

CLAUDE CODE

Claude Code overview

Learn about Claude Code, an agentic coding tool made by Anthropic. Currently in beta as a research preview.

Install **NodeJS 18+**, then run:

```
npm install -g @anthropic-ai/claude-code
```

⚠ **Do NOT use** `sudo npm install -g` as this can lead to permission issues and security risks. If you encounter permission errors, see **configure Claude Code** for recommended solutions.

Claude Code is an agentic coding tool that lives in your terminal, understands your codebase, and helps you code faster through natural language commands. By integrating directly with your development environment, Claude Code streamlines your workflow without requiring additional servers or complex setup.

Claude Code's key capabilities include:

Editing files and fixing bugs across your codebase

Answering questions about your code's architecture and logic

Executing and fixing tests, linting, and other commands

Searching through git history, resolving merge conflicts, and creating commits and PRs

ⓘ **Research preview**

Code is in beta as a research preview. We're gathering developer feedback on AI collaboration preferences, which workflows benefit most from AI assistance, and how to improve the agent experience.

This early version will evolve based on user feedback. We plan to enhance tool execution reliability, support for long-running commands, terminal rendering, and Claude's self-knowledge of its capabilities in the coming weeks.

Report bugs directly with the `/bug` command or through our **GitHub repository**.

Before you begin

Check system requirements

Operating Systems: macOS 10.15+, Ubuntu 20.04+/Debian 10+, or Windows via WSL

Hardware: 4GB RAM minimum

Software:
Claude Code overview

Node.js 18+

git 2.23+ (optional)

GitHub or **GitLab** CLI for PR workflows (optional)

ripgrep (rg) for enhanced file search (optional)

Network: Internet connection required for authentication and AI processing

Location: Available only in **supported countries**

❗ **Troubleshooting WSL installation**

Currently, Claude Code does not run directly in Windows, and instead requires WSL. If you encounter issues in WSL:

1. **OS/platform detection issues:** If you receive an error during installation, WSL may be using Windows `npm`. Try:

Run `npm config set os linux` before installation

Install with `npm install -g @anthropic-ai/claude-code --force --no-os-check` (Do NOT use `sudo`)

2. **Node not found errors:** If you see `exec: node: not found` when running `claude`, your WSL environment may be using a Windows installation of Node.js. You can confirm this with `which npm` and `which node`, which should point to Linux paths starting with `/usr/` rather than `/mnt/c/`. To fix this, try installing Node via your Linux distribution's package manager or via [nvm](#).

Install and authenticate

1 Install Claude Code

Run in your terminal: `npm install -g @anthropic-ai/claude-code`

⚠ **Do NOT use `sudo npm install -g`** as this can lead to permission issues and security risks. If you encounter permission errors, see **configure Claude Code** for recommended solutions.

2 Navigate to your project

`cd your-project-directory`

3 Start Claude Code

Run `claude` to launch

4 Complete authentication

Follow the one-time OAuth process with your Console account. You'll need active billing at **console.anthropic.com**.

Core features and workflows

Claude Code operates directly in your terminal, understanding your project context and taking real actions. No need to manually add files to context

Claude Code overview

- Claude will explore your codebase as needed. Claude Code uses `claude-3-7-sonnet-20250219` by default.

Security and privacy by design

Your code’s security is paramount. Claude Code’s architecture ensures:

- Direct API connection:** Your queries go straight to Anthropic’s API without intermediate servers
- Works where you work:** Operates directly in your terminal
- Understands context:** Maintains awareness of your entire project structure
- Takes action:** Performs real operations like editing files and creating commits

From questions to solutions in seconds

```
# Ask questions about your codebase
claude
> how does our authentication system work?

# Create a commit with one command
claude commit

# Fix issues across multiple files
claude "fix the type errors in the auth module"
```

Initialize your project

For first-time users, we recommend:

1. Start Claude Code with `claude`
2. Try a simple command like `summarize this project`
3. Generate a CLAUDE.md project guide with `/init`
4. Ask Claude to commit the generated CLAUDE.md file to your repository

Use Claude Code for common tasks

Claude Code operates directly in your terminal, understanding your project context and taking real actions. No need to manually add files to context

- Claude will explore your codebase as needed.

