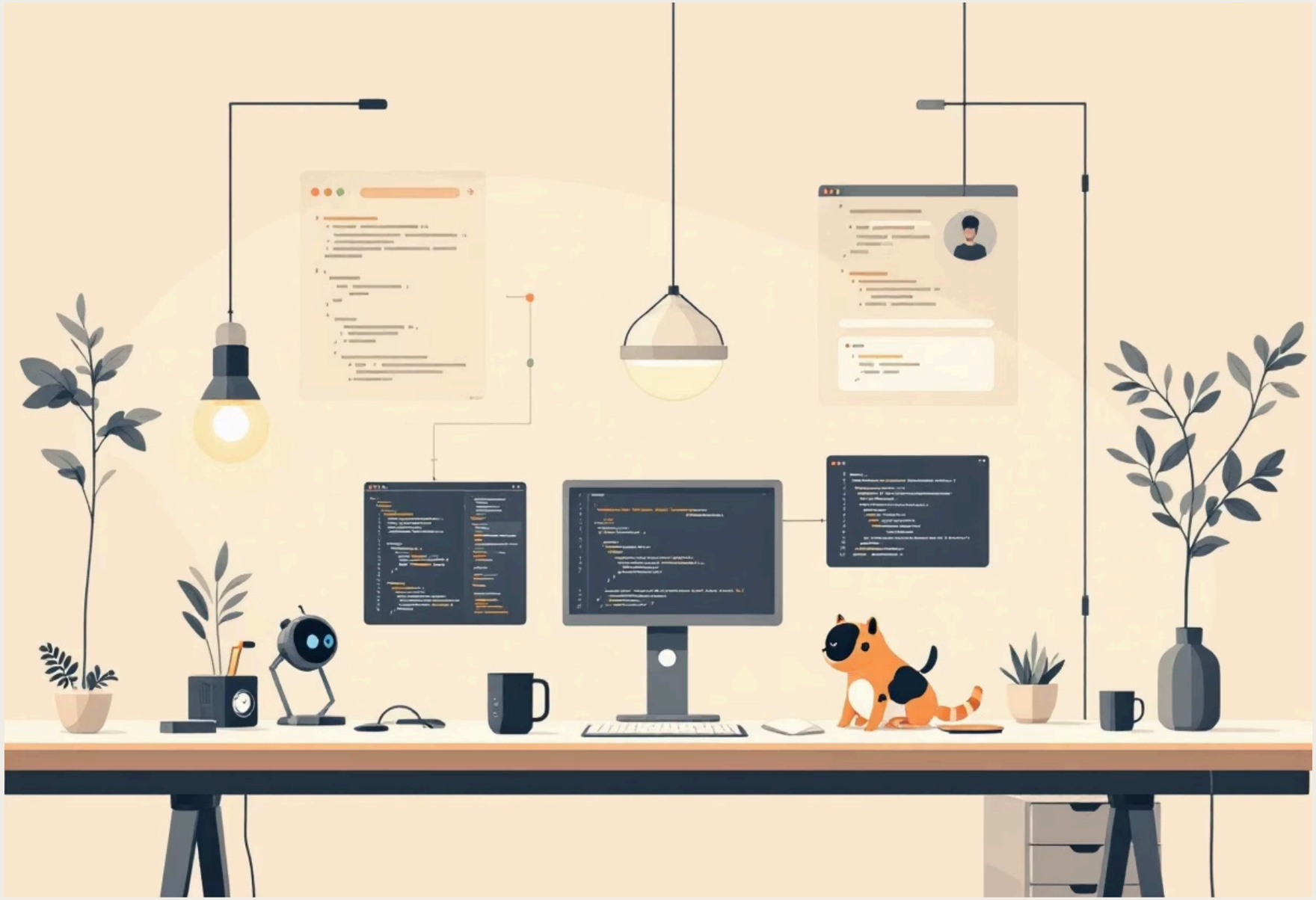
DevFlux AI Workflows



Setup & Usage Guide

Introduction

DevFlux AI Workflows are a systematic approach to using AI coding assistants that dramatically improves success rates from ~10% to 90%+.

