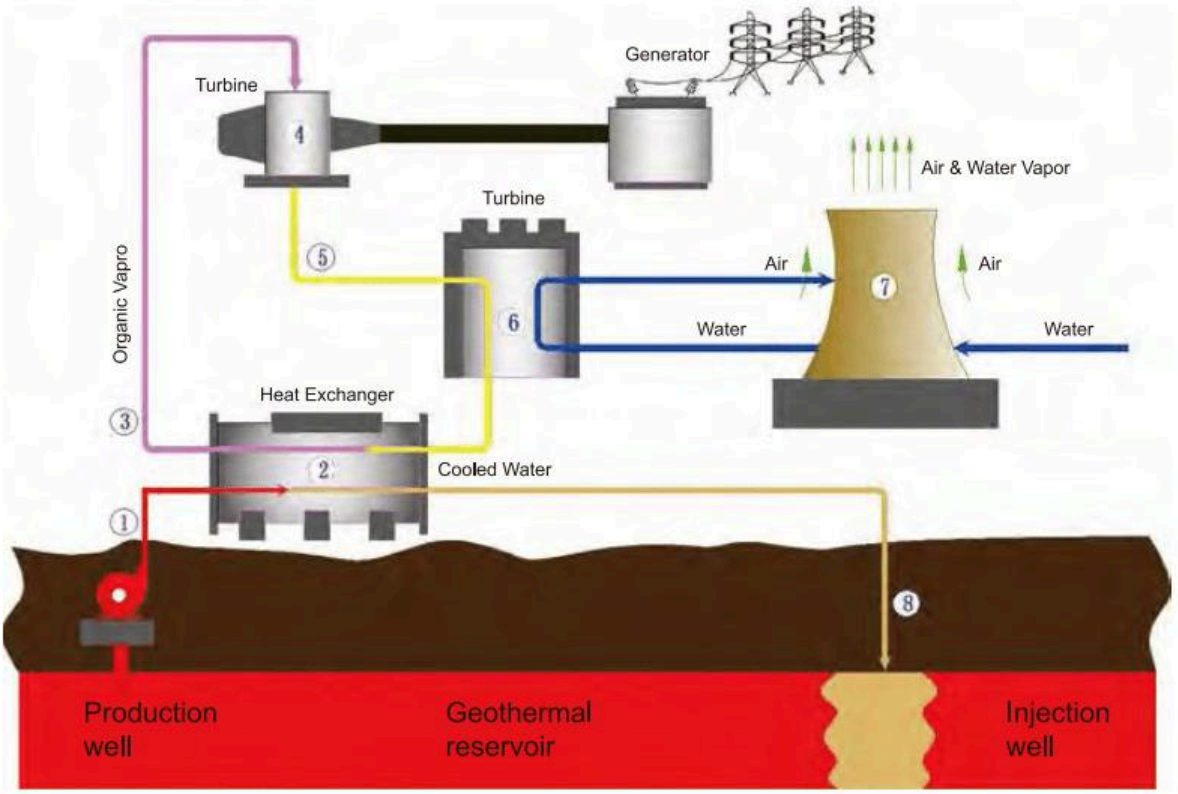
และพลังงานกลเป็นพลังงานไฟฟ้า ระบบควบคุมได้รับการออกแบบมาเพื่อให้แน่ใจว่าระบบทำงานได้อย่างปลอดภัย เหมาะสม และประหยัด



กระบวนการโรงไฟฟ้าถ่านหิน

โรงไฟฟ้าพลังความร้อนใต้พิภพแบบไบนารี

โรงไฟฟ้าพลังงานความร้อนใต้พิภพแบบไบนารี่ ไซเคิล แสดงถึงเทคโนโลยีล่าสุดในการใช้งานความร้อนใต้พิภพ เครื่องแลกเปลี่ยนความร้อนใช้เพื่อถ่ายโอนพลังงานน้ำร้อนไปยังของไหลที่นำความร้อนซึ่งมีจุดเดือดต่ำ ซึ่งนำไปสู่ประสิทธิภาพที่มากขึ้น น้ำจากโลกไม่เคยสัมผัสกับกังหัน ซึ่งช่วยลดการปล่อยก๊าซที่ติดตามสู่ชั้นบรรยากาศ

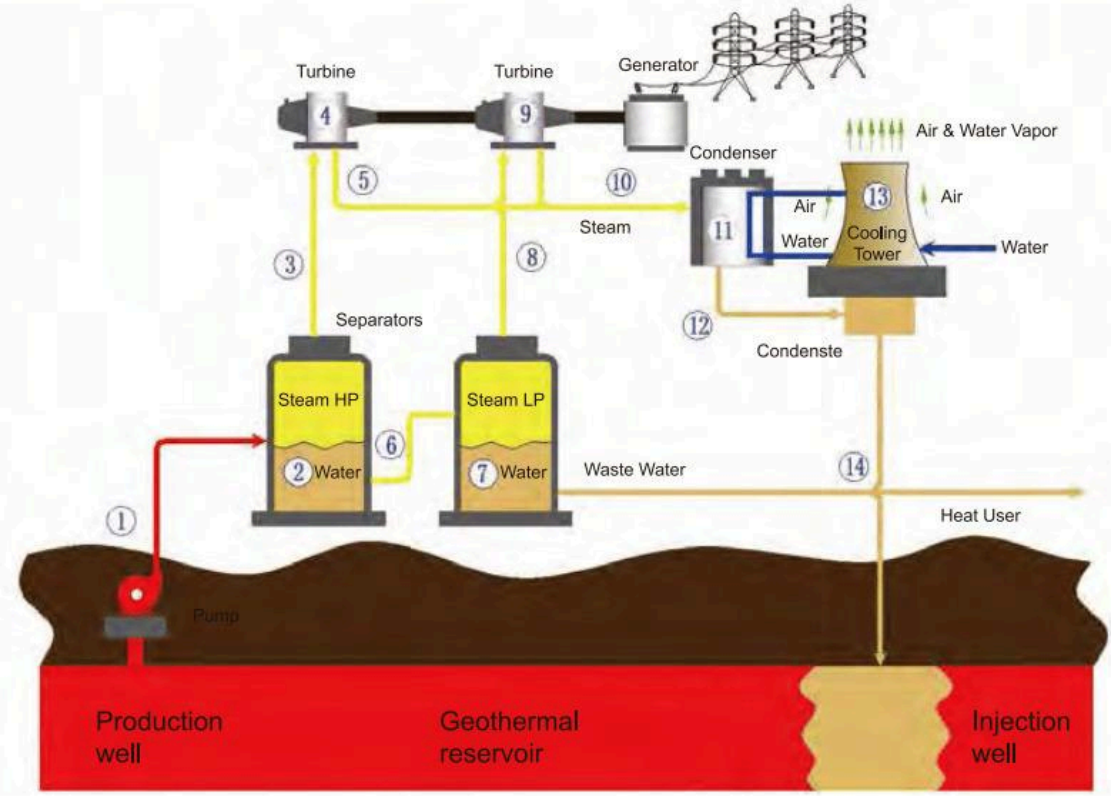


กระบวนการผลิตโรงไฟฟ้าพลังงานความร้อนใต้พิภพแบบไบนารี

โรงไฟฟ้าพลังงานความร้อนใต้พิภพแบบระเหยแฟลช

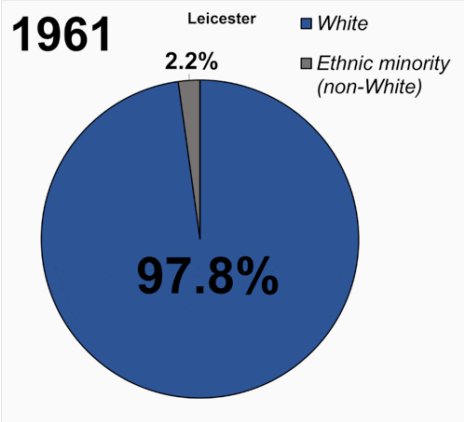
โรงไฟฟ้าพลังงานความร้อนใต้พิภพแบบระเหยวาบเป็นโรงไฟฟ้าพลังงานความร้อนใต้พิภพประเภทที่พบมากที่สุด โดยมีการสูบน้ำร้อนที่แรงดันสูงขึ้นสู่พื้นผิว สิ่งนี้ต้องการวาล์วและแอคชูเอเตอร์ที่สามารถจัดการกับเงื่อนไขได้ น้ำร้อนถูกสกัดด้วยภาชนะความดันที่มีความดันภายในต่ำกว่าน้ำร้อน

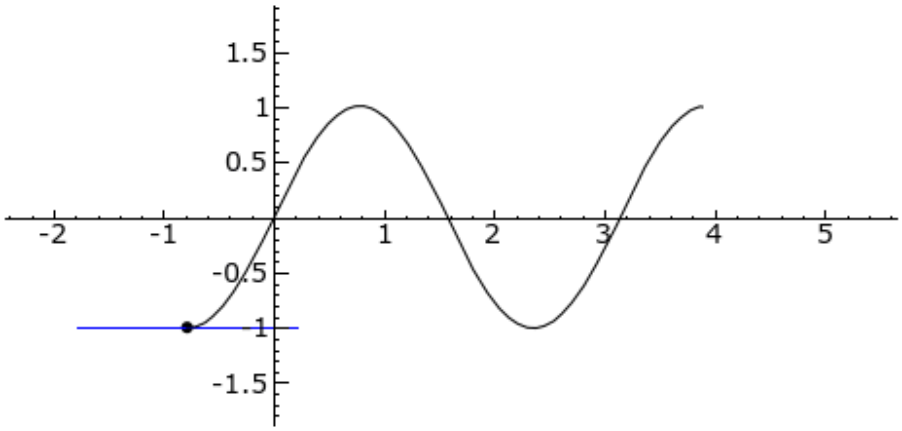
ในขณะที่ความแตกต่างของความดันจะเปลี่ยนน้ำให้กลายเป็นไอน้ำ ไอน้ำเข้ากังหันให้ทำงาน ก่อนควบแน่น ในที่สุดมันก็กลับสู่แหล่งกักเก็บความร้อนใต้พิภพผ่านบ่อฉีด

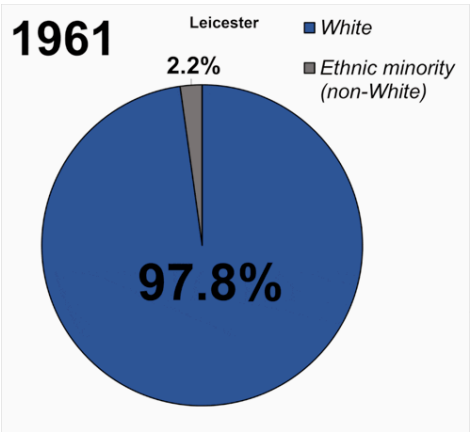


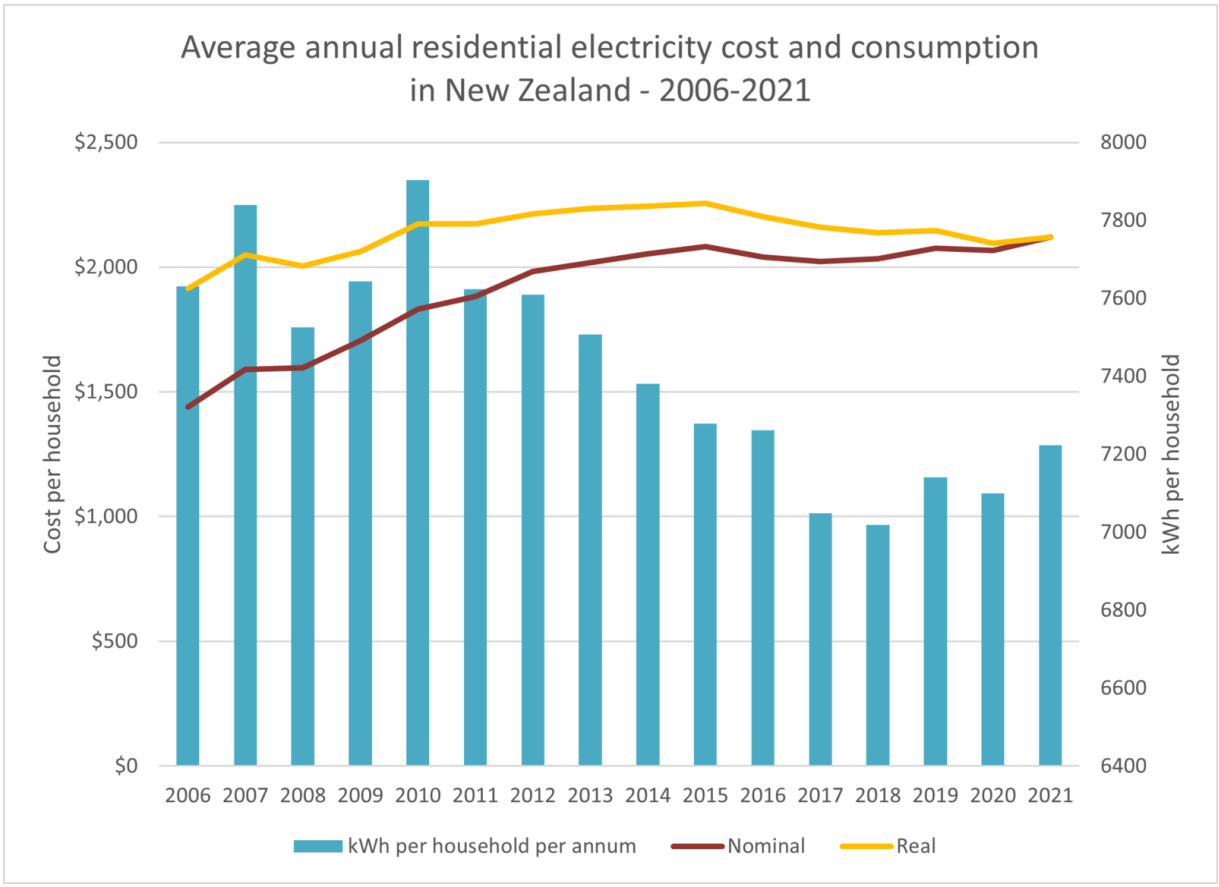
กระบวนการโรงไฟฟ้าพลังงานความร้อนใต้พิภพด้วยการระเหยอย่างรวดเร็ว

พบกับทีมงานของเราและเหตุใดคุณจึงเลือก THINKTANK


1961
Leicester
White
2.2%
Ethnic minority
(non-White)
97.8%


1.5
1
0.5
-2
-1
1
2
3
4
5
-0.5
-1
-1.5


1961
Leicester
White
2.2%
Ethnic minority
(non-White)
97.8%


Average annual residential electricity cost and consumption in New Zealand - 2006-2021
$2,500
$2,000
$1,500
$1,000
$500
$0
Cost per household
8000
7800
7600
7400
7200
7000
6800
6600
6400
kWh per household
2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021
kWh per household per annum
Nominal
Real

Visual Studio Code for the Web

Visual Studio Code for the Web provides a free, zero-install Microsoft Visual Studio Code experience running entirely in your browser, allowing you to quickly and safely browse source code repositories and make lightweight code changes. To get started, go to https://vscode.dev (https://vscode.dev) in your browser.

VS Code for the Web has many of the features of VS Code Desktop that you love, including search and syntax highlighting while browsing and editing, along with extension support to work on your codebase and make simpler edits. In addition to opening repositories, forks, and pull requests from source control providers like GitHub and Azure Repos, you can also work with code that is stored on your local machine.

VS Code for the Web runs entirely in your web browser, so there are certain limitations compared to the desktop experience, which you can read more about below.

The following video gives a quick overview of Visual Studio Code for the Web.

ดูใน

Relationship to VS Code Desktop

VS Code for the Web provides a browser-based experience for navigating files and repositories and committing lightweight code changes. However, if you need access to a runtime to run, build, or debug your code, you want to use platform features such as a terminal, or you want to run extensions that aren't supported in the web, we

recommend moving your work to the desktop application, GitHub Codespaces (https://github.com/features/codespaces), or using Remote - Tunnels for the full capabilities of VS Code. In addition, VS Code Desktop lets you use a full set of keyboard shortcuts not limited by your browser.

When you're ready to switch, you'll be able to "upgrade" to the full VS Code experience with a few clicks.

You can also switch between the Stable and Insiders versions of VS Code for the Web by selecting the gear icon, then **Switch to Insiders Version...**, or by navigating directly to https://insiders.vscode.dev (https://insiders.vscode.dev).

Opening a project

By navigating to https://vscode.dev (https://vscode.dev), you can create a new local file or project, work on an existing local project, or access source code repositories hosted elsewhere, such as on GitHub and Azure Repos (part of Azure DevOps).

You can create a new local file in the web just as you would in a VS Code Desktop environment, using **File** > **New File** from the Command Palette (`F1`).

GitHub repos

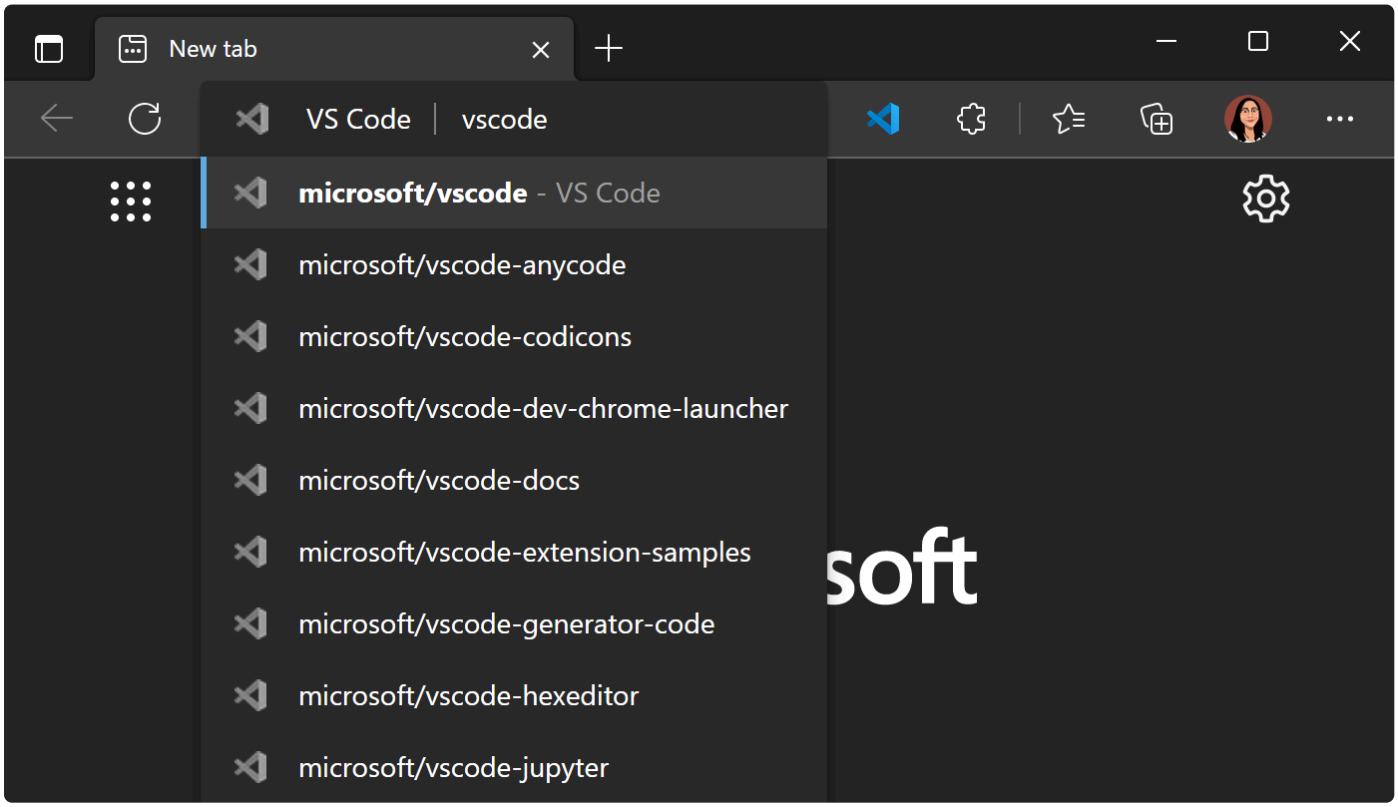
You can open a GitHub repository in VS Code for the Web directly from a URL, following the schema: `https://vscode.dev/github/<organization>/<repo>` . Using the VS Code repository (https://github.com/microsoft/vscode) as an example, this would look like: `https://vscode.dev/github/microsoft/vscode` .

This experience is delivered at a custom `vscode.dev/github` URL, which is powered by the GitHub Repositories (https://marketplace.visualstudio.com/items?itemName=GitHub.remotehub) extension (which is part of the broader Remote Repositories (https://marketplace.visualstudio.com/items?itemName=ms-vscode.remote-repositories) extension).

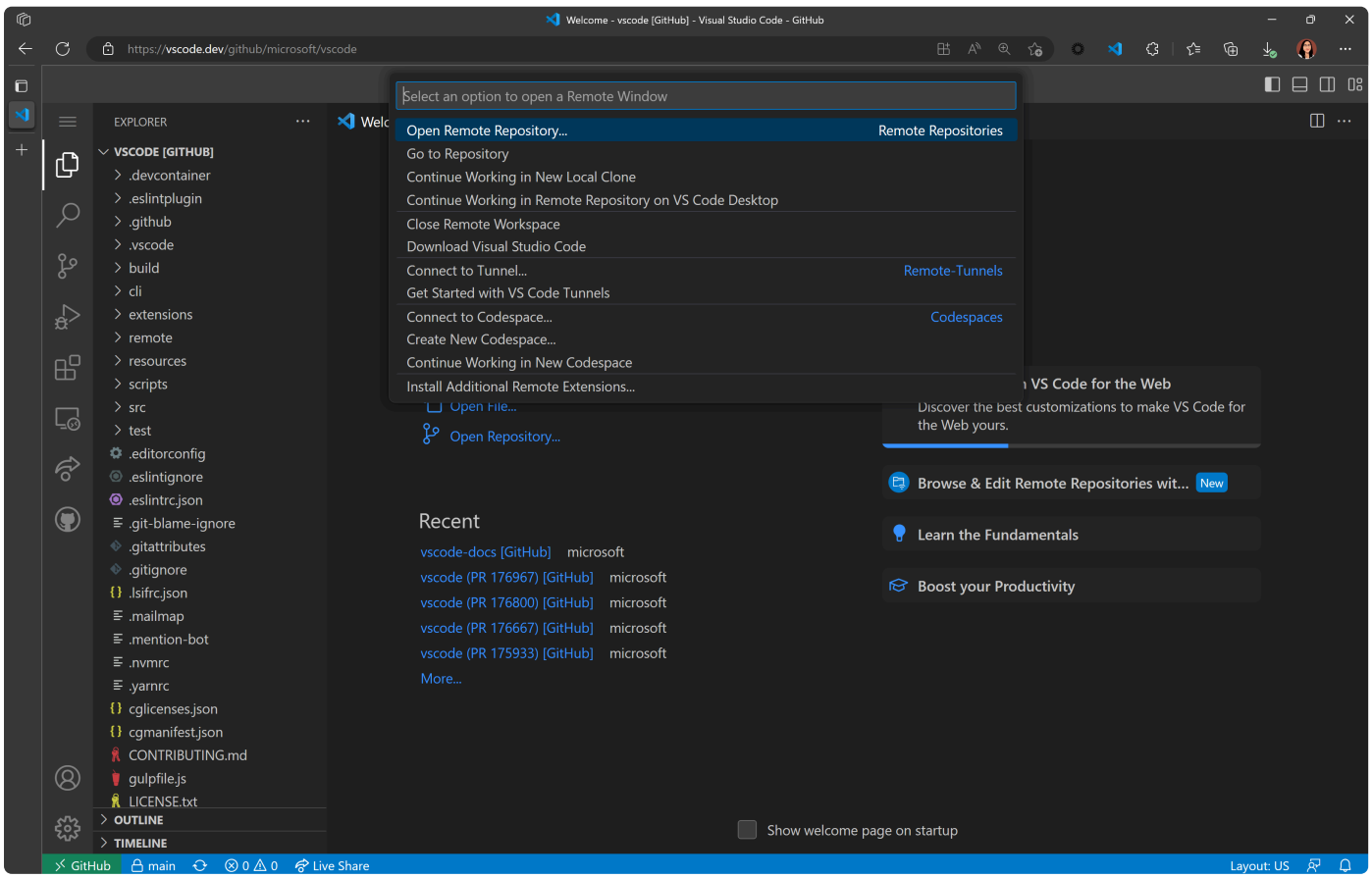
GitHub Repositories allows you to remotely browse and edit a repository from within the editor, without needing to pull code onto your local machine. You can learn more about the extension and how it works in our GitHub Repositories (/docs/sourcecontrol/github#_github-repositories-extension) guide.

> **Note**: The GitHub Repositories (https://marketplace.visualstudio.com/items?itemName=GitHub.remotehub) extension works in VS Code Desktop as well to provide fast repository browsing and editing. Once you have the extension installed, you can open a repo with the **GitHub Repositories: Open Repository...** command.

You can also open GitHub repositories in `vscode.dev` through your browser's search bar (aka omnibox) by installing the `vscode.dev` extension (https://chrome.google.com/webstore/detail/vs-code/kobakmhnkfaghloikphojodjebdelppk) for Chrome and Edge. Then, type `code` to activate the omnibox, followed by your repository's name. Suggestions are populated by your browser search history, so if the repo you want doesn't come up, you can also type in the fully qualified `<owner>/<repo>` name to open it, for example `microsoft/vscode` .



If you're already in VS Code for the Web at https://vscode.dev (https://vscode.dev), you can alternatively navigate to different repos via the Remote Repositories (https://marketplace.visualstudio.com/items?itemName=ms-vscode.remote-repositories) extension commands. Select the remote indicator in the lower left of the Status bar, and you'll be presented with the **Open Remote Repository...** command.



Azure Repos

You can open Azure Repos just like Github repos in VS Code for the Web.

When you navigate to a URL with the schema `https://vscode.dev/azurerepos/<organization>/<project>/<repo>`, you will be able to read, search the files in the repo, and commit your changes to Azure Repos. You can fetch, pull, and sync changes, and view branches.

You can open any repository, branch, or tag from Azure Repos in VS Code for the Web by prefixing `vscode.dev` to the Azure Repos URL.

Alternatively, when you are on an Azure DevOps repository or pull request, you can press (`.`) to open it in VS Code for the Web.

More custom URLs

Like in the desktop, you can customize VS Code for the Web through a rich ecosystem of extensions that support just about every back end, language, and service. `vscode.dev` includes URLs that provide shortcuts to common experiences.

We've explored a couple of URLs already (`vscode.dev/github` and `vscode.dev/azurerepos`). Here's a more complete list:

Expand table

Service	URL Structure	Docs
GitHub	`/github/<org>/<repo>`	More info above
Azure Repos	`/azurerepos/<org>/<project>/<repo>`	More info above
Visual Studio Live Share	`/editor/liveshare/<sessionId>`	More info below
Visual Studio Marketplace	`/editor/marketplace/<marketplacePublisher>/<extensionId>/<extensionVersion>`	Example route (https://insiders.vscode.dev/editor/marketplace/Brigit/devcontainer-image-convert/0.0.1) to edit this extension (https://marketplace.visualstudio.com/items?itemName=Brigit.devcontainer-image-convert)
Power Pages	`/power/pages`	Power Pages docs (https://learn.microsoft.com/power-pages/configure/visual-studio-code-editor)
Profiles	`/editor/profile/github/<GUID>`	Profiles docs (/docs/configure/profiles#_save-as-a-github-gist)
Themes	`/editor/theme/<extensionId>`	More info below

Service	URL Structure	Docs
MakeCode	`/edu/makecode`	MakeCode docs (https://arcade.makecode.com/vscode)
VS Code for Education	`/edu`	VS Code for Education landing page (https://vscodeedu.com/)
Azure Machine Learning (AML)	`/+ms-toolsai.vscode-ai-remote-web`	AML docs (https://learn.microsoft.com/azure/machine-learning/how-to-launch-vs-code-remote?view=azureml-api-2&tabs=vscode-web)
Azure	`/azure`	VS Code for Azure (https://code.visualstudio.com/docs/azure/vscodeforweb)

Please note that some URLs must be entered in a specific way (for example, `vscode.dev/editor/liveshare` requires an active Live Share session). Please review each service's documentation for specific access and usage information.

There's more information on some of these URLs below.

Themes

You can share and experience color themes through VS Code for the Web through the URL schema: `https://vscode.dev/editor/theme/<extensionId>` .

For instance, you can go to https://vscode.dev/editor/theme/sdras.night-owl (https://vscode.dev/editor/theme/sdras.night-owl) to experience the Night Owl theme (https://marketplace.visualstudio.com/items?itemName=sdras.night-owl) without having to go through the download and install process.

> Note: The color theme URL schema works for themes that are fully declarative (no code).

An extension can define multiple themes. You can use the schema `/editor/theme/<extensionId>/<themeName>` . If no `themeName` is specified, VS Code for the Web will take the first theme.

As a theme author, you can add the following badge to your extension readme to allow users to easily try out your theme in VS Code for the Web (replacing `<extensionId>` with your theme extension's unique identifier):

Markdown

```
[![Preview in vscode.dev](https://img.shields.io/badge/preview%20in-vscode.dev-blue)]
(https://vscode.dev/editor/theme/<extensionId>)
```

Visual Studio Live Share

Live Share (https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare) guest sessions are available in the browser through the `https://vscode.dev/editor/liveshare` URL. The `sessionId` will be passed to the extension to make joining a seamless experience.

Continue working in a different environment

In some cases, you will want to access a different environment that has the ability to run code. You can switch to working on a repository in a development environment that has support for a local file system and full language and development tooling.

The GitHub Repositories extension makes it easy for you to clone the repository locally, reopen it on the desktop, or create a GitHub codespace for the current repository (if you have the GitHub Codespaces (https://marketplace.visualstudio.com/items?itemName=GitHub.codespaces) extension installed and access to create GitHub codespaces). To do this, use the **Continue Working On...** command available from the Command Palette (`F1`) or click on the Remote indicator in the Status bar.

Saving and sharing work

When working on a local file in the web, your work is saved automatically if you have Auto Save (/docs/editing/codebasics#_save--auto-save) enabled. You can also save manually as you do when working in desktop VS Code (for example **File** > **Save**).

When working on a remote repository, your work is saved in the browser's local storage until you commit it. If you open a repo or pull request using GitHub Repositories, you can push your changes in the Source Control view to persist any new work.

You can also continue working in other environments via Continue Working On.

The first time that you use **Continue Working On** with uncommitted changes, you will have the option to bring your edits to your selected development environment using **Cloud Changes**, which uses a VS Code service to store your pending changes. This is described further in the GitHub Repositories (/docs/sourcecontrol/github#_continue-working-on) doc.

Use your own compute instance with Remote Tunnels

You may develop against another machine in VS Code for the Web using the Remote - Tunnels (https://marketplace.visualstudio.com/items?itemName=ms-vscode.remote-server) extension.

The Remote - Tunnels extension lets you connect to a remote machine, like a desktop PC or virtual machine (VM), via a secure tunnel. You can then securely connect to that machine from anywhere, without the requirement of SSH. This lets you "bring your own compute" to vscode.dev, enabling additional scenarios like running your code in the browser.