Understand unfamiliar code

```
> what does the payment processing system do?
> find where user permissions are checked
> explain how the caching layer works
```

Automate Git operations

Claude Code overview

- > commit my changes
- > create a pr
- > which commit added tests for markdown back in December?
- > rebase on main and resolve any merge conflicts

Edit code intelligently

- > add input validation to the signup form
- > refactor the logger to use the new API
- > fix the race condition in the worker queue

Test and debug your code

- > run tests for the auth module and fix failures
- > find and fix security vulnerabilities
- > explain why this test is failing

Encourage deeper thinking

For complex problems, explicitly ask Claude to think more deeply:

- > think about how we should architect the new payment service
- > think hard about the edge cases in our authentication flow

Claude Code will show when Claude (3.7 Sonnet) is using extended thinking. You can proactively prompt Claude to “think” or “think deeply” for more planning-intensive tasks. We suggest that you first tell Claude about your task and let it gather context from your project. Then, ask it to “think” to create a plan.

💡 Claude will think more based on the words you use. For example, “think hard” will trigger more extended thinking than saying “think” alone.

For more tips, see **Extended thinking tips**.

Automate CI and infra workflows

Claude Code comes with a non-interactive mode for headless execution. This is especially useful for running Claude Code in non-interactive contexts like scripts, pipelines, and Github Actions.

Use `--print (-p)` to run Claude in non-interactive mode. In this mode, you can set the `ANTHROPIC_API_KEY` environment variable to provide a custom API key.

Non-interactive mode is especially useful when you pre-configure the set of commands Claude is allowed to use:

Claude Code overview

export ANTHROPIC_API_KEY=sk_...

claude -p "update the README with the latest changes" --allowedTools "Bash(git diff:*)" "Bash(git log:*)" Edit

Control Claude Code with commands

CLI commands

Command	Description	Example
claude	Start interactive REPL	claude
claude "query"	Start REPL with initial prompt	claude "explain this project"
claude -p "query"	Run one-off query, then exit	claude -p "explain this function"
cat file claude -p "query"	Process piped content	cat logs.txt claude -p "explain"
claude config	Configure settings	claude config set --global theme dark
claude update	Update to latest version	claude update
claude mcp	Configure Model Context Protocol servers	See MCP section in tutorials

CLI flags:

- print : Print response without interactive mode
- verbose : Enable verbose logging
- dangerously-skip-permissions : Skip permission prompts (only in Docker containers without internet)

Slash commands

Control Claude’s behavior within a session:

Command	Purpose
/bug	Report bugs (sends conversation to Anthropic)
/clear	Clear conversation history
/compact [instructions]	Compact conversation with optional focus instructions
/config	View/modify configuration
/cost	Show token usage statistics
/doctor	Checks the health of your Claude Code installation
/help	Get usage help
/init	Initialize project with CLAUDE.md guide
/login	Switch Anthropic accounts
/logout	Sign out from your Anthropic account
/memory	Edit CLAUDE.md memory files
/pr_comments	View pull request comments
/review	Request code review

Command	Purpose
Claude Code overview	
<code>/terminal-setup</code>	Install Shift+Enter key binding for newlines (iTerm2 and VSCode only)
<code>/vim</code>	Enter vim mode for alternating insert and command modes

Manage Claude’s memory

Claude Code can remember your preferences across sessions, like style guidelines and common commands in your workflow.

Determine memory type

Claude Code offers three memory locations, each serving a different purpose:

Memory Type	Location	Purpose	Use Case Examples
Project memory	<code>./CLAUDE.md</code>	Team-shared conventions and knowledge	Project architecture, coding standards, common workflows
Project memory (local)	<code>./CLAUDE.local.md</code>	Personal project-specific preferences	Your sandbox URLs, preferred test data
User memory	<code>~/.claude/CLAUDE.md</code>	Global personal preferences	Code styling preferences, personal tooling shortcuts

All memory files are automatically loaded into Claude Code’s context when launched.

Quickly add memories with the `#` shortcut

The fastest way to add a memory is to start your input with the `#` character:

```
# Always use descriptive variable names
```

You’ll be prompted to select which memory file to store this in.

Directly edit memories with `/memory`

Use the `/memory` slash command during a session to open any memory file in your system editor for more extensive additions or organization.

Memory best practices

- Be specific:** “Use 2-space indentation” is better than “Format code properly”.
- Use structure to organize:** Format each individual memory as a bullet point and group related memories under descriptive markdown headings.
- Review periodically:** Update memories as your project evolves to ensure Claude is always using the most up to date information and context.

