Claude Code
Features Guide

Part 3 of the Claude Code Series

Claude Code Checkpoints and Rewind: Never Fear a Bad AI Edit Again

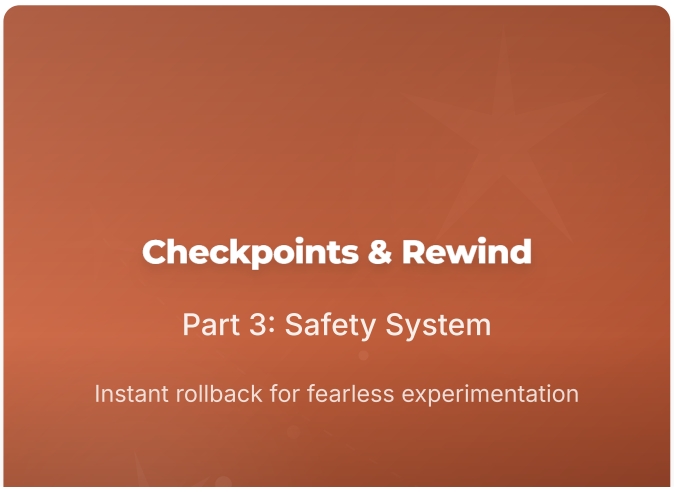
Learn how Claude Code's checkpoints and rewind feature automatically saves your code before every AI edit, letting you instantly roll back changes. Complete guide for developers and agencies.

9 February 2026 · 11 min read

There is a moment every developer knows well. You ask an AI assistant to refactor a function, it touches fifteen files, and suddenly nothing compiles. Your stomach drops. How far back do you need to go? What exactly changed? Can you even recover the original state?

Claude Code's checkpoint and rewind system eliminates that anxiety entirely. Every single edit the AI makes is automatically saved as a checkpoint. If anything goes wrong, you press Escape twice or type `` `/rewind` ``, and you are back to exactly where you were.

This is not just a convenience feature. It fundamentally changes how ambitious you can be when working with AI coding tools. When rollback is instant, risk is no longer a reason to hold back.



What Are Claude Code Checkpoints?

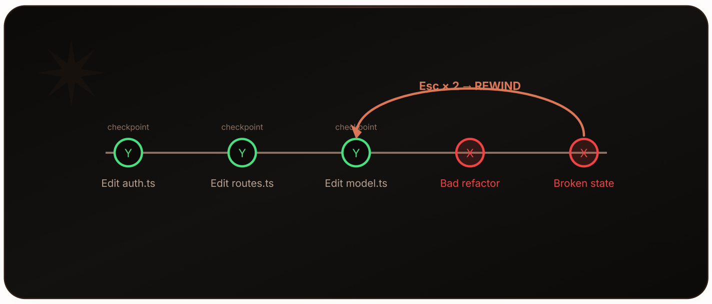
Checkpoints are automatic, per-edit snapshots of your code that Claude Code creates before every change it makes. Think of them as a detailed undo history, but far more granular than anything you would get from Git alone.

How they work:

- Before Claude Code edits any file, it saves a checkpoint of the current state
- Each checkpoint captures the exact state of every file in the session
- Checkpoints are created silently in the background with zero interruption to your workflow
- They persist for the duration of your current coding session

You do not need to configure anything, remember to save, or manually create restore points. The system is entirely automatic. Every time Claude Code touches your code, it has already recorded what was there before.

Checkpoint Timeline with Rewind



How to Use Rewind

Accessing your checkpoints is straightforward. There are two methods:

Method 1: Double Escape

Press `` `Esc` `` twice in quick succession. This opens the checkpoint timeline, showing you every change Claude Code has made during your session.

Method 2: The /rewind Command

Type `` `/rewind` `` in the Claude Code terminal. This gives you the same checkpoint timeline with a scrollable history of all changes.

Three Restore Options

When you select a checkpoint, Claude Code presents three choices:

1. **Conversation only** - Rewinds the chat history to that point but leaves your files as they are. Useful when the AI went down a wrong conversational path but the code changes are actually fine.

2. **Code only** - Restores your files to the checkpoint state but keeps the full conversation history. This is the most common choice: the discussion was productive, but the implementation needs another go.

3. **Both** - Rewinds everything, conversation and code, back to the checkpoint. A complete reset to that moment in time.

This granularity matters. In most cases, you do not want to lose the entire conversation. You want to keep the context, the reasoning, the decisions that were made, and simply try a different implementation approach.