You may learn more about Remote - Tunnels in its documentation (/docs/remote/tunnels).

Safe exploration

VS Code for the Web runs entirely in your web browser's sandbox and offers a very limited execution environment.

When accessing code from remote repositories, the web editor doesn't "clone" the repo, but instead loads the code by invoking the services' APIs directly from your browser; this further reduces the attack surface when cloning untrusted repositories.

When working with local files, VS Code for the Web loads them through your browser's file system access APIs, which limit the scope of what the browser can access.

Run anywhere

Similar to GitHub Codespaces (/docs/remote/codespaces), VS Code for the Web can run on tablets, like iPads.

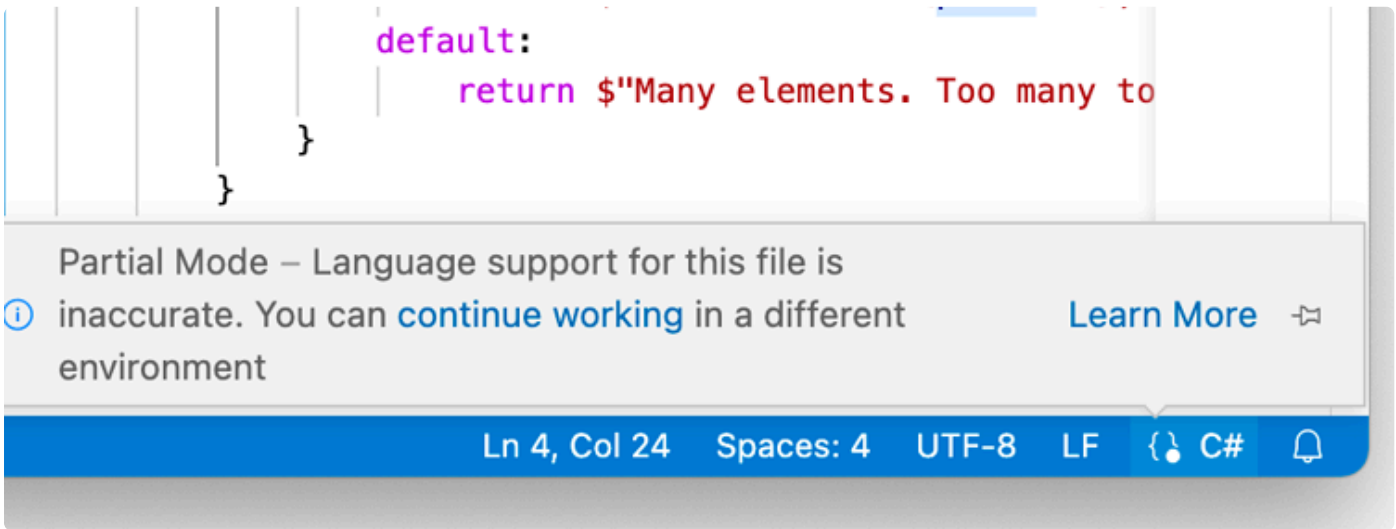
Language support

Language support is a bit more nuanced on the web, including code editing, navigation, and browsing. The desktop experiences are typically powered by language services and compilers that expect a file system, runtime, and compute environment. In the browser, these experiences are powered by language services running in the browser that provide source code tokenization and syntax colorization, completions, and many single-file operations.

Generally, experiences fall into the following categories:

- **Good:** For most programming languages, VS Code for the Web gives you code syntax colorization, text-based completions, and bracket pair colorization. Using a Tree-sitter (https://tree-sitter.github.io/tree-sitter) syntax tree through the anycode extension (https://marketplace.visualstudio.com/items?itemName=ms-vscode.anycode), we're able to provide additional experiences such as **Outline/Go to Symbol** and **Symbol Search** for popular languages such as C/C++, C#, Java, PHP, Rust, and Go.
- **Better:** The TypeScript, JavaScript, and Python experiences are all powered by language services that run natively in the browser. With these programming languages, you'll get the "**Good**" experience plus rich single file completions, semantic highlighting, syntax errors, and more.
- **Best:** For many "webby" languages, such as JSON, HTML, CSS, and LESS, etc., the coding experience in vscode.dev is nearly identical to the desktop (including Markdown preview!).

You can determine the level of language support in your current file through the Language Status Indicator in the Status bar:

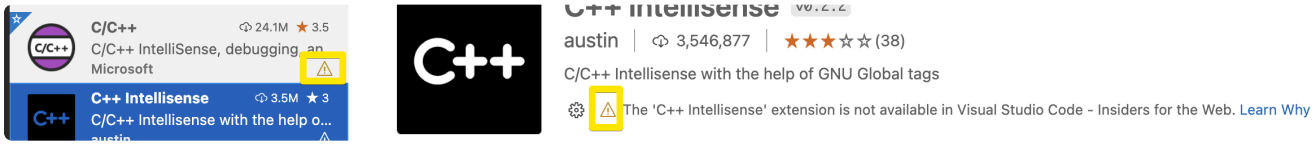


Limitations

Since VS Code for the Web runs completely within the browser, some experiences will naturally be more constrained when compared to what you can do in the desktop app. For example, the terminal and debugger are not available, which makes sense since you can't compile, run, and debug a Rust or Go application within the browser sandbox.

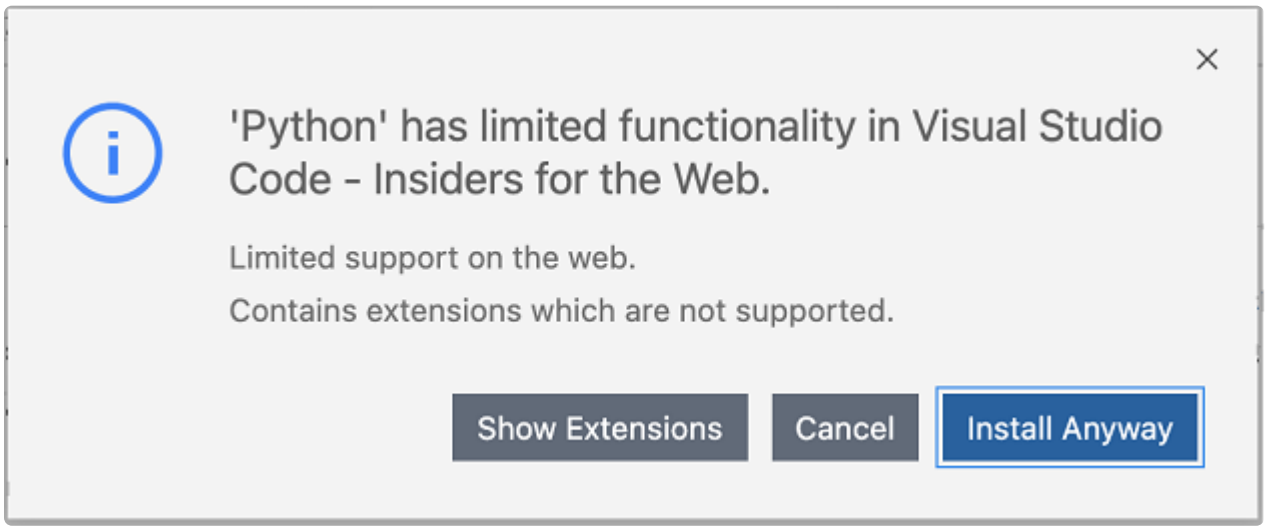
Extensions

Only a subset of extensions can run in the browser. You can use the Extensions view to install extensions in the web, and extensions that cannot be installed will have a warning icon and **Learn Why** link. We expect more extensions to become enabled over time.



When you install an extension, it is saved in the browser's local storage. You can ensure your extensions are synced across VS Code instances, including different browsers and even the desktop, by enabling Settings Sync (/docs/configure/settings-sync).

When an Extension Pack contains extensions that do not run in the browser sandbox, you will get an informational message with the option to see the extensions included in the pack.



When extensions are executed in the browser sandbox, they are more restricted. Extensions that are purely declarative, such as most themes, snippets, or grammars, can run unmodified and are available in VS Code for the Web without any modification from the extension authors. Extensions that are running code need to be updated to support running in the browser sandbox. You can read more about what is involved to support extensions in the browser in the web extension authors guide (/api/extension-guides/web-extensions).

There are also extensions that run in the browser with partial support only. A good example is a language extension that restricts its support (/docs/nodejs/working-with-javascript#_partial-intellisense-mode) to single files or the currently opened files.

File system API

Edge and Chrome today support the File System API (https://developer.mozilla.org/docs/Web/API/File_System_Access_API), allowing web pages to access the local file system. If your browser does not support the File System API, you cannot open a folder locally, but you can

open files instead.

Browser support

You can use VS Code for the Web in the latest versions of Chrome, Edge, Firefox, and Safari. Older versions of each browser may not work - we only guarantee support for the latest version.

> **Tip:** One way to check the compatible browser version is to look at the version of Playwright (https://playwright.dev/) currently used for testing VS Code and review its supported browser versions. You can find the currently used Playwright version in the VS Code repo's package.json (https://github.com/microsoft/vscode/blob/main/package.json) file at `devDependencies/@playwright/test`. Once you know the Playwright version, for example `1.37`, you can then review the **Browser Versions** section in their Release notes (https://playwright.dev/docs/release-notes).

Webviews might appear differently or have some unexpected behavior in Firefox and Safari. You can view issue queries in the VS Code GitHub repo to track issues related to specific browsers, such as with the Safari label (https://github.com/microsoft/vscode/labels/safari) and Firefox label (https://github.com/microsoft/vscode/labels/firefox).

There are additional steps you can take to improve your browser experience using VS Code for the Web. Review the Additional browser setup section for more information.

Mobile support

You can use VS Code for the Web on mobile devices, but smaller screens may have certain limitations.

Keyboard shortcuts

Certain keyboard shortcuts may also work differently in the web.

Expand table

Issue	Reason
`Ctrl+Shift+P` won't launch the Command Palette in Firefox.	`Ctrl+Shift+P` is reserved in Firefox. As a workaround, use `F1` to launch the Command Palette.
`Ctrl+N` for new file doesn't work in web.	`Ctrl+N` opens a new window instead. As a workaround, you can use `Ctrl+Alt+N` (`Cmd+Alt+N` on macOS).
`Ctrl+W` for closing an editor doesn't work in web.	`Ctrl+W` closes the current tab in browsers. As a workaround, you can use `Ctrl+Shift+Alt+N` (`Cmd+Shift+Alt+N` on macOS).
`Ctrl+Shift+B` will not toggle the favorites bar in the browser.	VS Code for the Web overrides this and redirects to the "Build" menu in the Command Palette.

Issue	Reason
`Alt+Left` and `Alt+Right` should navigate within the editor but may incorrectly trigger tab history navigation.	If focus is outside the editor, these shortcuts trigger tab history navigation instead.

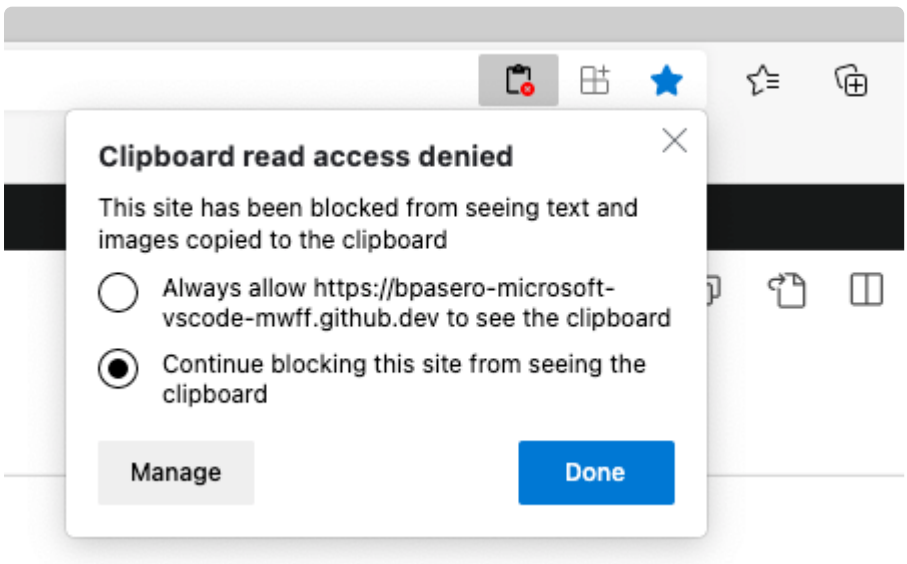
Additional browser setup

There are additional browser configuration steps you can take when working with VS Code in a browser.

Opening new tabs and windows

In certain cases, you may need to open a new tab or window while working in VS Code for the Web. VS Code might ask you for permission to access the clipboard when reading from it. Depending on your browser, you may grant access to the clipboard or otherwise allow for pop-up windows in different ways:

- Chrome, Edge, Firefox: Search for "site permissions" in your browser's settings, or look for the following option in the address bar on the right:



- Safari: In the Safari browser, go to **Preferences...** > **Websites** > **Pop-up Windows** > the domain you're accessing (for example, `vscode.dev`), and select **Allow** from the dropdown.

Was this documentation helpful?

Yes No

Still need help?

Ask the community(https://stackoverflow.com/questions/tagged/vscode)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/setup/vscode-web.md).

01/08/2026

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

Network Connections in Visual Studio Code

Visual Studio Code is built on top of Electron (https://www.electronjs.org) and benefits from all the networking stack capabilities of Chromium (https://www.chromium.org/). This also means that VS Code users get much of the networking support available in Google Chrome (https://www.google.com/chrome/index.html).

Common hostnames

A handful of features within VS Code require network communication to work, such as the auto-update mechanism, querying and installing extensions, and telemetry. For these features to work properly in a proxy environment, you must have the product correctly configured.

If you are behind a firewall that needs to allow specific domains used by VS Code, here's the list of hostnames you should allow communication to go through:

- `update.code.visualstudio.com` - Visual Studio Code download and update server
- `code.visualstudio.com` - Visual Studio Code documentation
- `go.microsoft.com` - Microsoft link forwarding service
- `marketplace.visualstudio.com` - Visual Studio Marketplace
- `*.gallery.vsassets.io` - Visual Studio Marketplace
- `*.gallerycdn.vsassets.io` - Visual Studio Marketplace
- `rink.hockeyapp.net` - Crash reporting service
- `bingsettingssearch.trafficmanager.net` - In-product settings search
- `vscode.search.windows.net` - In-product settings search
- `raw.githubusercontent.com` - GitHub repository raw file access
- `vsmarketplacebadges.dev` - Visual Studio Marketplace badge service
- `*.vscode-cdn.net` - Visual Studio Code CDN
- `vscode.download.prss.microsoft.com` - Visual Studio Code download CDN
- `download.visualstudio.microsoft.com` - Visual Studio download server, provides dependencies for some VS Code extensions (C++, C#)
- `vscode-sync.trafficmanager.net` - Visual Studio Code Settings Sync service
- `vscode-sync-insiders.trafficmanager.net` - Visual Studio Code Settings Sync service (Insiders)
- `vscode.dev` - Used as a fallback when logging in with GitHub or Microsoft for an extension or Settings Sync (just `vscode.dev/redirect`)
- `*.vscode-unpkg.net` - Used when loading web extensions
- `default.exp-tas.com` - Visual Studio Code Experiment Service, used to provide experimental user experiences

Proxy server support

VS Code has exactly the same proxy server support as Google Chromium. Here's a snippet from Chromium's documentation (https://www.chromium.org/developers/design-documents/network-settings):

```Text
"The Chromium network stack uses the system network settings so that users and admini
strators can control the network settings of all applications easily. The network set
tings include:

 - proxy settings
 - SSL/TLS settings
 - certificate revocation check settings
 - certificate and private key stores"
```

This means that your proxy settings should be picked up automatically.

Otherwise, you can use the following command-line arguments to control your proxy settings:

```Bash
# Disable proxy
--no-proxy-server

# Manual proxy address
--proxy-server=<scheme>=<uri>[:<port>][;...] | <uri>[:<port>] | "direct://"

# Manual PAC address
--proxy-pac-url=<pac-file-url>

# Disable proxy per host
--proxy-bypass-list=(<trailing_domain>|<ip-address>)[:<port>][;...]
```

To learn more about these command-line arguments, see Chromium Network Settings (https://www.chromium.org/developers/design-documents/network-settings).

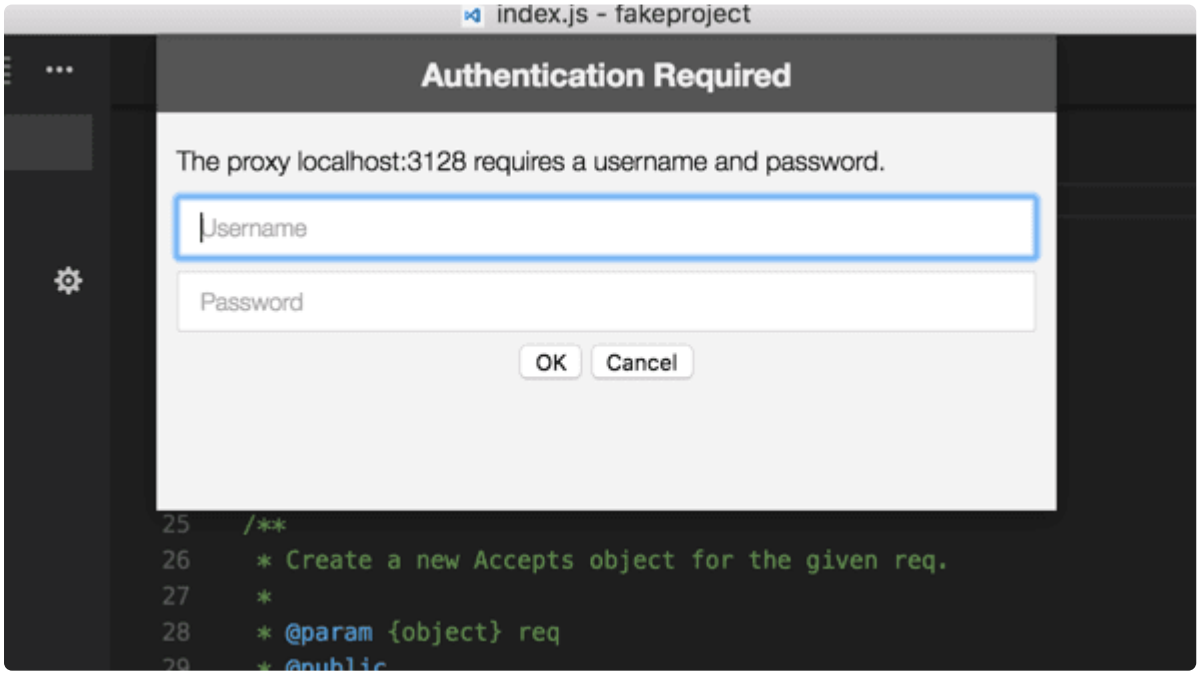
Authenticated proxies

Authenticated proxies should work seamlessly within VS Code with the addition of PR #22369 (https://github.com/microsoft/vscode/pull/22369).

The authentication methods supported are:

- Basic
- Digest
- NTLM
- Negotiate

When using VS Code behind an authenticated HTTP proxy, the following authentication popup should appear:



Note that SOCKS5 proxy authentication support isn't implemented yet; you can follow the issue in Chromium's issue tracker (https://bugs.chromium.org/p/chromium/issues/detail?id=256785).

See Chromium HTTP authentication (https://www.chromium.org/developers/design-documents/http-authentication) to read more about HTTP proxy authentication within VS Code.

SSL certificates

Often HTTPS proxies rewrite SSL certificates of the incoming requests. Chromium was designed to reject responses which are signed by certificates which it doesn't trust. If you hit any SSL trust issues, there are a few options available for you:

- Since Chromium uses the OS's certificate trust infrastructure, the preferred option is to add your proxy's certificate to your OS's trust chain. See the Chromium Root Certificate Policy (https://www.chromium.org/Home/chromium-security/root-ca-policy) documentation to learn more.
- If your proxy runs in `localhost`, you can always try the --allow-insecure-localhost (https://peter.sh/experiments/chromium-command-line-switches/#allow-insecure-localhost) command-line flag.
- If all else fails, you can tell VS Code to ignore all certificate errors using the --ignore-certificate-errors (https://peter.sh/experiments/chromium-command-line-switches/#ignore-certificate-errors) command-line flag. **Warning:** This is **dangerous** and **not recommended**, since it opens the door to security issues.

> **Note for Linux users**: To add your proxy's certificate on Linux, you need to add it to the system trust store and the NSS trust store. The exact steps vary by distribution:
>
> - For Ubuntu/Debian: Copy the certificate to `/usr/local/share/ca-certificates/` and run `sudo update-ca-certificates`
> - For RHEL/CentOS/Fedora: Use `sudo trust anchor --store <certificate-file>` or place in `/etc/pki/ca-trust/source/anchors/` and run `sudo update-ca-trust`
> - Additionally, use `certutil -A -n "ProxyCA" -t "CT,," -i <certificate-file> -d sql:$HOME/.pki/nssdb` to add it to the NSS trust store.

Legacy proxy server support

Extensions don't benefit yet from the same proxy support that VS Code supports. You can follow this issue's development in GitHub (https://github.com/microsoft/vscode/issues/12588).

Similarly to extensions, a few other VS Code features don't yet fully support proxy networking, namely the CLI interface. The CLI interface is what you get when running `code --install-extension vscodevim.vim` from a command prompt or terminal. You can follow this issue's development in GitHub (https://github.com/microsoft/vscode/issues/29910).

Due to both of these constraints, the ⚙ `http.proxy (vscode://settings/http.proxy)`, ⚙ `http.proxyStrictSSL (vscode://settings/http.proxyStrictSSL)` and ⚙ `http.proxyAuthorization (vscode://settings/http.proxyAuthorization)` variables are still part of VS Code's settings, yet they are only respected in these two scenarios.

Troubleshooting

Here are some helpful links that might help you troubleshoot networking issues in VS Code:

- Network Settings (https://www.chromium.org/developers/design-documents/network-settings)
- Debugging problems with the network proxy (https://www.chromium.org/developers/design-documents/network-stack/debugging-net-proxy)
- Configuring a SOCKS proxy server in Chrome (https://www.chromium.org/developers/design-documents/network-stack/socks-proxy)
- Proxy settings and fallback (Windows) (https://www.chromium.org/developers/design-documents/network-stack/proxy-settings-fallback)

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

Visual Studio Code on Linux

Installation

1 Download and install Visual Studio Code for your Linux distribution

> Note
> VS Code ships monthly releases and supports auto-update when a new release is available.

2 Install additional components (/docs/setup/additional-components)

Install Git, Node.js, TypeScript, language runtimes, and more.

3 Install VS Code extensions from the Visual Studio Marketplace (https://marketplace.visualstudio.com/VSCode)

Customize VS Code with themes, formatters, language extensions and debuggers for your favorite languages, and more.

4 Enable AI features (/docs/copilot/setup)

> Tip
> If you don't yet have a Copilot subscription, you can use Copilot for free by signing up for the Copilot Free plan (https://github.com/github-copilot/signup) and get a monthly limit of inline suggestions and chat interactions.

5 Get started with the VS Code tutorial (/docs/getstarted/getting-started)

Discover the user interface and key features of VS Code.

Install VS Code on Linux

Debian and Ubuntu based distributions

1 The easiest way to install Visual Studio Code for Debian/Ubuntu based distributions is to download and install the .deb package (64-bit) (https://go.microsoft.com/fwlink/?LinkID=760868), either through the graphical software center if it's available, or through the command line with:

Bash

```
sudo apt install ./<file>.deb

# If you're on an older Linux distribution, you will need to run this instead:
# sudo dpkg -i <file>.deb
# sudo apt-get install -f # Install dependencies
```

> **Note**
> Other binaries are also available on the VS Code download page (/Download).

When you install the .deb package, it prompts to install the apt repository and signing key to enable auto-updating using the system's package manager.

2 To automatically install the apt repository and signing key, such as on a non-interactive terminal, run the following command first:

Bash

```
echo "code code/add-microsoft-repo boolean true" | sudo debconf-set-selections
```

3 To manually install the apt repository:

1 Run the following script to install the signing key:

Bash

```
sudo apt-get install wget gpg &&
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor
> microsoft.gpg &&
sudo install -D -o root -g root -m 644 microsoft.gpg /usr/share/keyrings/mi
crosoft.gpg &&
rm -f microsoft.gpg
```

2 Create a `/etc/apt/sources.list.d/vscode.sources` file with the following contents to add a reference to the upstream package repository:

Text

```
Types: deb
URIs: https://packages.microsoft.com/repos/code
Suites: stable
Components: main
Architectures: amd64,arm64,armhf
Signed-By: /usr/share/keyrings/microsoft.gpg
```

3 Lastly, update the package cache and install the package:

```bash
sudo apt install apt-transport-https &&
sudo apt update &&
sudo apt install code # or code-insiders
```

> **Note**
> Due to the manual signing process and the publishing system we use, the Debian repo could lag behind by up to three hours and not immediately get the latest version of VS Code.

RHEL, Fedora, and CentOS based distributions

We currently ship the stable 64-bit VS Code for RHEL, Fedora, or CentOS based distributions in a yum repository.

1 Install the key and yum repository by running the following script:

```bash
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc &&
echo -e "[code]\nname=Visual Studio Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\nautorefresh=1\ntype=rpm-md\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc" | sudo tee /etc/yum.repos.d/vscode.repo > /dev/null
```

2 Then update the package cache and install the package using `dnf` (Fedora 22 and above):

```bash
dnf check-update &&
sudo dnf install code # or code-insiders
```

Or on older versions using `yum`:

```bash
yum check-update &&
sudo yum install code # or code-insiders
```

> **Note**
> Due to the manual signing process and the publishing system we use, the yum repo could lag behind by up to three hours and not immediately get the latest version of VS Code.

Snap

VS Code is officially distributed as a Snap package in the Snap Store (https://snapcraft.io/store)

(https://snapcraft.io/code)

You can install it by running:

```Bash
sudo snap install --classic code # or code-insiders
```

Once installed, the Snap daemon takes care of automatically updating VS Code in the background. You get an in-product update notification whenever a new update is available.

> **Note**
> If `snap` isn't available in your Linux distribution, check the Installing snapd guide (https://docs.snapcraft.io/installing-snapd), which can help you get that set up.

Learn more about *snaps* from the official Snap Documentation (https://docs.snapcraft.io).

openSUSE and SLE-based distributions

The yum repository mentioned previously also works for openSUSE and SLE-based systems.

1. Install the key and yum repository by running the following script:

```Bash
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc &&
echo -e "[code]\nname=Visual Studio Code\nbaseurl=https://packages.microsoft.co
m/yumrepos/vscode\nenabled=1\nautorefresh=1\ntype=rpm-md\ngpgcheck=1\ngpgkey=htt
ps://packages.microsoft.com/keys/microsoft.asc" | sudo tee /etc/zypp/repos.d/vsc
ode.repo > /dev/null
```

2. Then update the package cache and install the package using:

```Bash
sudo zypper install code
```

AUR package for Arch Linux

There is a community-maintained Arch User Repository package for VS Code (https://aur.archlinux.org/packages/visual-studio-code-bin).

To get more information about the installation from the AUR, consult the following wiki entry: Install AUR Packages (https://wiki.archlinux.org/index.php/Arch_User_Repository).

Nix package for NixOS (or any Linux distribution using Nix package manager)

There is a community-maintained VS Code Nix package (https://github.com/NixOS/nixpkgs/blob/master/pkgs/applications/editors/vscode/vscode.nix) in the nixpkgs repository.

To install it by using Nix:

1. Set `allowUnfree` option to true in your `config.nix`
2. Run the following command:

```Bash
nix-env -i vscode
```

Install the .rpm package manually

You can manually download and install the VS Code .rpm package (64-bit) (https://go.microsoft.com/fwlink/?LinkID=760867), however, auto-updating won't work unless the repository above is installed.

Once downloaded, the `.rpm` package can be installed by using your package manager, for example with `dnf`:

```Bash
sudo dnf install <file>.rpm
```

> Note
> Other binaries are also available on the VS Code download page (/Download).

Updates

VS Code ships monthly and you can see when a new release is available by checking the release notes (/updates). If the VS Code repository was installed correctly, then your system package manager should handle auto-updating in the same way as other packages on the system.

> Note
> Updates are automatic and run in the background for the Snap package.

Configure VS Code as the default text editor

xdg-open

You can set the default text editor for text files (`text/plain`) that is used by `xdg-open` with the following command:

```bash
xdg-mime default code.desktop text/plain
```

Debian alternatives system

Debian-based distributions allow setting a default **editor** by using the Debian alternatives system (https://wiki.debian.org/DebianAlternatives), without concern for the MIME type. You can set this by running the following command and selecting `code`:

```bash
sudo update-alternatives --set editor /usr/bin/code
```

If you've installed VS Code with the Snap package, use this command instead:

```bash
sudo update-alternatives --set editor /snap/bin/code
```

If VS Code doesn't show up as an alternative to the default `editor`, you need to register it:

```bash
sudo update-alternatives --install /usr/bin/editor editor $(which code) 10
```

Use the custom title bar

The custom title bar provides many benefits, including great theming support and better accessibility through keyboard navigation and screen readers. These benefits might not always translate as well to the Linux platform. Linux has various desktop environments and window managers that can make the VS Code theming look foreign to users. Therefore, the custom title bar isn't enabled by default on Linux.

For users needing the accessibility improvements, we recommend enabling the custom title bar when running in accessibility mode using a screen reader.

You can manually configure the title bar with the **Window: Title Bar Style** (`window.titleBarStyle (vscode://settings/window.titleBarStyle)`) setting:

- `custom`: Use the custom title bar.
- `native`: Use the operating system's title bar.

Windows as a Linux developer machine

Another option for Linux development with VS Code is to use a Windows machine with the Windows Subsystem for Linux (https://learn.microsoft.com/windows/wsl/install) (WSL).

Windows Subsystem for Linux

With WSL, you can install and run Linux distributions on Windows to develop and test your source code on Linux, while still working locally on a Windows machine. WSL supports Linux distributions such as Ubuntu, Debian, SUSE, and Alpine available from the Microsoft Store.

When coupled with the WSL (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl) extension, you get full VS Code editing and debugging support while running in the context of a Linux distro on WSL.

See the Developing in WSL (/docs/remote/wsl) documentation to learn more, or try the Working in WSL (/docs/remote/wsl-tutorial) introductory tutorial.

Next steps

Once you have installed VS Code, these topics will help you learn more about it:

- VS Code tutorial (/docs/getstarted/getting-started) - A quick hands-on tour of the key features of VS Code.
- Tips and Tricks (/docs/getstarted/tips-and-tricks) - A collection of productivity tips for working with VS Code.
- AI-assisted coding (/docs/copilot/overview) - Learn about using GitHub Copilot in VS Code to help you write code faster.

Common questions

Debian and moving files to trash

If you see an error when deleting files from the VS Code Explorer on the Debian operating system, it might be because the trash implementation that VS Code is using is not there.