Manage permissions and security

Claude Code uses a tiered permission system to balance power and safety:

Tool Type	Example	Approval Required	"Yes, don't ask again" Behavior
Claude Code overview			
Read-only	File reads, LS, Grep	No	N/A
Bash Commands	Shell execution	Yes	Permanently per project directory and command
File Modification	Edit/write files	Yes	Until session end

Tools available to Claude

Claude Code has access to a set of powerful tools that help it understand and modify your codebase:

Tool	Description	Permission Required
AgentTool	Runs a sub-agent to handle complex, multi-step tasks	No
BashTool	Executes shell commands in your environment	Yes
GlobTool	Finds files based on pattern matching	No
GrepTool	Searches for patterns in file contents	No
LSTool	Lists files and directories	No
FileReadTool	Reads the contents of files	No
FileEditTool	Makes targeted edits to specific files	Yes
FileWriteTool	Creates or overwrites files	Yes
NotebookReadTool	Reads and displays Jupyter notebook contents	No
NotebookEditTool	Modifies Jupyter notebook cells	Yes

Protect against prompt injection

Prompt injection is a technique where an attacker attempts to override or manipulate an AI assistant’s instructions by inserting malicious text. Claude Code includes several safeguards against these attacks:

- Permission system:** Sensitive operations require explicit approval
- Context-aware analysis:** Detects potentially harmful instructions by analyzing the full request
- Input sanitization:** Prevents command injection by processing user inputs
- Command blacklist:** Blocks risky commands that fetch arbitrary content from the web like `curl` and `wget`

Best practices for working with untrusted content:

- Review suggested commands before approval
- Avoid piping untrusted content directly to Claude
- Verify proposed changes to critical files
- Report suspicious behavior with `/bug`

⚠ While these protections significantly reduce risk, no system is completely immune to all attacks. Always maintain good security practices when working with any AI tool.

Configure network access

Claude Code requires access to:
Claude Code overview

api.anthropic.com

statsig.anthropic.com

sentry.io

Allowlist these URLs when using Claude Code in containerized environments.

Configure Claude Code

Configure Claude Code by running `claude config` in your terminal, or the `/config` command when using the interactive REPL.

Configuration options

Claude Code supports global and project-level configuration.

To manage your configurations, use the following commands:

List settings: `claude config list`

See a setting: `claude config get <key>`

Change a setting: `claude config set <key> <value>`

Push to a setting (for lists): `claude config add <key> <value>`

Remove from a setting (for lists): `claude config remove <key> <value>`

By default `config` changes your project configuration. To manage your global configuration, use the `--global` (or `-g`) flag.

Global configuration

To set a global configuration, use `claude config set -g <key> <value>` :

Key	Value	Description
autoUpdaterStatus	disabled or enabled	Enable or disable the auto-updater (default: enabled)
preferredNotifChannel	item2 , item2_with_bell , terminal_bell , or notifications_disabled	Where you want to receive notifications (default: item2)
theme	dark , light , light-daltonized , or dark-daltonized	Color theme
verbose	true or false	Whether to show full bash and command outputs (default: false)

Auto-updater permission options

When Claude Code detects that it doesn’t have sufficient permissions to write to your global npm prefix directory (required for automatic updates), you’ll see a warning that points to this documentation page. For detailed solutions to auto-updater issues, see the **troubleshooting guide**.

Recommended: Create a new user-writable npm prefix

```
# First, save a list of your existing global packages for later migration
npm list -g --depth=0 > ~/npm-global-packages.txt
```


Create a directory for your global packages
Claude Code overview
mkdir -p ~/.npm-global

Configure npm to use the new directory path
npm config set prefix ~/.npm-global

Note: Replace ~/.bashrc with ~/.zshrc, ~/.profile, or other appropriate file for your shell
echo 'export PATH=~/.npm-global/bin:\$PATH' >> ~/.bashrc

Apply the new PATH setting
source ~/.bashrc

Now reinstall Claude Code in the new location
npm install -g @anthropic-ai/claude-code

Optional: Reinstall your previous global packages in the new location
Look at ~/.npm-global-packages.txt and install packages you want to keep
npm install -g package1 package2 package3...

Why we recommend this option:

- Avoids modifying system directory permissions
- Creates a clean, dedicated location for your global npm packages
- Follows security best practices

Since Claude Code is actively developing, we recommend setting up auto-updates using the recommended option above.