Why These Workflows Work

AI coding assistants have predictable failure modes:

AI Failure Mode	Workflow Defense
Jumping to conclusions	Context-First mandatory stop
Hallucinating code paths	100% Validation Checklist
Losing track of goal	RE-ANCHOR at every step
Missing edge cases	Chronological Timeline Debugging
Assuming patterns	"Claims require evidence" with [file:line] citations

Quick Start

01

Download the Workflow Files

You should have these 6 workflow files:

- quick-fix.md
- complex-issue.md
- story-implementation.md
- bigcodechange.md
- release-upgrade.md
- test-writing.md

02

Install in Your IDE

Follow the IDE-specific instructions below.

03

Invoke a Workflow

Type `/quick-fix` or `/complex-issue` in your AI chat to start.

IDE Setup Instructions

Cursor Setup

Cursor uses Custom Slash Commands stored as Markdown files.

Project-Level Setup (Team Sharing)

Create the commands directory in your project:

```
your-project/
└── .cursor/
    └── commands/
        ├── quick-fix.md
        ├── complex-issue.md
        ├── story-implementation.md
        ├── bigcodechange.md
        ├── release-upgrade.md
        └── test-writing.md
```

Steps:

```
# Create the directory
mkdir -p .cursor/commands

# Copy your workflow files
cp path/to/workflows/*.md .cursor/commands/
```

Commands stored here are:

- ✅ Version controlled with your project
- ✅ Shared with your team automatically
- ✅ Available when anyone opens the project

Global Setup (Personal Use)

Store commands in your home directory for use across all projects:

```
~/.cursor/
└── commands/
    ├── quick-fix.md
    ├── complex-issue.md
    ├── story-implementation.md
    ├── bigcodechange.md
    ├── release-upgrade.md
    └── test-writing.md
```

Steps:

```
# macOS/Linux
mkdir -p ~/.cursor/commands
cp path/to/workflows/*.md ~/.cursor/commands/

# Windows (PowerShell)
mkdir -Force "$env:USERPROFILE\.cursor\commands"
Copy-Item path\to\workflows\*.md "$env:USERPROFILE\.cursor\commands\"
```

Using Commands in Cursor

1. Open Cursor's AI chat (Cmd+L / Ctrl+L) or Agent mode
2. Type / to see all available commands
3. Select your workflow (e.g., /quick-fix)
4. The workflow instructions are inserted into the chat
5. Add your specific context and press Enter

Windsurf Setup

Windsurf uses Workflows stored as Markdown files.

Global Setup

Store workflows in your Codeium user directory for use across ALL projects:

macOS

```
~/.codeium/windsurf/global_workflows/
```

Full path:

```
/Users/<username>/.codeium/windsurf/global_workflows/
```

Steps:

```
# Create the directory
mkdir -p ~/.codeium/windsurf/global_workflows

# Copy your workflow files
cp path/to/workflows/*.md ~/.codeium/windsurf/global_workflows/
```

Windows

```
C:\Users\<username>\.codeium\windsurf\global_workflows\
```

```
mkdir -Force "$env:USERPROFILE\.codeium\windsurf\global_workflows"
Copy-Item path\to\workflows\*.md "$env:USERPROFILE\.codeium\windsurf\global_workflows\"
```

Linux

```
mkdir -p ~/.codeium/windsurf/global_workflows
cp path/to/workflows/*.md ~/.codeium/windsurf/global_workflows/
```

Using Workflows in Windsurf

1. Open Cascade (Cmd+L / Ctrl+L)
2. Type / followed by workflow name: `/quick-fix`
3. Press Enter to invoke
4. Add your specific context

Other AI Tools

Claude (API/Projects)

Claude Projects:

1. Create a new Project in claude.ai
2. Go to Project Knowledge
3. Upload all workflow files
4. Start conversations with: "Use the Complex Issue workflow"

Per-Conversation:

```
I want to use the Quick Fix workflow.

[paste workflow content]

Here's my issue: ...
```

GitHub Copilot Chat

Add to your prompt:

```
@workspace Follow this workflow:

[paste workflow content]

Issue: ...
```

Cline / Continue

Add workflows to your system prompt configuration or .continuerules file.