Run these commands to solve this issue:

Bash

```
sudo apt-get install gvfs libglib2.0-bin
```

Conflicts with VS Code packages from other repositories

Some distributions, for example Pop!_OS (https://pop.system76.com) provide their own `code` package. To ensure the official VS Code repository is used, create a file named `/etc/apt/preferences.d/code` with the following content:

Text

```
Package: code
Pin: origin "packages.microsoft.com"
Pin-Priority: 9999
```

"Visual Studio Code is unable to watch for file changes in this large workspace" (error ENOSPC)

When you see this notification, it indicates that the VS Code file watcher is running out of file handles that are needed to implement file watching. Most often this can happen when opening a workspace that is large and contains many files. Before adjusting platform limits, make sure that potentially large folders, such as Python `.venv`, are added to the `files.watcherExclude (vscode://settings/files.watcherExclude)` setting (more details below). It is also possible that other running applications consume so many file handles that none are left for VS Code to use. In that case, it might help to close these other applications.

The current limit can be viewed by running:

Bash

```
cat /proc/sys/fs/inotify/max_user_watches
```

The limit can be increased to its maximum by editing `/etc/sysctl.conf` (except on Arch Linux and Ubuntu 24.10 and later, read below) and adding this line to the end of the file:

Bash

```
fs.inotify.max_user_watches=524288
```

The new value can then be loaded in by running `sudo sysctl -p`.

While 524,288 is the maximum number of files that can be watched, if you're in an environment that is particularly memory-constrained, you might want to lower the number. Each file watch takes up 1,080 bytes (https://stackoverflow.com/a/7091897/1156119), so assuming that all 524,288 watches are consumed, that results in an upper bound of around 540 MiB.

Arch (https://www.archlinux.org/)-based distros (including Manjaro) and Ubuntu-based distros starting with 24.10 require you to change a different file; follow these steps (https://gist.github.com/tbjgolden/c53ca37f3bc2fab8c930183310918c8c) instead.

Another option is to exclude specific workspace directories from the VS Code file watcher with the `files.watcherExclude (vscode://settings/files.watcherExclude)` setting (/docs/configure/settings). The default for `files.watcherExclude (vscode://settings/files.watcherExclude)` excludes `node_modules` and some folders under `.git`, but you can add other directories that you don't want VS Code to track.

JSON

```
"files.watcherExclude": {
    "**/.git/objects/**": true,
    "**/.git/subtree-cache/**": true,
    "**/node_modules/*/**": true
  }
```

I can't see Chinese characters in Ubuntu

We're working on a fix. In the meantime, open the application menu, then choose **File** > **Preferences** > **Settings**. In the **Text Editor** > **Font** section, set "Font Family" to `Droid Sans Mono, Droid Sans Fallback`. If you'd rather edit the `settings.json` file directly, set `editor.fontFamily (vscode://settings/editor.fontFamily)` as shown:

JSON

```
    "editor.fontFamily": "Droid Sans Mono, Droid Sans Fallback"
```

Package git is not installed

This error can appear during installation and is typically caused by the package manager's lists being out of date. Try updating them and installing again:

Bash

```
# For .deb
sudo apt-get update

# For .rpm (Fedora 21 and below)
sudo yum check-update

# For .rpm (Fedora 22 and above)
sudo dnf check-update
```

The code bin command does not bring the window to the foreground on Ubuntu

Running `code .` on Ubuntu when VS Code is already open in the current directory will not bring VS Code into the foreground. This is a feature of the OS which can be disabled using `ccsm`.

Bash

```
# Install
sudo apt-get update
sudo apt-get install compizconfig-settings-manager

# Run
ccsm
```

Under **General** > **General Options** > **Focus & Raise Behavior**, set "Focus Prevention Level" to "Off". Remember this is an OS-level setting that will apply to all applications, not just VS Code.

Cannot install .deb package due to "/etc/apt/sources.list.d/vscode.list: No such file or directory"

This can happen when `sources.list.d` doesn't exist or you don't have access to create the file. To fix this, try manually creating the folder and an empty `vscode.list` file:

```Bash
sudo mkdir /etc/apt/sources.list.d
sudo touch /etc/apt/sources.list.d/vscode.list
```

Cannot move or resize the window while X forwarding a remote window

If you are using X forwarding to use VS Code remotely, you will need to use the native title bar to ensure you can properly manipulate the window. You can switch to using it by setting `window.titleBarStyle (vscode://settings/window.titleBarStyle)` to `native`.

Repository changed its origin value

If you receive an error similar to the following:

```Text
E: Repository '...' changed its 'Origin' value from '...' to '...'
N: This must be accepted explicitly before updates for this repository can be applie
d. See apt-secure(8) manpage for details.
```

Use `apt` instead of `apt-get` and you will be prompted to accept the origin change:

```Bash
sudo apt update
```

Was this documentation helpful?

Yes No

Still need help?

Ask the community(https://stackoverflow.com/questions/tagged/vscode)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/setup/linux.md).

01/08/2026

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

Visual Studio Code on Windows

Installation

1 Download and install Visual Studio Code

> Note
> VS Code ships monthly releases and supports auto-update when a new release is available.

2 Install additional components (/docs/setup/additional-components)

Install Git, Node.js, TypeScript, language runtimes, and more.

3 Install VS Code extensions from the Visual Studio Marketplace (https://marketplace.visualstudio.com/VSCode)

Customize VS Code with themes, formatters, language extensions and debuggers for your favorite languages, and more.

4 Enable AI features (/docs/copilot/setup)

> Tip
> If you don't yet have a Copilot subscription, you can use Copilot for free by signing up for the Copilot Free plan (https://github.com/github-copilot/signup) and get a monthly limit of inline suggestions and chat interactions.

5 Get started with the VS Code tutorial (/docs/getstarted/getting-started)

Discover the user interface and key features of VS Code.

Install VS Code on Windows

Use the Windows installer

1 Download the Visual Studio Code installer (https://go.microsoft.com/fwlink/?LinkID=534107) for Windows

2 Once it is downloaded, run the installer (VSCodeUserSetup-{version}.exe)

By default, VS Code is installed under `C:\Users\{Username}\AppData\Local\Programs\Microsoft VS Code`.

> **Tip**
> Setup adds Visual Studio Code to your `%PATH%` environment variable, to let you type 'code .' in the console to open VS Code on that folder. You need to restart your console after the installation for the change to the `%PATH%` environmental variable to take effect.

Use the ZIP file

1. Download the Visual Studio Code Zip archive (/docs/?dv=winzip)
2. Extract the Zip archive, and run VS Code from there

User setup versus system setup

VS Code provides both Windows **user** and **system** level setups.

Expand table

Setup Type	Description
User setup (https://go.microsoft.com/fwlink/?LinkID=534107)	Does not require administrator privileges to run, as the location is under your user Local AppData (`LOCALAPPDATA`) folder. Since it requires no elevation, the user setup is able to provide a smoother background update experience. This is the preferred way to install VS Code on Windows. **Note:** When running VS Code as Administrator in a user setup installation, updates are disabled.
System setup (https://go.microsoft.com/fwlink/?linkid=852157)	Requires elevation to administrator privileges to run and places the installation under the system's `Program Files` . The in-product update flow also requires elevation, making it less streamlined than the user setup. On the other hand, installing VS Code using the system setup means that it is available to all users in the system.

See the Download Visual Studio Code (/download) page for a complete list of available installation options.

Updates

VS Code ships monthly releases (/updates) and supports auto-update when a new release is available. If you're prompted by VS Code, accept the newest update and it will be installed (you won't need to do anything else to get the latest bits).

> **Note**

> You can disable auto-update (/docs/supporting/faq#_how-do-i-opt-out-of-vs-code-autoupdates) if you prefer to update VS Code on your own schedule.

Windows as a developer machine

Windows is a popular operating system and it can also be a great cross-platform development environment. This section describes cross-platform features such as the Windows Subsystem for Linux (https://learn.microsoft.com/windows/wsl/install) (WSL) and the Windows Terminal.

> Note
> Make sure you are on a recent Windows build. Check **Settings** > **Windows Update** to see if you are up-to-date.

Windows Subsystem for Linux

With WSL, you can install and run Linux distributions on Windows to develop and test your source code on Linux, while still working locally on your Windows machine.

When coupled with the WSL (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl) extension, you get full VS Code editing and debugging support while running in the context of WSL.

See the Developing in WSL (/docs/remote/wsl) documentation to learn more, or try the Working in WSL (/docs/remote/wsl-tutorial) introductory tutorial.

Windows Terminal

The Windows Terminal (https://apps.microsoft.com/detail/9n0dx20hk701), available from the Microsoft Store, is a terminal application for users of command-line tools and shells like Command Prompt, PowerShell, and WSL. Its main features include multiple tabs, panes, Unicode and UTF-8 character support, a GPU accelerated text rendering engine, and custom themes, styles, and configurations.

Next steps

Once you have installed VS Code, these topics will help you learn more about it:

- VS Code tutorial (/docs/getstarted/getting-started) - A quick hands-on tour of the key features of VS Code.
- Tips and Tricks (/docs/getstarted/tips-and-tricks) - A collection of productivity tips for working with VS Code.
- AI-assisted coding (/docs/copilot/overview) - Learn about using GitHub Copilot in VS Code to help you write code faster.

Common questions

What command-line arguments are supported by the Windows Setup?

VS Code uses Inno Setup (https://www.jrsoftware.org/isinfo.php) to create its setup package for Windows. Thus, all the Inno Setup command-line switches (https://www.jrsoftware.org/ishelp/index.php?topic=setupcmdline) are available for use.

Additionally, you can prevent the Setup from launching VS Code after completion with `/mergetasks=!runcode`.

I'm having trouble with the installer

Try using the zip file (/docs/?dv=winzip) instead of the installer. To use this, unzip VS Code in your `AppData\Local\Programs` folder.

> **Note**
> When VS Code is installed via a Zip file, you will need to manually update it for each release (/updates).

Unable to run as admin when AppLocker is enabled

With the introduction of process sandboxing (discussed in this blog post (https://code.visualstudio.com/blogs/2022/11/28/vscode-sandbox)) running as administrator is currently unsupported when AppLocker is configured due to a limitation of the runtime sandbox.

If your work requires that you run VS Code from an elevated terminal:

1. In VS Code, run the **Preferences: Configure Runtime Arguments** command in the Command Palette (`Ctrl+Shift+P`)

   This command opens an `argv.json` file to configure runtime arguments for VS Code. You might see some default arguments there already.

2. Add `"disable-chromium-sandbox": true` to the `argv.json` file.

3. Restart VS Code. You should now be able to run VS Code in an elevated terminal.

Subscribe to issue #122951 (https://github.com/microsoft/vscode/issues/122951) to receive updates.

Working with UNC paths

As of version `1.78.1`, VS Code on Windows only allows access to UNC paths (these begin with a leading `\\`) that were either approved by the user on startup or where the host name is configured to be allowed via the ⚙ `security.allowedUNCHosts (vscode://settings/security.allowedUNCHosts)` setting.

If you rely on using UNC paths in VS Code, you can either:

- Configure the host to be allowed via the ⚙ `security.allowedUNCHosts (vscode://settings/security.allowedUNCHosts)` setting. For example, add `server-a` when you open a path such as `\\server-a\path`.
- Map the UNC path as a network drive (https://support.microsoft.com/en-us/windows/map-a-network-drive-in-windows-29ce55d1-34e3-a7e2-4801-131475f9557d), and use the drive letter instead of the UNC path.
- Define a global environment variable `NODE_UNC_HOST_ALLOWLIST` with the backslash-separated list of host names to allow. For example, `server-a\server-b` to allow the hosts `server-a` and `server-b`.

> **Note**

> If you are using any of the remote extensions to connect to a workspace remotely (such as SSH), the `security.allowedUNCHosts (vscode://settings/security.allowedUNCHosts)` has to be configured on the remote machine and not the local machine.

This change was done to improve the security when using VS Code with UNC paths. Please refer to the associated security advisory (https://github.com/microsoft/vscode/security/advisories/GHSA-mmfh-4pv3-39hr) for more information.

Was this documentation helpful?

Yes **No**

Still need help?

Ask the community(https://stackoverflow.com/questions/tagged/vscode)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/setup/windows.md).

01/08/2026

(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

Additional components and tools

Visual Studio Code is a small download by design and only includes the minimum number of components shared across most development workflows. Basic functionality like the editor, file management, window management, and preference settings are included. A JavaScript/TypeScript language service and Node.js debugger are also part of the base install.

If you are used to working with larger, monolithic development tools (IDEs), you may be surprised that your scenarios aren't completely supported out of the box. For example, there isn't a **File** > **New Project** dialog with pre-installed project templates. Most VS Code users will need to install additional components depending on their specific needs.

Commonly used components

Here are a few commonly installed components:

- Git (https://git-scm.com/download) - VS Code has built-in support for source code control using Git but requires Git to be installed separately.
- Node.js (includes npm) (https://nodejs.org/) - A cross platform runtime for building and running JavaScript applications.
- TypeScript (https://www.typescriptlang.org) - The TypeScript compiler, `tsc`, for transpiling TypeScript to JavaScript.

You'll find the components above mentioned often in our documentation and walkthroughs.

VS Code extensions

You can extend the VS Code editor itself through extensions (/docs/configure/extensions/extension-marketplace). The VS Code community has built thousands of useful extensions available on the VS Code Marketplace (https://marketplace.visualstudio.com/VSCode).

The following list shows some of the popular extensions in the VS Code Marketplace. Select an extension tile to view the extension details.

Python
ms-python 199.9M

C/C++
ms-vscode 93.4M

GitHub Copilot
GitHub 68.0M

Extension Pack for J...
vscjava 40.9M

(https://marketplace.visualstudio.com/items?itemName=ms-python.python)	(https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools)	(https://marketplace.visualstudio.com/items?itemName=GitHub.copilot)	(https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack)

Additional tools

Visual Studio Code integrates with existing tool chains. We think the following tools will enhance your development experiences.

- Yeoman (https://yeoman.io/) - An application scaffolding tool, a command line version of **File** > **New Project**.
- generator-hottowel (https://github.com/johnpapa/generator-hottowel) - A Yeoman generator for quickly creating AngularJS applications.
- Express (https://expressjs.com/) - An application framework for Node.js applications using the Pug template engine.
- Gulp (https://gulpjs.com/) - A streaming task runner system which integrates easily with VS Code tasks.
- Mocha (https://mochajs.org/) - A JavaScript test framework that runs on Node.js.
- Yarn (https://yarnpkg.com/) - A dependency manager and alternative to npm.

> **Note:** Most of these tools require Node.js and the npm package manager to install and use.

Next steps

- User Interface (/docs/getstarted/userinterface) - A quick orientation around VS Code.
- User/Workspace Settings (/docs/configure/settings) - Learn how to configure VS Code to your preferences through settings.
- Languages (/docs/languages/overview) - VS Code supports many programming languages out-of-the-box as well as many more through community created extensions.

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/setup/additional-components.md).

01/08/2026

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

GitHub Copilot in VS Code cheat sheet

GitHub Copilot in Visual Studio Code provides AI-powered features to help you write code faster and with less effort. This cheat sheet provides a quick overview of the features for GitHub Copilot in Visual Studio Code.

> Tip
> If you don't yet have a Copilot subscription, you can use Copilot for free by signing up for the Copilot Free plan (https://github.com/github-copilot/signup) and get a monthly limit of inline suggestions and chat interactions.

Essential keyboard shortcuts

- `Ctrl+Alt+I` - Open the Chat view
- `Ctrl+I` - Enter voice chat prompt in Chat view
- `Ctrl+N` - Start a new chat session in Chat view
- `Ctrl+Shift+Alt+I` - Switch to using agents in Chat view
- `Ctrl+I` - Start inline chat in the editor or terminal
- `Ctrl+I` (hold) - Start inline voice chat
- `Tab` - Accept inline suggestion or navigate to the next edit suggestion
- `Escape` - Dismiss inline suggestion

Access AI in VS Code

- Start a chat conversation using natural language
  - Chat view (`Ctrl+Alt+I`): keep an ongoing chat conversation in the Secondary Side Bar
  - Inline chat in the editor or terminal (`Ctrl+I`): ask questions while you're in the flow
  - Quick Chat (`Ctrl+Shift+Alt+L`): ask quick questions without leaving your current task
- AI in the editor (/docs/copilot/ai-powered-suggestions)
  - Inline suggestions: get suggestions as you type, press `Tab` to accept a suggestion
  - Edit context menu actions: access common AI actions like explaining or fixing code, generating tests, or reviewing a text selection
  - Code actions: get editor code actions (lightbulb) to fix linting and compiler errors
- Task-specific smart actions (/docs/copilot/copilot-smart-actions) across VS Code
  - Generate commit messages and pull request titles and descriptions
  - Fix testing errors
  - Semantic file search suggestions

Chat experience in VS Code

Start a natural language chat conversation to get help with coding tasks. For example, ask to explain a block of code or a programming concept, refactor a piece of code, or implement a new feature. Get more information about using Copilot Chat (/docs/copilot/chat/copilot-chat).

Expand table

Action	Description
`Ctrl+Alt+I`	Open the Chat view (/docs/copilot/chat/copilot-chat) in the Secondary Side Bar.
`Ctrl+I`	Start inline chat (/docs/copilot/chat/inline-chat) to open chat in the editor or terminal.
`Ctrl+Shift+Alt+L`	Open Quick Chat (/docs/copilot/chat/copilot-chat) without interrupting your workflow.
`Ctrl+N`	Start a new chat session in the Chat view.
`unassigned`	Toggle between different agents (/docs/copilot/customization/custom-agents) in the Chat view.
`Ctrl+Alt+.`	Show the model picker to select a different AI model (/docs/copilot/customization/language-models) for chat.
`Add Context...`	Attach different types of context to your chat prompt (/docs/copilot/chat/copilot-chat-context).
`/` -command	Use slash commands for common tasks or invoke a reusable chat prompt (/docs/copilot/customization/overview).
`#` -mention	Reference common tools or chat variables to provide context (/docs/copilot/chat/copilot-chat-context) within in your prompt.
`@` -mention	Reference chat participants to handle domain-specific requests.
Edit (✎)	Edit a previous chat prompt (/docs/copilot/chat/chat-checkpoints#_edit-a-previous-chat-request) and revert changes.

Action	Description
History ( )	Access your history of chat sessions.
Voice ( )	Enter a chat prompt by using speech (voice chat). The chat response is read out aloud.
KaTeX (https://katex.org)	Render mathematical equations in chat responses. Enable with `chat.math.enabled` (`vscode://settings/chat.math.enabled`). Right-click on a math expression to copy the source expression.

Tips

- Use `#` -mentions to add more context to your chat prompt.
- Use `/` commands and `@` participants to get more precise and relevant answers.
- Be specific, keep it simple, and ask follow-up questions to get the best results.
- Choose an agent that fits your needs: Ask, Edit, Agent, or create a custom agent.

Add context to your prompt

Get more relevant responses by providing context to your chat prompt (/docs/copilot/chat/copilot-chat-context). Choose from different context types, such as files, symbols, editor selections, source control commits, test failures, and more.

Expand table

Action	Description
Add Context	Open a Quick Pick to select relevant context for your chat prompt. Choose from different context types, such as workspace files, symbols, current editor selection, terminal selection, and more.
Drag & drop files	Drag & drop a file from the Explorer or Search view, or drag an editor tab onto the Chat view.
Drag & drop folders	Drag & drop a folder onto the Chat view to attach the files within it.
Drag & drop problem	Drag & drop an item from the Problems panel.
`#<file|folder|symbol>`	Type `#`, followed by a file, folder, or symbol name, to add it as chat context.
`#` -mention	Type `#`, followed by a chat tool to add a specific context type or tool.

Chat tools

Use tools (/docs/copilot/chat/chat-tools) in chat to accomplish specialized tasks while processing a user request. Examples of such tasks are listing the files in a directory, editing a file in your workspace, running a terminal command, getting the output from the terminal, and more.

VS Code provides built-in tools, and you can extend chat with tools from MCP servers (/docs/copilot/customization/mcp-servers) and extensions (/api/extension-guides/ai/tools). Learn more about types of tools (/docs/copilot/chat/chat-tools#_types-of-tools).

The following table lists the VS Code built-in tools:

Expand table

Chat variable/Tool	Description
`#changes`	List of source control changes.
`#codebase`	Perform a code search in the current workspace to automatically find relevant context for the chat prompt.
`#createAndRunTask`	Create and run a new task (/docs/debugtest/tasks) in the workspace.
`#createDirectory`	Create a new directory in the workspace.
`#createFile`	Create a new file in the workspace.
`#edit` (tool set)	Enable modifications in the workspace.
`#editFiles`	Apply edits to files in the workspace.
`#editNotebook`	Make edits to a notebook.
`#extensions`	Search for and ask about VS Code extensions. For example, "how to get started with Python #extensions?"
`#fetch`	Fetch the content from a given web page. For example, "Summarize #fetch code.visualstudio.com/updates."
`#fileSearch`	Search for files in the workspace by using glob patterns and returns their path.
`#getNotebookSummary`	Get the list of notebook cells and their details.

Chat variable/Tool	Description
`#getProjectSetupInfo`	Provide instructions and configuration for scaffolding different types of projects.
`#getTaskOutput`	Get the output from running a task (/docs/debugtest/tasks) in the workspace.
`#getTerminalOutput`	Get the output from running a terminal command in the workspace.
`#githubRepo`	Perform a code search in a GitHub repo. For example, "what is a global snippet #githubRepo microsoft/vscode."
`#installExtension`	Install a VS Code extension.
`#listDirectory`	List files in a directory in the workspace.
`#new`	Scaffold a new VS Code workspace, preconfigured with debug and run configurations.
`#newJupyterNotebook`	Scaffold a new Jupyter notebook given a description.
`#newWorkspace`	Create a new workspace.
`#openSimpleBrowser`	Open the built-in Simple Browser and preview a locally-deployed web app.
`#problems`	Add workspace issues and problems from the **Problems** panel as context. Useful while fixing code or debugging.
`#readFile`	Read the content of a file in the workspace.
`#readNotebookCellOutput`	Read the output from a notebook cell execution.
`#runCell`	Run a notebook cell.
`#runCommands` (tool set)	Enable running commands in the terminal and reading the output.
`#runInTerminal`	Run a shell command in the integrated terminal.
`#runNotebooks` (tool set)	Enable running notebook cells.
`#runTask`	Run an existing task (/docs/debugtest/tasks) in the workspace.
`#runTasks` (tool set)	Enable running tasks (/docs/debugtest/tasks) in the workspace and reading the output.

Chat variable/Tool	Description
`#runSubagent`	Run a task in an isolated subagent context (/docs/copilot/chat/chat-sessions#_subagents). Helps to improve the context management of the main agent thread.
`#runTests`	Run unit tests (/docs/debugtest/testing) in the workspace.
`#runVscodeCommand`	Run a VS Code command. For example, "Enable zen mode #runVscodeCommand."
`#search` (tool set)	Enable searching for files in the current workspace.
`#searchResults`	Get the search results from the Search view.
`#selection`	Get the current editor selection (only available when text is selected).
`#terminalLastCommand`	Get the last run terminal command and its output.
`#terminalSelection`	Get the current terminal selection.
`#testFailure`	Get unit test failure information. Useful when running and diagnosing tests (/docs/debugtest/testing).
`#textSearch`	Find text in files.
`#todos`	Track implementation and progress of a chat request with a todo list.
`#usages`	Combination of "Find All References", "Find Implementation", and "Go to Definition".
`#VSCodeAPI`	Ask about VS Code functionality and extension development.

Slash commands

Slash commands are shortcuts to specific functionality within the chat. You can use them to quickly perform actions, like fixing issues, generating tests, or explaining code.

Expand table

Slash command	Description
`/doc`	Generate code documentation comments from editor inline chat.
`/explain`	Explain a code block, file, or programming concept.

Slash command	Description
`/fix`	Ask to fix a code block or resolve compiler or linting errors.
`/tests`	Generate tests for all or only the selected methods and functions in the editor.
`/setupTests`	Get help setting up a testing framework for your code. Get recommendation for a relevant testing framework, steps to set up and configure it, and suggestions for VS Code testing extensions.
`/clear`	Start a new chat session in the Chat view.
`/new`	Scaffold a new VS Code workspace or file. Use natural language to describe the type of project/file you need, and preview the scaffolded content before creating it.
`/newNotebook`	Scaffold a new Jupyter notebook based on your requirements. Use natural language to describe what the notebook should contain.
`/search`	Generate a search query for the Search view. Use natural language to describe what you want to search for.
`/startDebugging`	Generate a `launch.json` debug configuration file and start a debugging session from the Chat view.
`/<prompt name>`	Run a reusable prompt (/docs/copilot/customization/prompt-files) in chat.

Chat participants

Use chat participants to handle domain-specific requests in chat. Chat participants are prefixed with `@` and can be used to ask questions about specific topics. VS Code provides built-in chat participants, such as `@github`, `@terminal`, and `@vscode`, and extensions can provide additional participants.

Expand table

Chat participant	Description
`@github`	Use the `@github` participant to ask questions about GitHub repositories, issues, pull requests, and more. Get more information about the available GitHub skills (https://docs.github.com/en/copilot/using-github-copilot/asking-github-copilot-questions-in-your-ide#currently-available-skills). Example: `@github What are all of the open PRs assigned to me?`, `@github Show me the recent merged PRs from @dancing-mona`
`@terminal`	Use the `@terminal` participant to ask questions about the integrated terminal or shell commands. Example: `@terminal list the 5 largest files in this workspace`
`@vscode`	Use the `@vscode` participant to ask questions about VS Code features, settings, and the VS Code extension APIs. Example: `@vscode how to enable word wrapping?`
`@workspace`	Use the `@workspace` participant to ask questions about the current workspace. Example: `@workspace how is authentication implemented?`

Use agents

When using agents (/docs/copilot/chat/copilot-chat#_built-in-agents), you can use natural language to specify a high-level task, and let AI autonomously reason about the request, plan the work needed, and apply the changes to your codebase. Agents use a combination of code editing and tool invocation to accomplish the task you specified. As it processes your request, it monitors the outcome of edits and tools, and iterates to resolve any issues that arise.

Expand table

Action	Description
`Ctrl+Shift+Alt+I`	Switch to using agents in the Chat view
Tools (🛠)	Configure which tools are available when using agents. Select from built-in tools, MCP servers, and extension-provided tools.

Action	Description
Auto-approve tools *(Experimental)*	Enable auto-approval of all tools (/docs/copilot/chat/chat-tools#_auto-approve-all-tools) when using agents (⚙ `chat.tools.autoApprove` (`vscode://settings/chat.tools.autoApprove`)).
Auto-approve terminal commands *(Experimental)*	Enable auto-approval of terminal commands (/docs/copilot/chat/chat-tools#_automatically-approve-terminal-commands) when using agents (⚙ `chat.tools.terminal.autoApprove` (`vscode://settings/chat.tools.terminal.autoApprove`)).
MCP	Configure MCP servers (/docs/copilot/customization/mcp-servers) to extend agent capabilities and tools.

Tips

- Add extra tools when using agents to extend its capabilities.
- Configure custom agents to define how the agent should operate, for example to implement a read-only planning mode.
- Define custom instructions to guide agents on how to generate and structure code.

Planning

Use the plan agent (/docs/copilot/chat/chat-planning) in VS Code chat to create detailed implementation plans before starting complex coding tasks. Hand off the approved plan to an implementation agent to start coding.

Expand table

Action	Description
Plan agent	Select the **Plan** agent from the agents dropdown in the Chat view to create a detailed implementation plan for complex coding tasks.
Todo list (Experimental)	Enable the `todos` tool in the tools picker to track progress on complex tasks with a todo list.

Customize your chat experience

Customize your chat experience to generate responses that match your coding style, tools, and developer workflow. There are several ways to customize your chat experience in VS Code:

- Custom instructions (/docs/copilot/customization/custom-instructions): Define common guidelines or rules for tasks like generating code, performing code reviews, or generating commit messages. Custom instructions describe the conditions in which the AI should operate (*how* a task should be done).
- Reusable prompt files (/docs/copilot/customization/prompt-files): Define reusable prompts for common tasks like generating code or performing a code review. Prompt files are standalone prompts that you can run directly in chat. They describe the task to be performed (*what* should be done).
- Custom agents (/docs/copilot/customization/custom-agents): Define how chat operates, which tools it can use, and how it interacts with the codebase. Each chat prompt is run within the boundaries of the agent, without having to configure tools and instructions for every request.

> **Tips**
>
> - Define language-specific instructions to get more accurate generated code for each language.
> - Store your instructions in your workspace to easily share them with your team.
> - Define reusable prompt files for common tasks to save time and help team members get started quickly.

Editor AI features

As you're coding in the editor, you can use Copilot to generate inline suggestions as you're typing. Invoke Inline Chat to ask questions and get help from Copilot, while staying in the flow of coding. For example, ask Copilot to generate unit tests for a function or method. Get more information about inline suggestions (/docs/copilot/ai-powered-suggestions) and Inline Chat (/docs/copilot/chat/inline-chat).

Expand table

Action	Description
Inline suggestions	Start typing in the editor and get inline suggestions (/docs/copilot/ai-powered-suggestions) that match your coding style and take your existing code into account.
Code comments	Provide an inline suggestions prompt by writing instructions in a code comment. Example: `# write a calculator class with methods for add, subtract, and multiply. Use static methods.`
`Ctrl+I`	Start editor inline chat to send a chat request directly from the editor. Use natural language and reference chat variables and slash commands to provide context.
`F2`	Get AI-powered suggestions when renaming symbols in your code.

Action	Description
Context menu actions	Use the editor context menu to access common AI actions, such as explaining code, generating tests, reviewing code, and more. Right-click in the editor to open the context menu and select **Generate Code**.
Code Actions (lightbulb)	Select the Code Action (lightbulb) in the editor for fixing linting or compiler errors in your code.

> **Tips**
>
> - Use meaningful method or function names to get better inline suggestions quicker.
> - Select a code block to scope your Inline Chat prompt or attach relevant context by attaching files or symbols.
> - Use the editor context menu options to access common AI-powered actions directly from the editor.

Source control and issues

Use AI to analyze the changes in your commits and pull requests and provide suggestions for commit messages and pull request descriptions.

Expand table

Action	Description
`#changes`	Add the current source control changes as context in your chat prompt.
Commit as context	Add a commit from the source control history as context in your chat prompt.
Commit message	Generate a commit message for the current changes in a source control commit.
Merge conflicts (Experimental)	Get help resolving Git merge conflicts with AI (/docs/sourcecontrol/overview#resolve-merge-conflicts-with-ai-experimental).
Pull request description	Generate a pull request title and description that correspond with the changes in your pull request.

Action	Description
`@github`	Use the `@github` participant in chat to ask about issues, pull requests, and more across your repositories. Get more information about the available GitHub skills (https://docs.github.com/en/copilot/using-github-copilot/asking-github-copilot-questions-in-your-ide#currently-available-skills). Example: `@github What are all of the open PRs assigned to me?`, `@github Show me the recent merged pr's from @dancing-mona`

Review code (experimental)

Use AI to do a quick review pass of a code block or perform a review of uncommitted changes in your workspace. Review feedback shows up as comments in the editor, where you can apply the suggestions.

Expand table

Action	Description
Review Selection *(Preview)*	Select a block of code, and select **Generate Code** > **Review** from the editor context menu for a quick review pass.
Code Review	Select the **Code Review** button in the Source Control view for a deeper review of all uncommitted changes.

Search and settings

Get semantically relevant search results in the Search view or help with searching for settings in the Settings editor.

