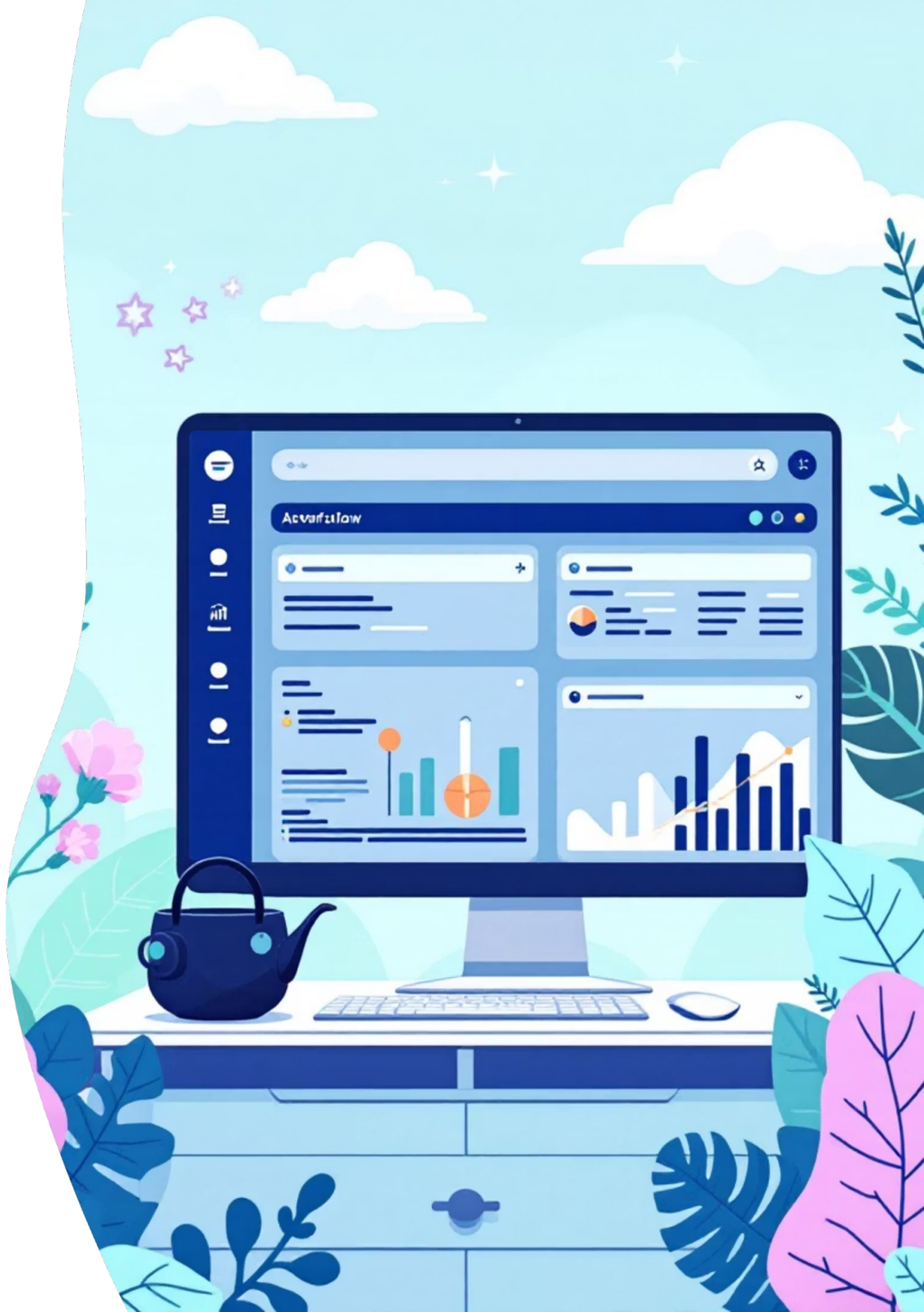
Slash Commands

Automate Your Workflow with Custom Commands



What Are Slash Commands?

Slash commands are custom prompts stored as Markdown files that you trigger using the `/command-name` syntax in Claude Code. They transform repetitive tasks into instant, reusable workflows.