Disabling the auto-updater

If you prefer to disable the auto-updater instead of fixing permissions, you can use:

claude config set -g autoUpdaterStatus disabled

Project configuration

Manage project configuration with `claude config set <key> <value>` (without the `-g` flag):

Key	Value	Description
allowedTools	array of tools	Which tools can run without manual approval
ignorePatterns	array of glob strings	Which files/directories are ignored when using tools

For example:

Let npm test to run without approval
claude config add allowedTools "Bash(npm test)"

Let npm test and any of its sub-commands to run without approval
claude config add allowedTools "Bash(npm test:*)"

Instruct Claude to ignore node_modules

Optimize your terminal setup

Claude Code works best when your terminal is properly configured. Follow these guidelines to optimize your experience.

Supported shells:

Bash

Zsh

Fish

Themes and appearance

Claude cannot control the theme of your terminal. That's handled by your terminal application. You can match Claude Code's theme to your terminal during onboarding or any time via the `/config` command

Line breaks

You have several options for entering linebreaks into Claude Code:

Quick escape: Type `\` followed by Enter to create a newline

Keyboard shortcut: Press Option+Enter (Meta+Enter) with proper configuration

To set up Option+Enter in your terminal:

For Mac Terminal.app:

1. Open Settings → Profiles → Keyboard
2. Check “Use Option as Meta Key”

For iTerm2 and VSCode terminal:

1. Open Settings → Profiles → Keys
2. Under General, set Left/Right Option key to “Esc+”

Tip for iTerm2 and VSCode users: Run `/terminal-setup` within Claude Code to automatically configure Shift+Enter as a more intuitive alternative.

Notification setup

Never miss when Claude completes a task with proper notification configuration:

Terminal bell notifications

Enable sound alerts when tasks complete:

```
claude config set --global preferredNotifChannel terminal_bell
```

For macOS users: Don't forget to enable notification permissions in System Settings → Notifications → [Your Terminal App].

iTerm 2 system notifications

For iTerm 2 alerts when tasks complete:

Claude Code overview

1. Open iTerm 2 Preferences
2. Navigate to Profiles → Terminal
3. Enable “Silence bell” and “Send notification when idle”
4. Set your preferred notification delay

Note that these notifications are specific to iTerm 2 and not available in the default macOS Terminal.

Handling large inputs

When working with extensive code or long instructions:

Avoid direct pasting: Claude Code may struggle with very long pasted content

Use file-based workflows: Write content to a file and ask Claude to read it

Be aware of VS Code limitations: The VS Code terminal is particularly prone to truncating long pastes

Vim Mode

Claude Code supports a subset of Vim keybindings that can be enabled with `/vim` or configured via `/config`.

The supported subset includes:

Mode switching: `Esc` (to NORMAL), `i` / `I` , `a` / `A` , `o` / `O` (to INSERT)

Navigation: `h` / `j` / `k` / `l` , `w` / `e` / `b` , `0` / `$` / `^` , `gg` / `G`

Editing: `x` , `dw` / `de` / `db` / `dd` / `D` , `cw` / `ce` / `cb` / `cc` / `C` , `.` (repeat)

Manage costs effectively

Claude Code consumes tokens for each interaction. The average cost is \$6 per developer per day, with daily costs remaining below \$12 for 90% of users.

Track your costs

Use `/cost` to see current session usage

Check **historical usage** in the Anthropic Console. Note: Users need Admin, Owner, or Billing roles to view Cost tab

Set **workspace spend limits** for the Claude Code workspace. Note: Users need Admin or Owner role to set spend limits.

Reduce token usage

Compact conversations:

Claude uses auto-compact by default when context exceeds 95% capacity

Toggle auto-compact: Run `/config` and navigate to “Auto-compact enabled”

Use `/compact` manually when context gets large

Add custom instructions: `/compact` Focus on code samples and API usage

```
# Summary instructions
```

```
When you are using compact, please focus on test output and code changes
```

Write specific queries: Avoid vague requests that trigger unnecessary scanning

Break down complex tasks: Split large tasks into focused interactions

Clear history between tasks: Use `/clear` to reset context

Costs can vary significantly based on:

- Size of codebase being analyzed

- Complexity of queries

- Number of files being searched or modified

- Length of conversation history

- Frequency of compacting conversations

❗ For team deployments, we recommend starting with a small pilot group to establish usage patterns before wider rollout.

Use with third-party APIs

❗ Claude Code requires access to both Claude 3.7 Sonnet and Claude 3.5 Haiku models, regardless of which API provider you use.