Expand table

Action	Description
Settings search	Include semantic search results in the Settings editor (⚙ `workbench.settings.showAISearchToggle` (`vscode://settings/workbench.settings.showAISearchToggle`)).
Semantic search *(Preview)*	Include semantic search results in the Search view (⚙ `search.searchView.semanticSearchBehavior` (`vscode://settings/search.searchView.semanticSearchBehavior`)).

Generate tests

VS Code can generate tests for functions and methods in your codebase by using slash commands in chat. Slash commands are a shorthand notation for common tasks that you can use in chat prompts. Type `/` followed by the command name to use a slash command.

Expand table

Action	Description
`/tests`	Generate tests for all or only the selected methods and functions in the editor. The generated tests are appended in an existing tests file or a new tests file is created.
`/setupTests`	Get help setting up a testing framework for your code. Get recommendation for a relevant testing framework, steps to set up and configure it, and suggestions for VS Code testing extensions.
`/fixTestFailure`	Ask Copilot for suggestions on how to fix failing tests.
Test coverage *(Experimental)*	Generate tests for functions and methods that are not yet covered by tests. Get more information (https://code.visualstudio.com/updates/v1_93#_generate-tests-based-on-test-coverage-experimental).

Tips

- Provide details about the testing frameworks or libraries to use.

Debug and fix problems

Use Copilot to help fix coding problems and to get help with configuring and starting debugging sessions in VS Code.

Expand table

Action	Description
`/fix`	Ask Copilot for suggestions on how to fix a block of code or how to resolve any compiler or linting errors in your code. For example, to help fix unresolved Node.js package names.

Action	Description
`/fixTestFailure`	Ask Copilot for suggestions on how to fix failing tests.
`/startDebugging` *(Experimental)*	Generate a `launch.json` debug configuration file and start a debugging session (/docs/copilot/guides/debug-with-copilot) from the Chat view.
`copilot-debug` command	Terminal command to help you debug your programs (/docs/copilot/guides/debug-with-copilot). Prefix a run command to start a debugging session for it (for example, `copilot-debug python foo.py`).

Tips

- Provide additional information about the type of fix you need, such as optimizing the memory consumption or performance.
- Watch for Copilot Code Actions in the editor that indicate suggestions for fixing problems in your code.

Scaffold a new project

Copilot can help you create a new project by generating a scaffold of the project structure, or generate a notebook based on your requirements.

Expand table

Action	Description
Agent	Use agents (/docs/copilot/chat/copilot-chat#_built-in-agents) and use a natural language prompt to create a new project or file. For example, `Create a svelte web application to track my tasks` .
`/new`	Use the `/new` command in the Chat view to scaffold a new project or a new file. Use natural language to describe the type of project/file you need, and preview the scaffolded content before creating it. Example: `/new Express app using typescript and svelte`
`/newNotebook`	Use the `/newNotebook` command in the Chat view to generate a new Jupyter notebook based on your requirements. Use natural language to describe what the notebook should contain. Example: `/newNotebook get census data and preview key insights with Seaborn` .

Terminal

Get help about shell commands and how to resolve errors when running commands in the terminal.

Expand table

Action	Description
`Ctrl+I`	Start terminal inline chat to use natural language for asking about shell commands and the terminal. Example: `how many cores on this machine?`
`@terminal`	Use the `@terminal` participant in the Chat view to ask questions about the integrated terminal or shell commands. Example: `@terminal list the 5 largest files in this workspace`
`@terminal /explain`	Use the `/explain` command in the Chat view to explain something from the terminal. Example: `@terminal /explain top shell command`

Python and notebook support

You can use chat to help you with Python programming tasks in the Native Python REPL and in Jupyter notebooks.

Expand table

Action	Description
✧ Generate `Ctrl+I`	Start Inline Chat in a notebook to generate a codeblock or Markdown block.
`#`	Attach variables from the Jupyter kernel in your chat prompt to get more relevant responses.
Native REPL + `Ctrl+I`	Start Inline Chat in the Native Python REPL and run the generated commands.
`Ctrl+Alt+I`	Open the **Chat view** and use agents to make notebook edits.
`/newNotebook`	Use the `/newNotebook` command in the Chat view to generate a new Jupyter notebook based on your requirements. Use natural language to describe what the notebook should contain. Example: `/newNotebook get census data and preview key insights with Seaborn.`

Next steps

- Tutorial: Get started with AI features in VS Code (/docs/copilot/getting-started)

Was this documentation helpful?

Yes No

Still need help?

Ask the community(https://stackoverflow.com/questions/tagged/vscode)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/copilot/reference/copilot-vscode-features.md).

01/08/2026

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Make chat an expert in your workspace

Chat becomes significantly more helpful when it has a deep understanding of your entire codebase, not just individual files. Workspace context is the underlying mechanism that enables the AI to search across your project, understand how components connect, and provide answers grounded in your actual code. This enables you to ask broad questions like "where is authentication handled?" or "how do I add a new API endpoint?" and get accurate answers based on your specific codebase.

This article explains how workspace context works, how to manage your workspace index for optimal results, and how to use `@workspace` and `#codebase` to leverage it in your prompts.

The intelligence behind workspace context automatically adjusts based on your project's size and setup, ensuring you get accurate results whether you're working on a small personal project or a large enterprise codebase.

How workspace context works

VS Code uses intelligent search strategies to find the most relevant code for your questions. Rather than using a single approach, it automatically selects the best method based on your project size and available resources. VS Code might run multiple strategies in parallel and then choose the one that produces the best results the fastest.

What sources are used for context?

Workspace context searches through the same sources a developer would use when navigating a codebase in VS Code:

- All indexable files in the workspace (workspace index), except those ignored by a `.gitignore` file
- Directory structure with nested folders and file names
- Code symbols and definitions (classes, functions, variables)
- Currently selected text or visible text in the active editor

The workspace index can be maintained remotely by GitHub or stored locally on your machine. See the workspace index section for more details.

> Important
> `.gitignore` is bypassed if you have a file open or have text selected within an ignored file.

Search strategy

For small projects, the entire workspace can be included directly in the chat context. For larger projects, VS Code uses different strategies to find the most relevant information to include in the chat context for your prompt.

The following steps outline how VS Code constructs the workspace context:

1. Determine which information from the workspace is needed to answer your question, also including the conversation history, workspace structure, and current editor selection.

2. Collect relevant code snippets from the workspace index by using various approaches:

    - GitHub's code search (https://github.blog/2023-02-06-the-technology-behind-githubs-new-code-search)
    - Local semantic search to find code that matches the meaning of your question, not just exact keywords
    - Text-based file-name and content search
    - VS Code's language IntelliSense to add details like function signatures, parameters, and more.

3. If the resulting context is too large to fit in the *context window*, only the most relevant parts are kept.

Workspace index

Chat in VS Code uses an index to quickly and accurately search your codebase for relevant code snippets. This index can either be maintained by GitHub or stored locally on your machine.

The remote index is built from the committed state of your repository on GitHub or Azure DevOps. This means that any uncommitted changes in your local workspace are not included in the remote index.

When you have local uncommitted changes, VS Code uses a hybrid approach combining the remote index with local file tracking. VS Code detects which files have been modified since the indexed commit and also reads the current file content from the editor for real-time content.

You can view the type of index that is being used and its indexing status in the Copilot status dashboard in the VS Code Status Bar.

Remote index

VS Code can use remote code search indexes to enable AI to search your codebase quickly, even for large codebases. Remote code search is currently available for workspaces that use GitHub or Azure DevOps repositories.

GitHub remote indexing

VS Code automatically builds and uses remote code search indexes for any GitHub-backed repositories in your workspace. Sign in with your GitHub account in VS Code and chat will automatically start using any available remote code search indexes.

Repositories are automatically indexed the first time `@workspace` or `#codebase` is used in chat. You can also force indexing by running the **Build Remote Workspace Index** command in the Command Palette (`Ctrl+Shift+P`).

The index only needs to be built once per repository. After that, the index is automatically kept up to date. Building the index is fast for small and medium sized projects, but may take a little time if your repository contains hundreds of thousands of files. The remote index works also best if GitHub has a relatively up-to-date version of your code, so make sure to push your code to GitHub regularly.

Currently remote indexing works for GitHub repositories hosted on GitHub.com or on GitHub Enterprise Cloud. It is not supported for repositories that use GitHub Enterprise Server.

Azure DevOps remote indexing

VS Code can also use remote indexes for Azure DevOps repositories. These indexes are automatically built and maintained. Sign in with your Microsoft account in VS Code for chat to start using the remote indexes. Check the Copilot Status Bar item for the current index status and to get a sign-in link if your account doesn't have the right permissions to access the Azure DevOps repository.

Local index

If you can't use a remote index, for example because you're not using a GitHub or Azure DevOps repository, VS Code can use an advanced semantic index that is stored on your local machine to provide fast, high quality search results. Currently, local indexes are limited to 2500 indexable files.

To build a local index:

- The project has less than 750 indexable files: VS Code automatically builds an advanced local index.
- The project has between 750 and 2500 indexable files: run the **Build local workspace index** command in the Command Palette (`Ctrl+Shift+P`) - this should only be run once.
- The project has more than 2500 indexable files: use a basic index.

It might take some time to build the initial local index or update the index if many files have changed, for example when switching git branches. You can monitor the current local index status in the Copilot status dashboard in the Status Bar.

Basic index

If your project does not have a remote index and has more than 2500 indexable files, VS Code falls back to using a basic index to search your codebase. This index uses simpler algorithms to search your codebase and is optimized to work locally for larger codebases.

The basic index should work just fine for many types of chat prompts. However, if you find that chat is struggling to provide relevant answers to questions about your codebase, consider upgrading to a remote index.

What content is included in the workspace index

VS Code indexes relevant text files that are part of your current project. This is not limited to specific file types or programming languages, however VS Code automatically skips over some common file types that are typically not relevant to workspace questions, such as `.tmp` or `.out` files.

The workspace index also excludes any files that are excluded from VS Code using the ⚙ `files.exclude (vscode://settings/files.exclude)` setting or that are part of the `.gitignore` file.

VS Code also currently does not index binary files, such as images or PDFs.

Use workspace context in chat

When you ask a workspace-related question in chat, the behavior for determining the workspace context depends on which agent you're using:

- **Agent/Plan**

  When using agents, the agent automatically performs an *agentic* codebase search based on your prompt. This means that after performing an initial search to determine the workspace context, depending on the results, the agent might decide to perform additional, more targeted searches to gather the information it needs to answer your question.

  You don't need to explicitly reference the `#codebase` tool in your prompt, but you can do so if you want to ensure that workspace context is used for your question. This is useful if your prompt is ambiguous and might be interpreted as not requiring workspace context.

- **Ask/Edit**

  In Ask or Edit, VS Code performs intent detection on your prompt to determine if it requires workspace context. If requires workspace context, VS Code performs a codebase search and adds the relevant code snippets to the chat context. As opposed to using agents, no follow-up searches are performed.

  You don't need to explicitly reference the `#codebase` tool in your prompt, but you can do so if you want to ensure that workspace context is used for your question. This is useful if your prompt is ambiguous and might be interpreted as not requiring workspace context.

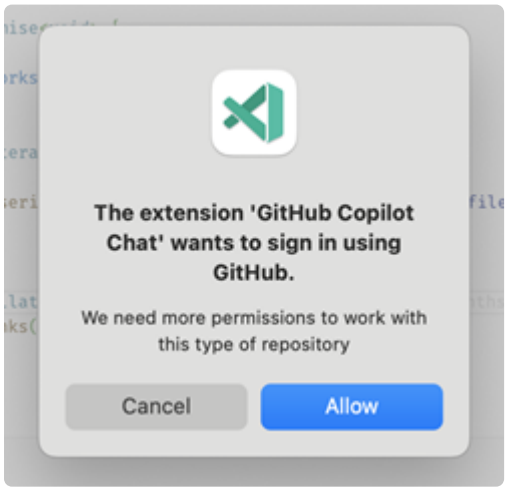
Tips for using workspace context

The way you phrase your question can significantly influence the quality of the context and the accuracy of the response. To optimize results, consider the following tips:

- Be specific and detailed in your question, avoiding vague or ambiguous terms like "what does this do", where "this" could be interpreted as the last answer, current file, or whole project.
- Incorporate terms and concepts in your prompt that are likely to appear in your code or its documentation.
- Explicitly include relevant context by selecting code, referencing files, or #-mentioning context items (/docs/copilot/chat/copilot-chat-context) such as debug context, terminal output, and more.
- Responses can draw from multiple references, such as "find exceptions without a catch block" or "provide examples of how handleError is called". However, don't anticipate a comprehensive code analysis across your codebase, such as "how many times is this function invoked?" or "rectify all bugs in this project".
- When asking about information beyond the code, such as "who contributed to this file?" or "summarize review comments for this folder", make sure to configure the relevant tools or MCP servers (/docs/copilot/chat/chat-tools) when using agents.

Private repositories

To enable more workspace search features for private repositories, we require additional permissions. If we detect that we don't have these permissions already, we will ask for them at startup. Once granted, we'll securely store the session for the future.



Learn more about security, privacy, and transparency in the GitHub Copilot Trust Center (https://resources.github.com/copilot-trust-center/).

Frequently asked questions

What is the difference between @workspace and #codebase?

Conceptually, both `@workspace` and `#codebase` enable you to ask questions about your entire codebase. However, there are some differences in how you can use them:

- `@workspace` is a chat participant (/docs/copilot/chat/copilot-chat-context#_atmentions)

  The `@workspace` participant is subject matter expert that is specialized to answering questions about your codebase. The language model hands off the entire chat prompt to the participant, which uses its knowledge of the codebase to provide an answer. The language model can't perform any additional processing or invoke other tools when using a chat participant. A chat prompt can only contain a single chat participant.

- `#codebase` is a chat tool (/docs/copilot/chat/chat-tools)

  The `#codebase` tool is specialized in searching your codebase for relevant information. It is one of many tools that the language model can choose to invoke when answering your chat prompt. The language model can decide to invoke the `#codebase` tool multiple times, interleaved with other tools, to gather the information it needs to answer your question. A chat prompt can contain multiple tools.

It's recommended to use `#codebase` in your chat prompts, as it provides more flexibility.

Was this documentation helpful?

Yes No

Still need help?

Ask the community(https://stackoverflow.com/questions/tagged/vscode)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/copilot/reference/workspace-context.md).

01/08/2026

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

Working with GitHub in VS Code

GitHub (https://github.com) is a cloud-based service for storing and sharing source code. Using GitHub with Visual Studio Code lets you share your source code and collaborate with others right within your editor. There are many ways to interact with GitHub, for example, via their website at https://github.com (https://github.com) or the Git (https://git-scm.com) command-line interface (CLI), but in VS Code, the rich GitHub integration is provided by the GitHub Pull Requests and Issues (https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github) extension.

In this topic, we'll demonstrate how you can use some of your favorite parts of GitHub without leaving VS Code.

> Tip
> If you're new to source control or want to learn more about VS Code's basic Git support, you can start with the Source Control (/docs/sourcecontrol/overview) topic.

Prerequisites

To get started with GitHub in VS Code, you need:

- Git is installed on your computer. Install Git version 2.0.0 or later (https://git-scm.com/download) on your machine.
- A GitHub account (https://docs.github.com/get-started/signing-up-for-github/signing-up-for-a-new-github-account).
- The GitHub Pull Requests and Issues (https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github) extension installed in VS Code.
- When you commit changes, Git uses your configured username and email. You can set these values with:

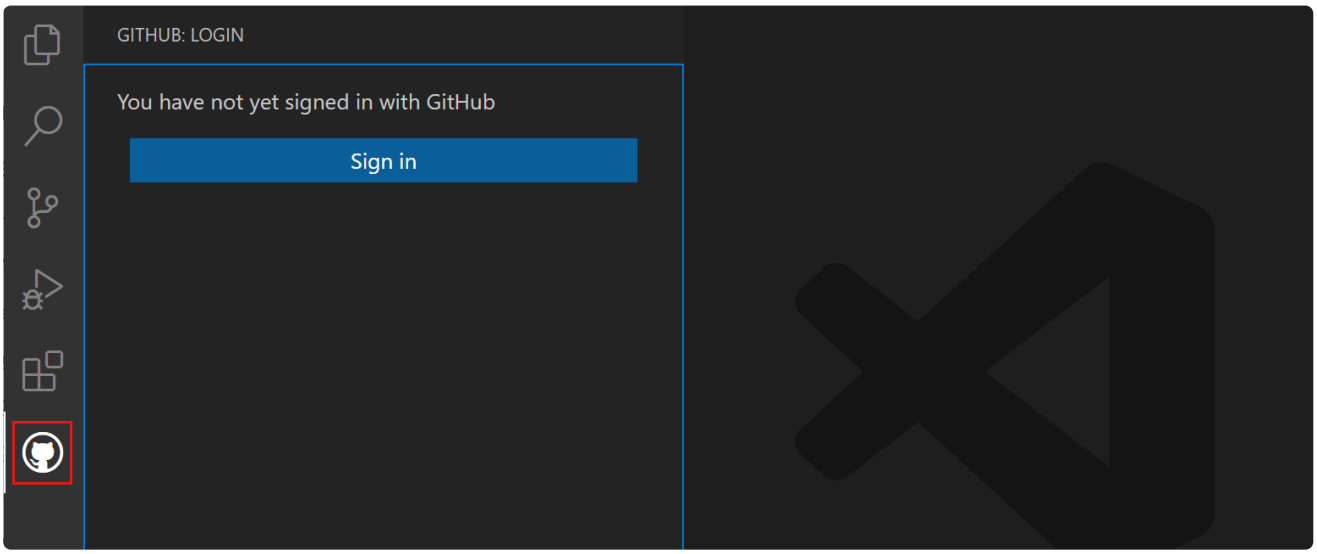
```bash
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

Getting started with GitHub Pull Requests and Issues

Once you've installed the GitHub Pull Requests and Issues (https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github) extension, you'll need to sign in.

1. Select the GitHub icon in the Activity Bar

2 Select **Sign In** and follow the prompts to authenticate with GitHub in the browser



3 You should be redirected back to VS Code

If you are not redirected to VS Code, you can add your authorization token manually:

1 In the browser window, copy your authorization token

2 In VS Code, select **Signing in to github.com...** in the Status Bar

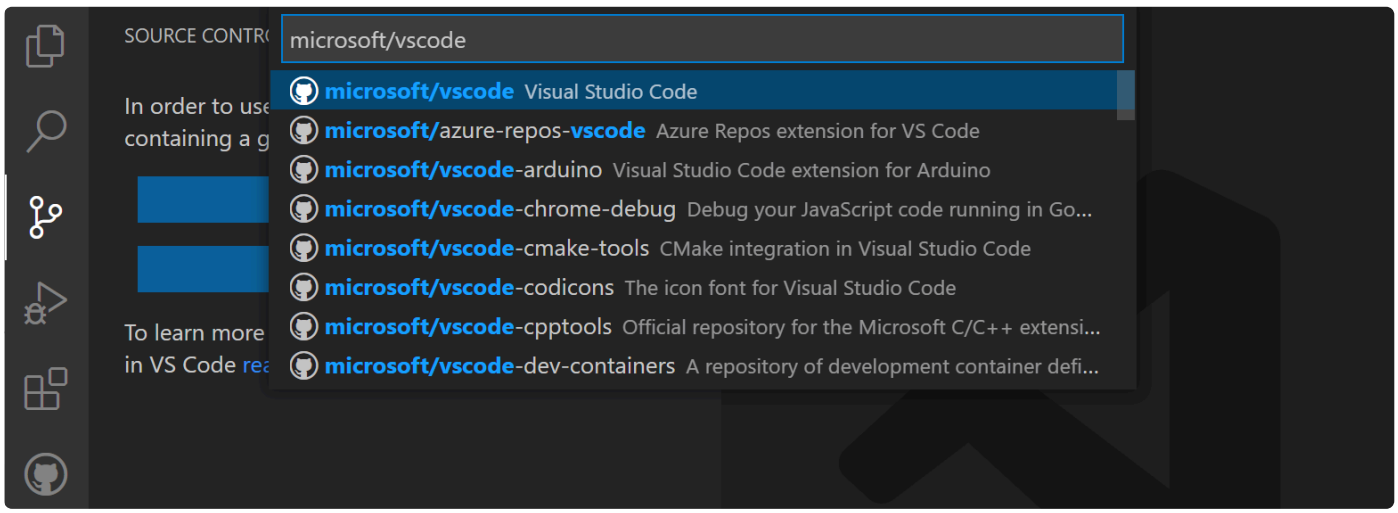
3 Paste the token and press `Enter` to complete the sign-in process

Setting up a repository

Cloning a repository

You can search for and clone a repository from GitHub using the **Git: Clone** command in the Command Palette (`Ctrl+Shift+P`) or by using the **Clone Repository** button in the Source Control view (available when you have no folder open).

From the GitHub repository dropdown you can filter and pick the repository you want to clone locally.

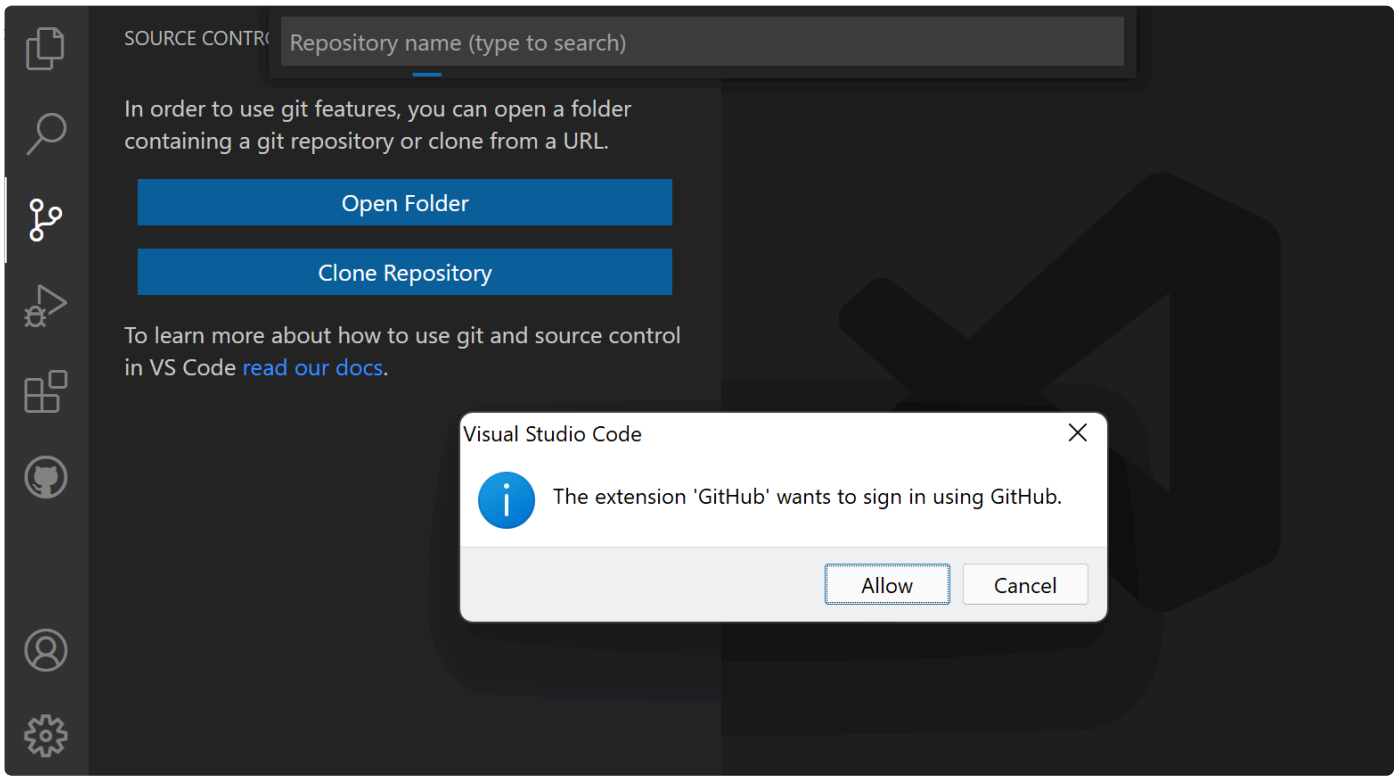


Learn more about cloning repositories and working with remotes (/docs/sourcecontrol/repos-remotes#_clone-repositories).

Authenticating with an existing repository

Enabling authentication through GitHub happens when you run any Git action in VS Code that requires GitHub authentication, such as pushing to a repository that you're a member of or cloning a private repository. You don't need to have any special extensions installed for authentication; it is built into VS Code so that you can efficiently manage your repository.

When you perform an action that requires GitHub authentication, VS Code prompts you to sign in. Follow the steps to sign into GitHub and return to VS Code.



Signing in with a personal access token (PAT) is only supported with GitHub Enterprise Server. If you're using GitHub Enterprise Server and want to use a PAT, you can select **Cancel** on the sign in prompts until you are prompted for a PAT.

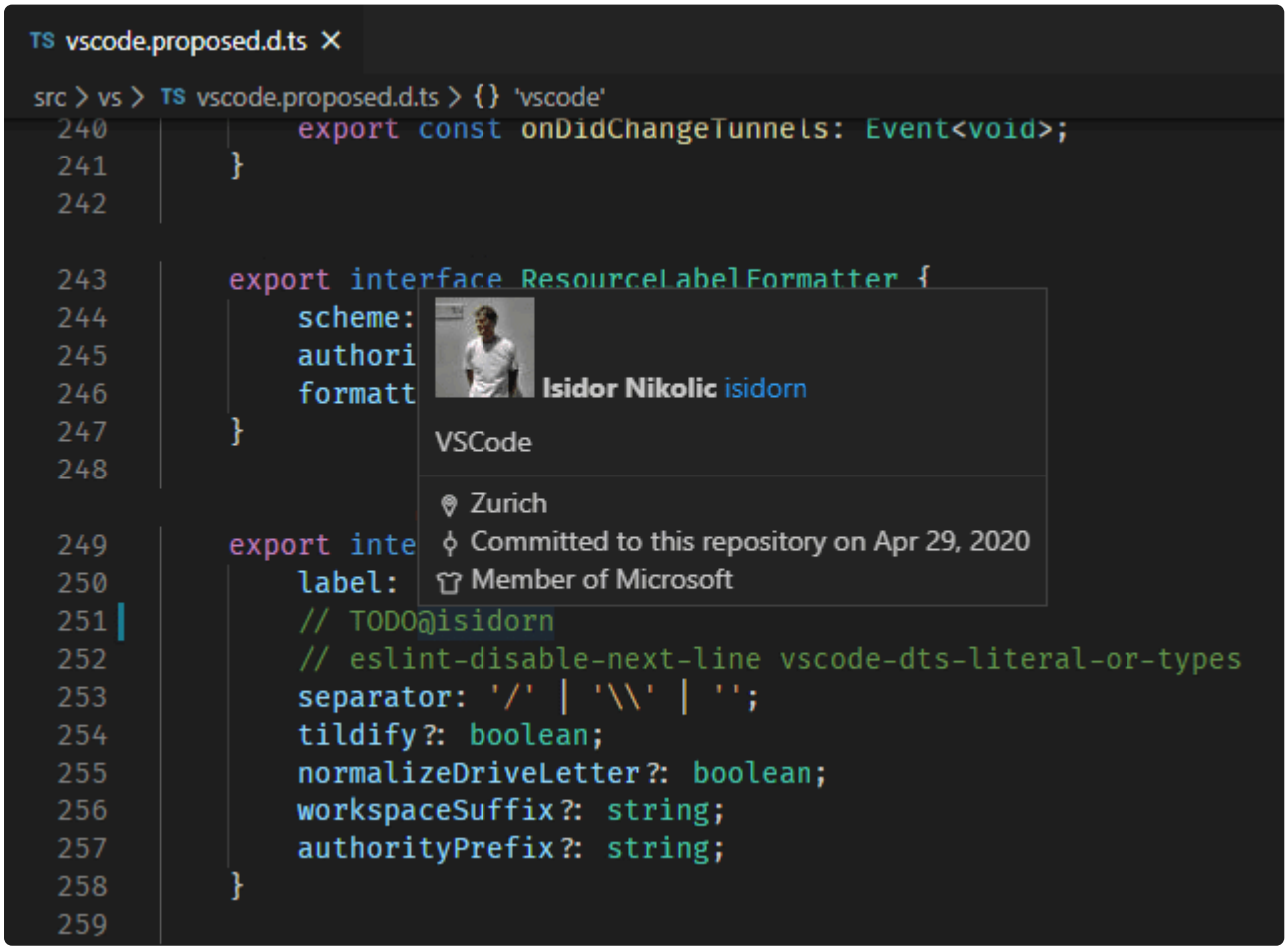
Note that there are several ways to authenticate to GitHub, including using your username and password with two-factor authentication (2FA), a personal access token, or an SSH key. See About authentication to GitHub (https://docs.github.com/github/authenticating-to-github/about-authentication-to-github) for more information and details about each option.

> **Note**
> If you'd like to work on a repository without cloning the contents to your local machine, you can install the GitHub Repositories (https://marketplace.visualstudio.com/items?itemName=github.remotehub) extension to browse and edit directly on GitHub. Learn more about the GitHub Repositories extension (/docs/sourcecontrol/github#_github-repositories-extension).

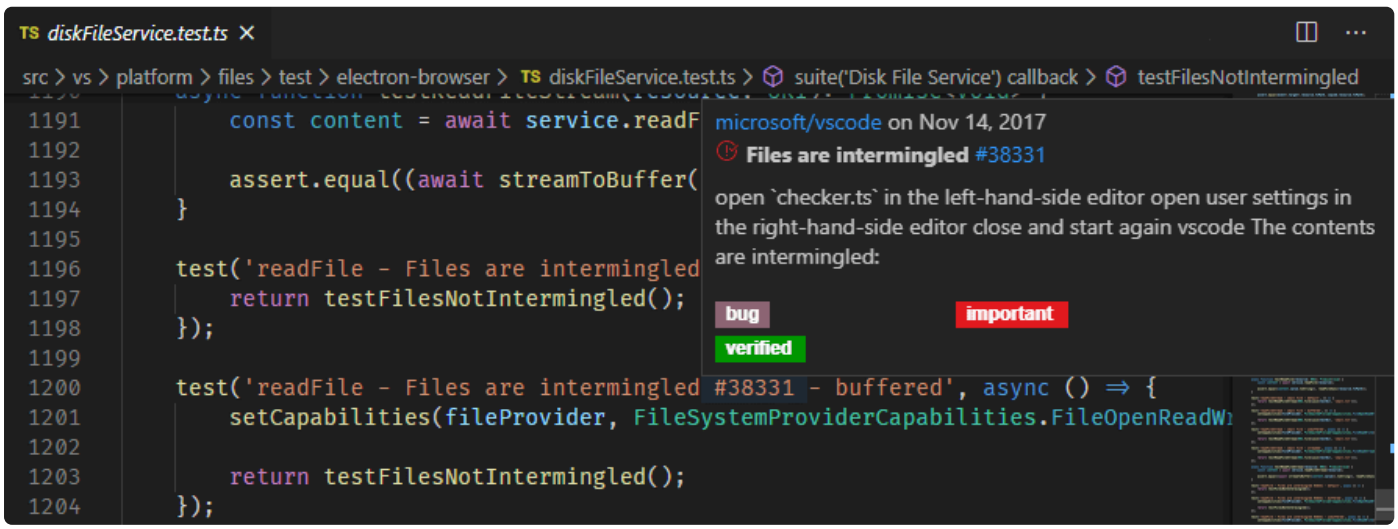
Editor integration

Hovers

When you have a repository open and a user is @-mentioned (for example, in a code comment), you can hover over that username and see a GitHub-style hover with the user's details.

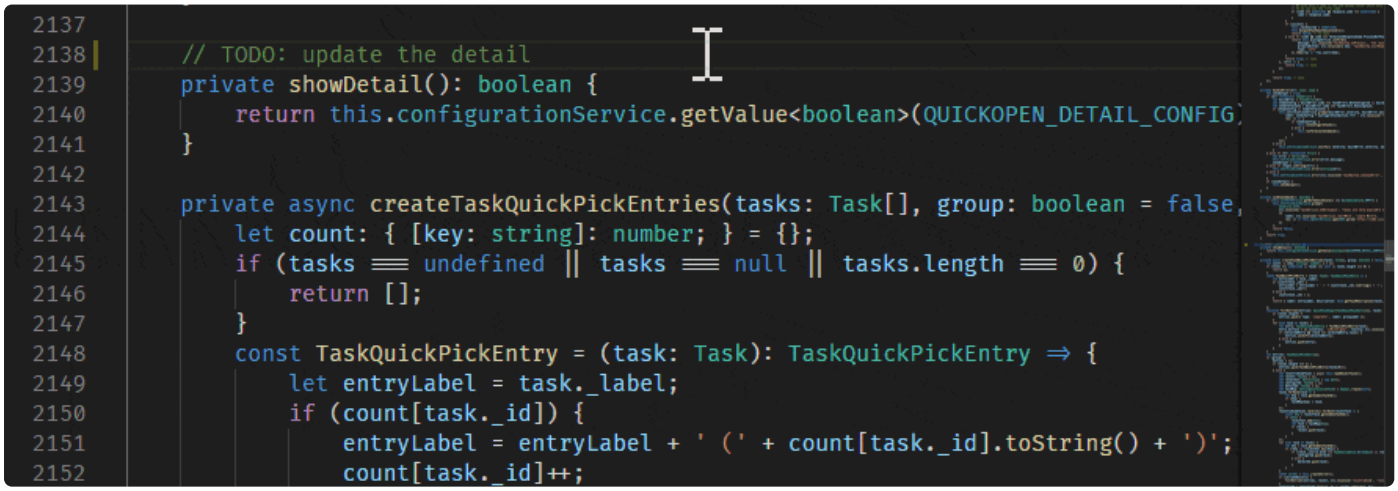


There is a similar hover for #-mentioned issue numbers, full GitHub issue URLs, and repository specified issues.



Suggestions

User suggestions are triggered by typing the "@" character and issue suggestions are triggered by typing the "#" character. Suggestions are available in the editor and in the Source Control commit message input box.