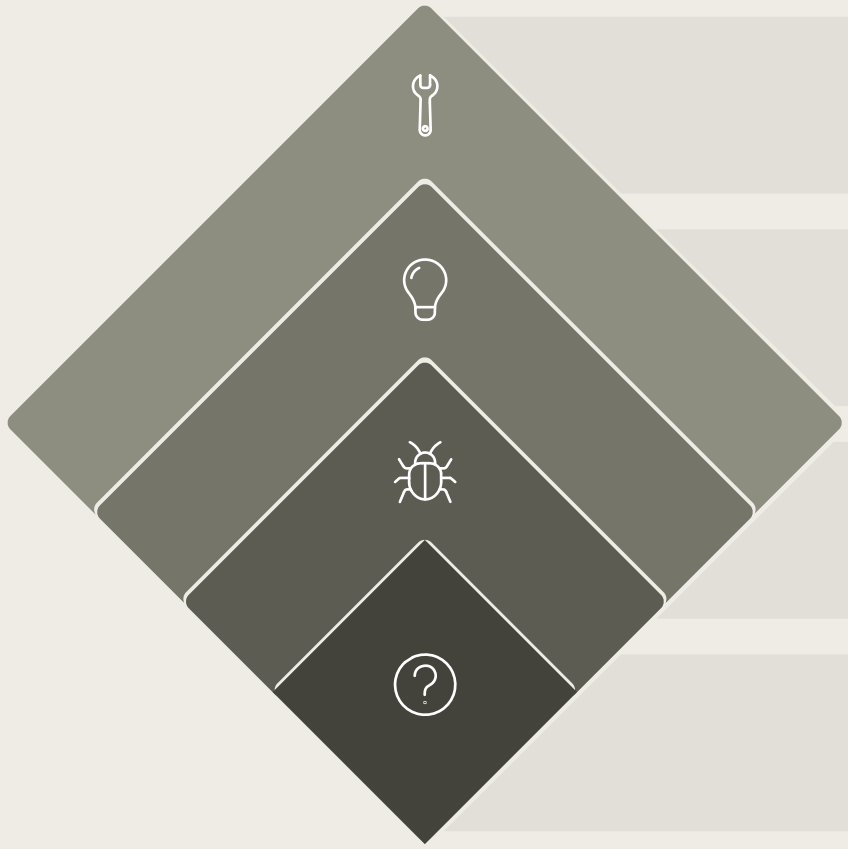
Aider

Add to your .aider.conf.yml or conventions file:

```
conventions: |
 Follow the DevFlux workflow system.
 [paste relevant workflow]
```

Workflow Overview

Choosing the Right Workflow



Maintenance

Upgrade or large refactor

Building New

Implement a story

Fixing Bugs

Simple or complex?

What are you doing?

Choose a main category

Quick Reference

Workflow	Use When	Invoke
Quick Fix	Simple bug, known area	`/quick-fix`
Complex Issue	Race conditions, intermittent bugs	`/complex-issue`
Story Implementation	New feature, user story	`/story-implementation`
Big Code Change	Refactor >10 files, migration	`/bigcodechange`
Release Upgrade	Bug after version upgrade	`/release-upgrade`
Test Writing	Adding/updating tests	`/test-writing`

Using the Workflows

Example: Quick Fix

Invoke the workflow

```
/quick-fix
```

AI asks context questions

```
AI: Before I search any code, I need to understand the issue:

1. What exactly is the problem? Expected vs actual behavior?
2. What triggers this issue?
3. Any specific file or module you suspect?
4. Any recent changes that might have caused this?
```

You provide context

```
1. Users see "undefined" instead of their name in greeting
2. Happens on every page load
3. Suspect UserGreeting component
4. Started after yesterday's API update
```

AI analyzes and proposes

```
AI: ISSUE: Greeting shows "undefined" instead of user's name

ROOT CAUSE: `user.firstName` changed to `user.name.first` in API
FILE: src/components/UserGreeting.tsx
LINES AFFECTED: Line 15
RISK: Low

**STOP POINT**: Approve this proposal?
```

You approve, AI implements

```
You: Approved

AI: [Implements exact change]
[Writes/updates tests]
[Verifies against proposal]

COMPLETE
```