Connect to Amazon Bedrock

```
CLAUDE_CODE_USE_BEDROCK=1
```

You can use environment variables to control which models Claude Code uses:

```
# Configure Sonnet
```

```
ANTHROPIC_MODEL='us.anthropic.claude-3-7-sonnet-20250219-v1:0'
```

```
# Configure Haiku
```

```
ANTHROPIC_SMALL_FAST_MODEL='us.anthropic.claude-3-5-haiku-20241022-v1:0'
```

If you'd like to access Claude Code via proxy, you can use the `ANTHROPIC_BEDROCK_BASE_URL` environment variable:

```
ANTHROPIC_BEDROCK_BASE_URL='https://your-proxy-url'
```

If you don't have prompt caching enabled, also set:
Claude Code overview

```
DISABLE_PROMPT_CACHING=1
```

Requires standard AWS SDK credentials (e.g., `~/.aws/credentials` or relevant environment variables like `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`). To set up AWS credentials, run:

```
aws configure
```

Contact Amazon Bedrock for prompt caching for reduced costs and higher rate limits.

❗ Users will need access to both Claude 3.7 Sonnet and Claude 3.5 Haiku models in their AWS account. If you have a model access role, you may need to request access to these models if they're not already available. Access to Bedrock in each region is necessary because inference profiles require cross-region capability.

Connect to Google Vertex AI

```
CLAUDE_CODE_USE_VERTEX=1
CLOUD_ML_REGION=us-east5
ANTHROPIC_VERTEX_PROJECT_ID=your-project-id
```

If you'd like to override the default model, you can pass in the `ANTHROPIC_MODEL` environment variable (Claude 3.7 Sonnet is used by default):

```
ANTHROPIC_MODEL='us.anthropic.claude-3-7-sonnet-20250219-v1:0'
```

If you'd like to access Claude Code via proxy, you can use the `ANTHROPIC_VERTEX_BASE_URL` environment variable:

```
ANTHROPIC_VERTEX_BASE_URL='https://your-proxy-url'
```

If you don't have prompt caching enabled, also set:

```
DISABLE_PROMPT_CACHING=1
```

❗ Claude Code on Vertex AI currently only supports the `us-east5` region. Make sure your project has quota allocated in this specific region.

❗ Users will need access to both Claude 3.7 Sonnet and Claude 3.5 Haiku models in their Vertex AI project.


Requires standard GCP credentials configured through `google-auth-library`. To set up GCP credentials, run:

For the best experience, contact Google for heightened rate limits.

Development container reference implementation

Claude Code provides a development container configuration for teams that need consistent, secure environments. This preconfigured **devcontainer setup** works seamlessly with VS Code’s Remote - Containers extension and similar tools.

The container’s enhanced security measures (isolation and firewall rules) allow you to run `claude --dangerously-skip-permissions` to bypass permission prompts for unattended operation. We’ve included a **reference implementation** that you can customize for your needs.

 While the devcontainer provides substantial protections, no system is completely immune to all attacks. Always maintain good security practices and monitor Claude’s activities.

Key features

- Production-ready Node.js:** Built on Node.js 20 with essential development dependencies
- Security by design:** Custom firewall restricting network access to only necessary services
- Developer-friendly tools:** Includes git, ZSH with productivity enhancements, fzf, and more
- Seamless VS Code integration:** Pre-configured extensions and optimized settings
- Session persistence:** Preserves command history and configurations between container restarts
- Works everywhere:** Compatible with macOS, Windows, and Linux development environments

Getting started in 4 steps

1. Install VS Code and the Remote - Containers extension
2. Clone the **Claude Code reference implementation** repository
3. Open the repository in VS Code
4. When prompted, click “Reopen in Container” (or use Command Palette: `Cmd+Shift+P` → “Remote-Containers: Reopen in Container”)

Configuration breakdown

The devcontainer setup consists of three primary components:

- devcontainer.json:** Controls container settings, extensions, and volume mounts
- Dockerfile:** Defines the container image and installed tools
- init-firewall.sh:** Establishes network security rules

Security features

The container implements a multi-layered security approach with its firewall configuration:

Precise access control: Restricts outbound connections to whitelisted domains only (npm registry, GitHub, Anthropic API, etc.)
Claude Code overview

Default-deny policy: Blocks all other external network access

Startup verification: Validates firewall rules when the container initializes

Isolation: Creates a secure development environment separated from your main system

Customization options

The devcontainer configuration is designed to be adaptable to your needs:

Add or remove VS Code extensions based on your workflow

Modify resource allocations for different hardware environments

Adjust network access permissions

Customize shell configurations and developer tooling