```
// TODO: update the detail
private showDetail(): boolean {
    return this.configurationService.getValue<boolean>(QUICKOPEN_DETAIL_CONFIG
}

private async createTaskQuickPickEntries(tasks: Task[], group: boolean = false,
    let count: { [key: string]: number; } = {};
    if (tasks === undefined || tasks === null || tasks.length === 0) {
        return [];
    }
    const TaskQuickPickEntry = (task: Task): TaskQuickPickEntry => {
        let entryLabel = task._label;
        if (count[task._id]) {
            entryLabel = entryLabel + ' (' + count[task._id].toString() + ')';
            count[task._id]++;
```

The issues that appear in the suggestion can be configured with the **GitHub Issues: Queries** (⚙ `githubIssues.queries (vscode://settings/githubIssues.queries)`) setting. The queries use the GitHub search syntax (https://docs.github.com/search-github/getting-started-with-searching-on-github/understanding-the-search-syntax).

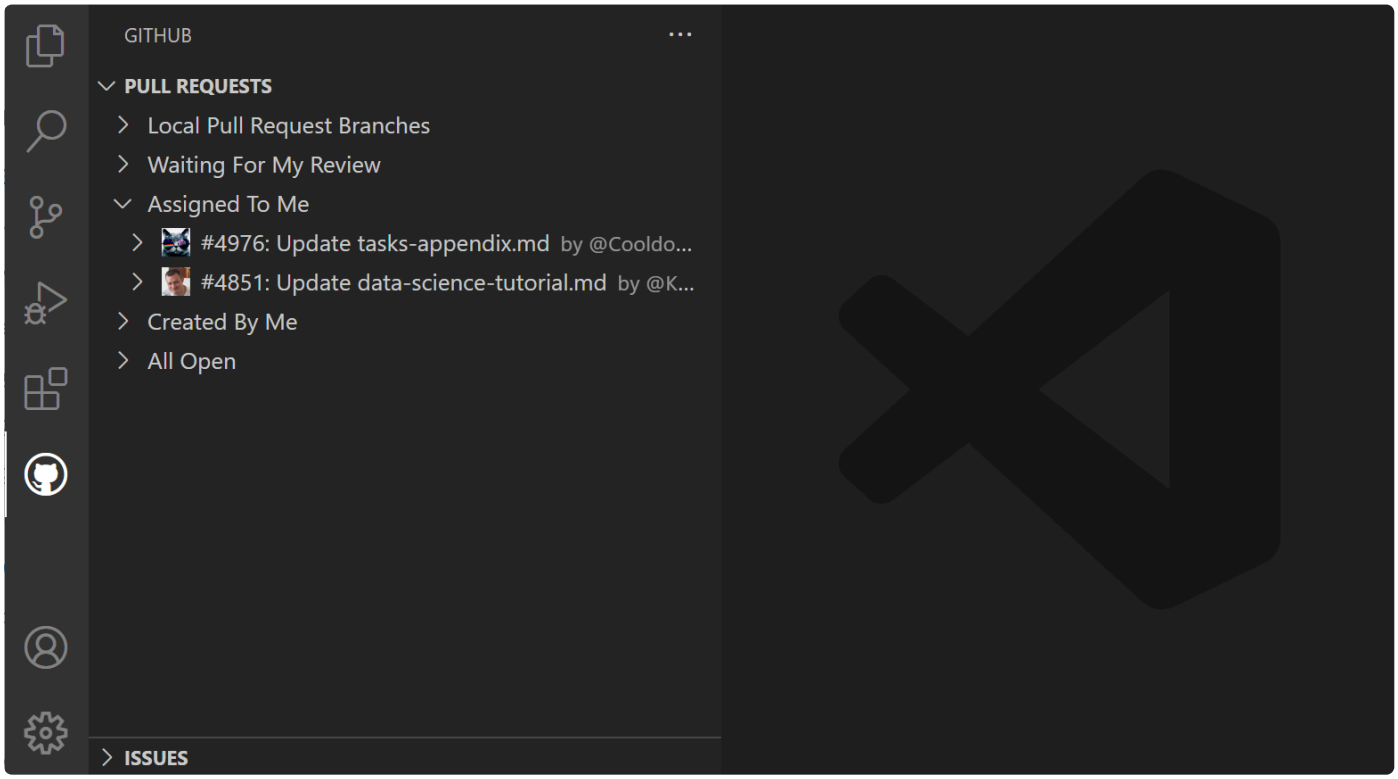
You can also configure which file types show these suggestions by using the settings **GitHub Issues: Ignore Completion Trigger** (⚙ `githubIssues.ignoreCompletionTrigger (vscode://settings/githubIssues.ignoreCompletionTrigger)`) and **GitHub Issues: Ignore User Completion Trigger** (⚙ `githubIssues.ignoreUserCompletionTrigger (vscode://settings/githubIssues.ignoreUserCompletionTrigger)`). These settings take an array of language identifiers (/docs/languages/identifiers) to specify the file types.

Jsonc

```
// Languages that the '#' character should not be used to trigger issue completion su
ggestions.
"githubIssues.ignoreCompletionTrigger": [
  "python"
]
```

Pull requests

From the **Pull Requests** view you can view, manage, and create pull requests.



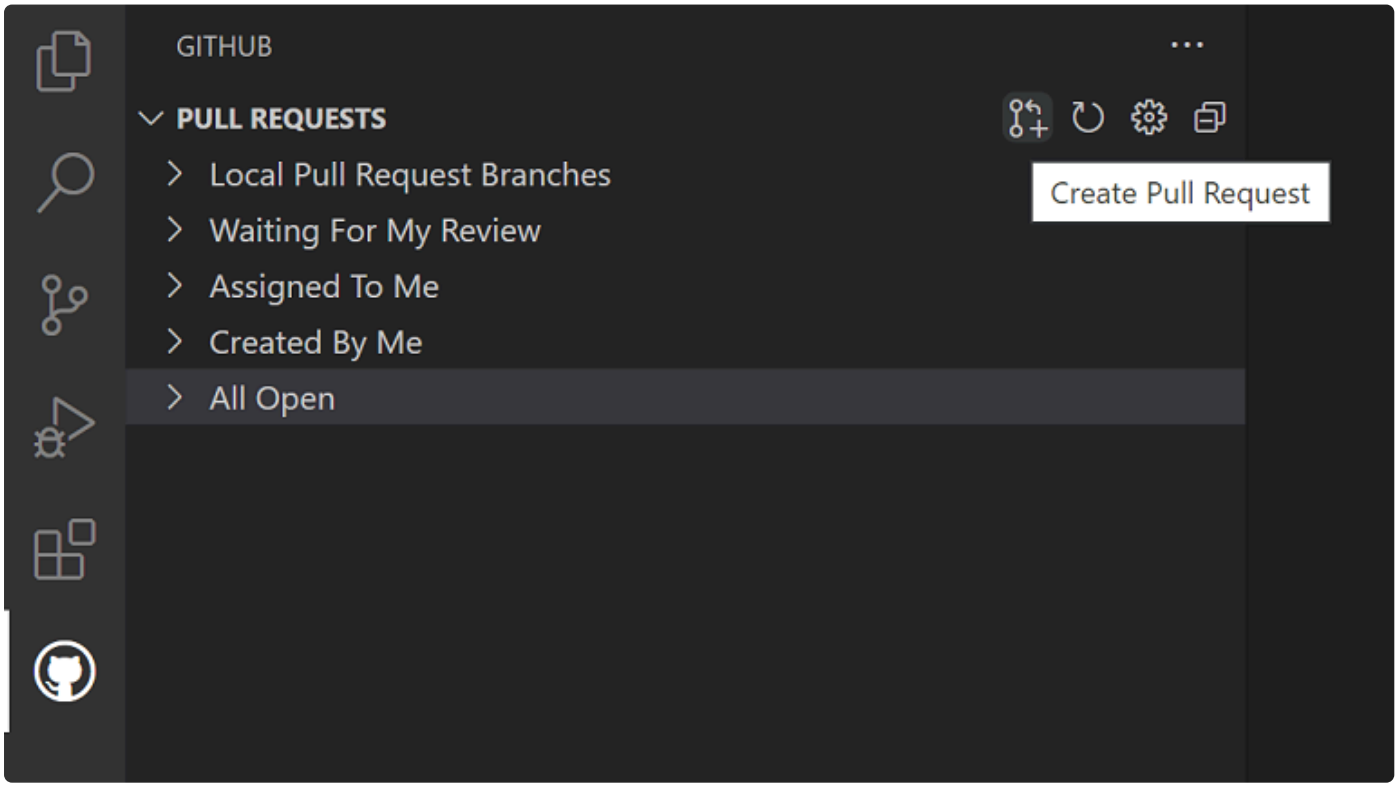
The queries used to display pull requests can be configured with the **GitHub Pull Requests: Queries** (`githubPullRequests.queries (vscode://settings/githubPullRequests.queries)`) setting and use the GitHub search syntax (https://docs.github.com/search-github/getting-started-with-searching-on-github/understanding-the-search-syntax).

JSON