Example: Complex Issue

Invoke:

```
/complex-issue

Issue: Payment sometimes fails during checkout
Frequency: ~5% of transactions
Scenario: High traffic periods, concurrent users
```

The workflow will:

1. Gather Context → Ask detailed questions (STOP POINT)
2. Map Call Chain → Create chronological timeline
3. Analyze Flow → Trace data through the system
4. Generate Hypotheses → At least 3, with evidence
5. Validate 100% → Complete code reading checklist
6. Present Findings → Only 100% validated hypotheses (STOP POINT)
7. Plan Implementation → Phased approach with rollback points
8. Implement → Execute plan, test each phase
9. Verify → Compare actual vs planned changes
10. Trigger Tests → Hand off to test-writing workflow

Key Concepts

TOON Format

All workflow artifacts use TOON (Token-Oriented Object Notation):

```
ISSUE_CONTEXT.toon
CALL_CHAIN.toon
HYPOTHESES.toon
IMPLEMENTATION_PLAN.toon
```

If unfamiliar, the AI will search for proper syntax before creating files.

Context-First Principle

Every workflow starts with understanding, not code search.

```
## Step 1: Gather Context (STOP POINT)

1. **STOP**: Ask the user questions
2. Create context document
3. **Do not proceed** until context is provided
```

This prevents AI from making assumptions.

RE-ANCHOR

Every step restates the goal explicitly:

```
**RE-ANCHOR**: Read `ISSUE_CONTEXT.toon`.
Output: "ISSUE: [one-line description]"
```

This prevents drift during long conversations.

STOP Points

Critical decision points requiring your approval:

```
## Step N: [Name] (STOP POINT)

1. **STOP**: Review the [artifact]
2. Approve to proceed
```

Never skip STOP points — they maintain your control.

// turbo

Steps marked // turbo chain automatically:

```
Step 2: Flow Analysis
// turbo
Step 3: Generate Hypotheses
// turbo
Step 4: Validate Hypotheses

Step 5: User Confirmation (STOP POINT) ← Stops here
```

100% Validation

Before presenting hypotheses:

```
**100% Validation Checklist**:
[ ] Every method: COMPLETE code read
[ ] Every variable: All usages traced
[ ] Every API call: Implementation verified
[ ] Every claim: [file:line] citation

**ONLY 100% VALIDATED hypotheses presented**
```

Auto-Detect Test Environment

Testing adapts to your project:

```
**Auto-Detect Test Environment**:
- Scan: package.json, pom.xml, requirements.txt, etc.
- Find existing test patterns
- Match framework, style, assertions exactly
- If unclear: ASK user
```

Best Practices

1. Answer Context Questions Thoroughly

❌ Bad: "It's broken sometimes"

✅ Good: "Payment fails ~5% of the time, only during high traffic (>100 concurrent users), started after we deployed the caching layer on Jan 15"

2. Don't Skip STOP Points

Even if confident:

- Catch AI mistakes early
- Prevent scope creep
- Maintain understanding

3. Use Git Tags for Rollback

```
git tag PRE_FIX_batch1
# If something goes wrong:
git reset --hard
PRE_FIX_batch1
```

4. Review Artifacts

The .toon files are valuable documentation:

- CALL_CHAIN.toon — System understanding
- HYPOTHESES.toon — Debugging record
- IMPLEMENTATION_PLAN.toon — Change log

Consider committing them.

5. Trust But Verify

Always:

- Run tests after changes
- Manual smoke test
- Code review the diff

Troubleshooting & Support

Common Issues

Workflows not appearing in IDE

- Verify file location matches IDE requirements
- Check file extensions are .md
- Restart IDE after adding files

AI not following workflow steps

- Ensure you're invoking with correct command
- Provide complete context at STOP points
- Re-invoke workflow if AI drifts

TOON files not being created

- AI will search for TOON syntax first
- Ensure workspace has write permissions
- Check if files are being created in correct directory

Support

For questions, customization, or team training:

DevFlux AI Workflow Consulting

Website: **devflux.pro**

Version 6.0 | DevFlux AI Workflows