Reusable Templates

Store common workflows once, execute instantly

Save Time

Eliminate repetitive typing and context switching

Team Collaboration

Share commands across your development team

Consistency

Standardize processes and improve output quality

Types of Slash Commands

Project-Scoped Commands

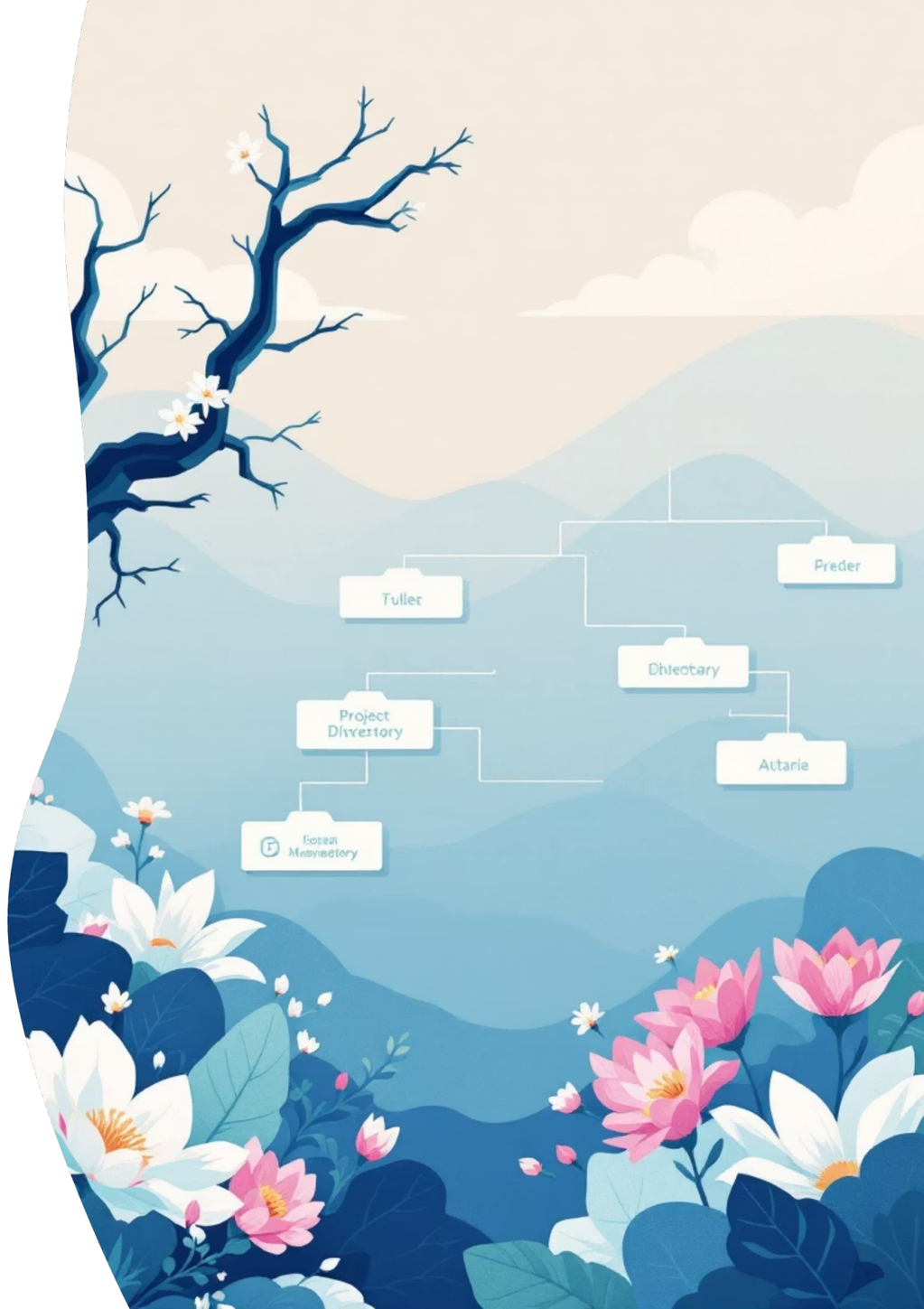
Stored in your project's `.claude/commands/` directory and shared with your team via version control.

- Display as "(project)" in /help
- Version-controlled with your code
- Enable team collaboration
- Standardize team workflows

User-Scoped Commands

Available across all your projects, stored in `~/.claude/commands/` for personal use.

- Display as "(user)" in /help
- Personal automation
- Consistent across all work
- Individual productivity boost



Built-in vs Custom Commands

/help List commands	**/add-dir** Expand workspace	**/mcp** Manage servers
/compact Compress chat	**/init** Initialize project	

The commands above are built-in and available by default. **Custom commands** extend this foundation—you create Markdown files to define your own workflows, arguments, and logic tailored to your specific needs.

Slash Command Structure

Every slash command follows a consistent structure to ensure clarity and functionality.

1

Title & Description

Start with `# Command Title` followed by a brief description explaining the command's purpose

2

Instructions

Use `## Instructions` section with numbered steps for execution

3

Arguments

Define dynamic values: `$ARGUMENTS` for all args, or `$1, $2, $3` for positional access

Example: `/fix-issue 123 high-priority` passes "123" to $1 and "high-priority" to $2, allowing targeted automation.

Command Frontmatter (Advanced)

Frontmatter provides metadata that controls how your command behaves within Claude Code.

1. **description**
   Text displayed in the command list when users type /help

2. **argument-hint**
   Shows expected parameters, guiding users how to invoke command

3. **model**
   Specify a particular Claude model version for this command's execution

4. **allowed-tools**
   Restrict which tools the command can access (e.g., git permissions)

`allowed-tools: Bash(git add:*), Bash(git commit:*)` restricts execution to specific Git operations for safety.

Best Practices

✓ Do

- Use clear, descriptive command names
- Include step-by-step instructions
- Add frontmatter descriptions
- Organize with namespaces (e.g., /code-review)
- Share project commands via git

✗ Don't

- Create overly complex single commands
- Forget to document arguments
- Mix project and user commands
- Hardcode sensitive information
- Skip testing before sharing

https://github.com/firstlink/claude-code/tree/main/slash-commands-and-agents/.claude/commands