```
"githubPullRequests.queries": [
    {
        "label": "Assigned To Me",
        "query": "is:open assignee:${user}"
    },
```

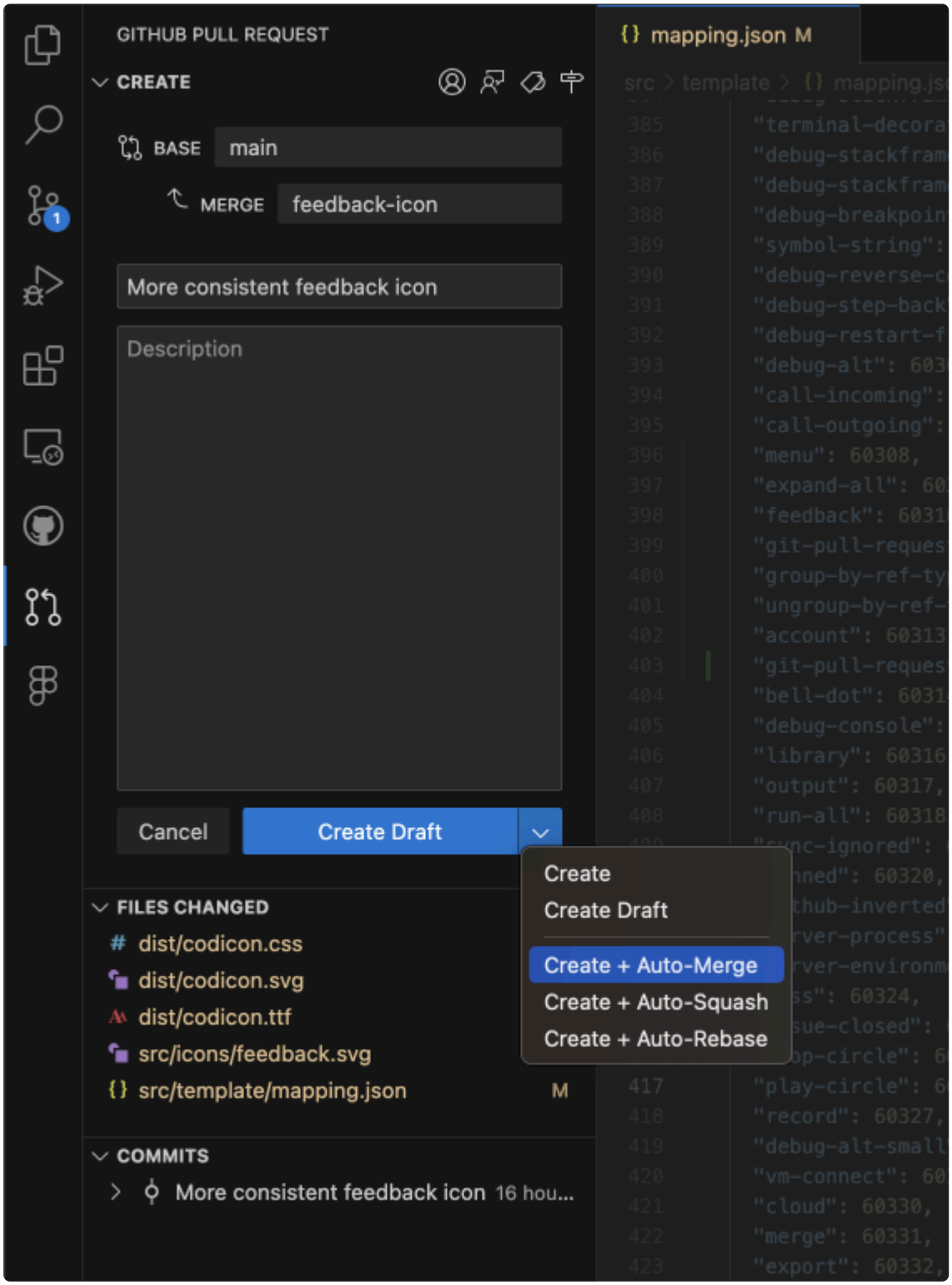
Creating pull requests

Once you have committed changes to your fork or branch, you can use the **GitHub Pull Requests: Create Pull Request** command or the **Create Pull Request** button in the **Pull Requests** view to create a pull request.



A new **Create** view will be displayed where you can select the base repository and base branch you'd like your pull request to target as well as fill in the title and description. If your repository has a pull request template, this will automatically be used for the description.

Use the buttons in the action bar at the top to add **Assignees**, **Reviewers**, **Labels** and a **Milestone**.

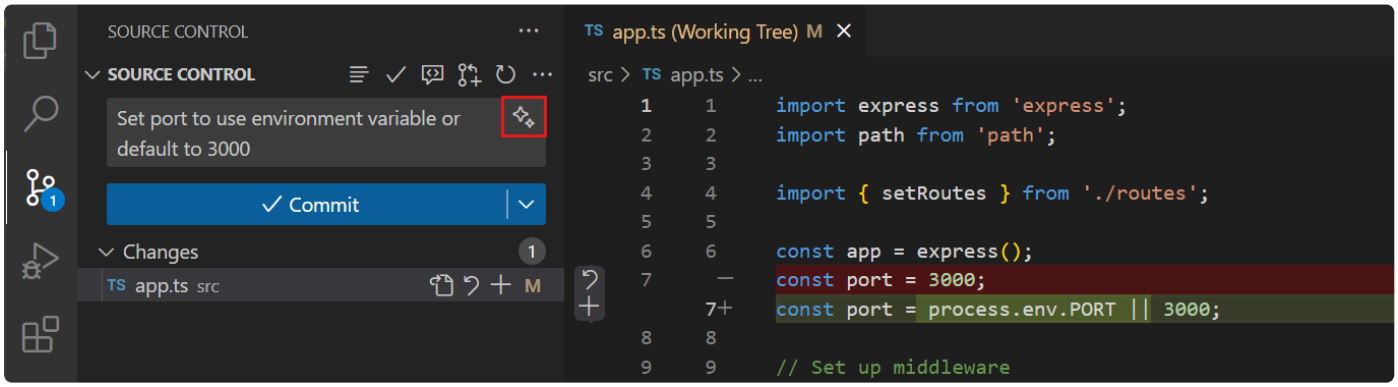


The **Create** button menu allows you to select alternative create options, such as **Create Draft** or enable an **Auto-Merge** method.

Once you select **Create**, if you have not already pushed your branch to a GitHub remote, the extension will ask if you'd like to publish the branch and provides a dropdown to select the specific remote.

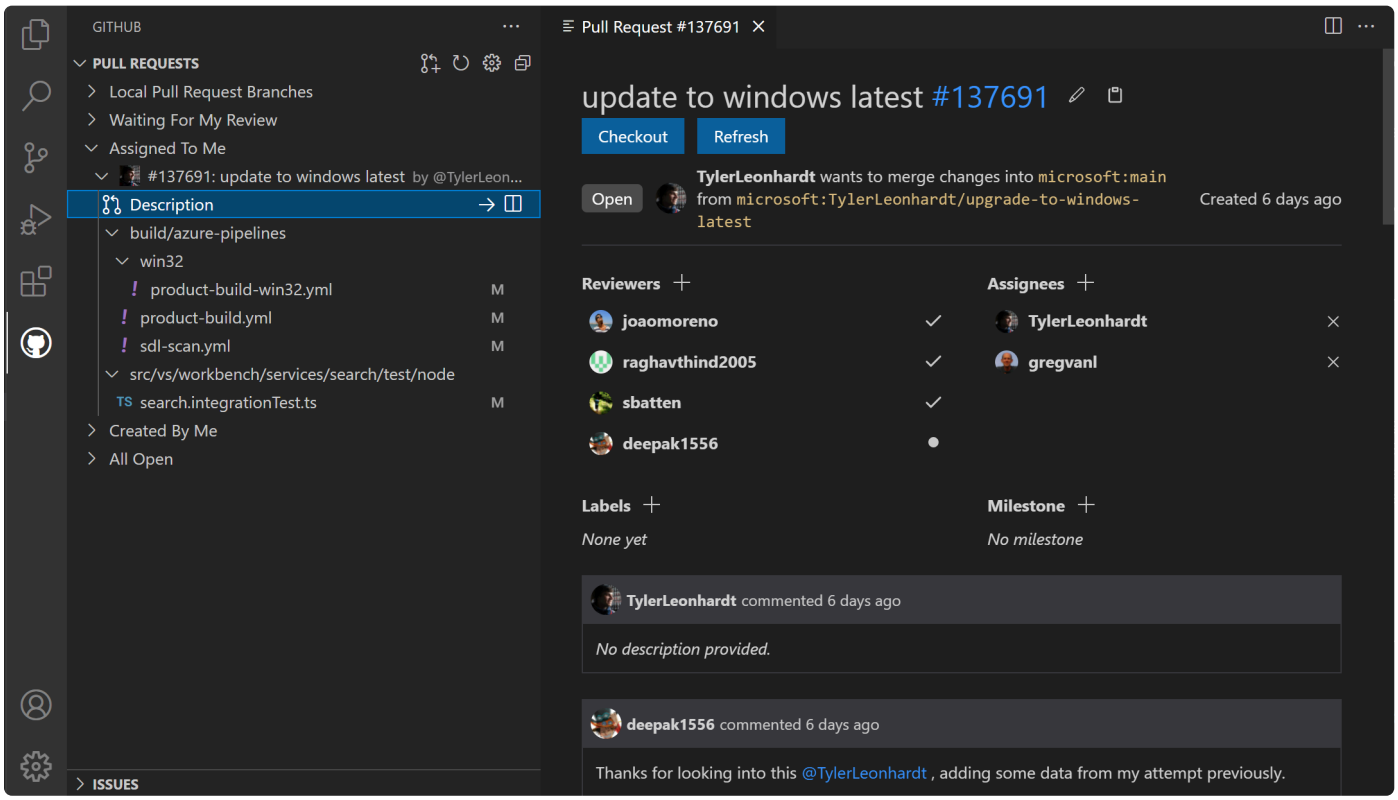
The **Create Pull Request** view now enters **Review Mode**, where you can review the details of the PR, add comments, and merge the PR once it's ready. After the PR is merged, you'll have the option to delete both the remote and local branch.

> Tip
> Use AI to generate a PR title and description, based on the commits that are included in the PR. Select the *sparkle* icon next to the PR title field to generate a PR title and description.

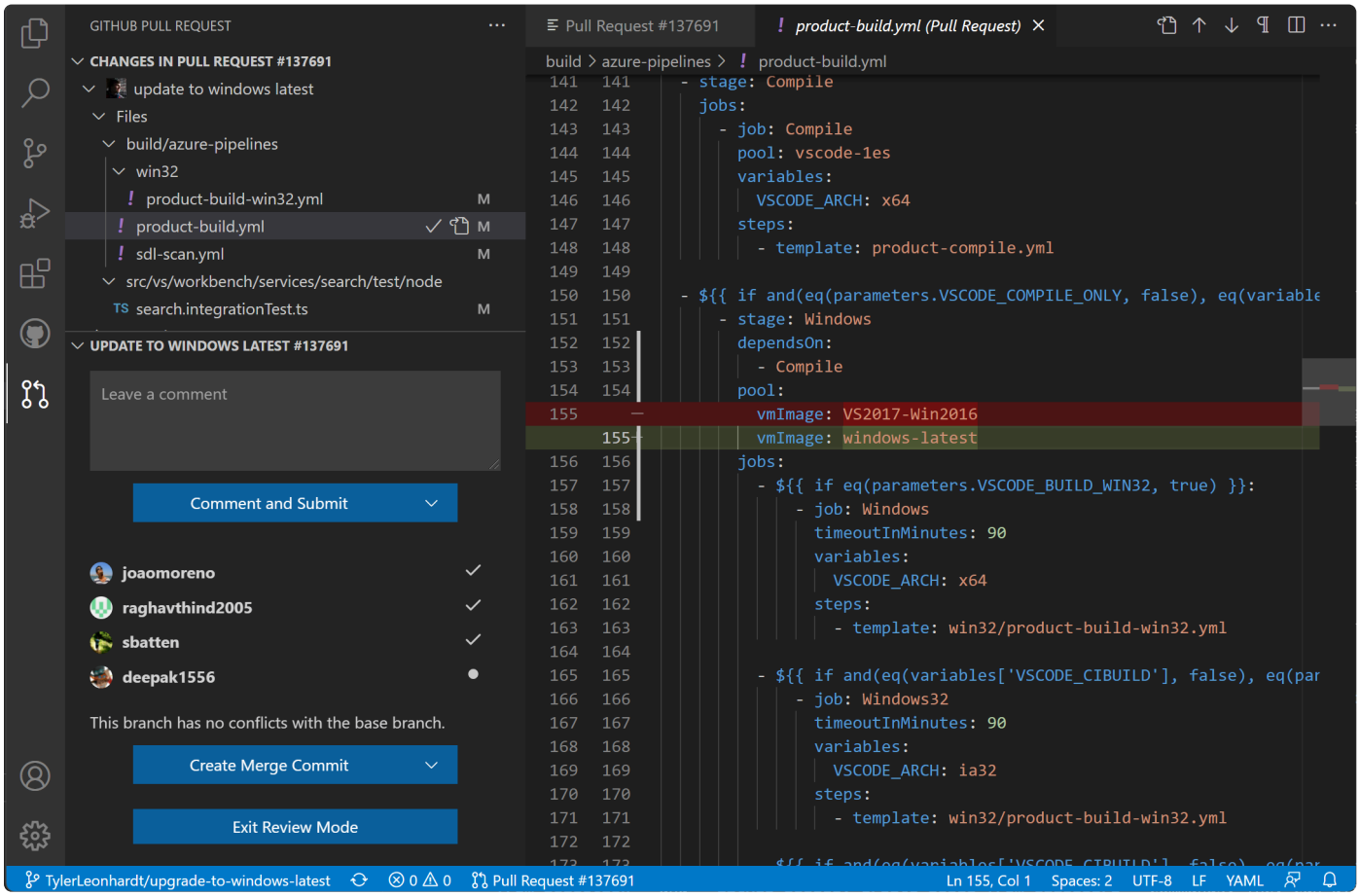


Reviewing

Pull requests can be reviewed from the **Pull Requests** view. You can assign reviewers and labels, add comments, approve, close, and merge all from the pull request **Description**.



From the **Description** page, you can also easily checkout the pull request locally using the **Checkout** button. This will switch VS Code to open the fork and branch of the pull request (visible in the Status Bar) in Review Mode and add a new **Changes in Pull Request** view from which you can view diffs of the current changes as well as all commits and the changes within these commits. Files that have been commented on are decorated with a diamond icon. To view the file on disk, you can use the **Open File** inline action.



The diff editors from this view use the local file, so file navigation, IntelliSense, and editing work as normal. You can add comments within the editor on these diffs. Both adding single comments and creating a whole review is supported.

When you are done reviewing the pull request changes you can merge the PR or select **Exit Review Mode** to go back to the previous branch you were working on.

> Tip
> You can also use AI to perform a code review of the PR (https://docs.github.com/en/copilot/using-github-copilot/code-review/using-copilot-code-review?tool=vscode) before you create it. Select the **Code Review** button in the GitHub Pull Request view.

Issues

Creating issues

Issues can be created from the **+** button in the **Issues** view and by using the **GitHub Issues: Create Issue from Selection** and **GitHub Issues: Create Issue from Clipboard** commands. They can also be created using a Code Action for "TODO" comments. When creating issues, you can take the default description or select the **Edit Description** pencil icon in the upper right to bring up an editor for the issue body.

You can configure the trigger for the Code Action using the **GitHub Issues: Create Issue Triggers** (`githubIssues.createIssueTriggers (vscode://settings/githubIssues.createIssueTriggers)`) setting.

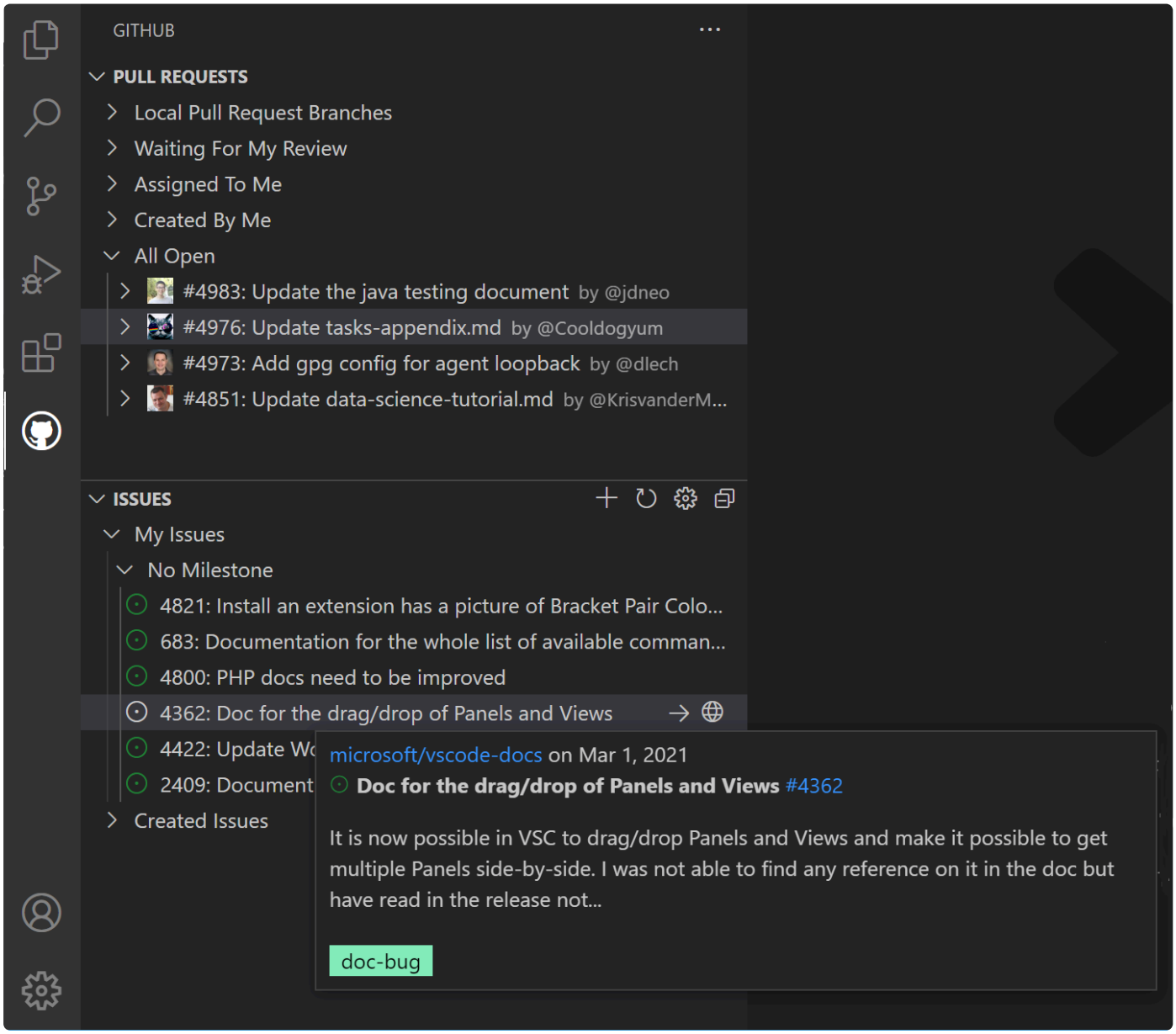
The default issue triggers are:

JSON

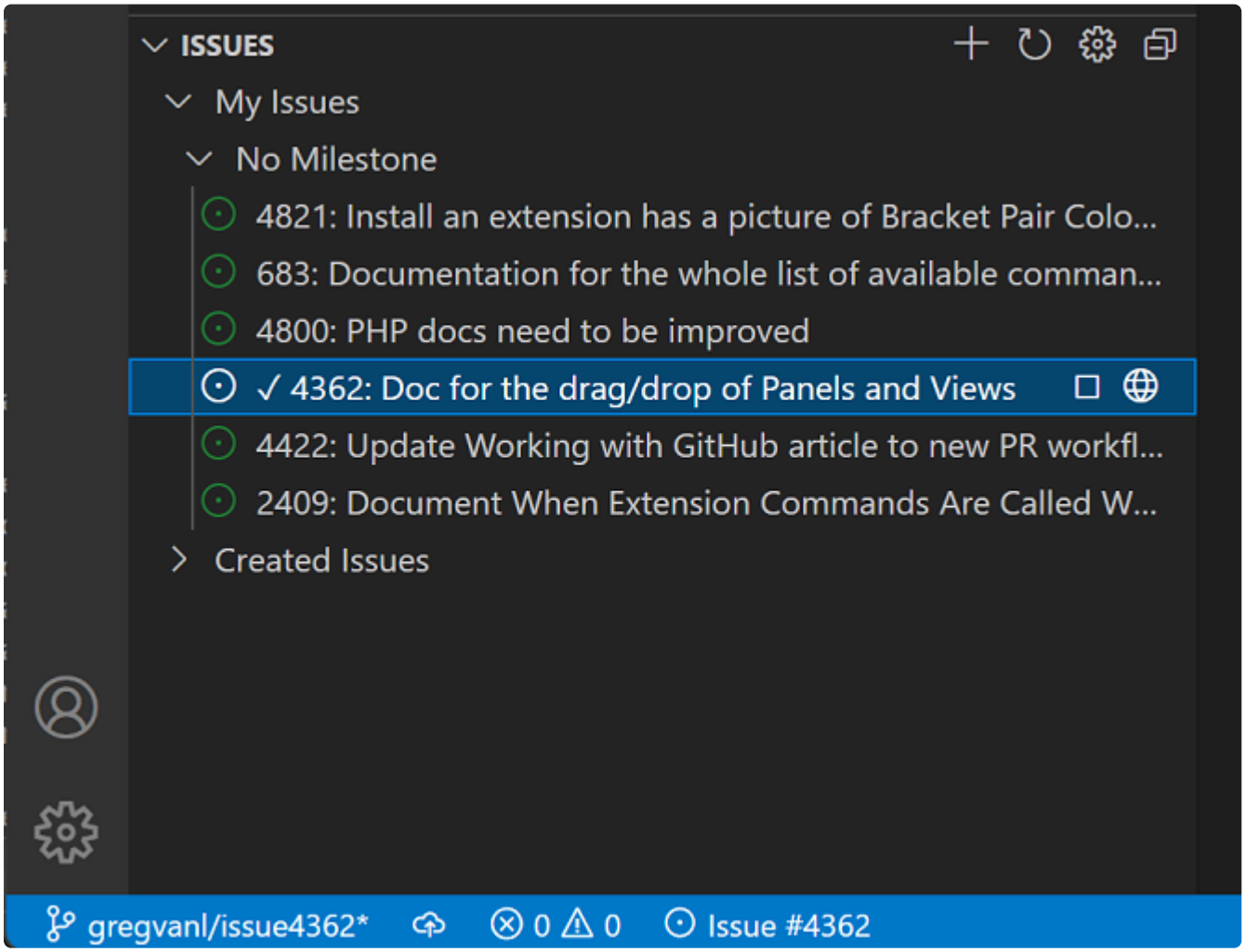
```
"githubIssues.createIssueTriggers": [
  "TODO",
  "todo",
  "BUG",
  "FIXME",
  "ISSUE",
  "HACK"
]
```

Working on issues

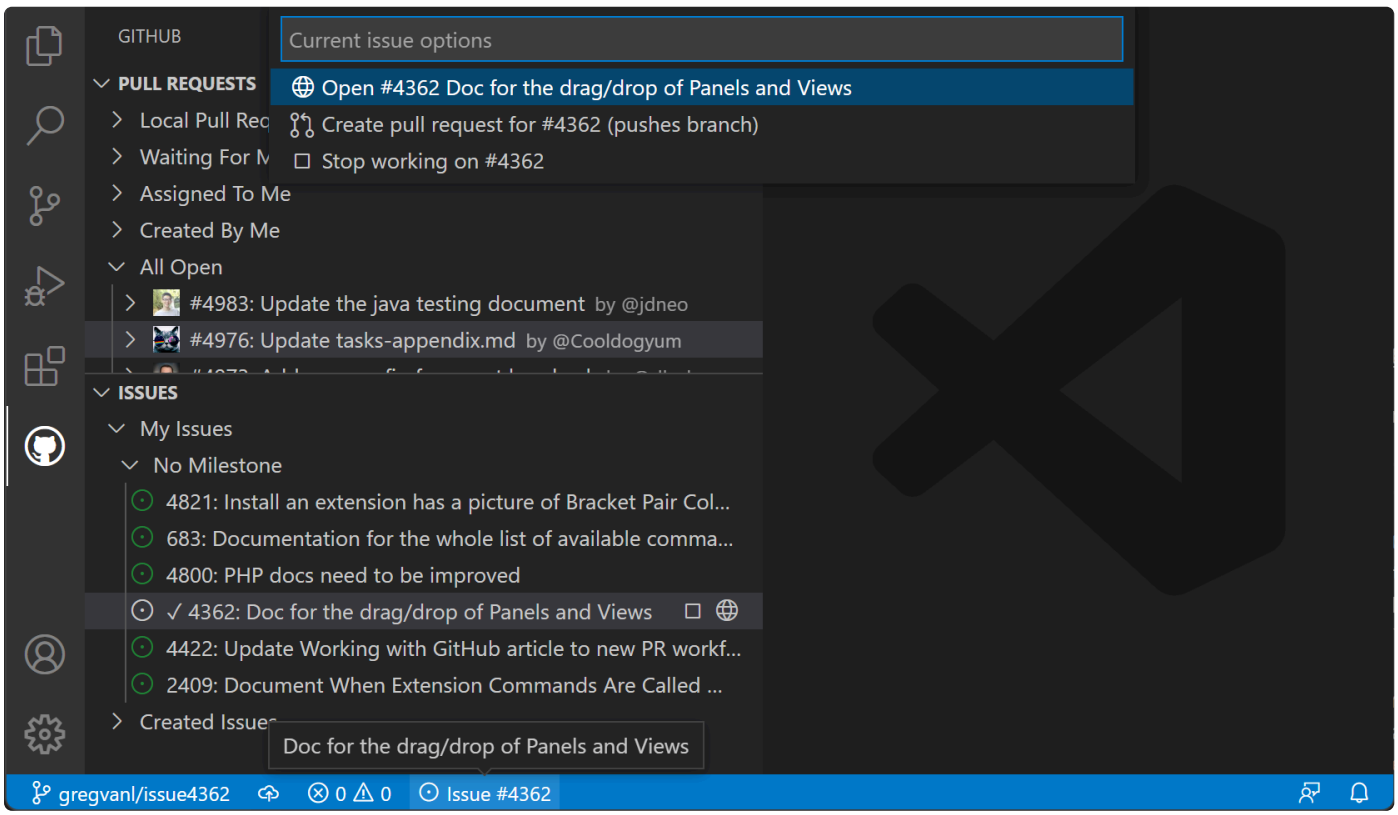
From the **Issues** view, you can see your issues and work on them.



By default, when you start working on an issue (**Start Working on Issue** context menu item), a branch will be created for you, as shown in the Status Bar in the image below.



The Status Bar also shows the active issue and if you select that item, a list of issue actions are available such as opening the issue on the GitHub website or creating a pull request.



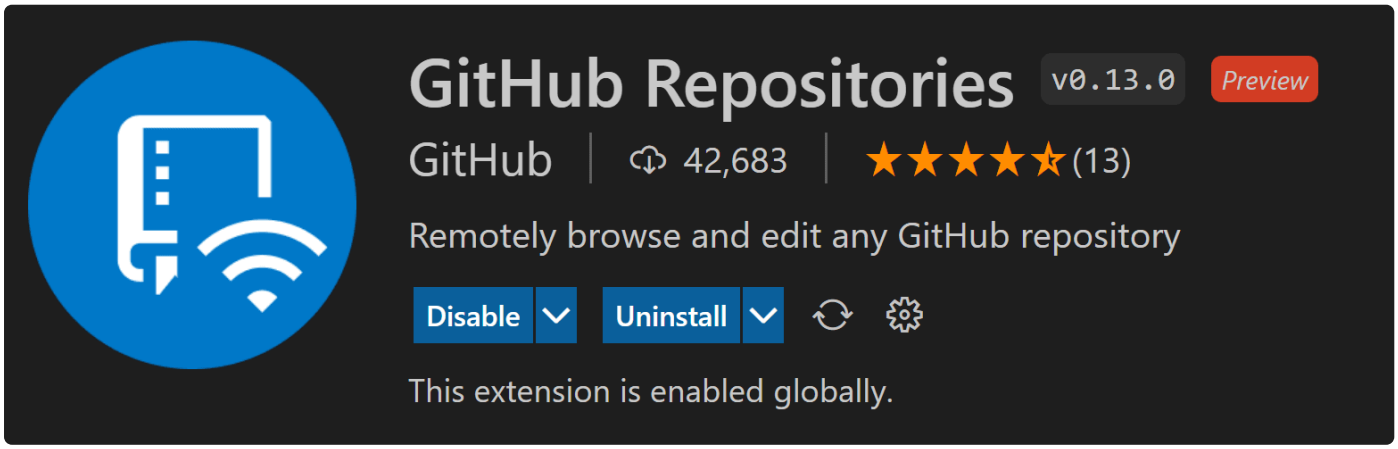
You can configure the name of the branch using the **GitHub Issues: Issue Branch Title** (`githubIssues.issueBranchTitle (vscode://settings/githubIssues.issueBranchTitle)`) setting. If your workflow doesn't involve creating a branch, or if you want to be prompted to enter a branch name every time, you can skip that step by turning off the **GitHub Issues: Use Branch For Issues** (`githubIssues.useBranchForIssues (vscode://settings/githubIssues.useBranchForIssues)`) setting.

> Tip
> Learn more about working with branches (/docs/sourcecontrol/branches-worktrees) to understand branch management, switching between branches, and organizing your development work.

Once you are done working on the issue and want to commit a change, the commit message input box in the **Source Control** view will be populated with a message, which can be configured with **GitHub Issues: Working Issue Format SCM** (`githubIssues.workingIssueFormatScm (vscode://settings/githubIssues.workingIssueFormatScm)`).

GitHub Repositories extension

The GitHub Repositories (https://marketplace.visualstudio.com/items?itemName=github.remotehub) extension lets you quickly browse, search, edit, and commit to any remote GitHub repository directly from within Visual Studio Code, without needing to clone the repository locally. This can be fast and convenient for many scenarios, where you just need to review source code or make a small change to a file or asset.

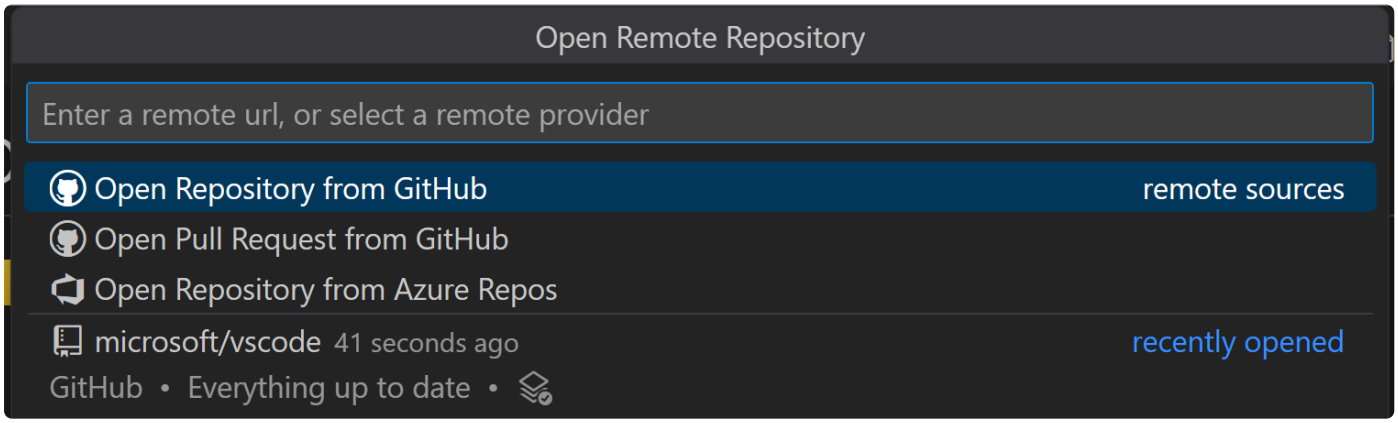


Opening a repository

Once you have installed the GitHub Repositories extension, you can open a repository with the **GitHub Repositories: Open Repository...** command from the Command Palette (`Ctrl+Shift+P`) or by clicking the Remote indicator in the lower left of the Status Bar.

When you run the **Open Repository** command, you then choose whether to open a repository from GitHub, open a Pull Request from GitHub, or reopen a repository that you had previously connected to.

If you haven't logged into GitHub from VS Code before, you'll be prompted to authenticate with your GitHub account.



You can provide the repository URL directly or search GitHub for the repository you want by typing in the text box.

Once you have selected a repository or Pull Request, the VS Code window will reload and you will see the repository contents in the File Explorer. You can then open files (with full syntax highlighting and bracket matching), make edits, and commit changes, just like you would working on a local clone of a repository.

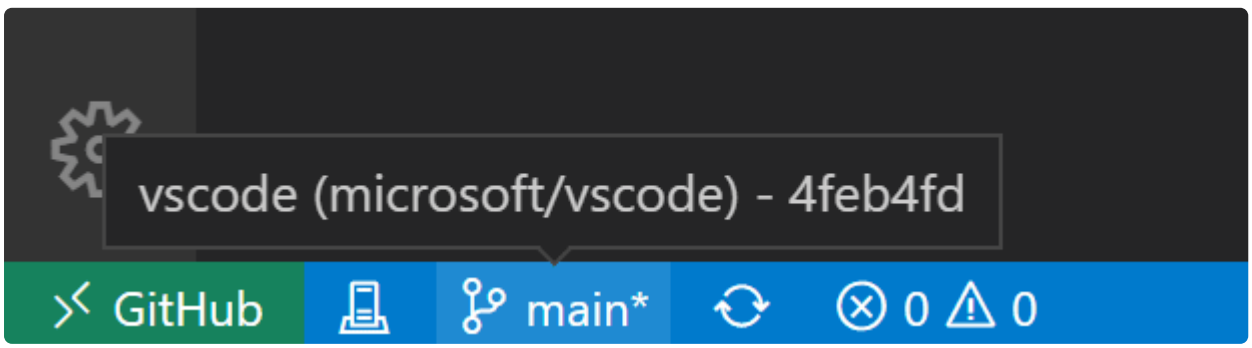
One difference from working with a local repository is that when you commit a change with the GitHub Repository extension, the changes are pushed directly to the remote repository, similar to if you were working in the GitHub web interface.

Another feature of the GitHub Repositories extension is that every time you open a repository or branch, you get the up-to-date sources available from GitHub. You don't need to remember to pull to refresh as you would with a local repository.

The GitHub Repositories extension supports viewing and even committing LFS-tracked files without needing to install Git LFS (https://git-lfs.github.com) (Large File System) locally. Add the file types you want tracked with LFS to a `.gitattributes` file (https://git-lfs.com), then commit your changes directly to GitHub using the Source Control view.

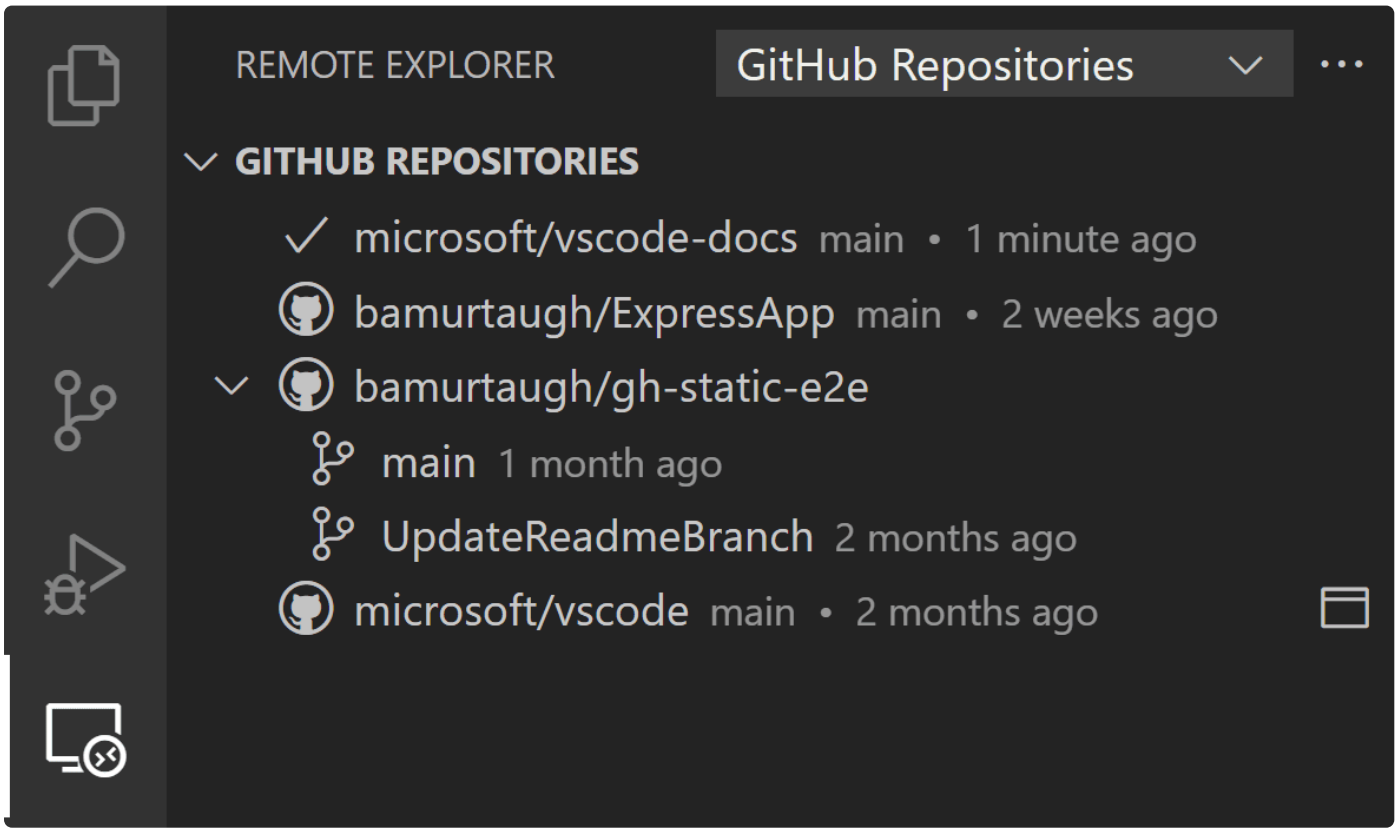
Switching branches

You can easily switch between branches by clicking on the branch indicator in the Status Bar. One great feature of the GitHub Repositories extension is that you can switch branches without needing to stash uncommitted changes. The extension remembers your changes and reapplies them when you switch branches.



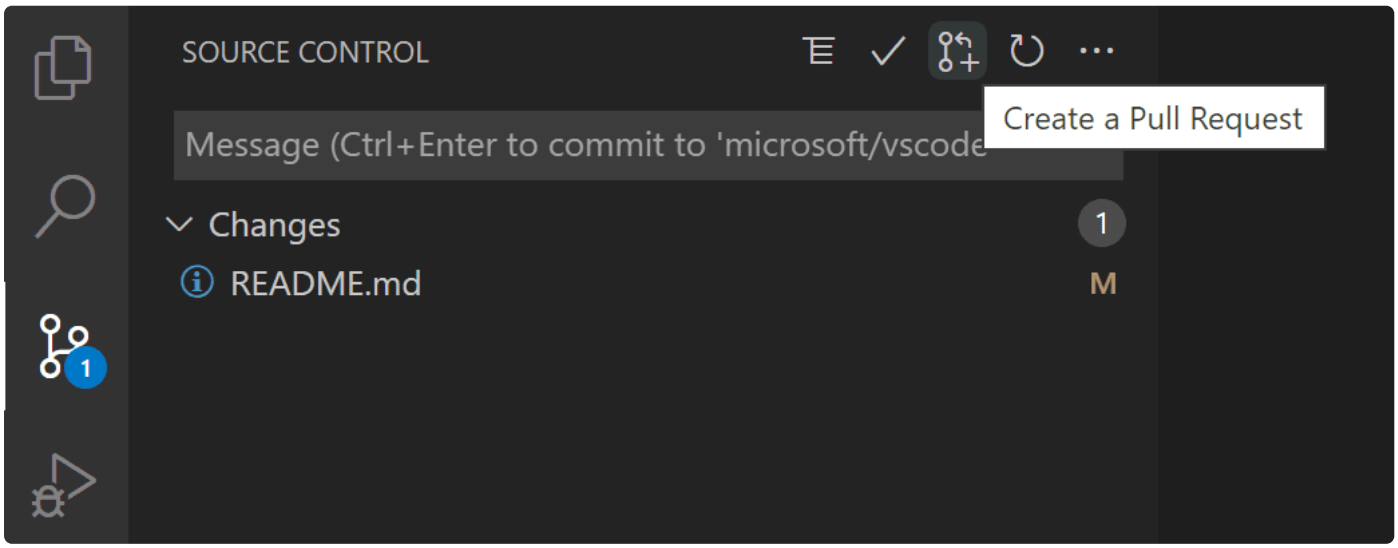
Remote Explorer

You can quickly reopen remote repositories with the Remote Explorer available on the Activity bar. This view shows you the previously opened repositories and branches.



Create pull requests

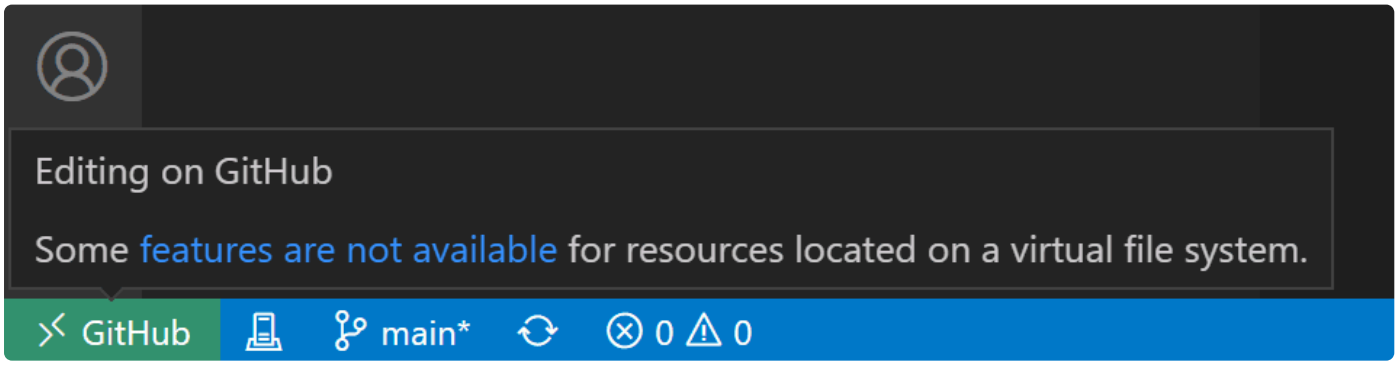
If your workflow uses Pull Requests, rather than direct commits to a repository, you can create a new PR from the Source Control view. You'll be prompted to provide a title and create a new branch.



Once you have created a Pull Request, you can use the GitHub Pull Request and Issues (https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github) extension to review, edit, and merge your PR as described earlier (/docs/sourcecontrol/github#_pull-requests) in this topic.

Virtual file system

Without a repository's files on your local machine, the GitHub Repositories extension creates a virtual file system in memory so you can view file contents and make edits. Using a virtual file system means that some operations and extensions which assume local files are not enabled or have limited functionality. Features such as tasks, debugging, and integrated terminals are not enabled and you can learn about the level of support for the virtual file system via the **features are not available** link in the Remote indicator hover.



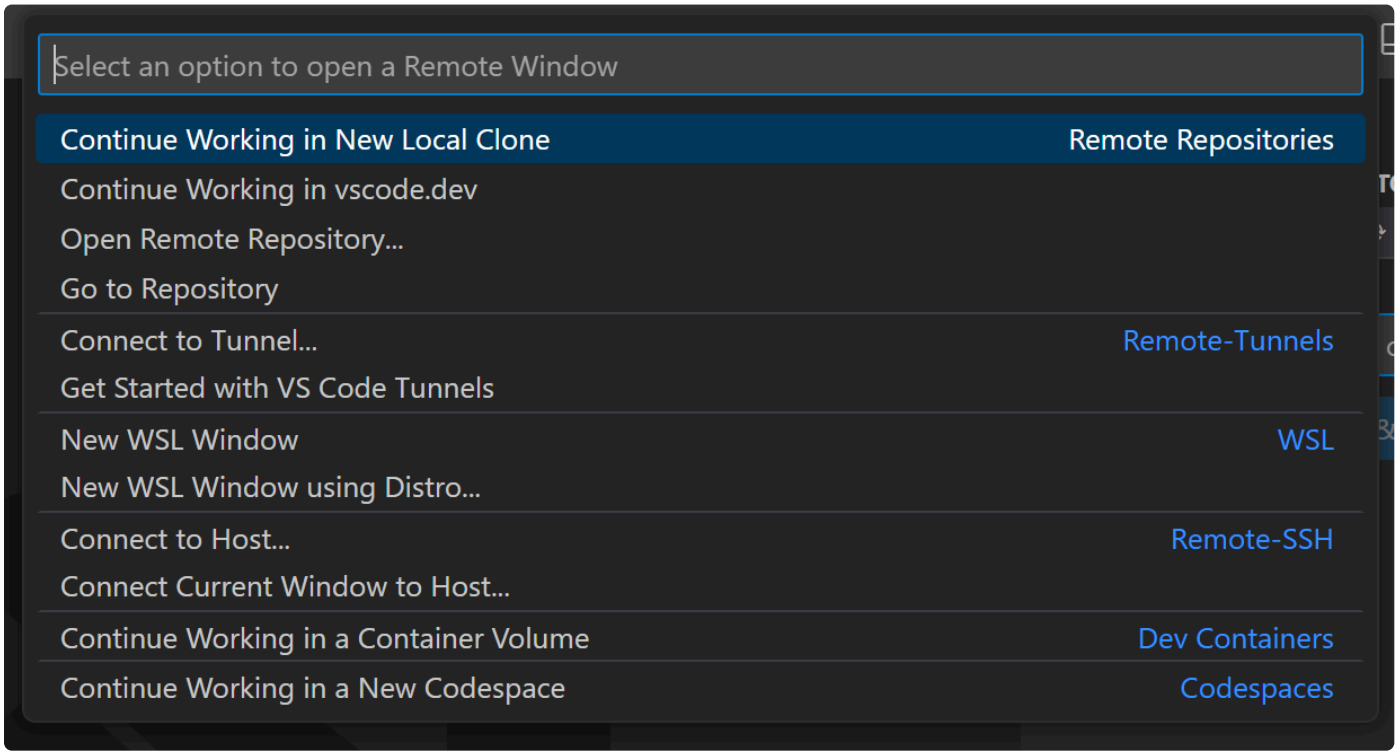
Extension authors can learn more about running in a virtual file system and workspace in the Virtual Workspaces extension author's guide (https://github.com/microsoft/vscode/wiki/Virtual-Workspaces).

Continue working on

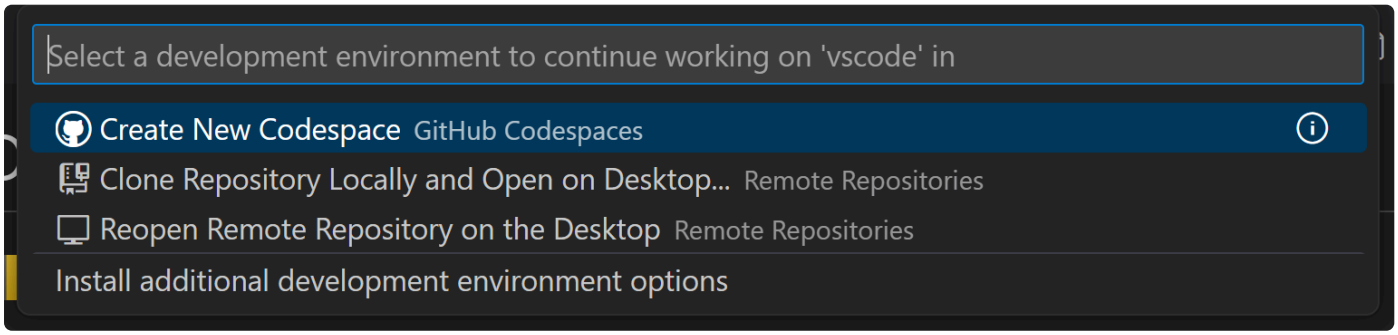
Sometimes you'll want to switch to working on a repository in a development environment with support for a local file system and full language and development tooling. The GitHub Repositories extension makes it easy for you to:

- Create a GitHub codespace (if you have the GitHub Codespaces extension (https://marketplace.visualstudio.com/items?itemName=GitHub.codespaces)).
- Clone the repository locally.
- Clone the repository into a Docker container (if you have Docker (https://docker.com/) and the Microsoft Container Tools extension (https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-containers) installed).

To switch development environments, use the **Continue Working On** command, available from the Command Palette (`Ctrl+Shift+P`) or by clicking on the Remote indicator in the Status Bar.



If you are using the browser-based editor (/docs/remote/codespaces#_browserbased-editor), the **"Continue Working On"** command has the options to open the repository locally or within a cloud-hosted environment in GitHub Codespaces (https://github.com/features/codespaces).



The first time that you use **Continue Working On** with uncommitted changes, you will have the option to bring your edits to your selected development environment using **Cloud Changes**, which stores your pending changes on the same VS Code service used for Settings Sync.

These changes are deleted from our service once they are applied to your target development environment. If you choose to continue without your uncommitted changes, you can always change this preference later by configuring the setting `"workbench.cloudChanges.continueOn": "prompt"`.

In the event that your pending changes are not automatically applied to your target development environment, you can view, manage, and delete your stored changes using the **Cloud Changes: Show Cloud Changes** command.

Next steps

- Learn more about AI in VS Code (/docs/copilot/overview) - Learn about AI features in VS Code.

Was this documentation helpful?

Yes No

Still need help?

Ask the community(https://stackoverflow.com/questions/tagged/vscode)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Help us improve

All VS Code docs are open source. See something that's wrong or unclear? Submit a pull request (https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/sourcecontrol/github.md).

01/08/2026

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)

Configure VS Code for Microsoft C++

In this tutorial, you configure Visual Studio Code to use the Microsoft Visual C++ compiler and debugger on Windows.

After configuring VS Code, you will compile and debug a simple Hello World program in VS Code. This tutorial does not teach you details about the Microsoft C++ toolset or the C++ language. For those subjects, there are many good resources available on the Web.

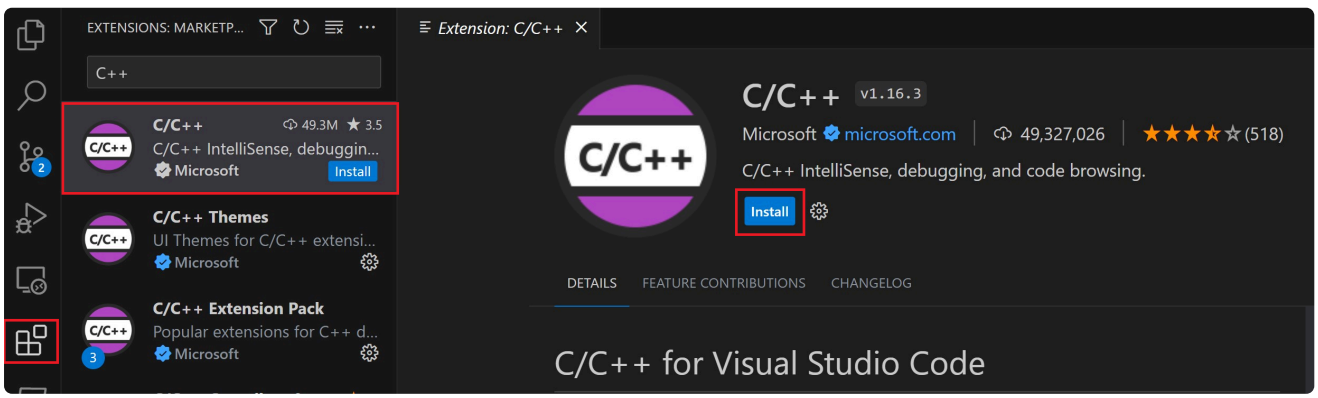
If you have any problems, feel free to file an issue for this tutorial in the VS Code documentation repository (https://github.com/microsoft/vscode-docs/issues).

Prerequisites

To successfully complete this tutorial, you must do the following:

1. Install Visual Studio Code (/download).

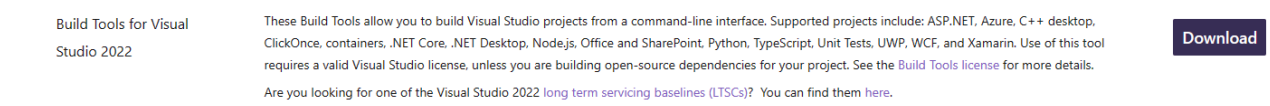
2. Install the C/C++ extension for VS Code (https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools). You can install the C/C++ extension by searching for 'c++' in the Extensions view (`Ctrl+Shift+X`).



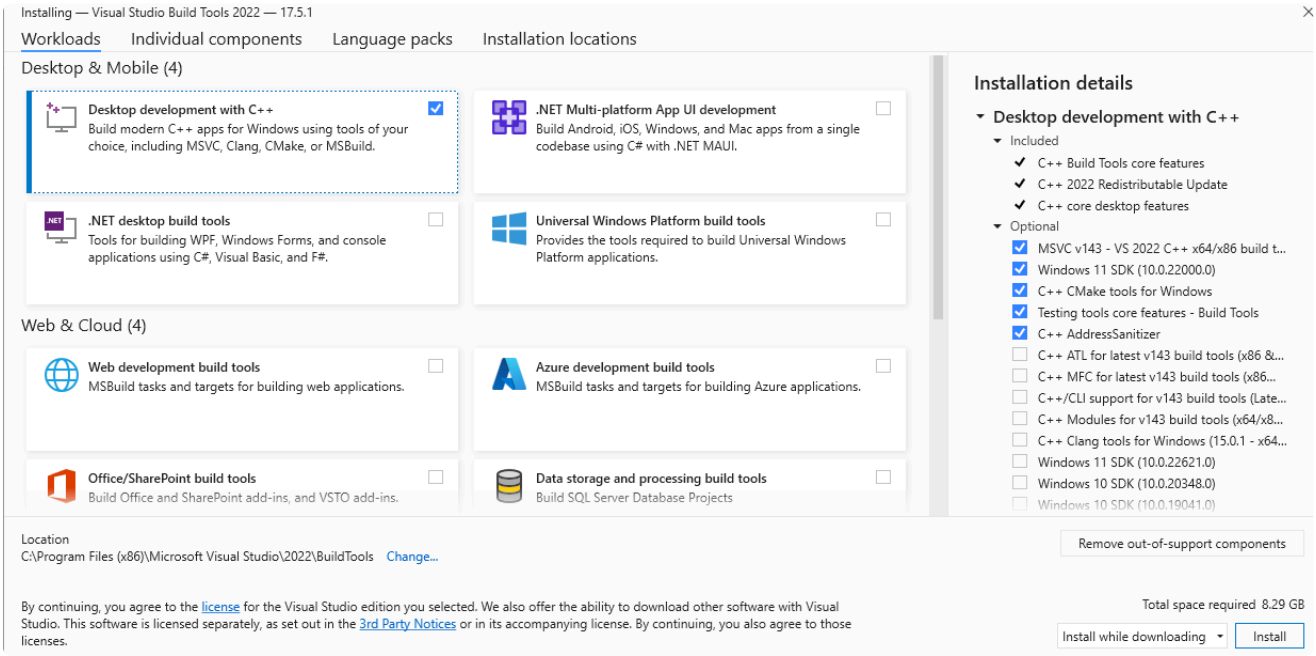
3. Install the Microsoft Visual C++ (MSVC) compiler toolset.

   If you have a recent version of Visual Studio, open the Visual Studio Installer from the Windows Start menu and verify that the C++ workload is checked. If it's not installed, then check the box and select the **Modify** button in the installer.

   You can also install the **Desktop development with C++** workload without a full Visual Studio IDE installation. From the Visual Studio Downloads (https://visualstudio.microsoft.com/downloads/#remote-tools-for-visual-studio-2022) page, scroll down until you see **Tools for Visual Studio** under the **All Downloads** section and select the download for **Build Tools for Visual Studio 2022**.



This will launch the Visual Studio Installer, which will bring up a dialog showing the available Visual Studio Build Tools workloads. Check the **Desktop development with C++** workload and select **Install**.

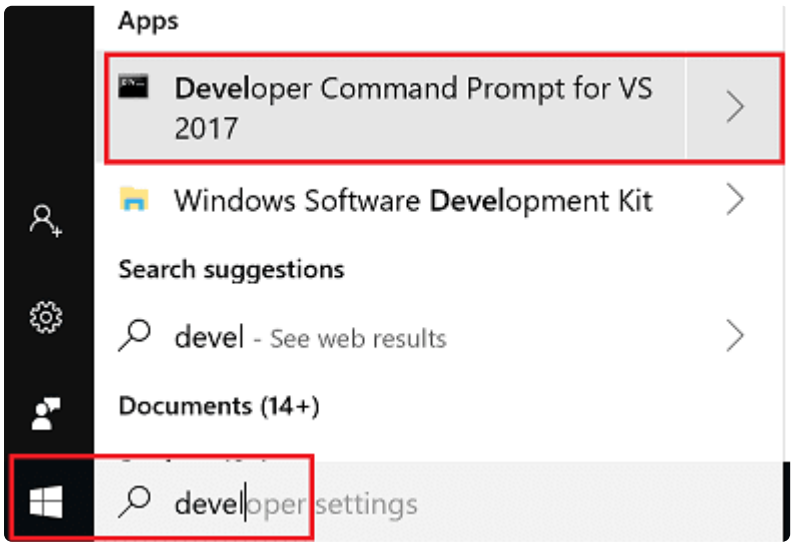


> **Note**: You can use the C++ toolset from Visual Studio Build Tools along with Visual Studio Code to develop, build, and test any C++ code as long as you also have a valid Visual Studio license (either Community, Pro, or Enterprise).

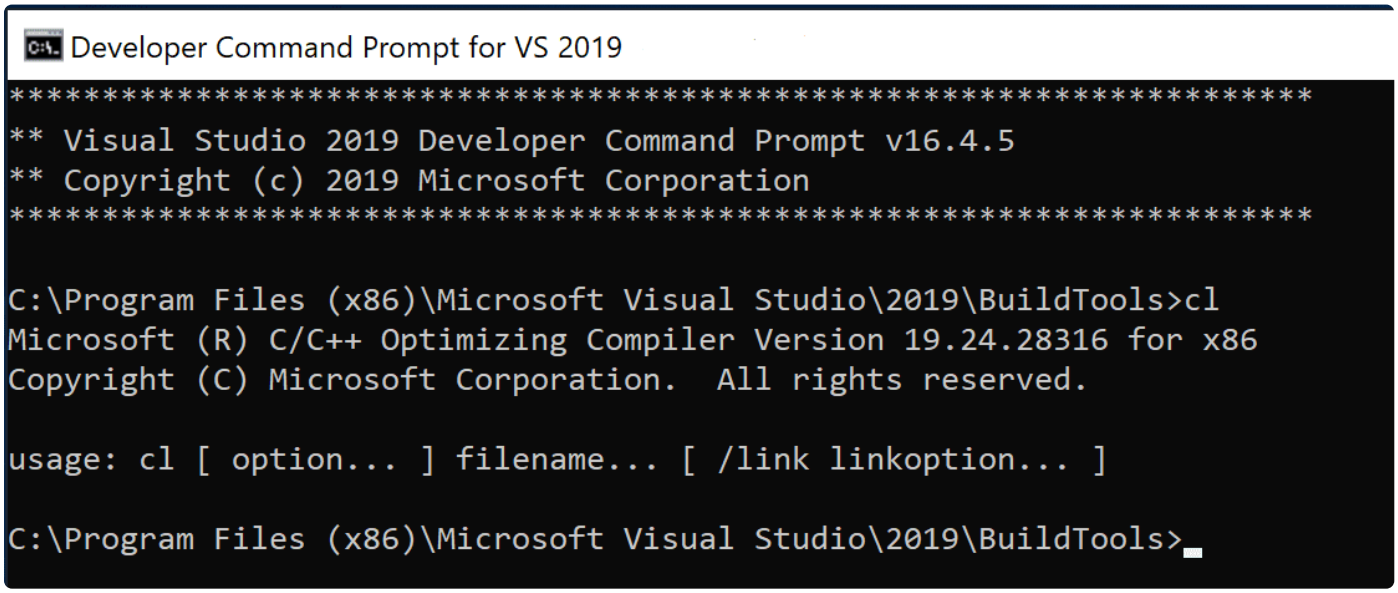
Check your Microsoft Visual C++ installation

To use MSVC from a command line or VS Code, you must run from a **Developer Command Prompt for Visual Studio**. An ordinary shell such as PowerShell, Bash, or the Windows command prompt does not have the necessary path environment variables set.

To open the Developer Command Prompt for VS, start typing 'developer' in the Windows Start menu, and you should see it appear in the list of suggestions. The exact name depends on which version of Visual Studio or the Visual Studio Build Tools you have installed. Select the item to open the prompt.



You can test that you have the C++ compiler, `cl.exe`, installed correctly by typing 'cl' and you should see a copyright message with the version and basic usage description.



If the Developer Command Prompt is using the BuildTools location as the starting directory (you wouldn't want to put projects there), navigate to your user folder (`C:\users\{your username}\`) before you start creating new projects.

> **Note**: If for some reason you can't run VS Code from a **Developer Command Prompt**, you can find a workaround for building C++ projects with VS Code in Run VS Code outside a Developer Command Prompt.

Create Hello World

From the Developer Command Prompt, create an empty folder called "projects" where you can store all your VS Code projects, then create a subfolder called "helloworld", navigate into it, and open VS Code (`code`) in that folder (`.`) by entering the following commands:

Bat

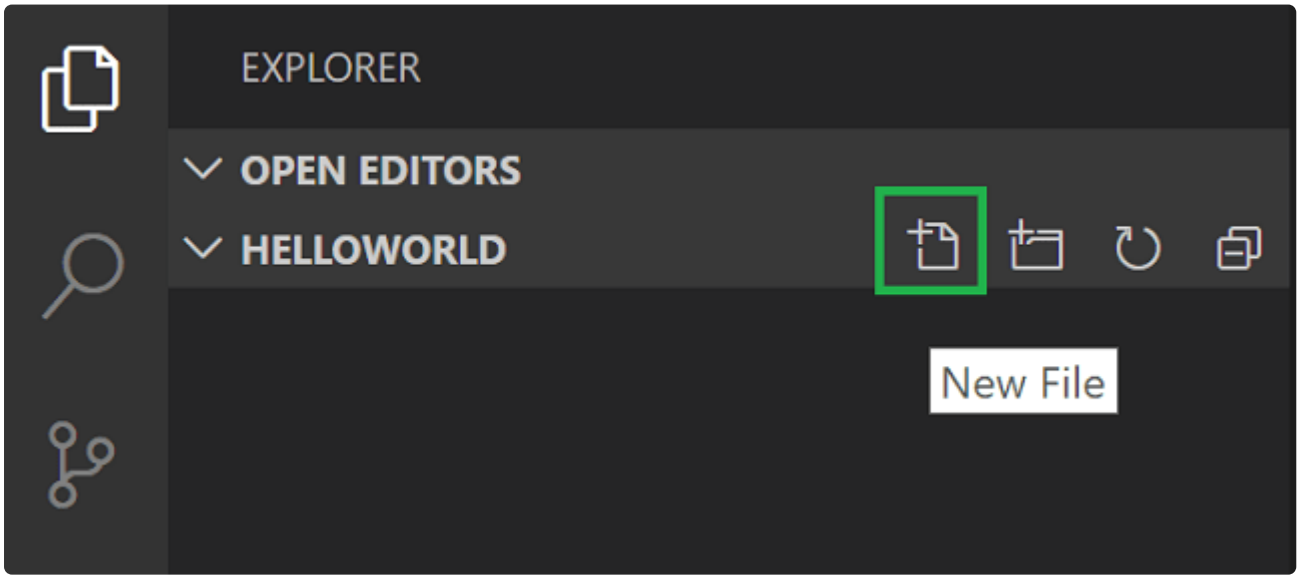
```
mkdir projects
cd projects
mkdir helloworld
cd helloworld
code .
```

The "code ." command opens VS Code in the current working folder, which becomes your "workspace". As you go through the tutorial, you will see three files created in a `.vscode` folder in the workspace:

- `tasks.json` (build instructions)
- `launch.json` (debugger settings)
- `c_cpp_properties.json` (compiler path and IntelliSense settings)

Add a source code file

In the File Explorer title bar, select the **New File** button and name the file `helloworld.cpp`.



Add hello world source code

Now paste in this source code:

C++

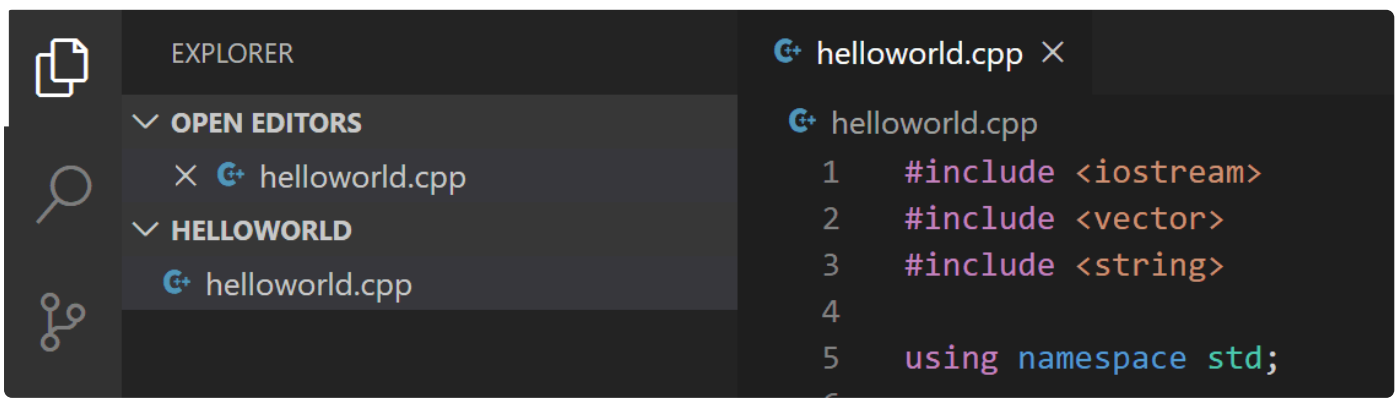
```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