Esc x 2

Instant Rewind

Two keystrokes to undo any AI change. Checkpoints automatically save before every edit — you're always one keystroke away from safety.

Why Checkpoints Change How You Work with AI

The real power of checkpoints is not in recovering from mistakes. It is in the permission they give you to be aggressive.

1. Fearless Refactoring

Without checkpoints, asking an AI to refactor a large codebase is a calculated risk. The AI might misunderstand the architecture, break dependency chains, or introduce subtle bugs across dozens of files.

With checkpoints, you can say "refactor the entire authentication module to use the new pattern" and know that if the result is not right, you are two keystrokes away from the original state. The cost of trying drops to nearly zero.

2. Exploring Multiple Approaches

Checkpoints turn coding into a branching exploration. You can ask Claude Code to implement a feature one way, evaluate the result, rewind, and then try a completely different approach. Compare the two implementations with full context of how each one felt during development.

This is something that Git branches technically allow, but the overhead of creating branches, switching, and managing them makes it impractical for quick experiments. Checkpoints are instant.

3. Recovering from Bad Suggestions

Every AI makes mistakes. Claude Code is remarkably capable, but it can still misinterpret requirements or make assumptions that do not match your codebase. Checkpoints mean that a bad suggestion is never more than a minor inconvenience. Rewind, provide better context, and try again.

4. Learning by Experimentation

For developers learning new patterns or frameworks, checkpoints create a safe sandbox. Try the AI's suggestion, see what happens, rewind if needed, and build understanding through iteration without the fear of leaving your codebase in a broken state.

Automatic Checkpoint Creation

Every time Claude edits a file, a checkpoint is silently created. No setup required, no manual saves. The safety net is always there.

Checkpoints vs Git: Complementary, Not Competing

A reasonable question: if you are already using Git, why do you need checkpoints?

The answer is that they operate at different levels of granularity and serve different purposes.

Aspect	Git	Claude Code Checkpoints
Granularity	Per-commit (you decide when)	Per-edit (every AI change)
Scope	Entire repository	Current session files
Purpose	Long-term version history	Short-term safety net

Aspect	Git	Claude Code Checkpoints
Overhead	Manual commits, messages, branches	Fully automatic
Persistence	Permanent	Session duration
Collaboration	Shared with team	Local to your session

Think of it this way: Git is your permanent record. Checkpoints are your working safety net.

During a typical coding session, Claude Code might make twenty or thirty individual edits across multiple files. Each of those edits gets a checkpoint. When you are satisfied with the overall result, you commit to Git. The checkpoints gave you the freedom to experiment; Git gives you the permanent history.

They are complementary tools at different scales. You would not stop using Git because you have checkpoints, and checkpoints solve problems that Git's commit-based model is not designed for.

Practical Scenarios

Scenario 1: The Risky Database Migration

Your agency has a client whose database schema needs updating. You ask Claude Code to write the migration scripts. The first attempt generates migrations that would break existing queries.

Without checkpoints: You would need to manually identify every change, figure out what went wrong, and carefully undo each modification. If you had already run the migrations in a development environment, you might need to restore from a backup.

With checkpoints: Press Escape twice, select the checkpoint from before the migration was generated, choose "Code only" (keeping the conversation for context), and try again with better instructions. Total recovery time: under five seconds.

Scenario 2: The Multi-file Refactor

You are restructuring a React application, moving from a monolithic component structure to a feature-based architecture. Claude Code needs to move files, update imports, modify routing, and adjust test files.

The refactor touches forty files. Most changes are correct, but the routing configuration has a subtle error.

Without checkpoints: You need to find the routing issue among forty changed files. Even with Git diff, this is time-consuming detective work.

With checkpoints: Rewind to before the routing changes, keep the earlier file movements, and give Claude Code more specific instructions for the routing portion. Surgical precision with zero detective work.

Scenario 3: The Exploratory Design Session

Your team is debating two different API designs for a new microservice. Rather than spending hours in a whiteboard session, you use Claude Code to prototype both approaches.

Build approach A, evaluate it, rewind to the starting checkpoint, build approach B, and now you have concrete implementations to compare rather than abstract diagrams. The whole exercise takes thirty minutes instead of a full afternoon.

---

3 RESTORE OPTIONS

1 Conversation only: go back to a prior

prompt

2 Code only: revert files, keep history

3 Both: full restore to prior point

How We Use Checkpoints at Jordan James Media

At our agency, we run AI-assisted development across multiple client projects simultaneously. Checkpoints have become an essential part of how we work.

