Bash tool

Copy page

The bash tool enables Claude to execute shell commands in a persistent bash session, allowing system operations, script execution, and command-line automation.

Overview

The bash tool provides Claude with:

- Persistent bash session that maintains state
- Ability to run any shell command
- Access to environment variables and working directory
- Command chaining and scripting capabilities

Model compatibility

Model	Tool Version
Claude 4 models and Sonnet 3.7 (deprecated)	`bash_20250124`

⚠ Older tool versions are not guaranteed to be backwards-compatible with newer models. Always use the tool version that corresponds to your model version.

Use cases

- **Development workflows**: Run build commands, tests, and development tools
- **System automation**: Execute scripts, manage files, automate tasks
- **Data processing**: Process files, run analysis scripts, manage datasets
- **Environment setup**: Install packages, configure environments

Claude Docs

Python

```python
import anthropic

client = anthropic.Anthropic()

response = client.messages.create(
    model="claude-sonnet-4-5",
    max_tokens=1024,
    tools=[
        {
            "type": "bash_20250124",
            "name": "bash"
        }
    ],
    messages=[
        {"role": "user", "content": "List all Python files in the current dire
    ]
)
```

How it works

The bash tool maintains a persistent session:

1. Claude determines what command to run
2. You execute the command in a bash shell
3. Return the output (stdout and stderr) to Claude
4. Session state persists between commands (environment variables, working directory)

Parameters

Parameter	Required	Description
`command`	Yes*	The bash command to run
`restart`	No	Set to `true` to restart the bash session

Example usage

Example: Multi-step automation

Claude can chain commands to complete complex tasks:

```
# User request
"Install the requests library and create a simple Python script that fetcnes a

# Claude's tool uses:
# 1. Install package
{"command": "pip install requests"}

# 2. Create script
{"command": "cat > fetch_joke.py << 'EOF'\nimport requests\nresponse = request

# 3. Run script
{"command": "python fetch_joke.py"}
```

The session maintains state between commands, so files created in step 2 are available in step 3.

Implement the bash tool

The bash tool is implemented as a schema-less tool. When using this tool, you don't need to provide an input schema as with other tools; the schema is built into Claude's model and can't be modified.

1 **Set up a bash environment**

Create a persistent bash session that Claude can interact with:

```
import queue

class BashSession:
    def __init__(self):
        self.process = subprocess.Popen(
            ['/bin/bash'],
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True,
            bufsize=0
        )
        self.output_queue = queue.Queue()
        self.error_queue = queue.Queue()
        self._start_readers()
```

2 Handle command execution

Create a function to execute commands and capture output:

```
def execute_command(self, command):
    # Send command to bash
    self.process.stdin.write(command + '\n')
    self.process.stdin.flush()

    # Capture output with timeout
    output = self._read_output(timeout=10)
    return output
```

3 Process Claude's tool calls

Extract and execute commands from Claude's responses:

```
        if content.input.get("restart"):
            bash_session.restart()
            result = "Bash session restarted"
        else:
            command = content.input.get("command")
            result = bash_session.execute_command(command)

        # Return result to Claude
        tool_result = {
            "type": "tool_result",
            "tool_use_id": content.id,
            "content": result
        }
```

4 Implement safety measures

Add validation and restrictions:

```
def validate_command(command):
    # Block dangerous commands
    dangerous_patterns = ['rm -rf /', 'format', ':(){:|:&};:']
    for pattern in dangerous_patterns:
        if pattern in command:
            return False, f"Command contains dangerous pattern: {pattern}"

    # Add more validation as needed
    return True, None
```

Handle errors

When implementing the bash tool, handle various error scenarios:

- **Command execution timeout**
- **Command not found**
- **Permission denied**

Claude Docs

- **Use command timeouts**
- **Maintain session state**
- **Handle large outputs**
- **Log all commands**
- **Sanitize outputs**

Security

⚠ The bash tool provides direct system access. Implement these essential safety measures:

- Running in isolated environments (Docker/VM)
- Implementing command filtering and allowlists
- Setting resource limits (CPU, memory, disk)
- Logging all executed commands

Key recommendations

- Use `ulimit` to set resource constraints
- Filter dangerous commands (`sudo` , `rm -rf` , etc.)
- Run with minimal user permissions
- Monitor and log all command execution

Pricing

The bash tool adds **245 input tokens** to your API calls.

Additional tokens are consumed by:

- Command outputs (stdout/stderr)

See tool use pricing for complete pricing details.

Common patterns

Development workflows

- Running tests: `pytest && coverage report`
- Building projects: `npm install && npm run build`
- Git operations: `git status && git add . && git commit -m "message"`

File operations

- Processing data: `wc -l *.csv && ls -lh *.csv`
- Searching files: `find . -name "*.py" | xargs grep "pattern"`
- Creating backups: `tar -czf backup.tar.gz ./data`

System tasks

- Checking resources: `df -h && free -m`
- Process management: `ps aux | grep python`
- Environment setup: `export PATH=$PATH:/new/path && echo $PATH`

Limitations

- **No interactive commands**: Cannot handle `vim`, `less`, or password prompts
- **No GUI applications**: Command-line only
- **Session scope**: Persists within conversation, lost between API calls
- **Output limits**: Large outputs may be truncated
- **No streaming**: Results returned after completion

Combining with other tools

The bash tool is most powerful when combined with the text editor and other tools.

Claude Docs

Tool use overview

Learn about tool use with Claude

Text editor tool

View and edit text files with Claude

Claude Docs

Solutions

AI agents
Code modernization
Coding
Customer support
Education
Financial services
Government
Life sciences

Partners

Amazon Bedrock
Google Cloud's Vertex AI

Learn

Blog
Catalog
Courses
Use cases
Connectors
Customer stories
Engineering at Anthropic
Events
Powered by Claude
Service partners
Startups program

Company

Anthropic
Careers

Claude Docs

News

Responsible Scaling Policy

Security and compliance

Transparency

Help and security

Availability

Status

Support

Discord

Terms and policies

Privacy policy

Responsible disclosure policy

Terms of service: Commercial

Terms of service: Consumer

Usage policy