int main()
{
    vector<string> msg {"Hello", "C++", "World", "from", "VS Code", "and the C++ exte
nsion!"};

    for (const string& word : msg)
    {
        cout << word << " ";
    }
    cout << endl;
}
```

Now press `Ctrl+S` to save the file. Notice how the file you just added appears in the **File Explorer** view (`Ctrl+Shift+E`) in the side bar of VS Code:



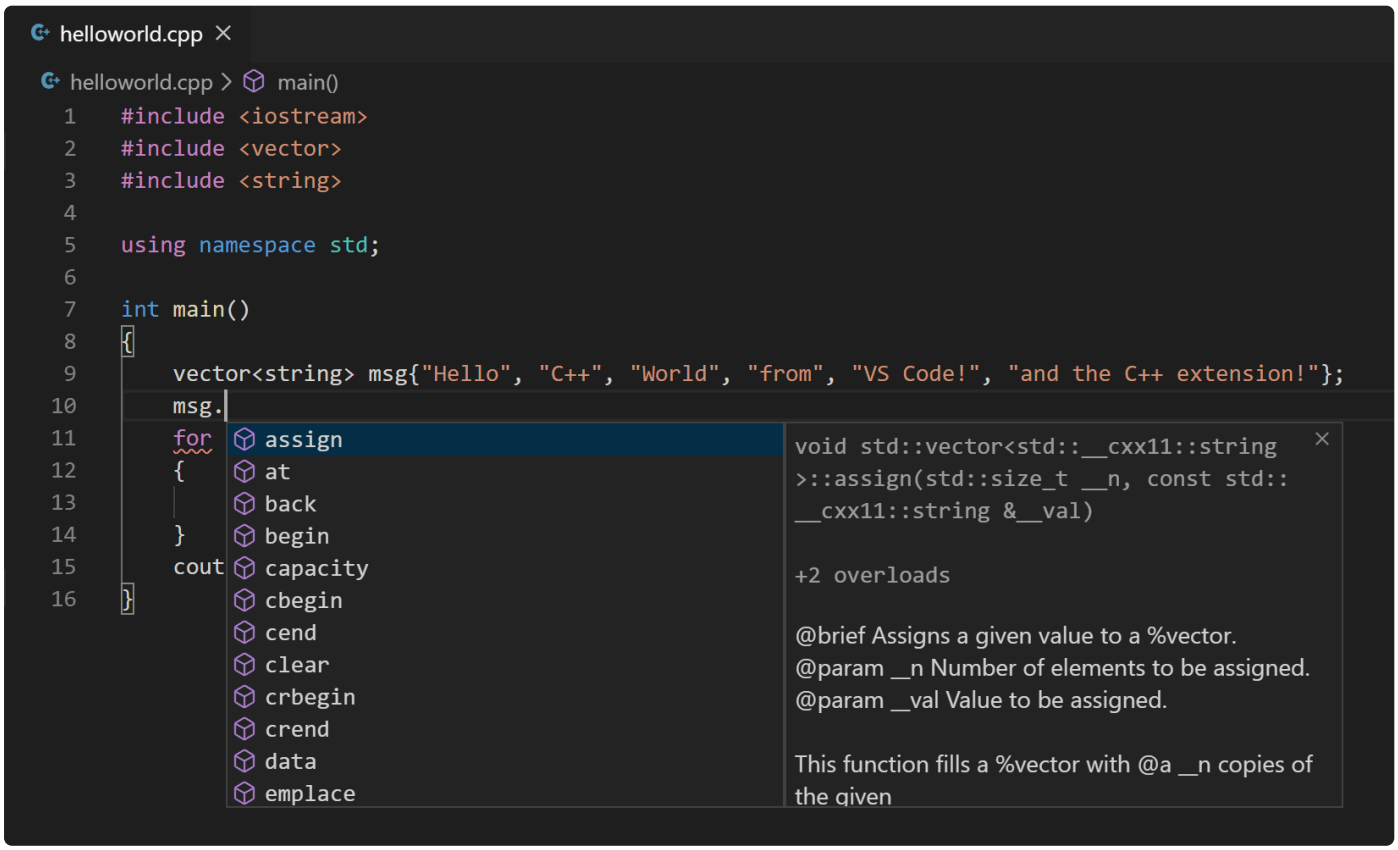
You can also enable Auto Save (/docs/editing/codebasics#_save-auto-save) to automatically save your file changes, by checking **Auto Save** in the main **File** menu.

The Activity Bar on the far left lets you open different views such as **Search**, **Source Control**, and **Run**. You'll look at the **Run** view later in this tutorial. You can find out more about the other views in the VS Code User Interface documentation (/docs/getstarted/userinterface).

> **Note**: When you save or open a C++ file, you may see a notification from the C/C++ extension about the availability of an Insiders version, which lets you test new features and fixes. You can ignore this notification by selecting the `X` (**Clear Notification**).

Explore IntelliSense

In your new `helloworld.cpp` file, hover over `vector` or `string` to see type information. After the declaration of the `msg` variable, start typing `msg.` as you would when calling a member function. You should immediately see a completion list that shows all the member functions, and a window that shows the type information for the `msg` object:



You can press the `Tab` key to insert the selected member; then, when you add the opening parenthesis, you will see information about any arguments that the function requires.

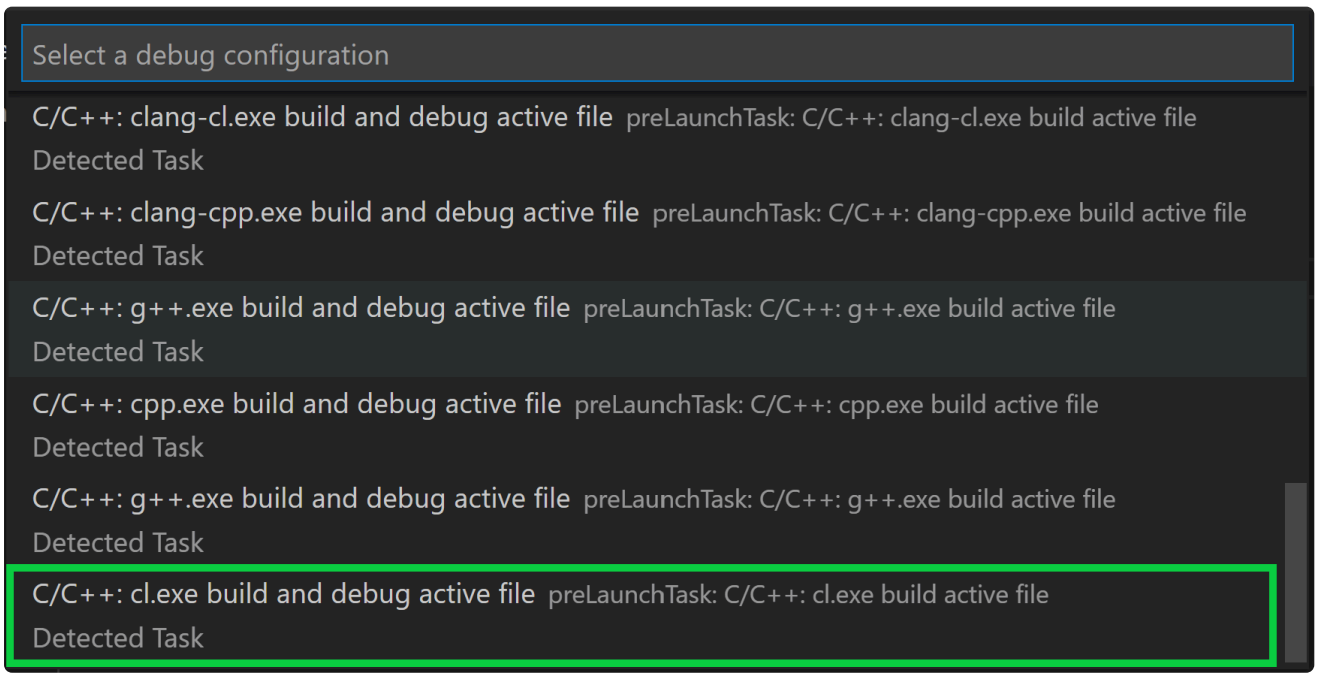
Run helloworld.cpp

Remember, the C++ extension uses the C++ compiler you have installed on your machine to build your program. Make sure you have a C++ compiler installed before attempting to run and debug `helloworld.cpp` in VS Code.

1. Open `helloworld.cpp` so that it is the active file.

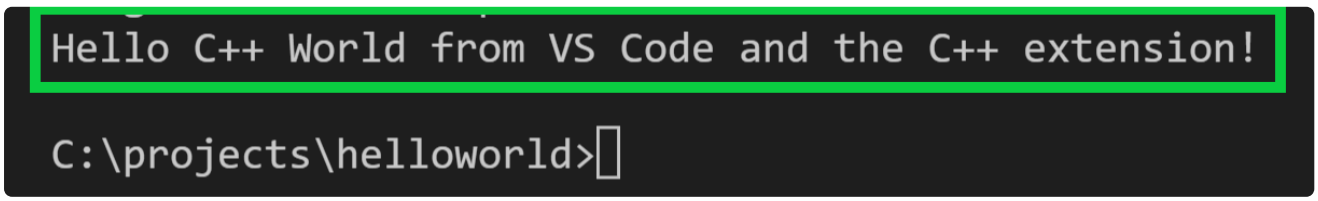
2 Press the play button in the top right corner of the editor.

3 Choose **C/C++: cl.exe build and debug active file** from the list of detected compilers on your system.

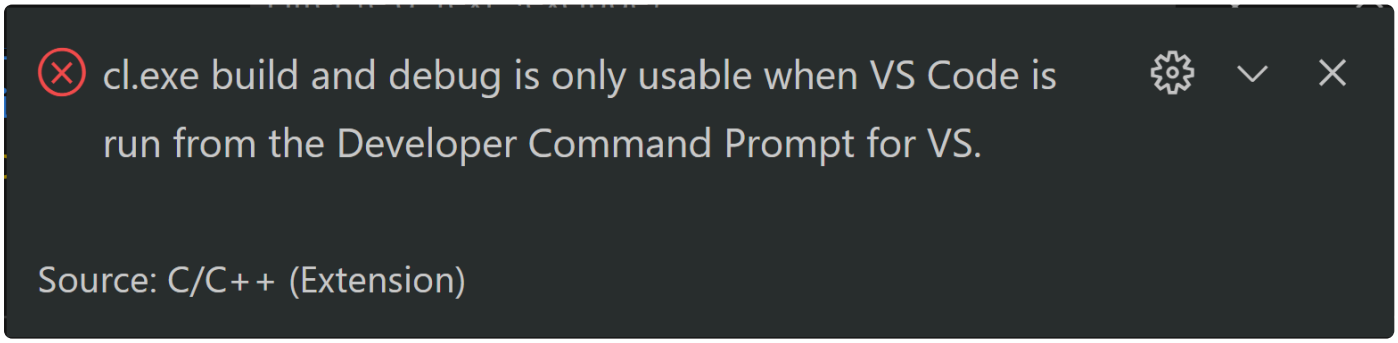


You'll only be asked to choose a compiler the first time you run `helloworld.cpp`. This compiler will be set as the "default" compiler in `tasks.json` file.

1 After the build succeeds, your program's output will appear in the integrated **Terminal**.



If you get an error trying to build and debug with cl.exe, make sure you have started VS Code from the Developer Command Prompt for Visual Studio using the `code .` shortcut.



The first time you run your program, the C++ extension creates `tasks.json`, which you'll find in your project's `.vscode` folder. `tasks.json` stores build configurations.

Your new `tasks.json` file should look similar to the JSON below:

JSON

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "shell",
      "label": "C/C++: cl.exe build active file",
      "command": "cl.exe",
      "args": [
        "/Zi",
        "/EHsc",
        "/Fe:",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "${file}"
      ],
      "problemMatcher": ["$msCompile"],
      "group": {
        "kind": "build",
        "isDefault": true
      },
      "detail": "Task generated by Debugger."
    }
  ]
}
```

> **Note**: You can learn more about `tasks.json` variables in the variables reference (/docs/reference/variables-reference).

The `command` setting specifies the program to run; in this case that is "cl.exe". The `args` array specifies the command-line arguments that will be passed to cl.exe. These arguments must be specified in the order expected by the compiler.

This task tells the C++ compiler to take the active file (`${file}`), compile it, and create an executable file (`/Fe:` switch) in the current directory (`${fileDirname}`) with the same name as the active file but with the `.exe` extension (`${fileBasenameNoExtension}.exe`), resulting in `helloworld.exe` for our example.

The `label` value is what you will see in the tasks list; you can name this whatever you like.

The `detail` value is what you will as the description of the task in the tasks list. It's highly recommended to rename this value to differentiate it from similar tasks.

The `problemMatcher` value selects the output parser to use for finding errors and warnings in the compiler output. For cl.exe, you'll get the best results if you use the `$msCompile` problem matcher.

From now on, the play button will read from `tasks.json` to figure out how to build and run your program. You can define multiple build tasks in `tasks.json` , and whichever task is marked as the default will be used by the play button. In case you need to change the default compiler, you can run **Tasks: Configure default build task**. Alternatively you can modify the `tasks.json` file and remove the default by replacing this segment:

JSON

```
"group": {
    "kind": "build",
    "isDefault": true
},
```

with this:

JSON

```
"group": "build",
```

Modifying tasks.json

You can modify your `tasks.json` to build multiple C++ files by using an argument like `"${workspaceFolder}/*.cpp"` instead of `"${file}"` .This will build all `.cpp` files in your current folder. You can also modify the output filename by replacing `"${fileDirname}\\${fileBasenameNoExtension}.exe"` with a hard-coded filename (for example `"${workspaceFolder}\\myProgram.exe"`).

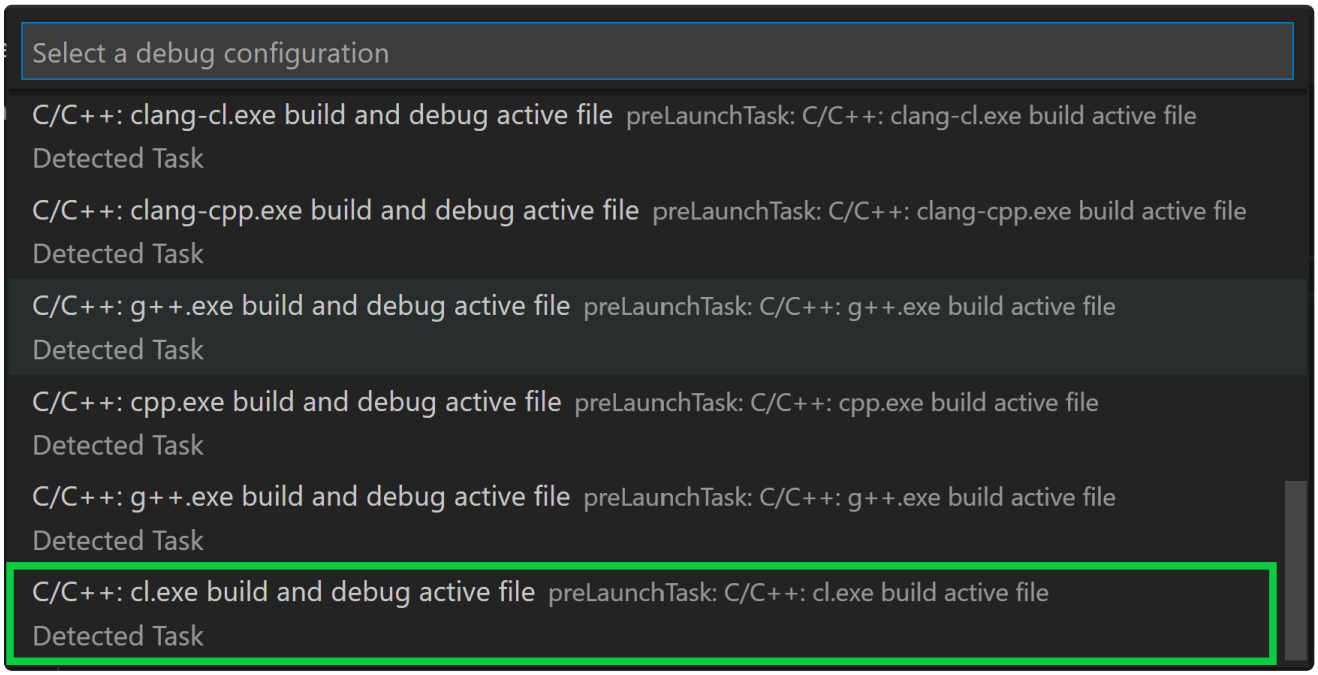
Debug helloworld.cpp

To debug your code,

1. Go back to `helloworld.cpp` so that it is the active file.
2. Set a breakpoint by clicking on the editor margin or using F9 on the current line.