Client Site Experimentation

When we are running SEO optimisations or implementing new features on client sites, checkpoints give us the confidence to try bold approaches. We recently ran a comprehensive audit across more than twenty microsites, and checkpoints meant the team could explore aggressive optimisation strategies knowing that any change could be instantly reverted.

This is particularly valuable in agency work where the stakes are higher. These are not personal projects; they are client businesses that depend on their websites functioning correctly. Checkpoints provide a safety net that makes AI-assisted work on client projects practical rather than risky.

Rapid Prototyping for Client Proposals

When a client asks "could you build something that does X?", we can prototype it in real time. If the first approach does not match what they had in mind, rewind and try another direction. The client sees options, not excuses.

Training Junior Developers

Checkpoints have been invaluable for training. Junior developers can work with Claude Code on real tasks, knowing that mistakes are trivially reversible. This removes the paralysis that often comes with inexperience and lets them learn through doing rather than just watching.

– WITHOUT CHECKPOINTS

- Manual git stash before experiments
- Fear of breaking working code
- Conservative, incremental changes
- Lost work from bad AI suggestions

WITH CHECKPOINTS

- Automatic saves before every edit
- Experiment freely, rewind instantly
- Try bold refactors without risk
- Every state is recoverable

Limitations to Be Aware Of

Checkpoints are powerful, but they have boundaries:

Session-scoped: Checkpoints exist for the duration of your current Claude Code session. Once you close the session, the checkpoint history is gone. This is why Git remains essential for permanent version control.

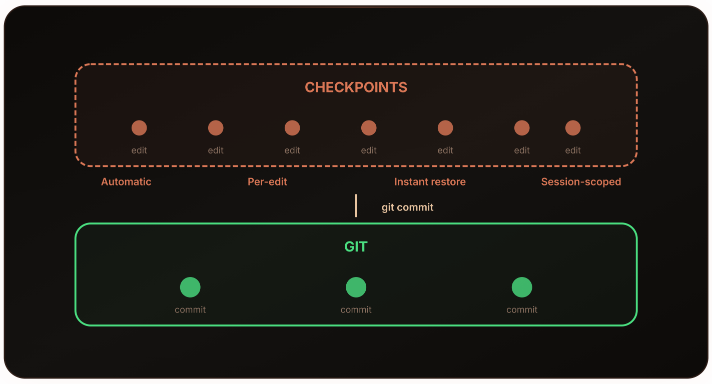
AI edits only: Checkpoints track changes made by Claude Code. If you manually edit files outside of Claude Code during a session, those manual changes are not captured in the checkpoint system. The system tracks what the AI does, not what you do independently.

File-based: Checkpoints capture file state. They do not capture external state like database changes, environment configurations, or running processes. If Claude Code helps you write a script that modifies a database, rewinding the code does not undo the database changes.

Local only: Checkpoints are not shared across machines or team members. Each developer's checkpoint history is their own.

Understanding these boundaries helps you use checkpoints effectively. They are a session-level safety net, not a replacement for comprehensive version control, backups, or deployment pipelines.

Checkpoints vs Git: Complementary Layers



Checkpoints and Plan Mode: A Safety Partnership

Checkpoints pair naturally with Claude Code's Plan Mode (covered in our Plan Mode guide). Here is how they work together:

1. **Plan Mode** lets you review what Claude Code intends to do before it does it
2. **Checkpoints** let you recover if the execution does not match expectations

Together, they create a two-layer safety system. Plan Mode is preventative. Checkpoints are corrective. Using both means you can work confidently with AI on tasks of any complexity.

PRO TIP

Three Restore Modes

Conversation only: Rewind the chat to a prior prompt but keep file changes. **Code only:** Revert files but keep conversation history. **Both:** Full restore to a prior state. Choose based on what you need to keep.

Getting Started with Checkpoints

There is nothing to set up. If you are using Claude Code, checkpoints are already active. But here are some tips for getting the most out of them:

1. Check the timeline regularly. Even when things are going well, glance at the checkpoint timeline occasionally. It gives you a clear picture of how many changes have been made and what the AI has touched.

2. Use "Code only" rewind most often. In our experience, preserving the conversation while rolling back code is the most useful pattern. You keep the context and reasoning; you just try a different implementation.

3. Commit to Git at natural breakpoints. Use checkpoints freely during active development, but commit to Git when you reach a stable state. This preserves your progress permanently.

4. Combine with Plan Mode for high-stakes work. When working on production code or client projects, use Plan Mode to review intentions and checkpoints as your safety net.

5. Do not be afraid to rewind multiple times. There is no penalty for rewinding. Some of our best implementations came after three or four rewind-and-retry cycles. Each attempt teaches the AI more about what you need.

The Complete Safety Stack

PS **Permission System**
Control what Claude can do
Layer 4

PM **Plan Mode**
Think before acting
Layer 3

CP **Checkpoints**
Instant rollback per-edit
Layer 2

GI **Git**
Permanent version history
Layer 1

Each layer reinforces the others. Together they make AI development safe by default.

Key Takeaways

1. **Checkpoints are automatic** - every AI edit creates a save point with zero effort from you
2. **Double Escape or /rewind** accesses your checkpoint timeline instantly
3. **Three restore options** let you rewind conversation, code, or both independently
4. **Fearless experimentation** becomes possible when rollback is instant
5. **Complementary to Git** - checkpoints handle per-edit safety, Git handles permanent history
6. **Session-scoped** - checkpoints exist for your current session only, so commit important work to Git
7. **Pairs with Plan Mode** for a two-layer safety system: prevention and correction
8. **Agency-essential** - working on client code demands the safety net that checkpoints provide

> “
> *Checkpoints changed how we work with AI. We stopped being cautious and started being ambitious — because the cost of failure dropped to zero.*
>
> —— Jordan James Media

Work with Confidence, Not Caution

Checkpoints and rewind transform AI-assisted development from a careful, measured process into an exploratory, ambitious one. When every change can be undone in seconds, the question shifts from "is this safe to try?" to "what should we try first?"

At Jordan James Media, we believe that the best results come from bold experimentation backed by reliable safety nets. Checkpoints provide exactly that.

Want to see how AI-powered development can accelerate your projects? Talk to our team at Jordan James Media about how we use Claude Code to deliver faster, better results for our clients.

Related Reading:

- Claude Code's Biggest Update: 7 Features That Changed How We Build
- Claude Code Plan Mode: Think Before You Build
- Claude Code Agent Teams: Parallel AI Development

Build With Confidence

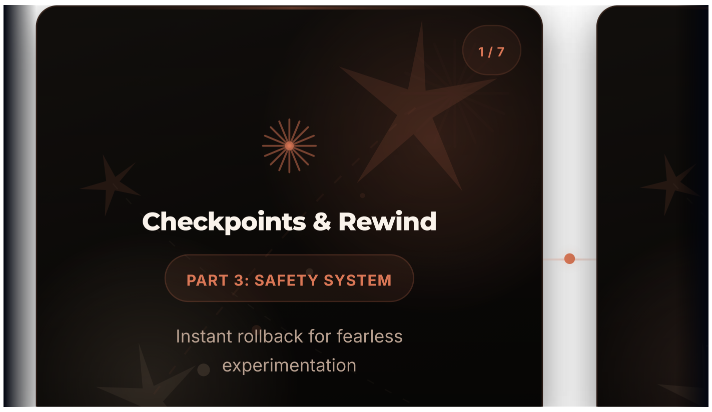
AI development with built-in safety

Get Your Free AI Strategy

Social Media Carousel

7 cards — Scroll to browse, click any card to download

PDF for LinkedIn | ZIP Images



Share This Article

Spread the knowledge

Twitter LinkedIn Copy

Build With Claude Code

We use Claude Code to ship faster, smarter, and at scale. Let us build your next project.

Get Your Free AI Strategy → | App Development

Claude Code
8 parts

- Part 0
- Part 1
- Part 2
- Part 3
- Part 4
- Part 5
- Part 6
- Part 7

View All

Quick Links

→ Series Home
→ App Development
→ Get In Touch

Claude Code Stats

Model	Opus 4.6
Features	7
Series Parts	8

Agency Tested

Real projects, real results

Ready to outperform your competition?

Get your free AI strategy consultation today.

Start Your Free Consultation →

Jordan James Media
GROWTH STUDIO

Australia's AI-first digital marketing agency. We build intelligent systems that drive exponential growth.

info@jordanjamesmedia.com

0475 897 737

Sydney, Australia

SERVICES

- SEO Services
- Google Ads
- Website Design
- Local SEO
- Content Marketing
- App Development

COMPANY

- About Us
- Blog
- Case Studies
- Pricing
- Tools
- Contact

LOCATIONS

- Sydney
- Melbourne
- Brisbane
- Perth
- Adelaide

All Locations

All Locations →

Privacy Policy

Terms of Service

Portal Login

Built with Claude 4.6 Opus