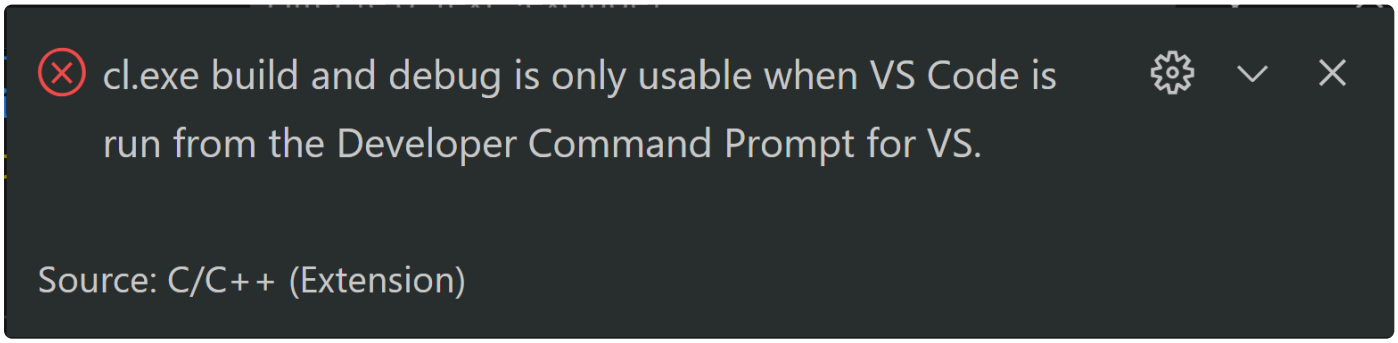
3 From the drop-down next to the play button, select **Debug C/C++ File**.

4 Choose **C/C++: cl.exe build and debug active file** from the list of detected compilers on your system (you'll only be asked to choose a compiler the first time you run or debug `helloworld.cpp`).



The play button has two modes: **Run C/C++ File** and **Debug C/C++ File**. It will default to the last-used mode. If you see the debug icon in the play button, you can select the play button to debug, instead of selecting the drop-down menu item.

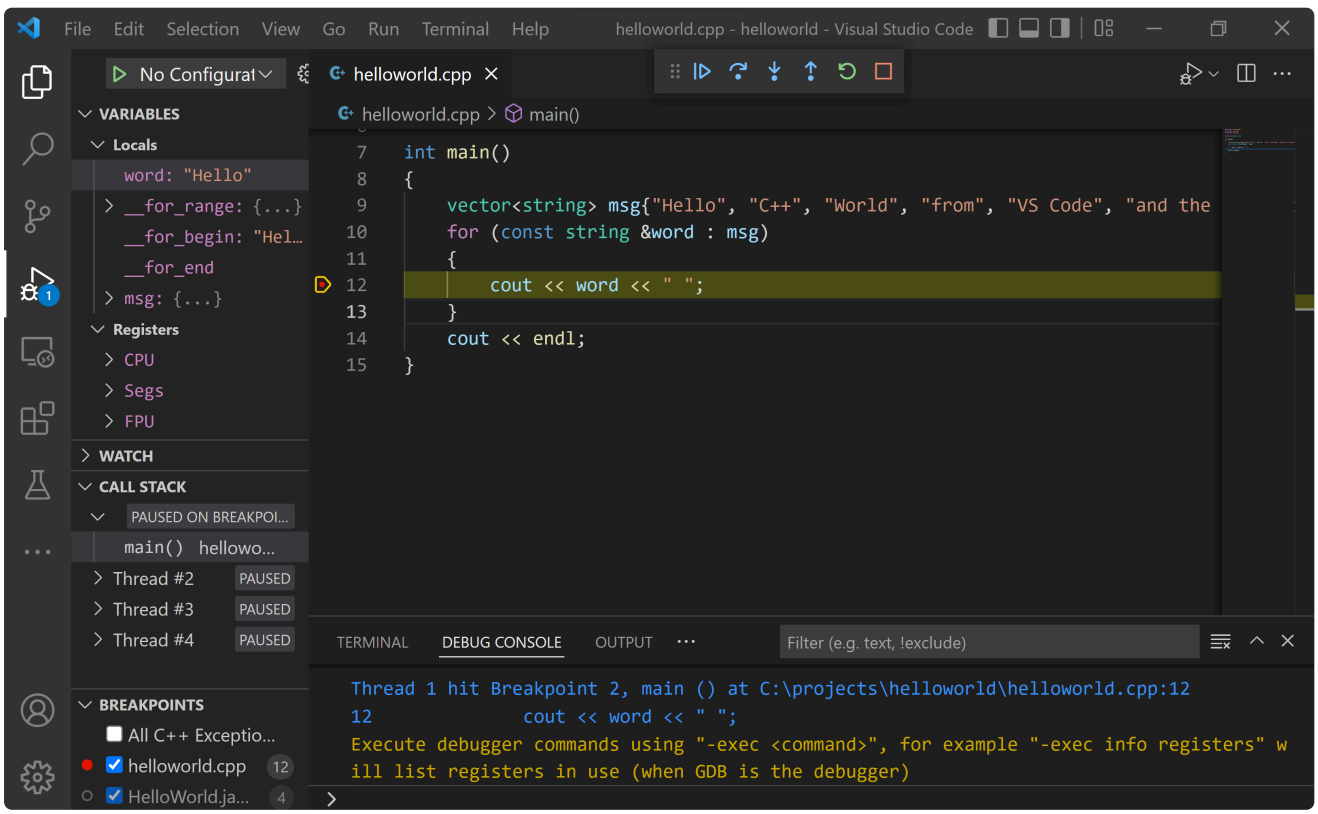
If you get an error trying to build and debug with cl.exe, make sure you have started VS Code from the Developer Command Prompt for Visual Studio using the `code .` shortcut.



Explore the debugger

Before you start stepping through the code, let's take a moment to notice several changes in the user interface:

- The Integrated Terminal appears at the bottom of the source code editor. In the **Debug Output** tab, you see output that indicates the debugger is up and running.
- The editor highlights the line where you set a breakpoint before starting the debugger:



- The **Run and Debug** view on the left shows debugging information. You'll see an example later in the tutorial.
- At the top of the code editor, a debugging control panel appears. You can move this around the screen by grabbing the dots on the left side.

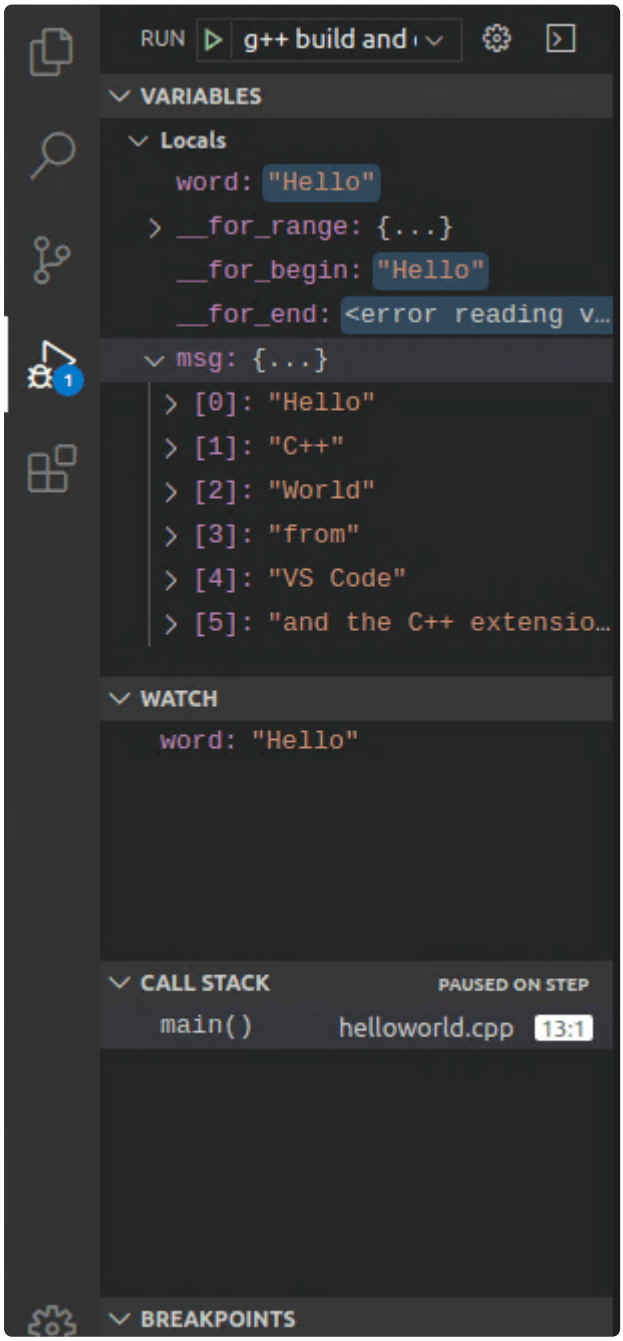
Step through the code

Now you're ready to start stepping through the code.

1. Click or press the **Step over** icon in the debugging control panel.

   This will advance program execution to the first line of the for loop, and skip over all the internal function calls within the `vector` and `string` classes that are invoked when the `msg` variable is created and initialized. Notice the change in the **Variables** window on the left.

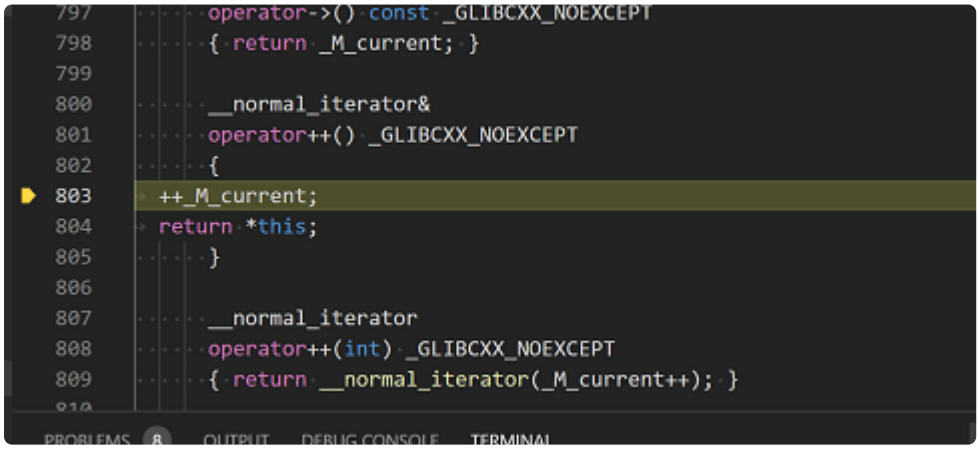


In this case, the errors are expected because, although the variable names for the loop are now visible to the debugger, the statement has not executed yet, so there is nothing to read at this point. The contents of `msg` are visible, however, because that statement has completed.

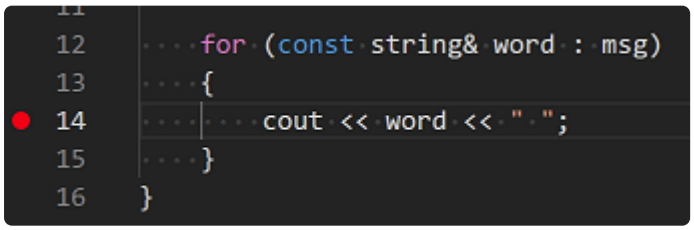
2 Press **Step over** again to advance to the next statement in this program (skipping over all the internal code that is executed to initialize the loop). Now, the **Variables** window shows information about the loop variables.

3 Press **Step over** again to execute the `cout` statement. (Note that the C++ extension does not print any output to the **Debug Console** until the loop exits.)

4 If you like, you can keep pressing **Step over** until all the words in the vector have been printed to the console. But if you are curious, try pressing the **Step Into** button to step through source code in the C++ standard library!



To return to your own code, one way is to keep pressing **Step over**. Another way is to set a breakpoint in your code by switching to the `helloworld.cpp` tab in the code editor, putting the insertion point somewhere on the `cout` statement inside the loop, and pressing `F9`. A red dot appears in the gutter on the left to indicate that a breakpoint has been set on this line.

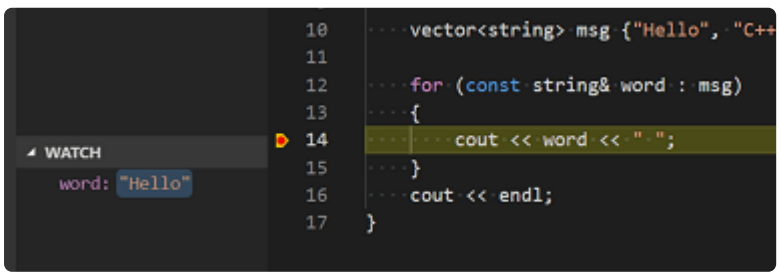


Then press `F5` to start execution from the current line in the standard library header. Execution will break on `cout`. If you like, you can press `F9` again to toggle off the breakpoint.

Set a watch

Sometimes you might want to keep track of the value of a variable as your program executes. You can do this by setting a **watch** on the variable.

1. Place the insertion point inside the loop. In the **Watch** window, select the plus sign and in the text box, type `word`, which is the name of the loop variable. Now view the Watch window as you step through the loop.



2. Add another watch by adding this statement before the loop: `int i = 0;`. Then, inside the loop, add this statement: `++i;`. Now add a watch for `i` as you did in the previous step.

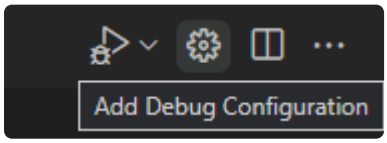
3. To quickly view the value of any variable while execution is paused on a breakpoint, you can hover over it with the mouse pointer.

Customize debugging with launch.json

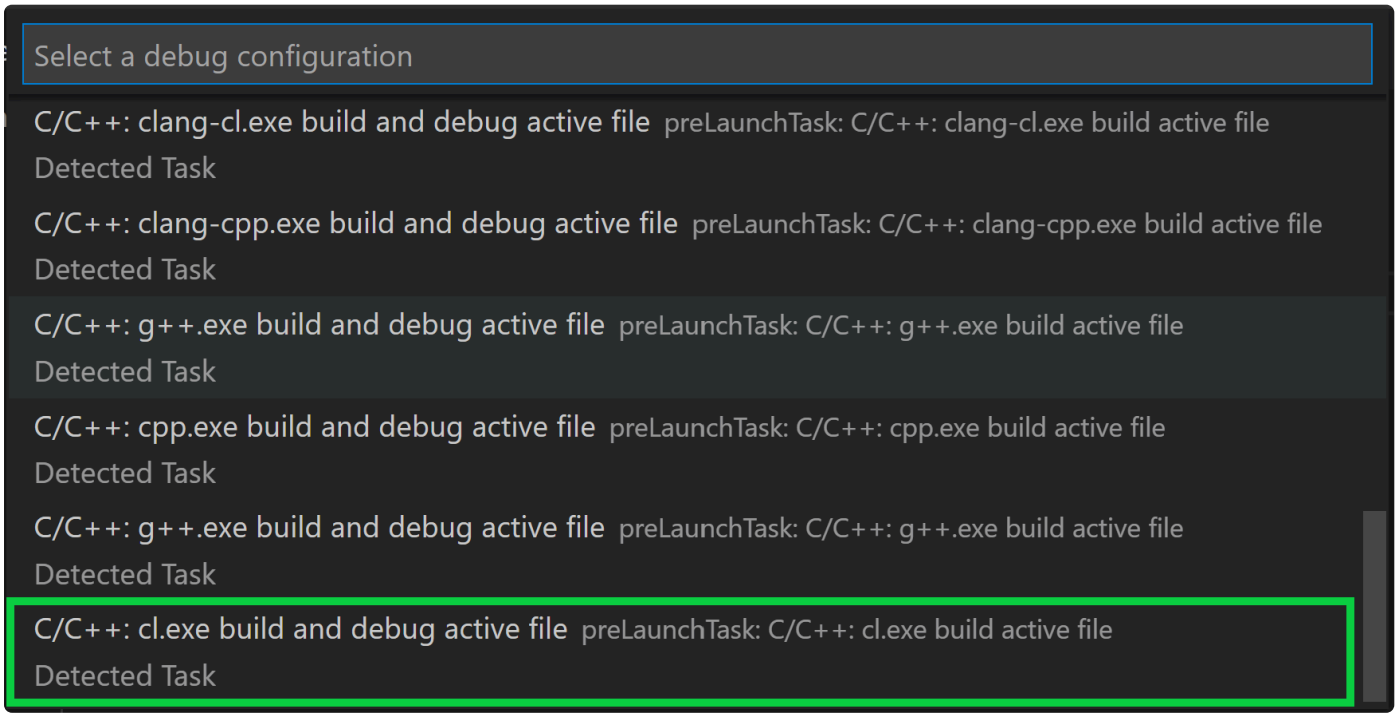
When you debug with the play button or `F5`, the C++ extension creates a dynamic debug configuration on the fly.

There are cases where you'd want to customize your debug configuration, such as specifying arguments to pass to the program at runtime. You can define custom debug configurations in a `launch.json` file.

To create `launch.json`, choose **Add Debug Configuration** from the play button drop-down menu.



You'll then see a dropdown for various predefined debugging configurations. Choose **C/C++: cl.exe build and debug active file**.



VS Code creates a `launch.json` file, which looks something like this:

JSON

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "C/C++: cl.exe build and debug active file",
      "type": "cppvsdbg",
      "request": "launch",
      "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false,
      "preLaunchTask": "C/C++: cl.exe build active file"
    }
  ]
}
```

In the JSON above, `program` specifies the program you want to debug. Here it is set to the active file folder (`${fileDirname}`) and active filename with the `.exe` extension (`${fileBasenameNoExtension}.exe`), which if `helloworld.cpp` is the active file will be `helloworld.exe` . The `args` property is an array of arguments to pass to the program at runtime.

By default, the C++ extension won't add any breakpoints to your source code and the `stopAtEntry` value is set to `false` .

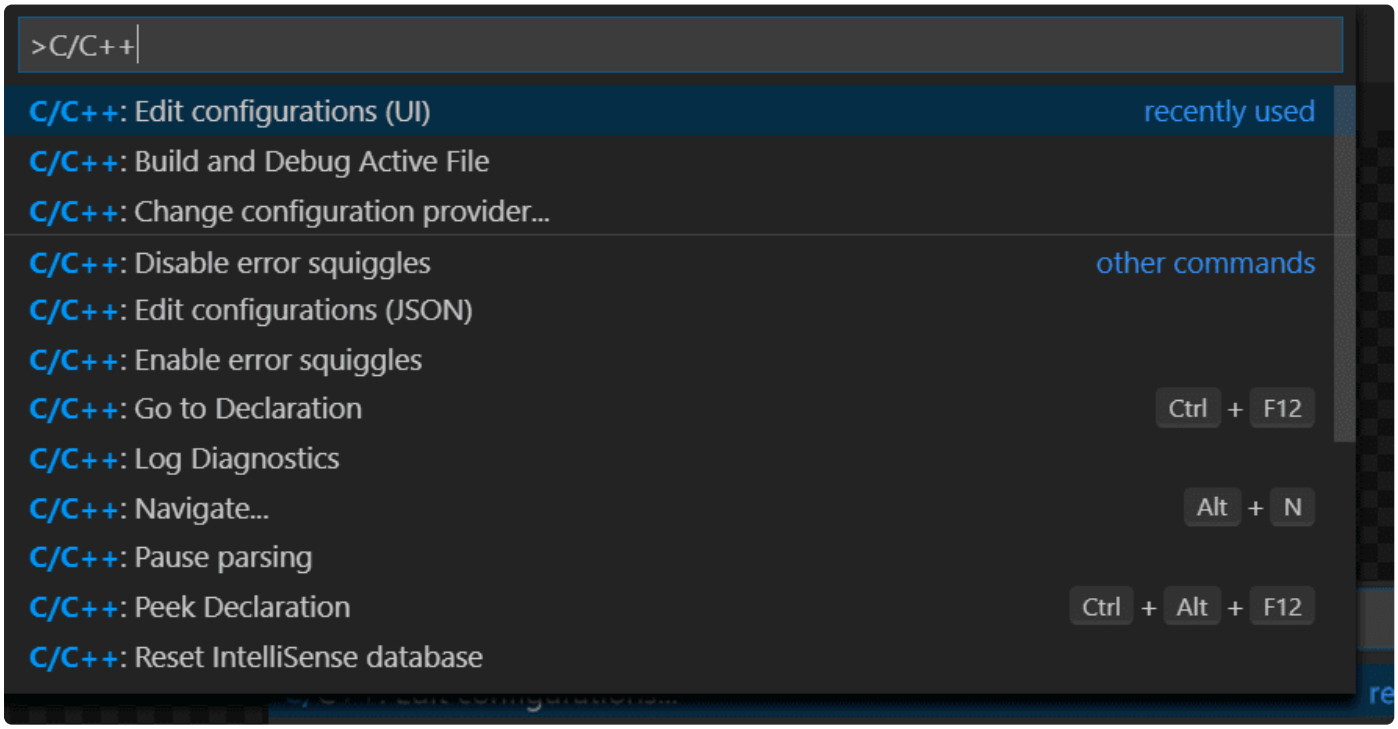
Change the `stopAtEntry` value to `true` to cause the debugger to stop on the `main` method when you start debugging.

> From now on, the play button and `F5` will read from your `launch.json` file when launching your program for debugging.

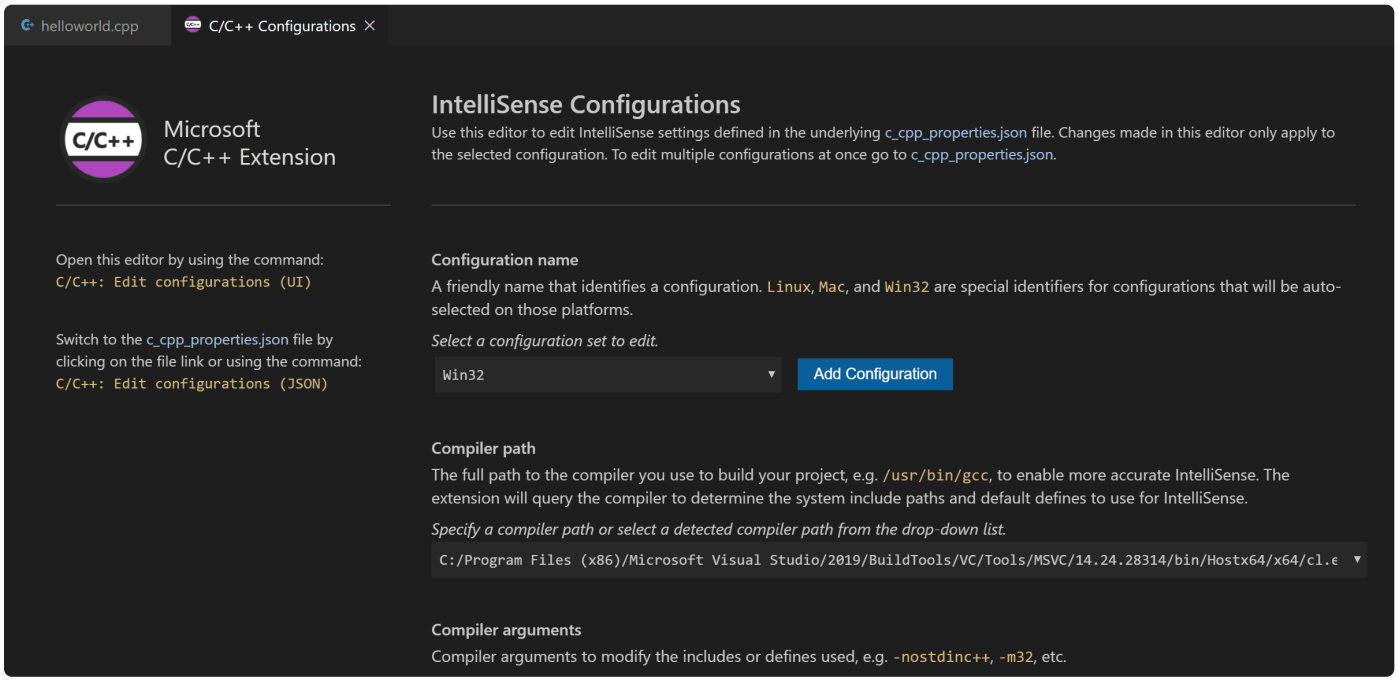
C/C++ configurations

If you want more control over the C/C++ extension, you can create a `c_cpp_properties.json` file, which will allow you to change settings such as the path to the compiler, include paths, C++ standard (default is C++17), and more.

You can view the C/C++ configuration UI by running the command **C/C++: Edit Configurations (UI)** from the Command Palette (`Ctrl+Shift+P`).



This opens the **C/C++ Configurations** page. When you make changes here, VS Code writes them to a file called `c_cpp_properties.json` in the `.vscode` folder.



Visual Studio Code places these settings in `.vscode\c_cpp_properties.json`. If you open that file directly, it should look something like this:

JSON

```
{
  "configurations": [
    {
      "name": "Win32",
      "includePath": ["${workspaceFolder}/**"],
      "defines": ["_DEBUG", "UNICODE", "_UNICODE"],
      "windowsSdkVersion": "10.0.18362.0",
      "compilerPath": "C:/Program Files (x86)/Microsoft Visual Studio/2019/BuildTool
s/VC/Tools/MSVC/14.24.28314/bin/Hostx64/x64/cl.exe",
      "cStandard": "c11",
      "cppStandard": "c++17",
      "intelliSenseMode": "msvc-x64"
    }
  ],
  "version": 4
}
```

You only need to add to the **Include path** array setting if your program includes header files that are not in your workspace or in the standard library path.

Compiler path

The `compilerPath` setting is an important setting in your configuration. The extension uses it to infer the path to the C++ standard library header files. When the extension knows where to find those files, it can provide useful features like smart completions and **Go to Definition** navigation.

The C/C++ extension attempts to populate `compilerPath` with the default compiler location based on what it finds on your system. The extension looks in several common compiler locations.

The `compilerPath` search order is:

- First check for the Microsoft Visual C++ compilerOpe
- Then look for g++ on Windows Subsystem for Linux (WSL)
- Then g++ for Mingw-w64.

If you have g++ or WSL installed, you might need to change `compilerPath` to match the preferred compiler for your project. For Microsoft C++, the path should look something like this, depending on which specific version you have installed: "C:/Program Files (x86)/Microsoft Visual Studio/2017/BuildTools/VC/Tools/MSVC/14.16.27023/bin/Hostx64/x64/cl.exe".

Reusing your C++ configuration

VS Code is now configured to use the Microsoft C++ compiler. The configuration applies to the current workspace. To reuse the configuration, just copy the JSON files to a `.vscode` folder in a new project folder (workspace) and change the names of the source file(s) and executable as needed.

Run VS Code outside the Developer Command Prompt

In certain circumstances, it isn't possible to run VS Code from **Developer Command Prompt for Visual Studio** (for example, in Remote Development through SSH scenarios). In that case, you can automate initialization of **Developer Command Prompt for Visual Studio** during the build using the following `tasks.json` configuration:

JSON

```
{
  "version": "2.0.0",
  "windows": {
    "options": {
      "shell": {
        "executable": "cmd.exe",
        "args": [
          "/C",
          // The path to VsDevCmd.bat depends on the version of Visual Studio you hav
e installed.
          "\"C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/Common7/To
ols/VsDevCmd.bat\"",
          "&&"
        ]
      }
    }
  },
  "tasks": [
    {
      "type": "shell",
      "label": "cl.exe build active file",
      "command": "cl.exe",
      "args": [
        "/Zi",
        "/EHsc",
        "/Fe:",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "${file}"
      ],
      "problemMatcher": ["$msCompile"],
      "group": {
        "kind": "build",
        "isDefault": true
      }
    }
  ]
}
```

> **Note**: The path to `VsDevCmd.bat` might be different depending on the Visual Studio version or installation path. You can find the path to `VsDevCmd.bat` by opening a Command Prompt and running `dir "\VsDevCmd*" /s`.

Troubleshooting

The term 'cl.exe' is not recognized

If you see the error "The term 'cl.exe' is not recognized as the name of a cmdlet, function, script file, or operable program.", this usually means you are running VS Code outside of a **Developer Command Prompt for Visual Studio** and VS Code doesn't know the path to the `cl.exe` compiler.

VS Code must either be started from the Developer Command Prompt for Visual Studio, or the task must be configured to run outside a Developer Command Prompt.

You can always check that you are running VS Code in the context of the Developer Command Prompt by opening a new Terminal (`Ctrl+Shift+``) and typing 'cl' to verify `cl.exe` is available to VS Code.

fatal error C1034: assert.h: no include path set

In this case, `cl.exe` is available to VS Code through the `PATH` environment variable, but VS Code still needs to either be started from the **Developer Command Prompt for Visual Studio**, or be configured to run outside the Developer Command Prompt. Otherwise, `cl.exe` does not have access to important environment variables such as `INCLUDE`.

Next steps

- Explore the VS Code User Guide (/docs/editing/codebasics).
- Review the Overview of the C++ extension (/docs/languages/cpp).
- Create a new workspace, copy your `.vscode` JSON files to it, adjust the necessary settings for the new workspace path, program name, and so on, and start coding!

3/7/2023

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://github.com/microsoft/vscode)
(https://go.microsoft.com/fwlink/?LinkID=533687)
(https://www.linkedin.com/showcase/vs-code)
(https://bsky.app/profile/vscode.dev)
(https://www.reddit.com/r/vscode/)
(https://www.vscodepodcast.com)
(https://www.tiktok.com/@vscode)
(https://www.youtube.com/@code)

(https://www.microsoft.com)

Support (https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d)
Privacy (https://go.microsoft.com/fwlink/?LinkId=521839)
Terms of Use (https://www.microsoft.com/legal/terms-of-use) License (/License)