Claude Code Hooks: Get Notified When Tasks Finish

14 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=G9OBGN0H6LE
https://www.youtube.com/watch?v=G9OBGn0h6LE

SUMMARY

Claude Code hooks allow users to automate actions based on specific trigger events within the Claude environment, enhancing productivity and ensuring consistency in tasks. By attaching actions to various triggers, users can streamline workflows and enforce coding conventions or organizational structures.

- *Hooks consist of two components: trigger events and corresponding actions.*
- *Users can set hooks to execute actions at specific points, such as starting a session or submitting a prompt.*
- *An example of a hook is a confetti animation that triggers upon completing a response.*
- *Hooks can enforce coding standards by automatically formatting code or checking for syntax compliance.*
- *A practical application includes organizing directory structures by preventing the creation of overlapping or duplicate folders.*
- *The effectiveness of hooks is determined by their deterministic nature, but subjective tasks may introduce latency.*
- *Users should be selective about the number of hooks to avoid workflow slowdowns.*
- *Filtering mechanisms can be implemented to ensure hooks only activate for specific commands or conditions.*

What can you do with Claude Code hooks? I'm going to type in hip hip here. Okay. Uh What do you usually say when someone says hip hip? Hooray. Great. Uh so, you can see that confetti there. Uh this is a really simple, silly uh example, but you can see that every time I type in something, um Claude Code is going to respond, and as soon as it is done responding, it plays this confetti animation on my computer. This was built using Claude Code, obviously, but specifically a hook.

And so, what is a hook? Uh a hook is a an action that you can tell Claude Code to take uh based on a trigger event that happens inside of Claude Code. So, your uh interaction with Claude Code goes through a few phases inside of when you're working inside Claude Code, right? Um I'm going to separate out two two sort of different components of a hook.

The first is the hook like trigger, and then the other is the hook action, right? So, let's look at hook trigger. I'm looking at Claude Code docs here, and you can see that uh under hooks reference, they have a really nice uh diagram of this like life cycle of all the different triggers that happen when you're using Claude Code. So, first you'll start a session. That's when you type in Claude, and it fires up the Claude uh CLI, right? Uh and then you will write something. You will ask Claude to do something, or you'll whatever, you'll you'll give it instructions and that's the user prompt.

You submit it and then there's like this agentic loop, right? If Claude has to go and use some tools like search your local files or do a bunch of web research or whatever it is, it's going to like think about using that tool and then say, "Okay, I'm going to use that tool." So that's the pre-tool use and then it might ask you for permission to use that tool. It'll use that tool and then after that tool be another trigger there and then it might start a sub-agent and then eventually it'll say, "Okay, this task has been completed."

It'll stop. At some point it'll compact its context window and then it'll end the session, right? So like this is like a nice summary of the life cycles and why did I go through all of these? Because you can attach a hook action to any of these triggers, right? So you can say, "Every time I start a session with Claude, run this action." Or every time I submit a prompt to Claude, do this.

The example that you saw with the confetti is um when it's it's it's actually not in here. That's funny. It's actually not in here, but it is here. So if you type in hooks, it's actually kind of a nice natural segue. You can actually see all of the different hooks the trigger points, right? So he says pre-tool use. I'll actually do this. There we go. Right here. This is the confetti one. So you can see stop hook and I run the confetti script right before Claude concludes its its response is when the trigger, so the hook trigger happens and as soon as it triggers, then it triggers this action that I've specified here, right?

Um you might notice that I have another hook and I'll show you this, but I just want to um go back and explain the reason why hooks are really powerful is that if you need to do something at a certain trigger point every single time then you want to create that turn that into a hook instead of having to prompt Claude every time. So, for example programmers use this a lot because they'll write some code or they'll get Claude to write some code, but then if it's not in the right format or the right syntax that they like to write the code in that there's like, you know, the their team might have some sort of formatting convention and Claude didn't follow it, they can run this hook every time a file gets updated or created to make sure that the the syntax is in line with how with their convention of how how they write code. And what makes hooks really powerful is that it's deterministic in that it's code. It's like every time Claude code stops or every time Claude code um starts a session every single time run this hook, right? There's nothing uh probabilistic about that and so you can uh sort of set these really defined guardrails around the way you you work, right? And so, I gave you an example of um the the formatting of the of the code that programmers would use, but what's like an equivalent for um for GTM, right? There isn't something that um you would necessarily do something like like linting or or formatting of of code. Um but here's an example, right? So, uh what I've noticed is that Claude often times will generate all these artifacts in terms of files and they'll and and they'll start creating folders like random folders here and there to to store these files, right? And when I go and look at all those folders, they're never very structured or organized.

And what I mean by that is that there's sometimes sort of a lot of overlap between the different folders. So, like one thing might say notes, another one might say docs, but it's not clear to me what how how notes and docs should be different, right? And so, I've written a hook here where every time a tool is about to be used, so write, edit, or bash, and these are all sort of um uh commands that modify or update or create new files or folders, right?

Whenever this happens, I'm going to run what I call a MEC directory check, right? And MEC here stands for mutually exclusive and collectively exhaustive, right? And it's just like a framework for thinking about how to ensure that the different categories you create can sort of stand alone at the same level of abstraction without any overlap in between and combined together collectively, they exhaust all of the options.

Um so, what do I mean by that? Let let me show you an example, right? So, for example, um if I say something like hey in the root directory, create a folder called fruit and inside of fruit create apples, oranges, and citrus. And then inside of citrus create oranges. So, you can see if I have a parent directory called fruit and there's apples and oranges but then citrus, well, there's a little bit of conflict between orange and citrus cuz orange is part of the citrus family, right? So, that's not really me see. And what this is going to do is going to um look at we're going to called call the bash tool, I believe, to create these folders. And then but before it runs that tool, it's going to run this script and check whether this passes the me see test.

So, let's see if it works. So, here it's asking create fruit directory structure bash command pre-tool use bash requires confirmation for this command new directory apples and fruit existing sibling citrus and oranges. Does this new directory overlap with any existing sibling? I'm just going to say yes here. And then this was created, right? So, if you look where is it? Fruit. Apples, citrus, oranges, and oranges, right? But um you can see that it had created a uh a permission to to ask me, "Hey, are you sure you want to do this, right?" Let's say um create a new folder in root there called uh um misc.

Did you just notice how like the miscellaneous folder just got created right away cuz Claude determined that this doesn't uh create any concerns in terms of overlap or duplicates or whatever, right? I I want to show you what this check actually does. Um so when a new directory is about to be created, this hook uses AI to evaluate whether it violates uh this uh against sibling directories, right? Um if AI detects a violation, it denies the tool call and feeds the analysis back to Claude.

Blah blah blah blah And it basically has this prompt, right? Like so check for these specific violations. Is the new directory a subset or superset, right? Is it a duplicate? Is it an overlap? Um and then respond with valid JSON, right? And it has um you know, Claude just created this basically uh to to to make sure that whenever uh a new folder is created that, you know, it's it's thoughtful, right? It's thoughtfully done.

Um So that's what it does, and I think you know, something like this would be useful in a go-to-market setting where, you know, you might want to say, "Hey, like have we already done research for this uh for this um prospect, for example?" Or have we already, you know, explored this theme in our um in our content, right? Uh and then like you can go out and search and look it up and stuff like that. Um Two things that I want to mention.

The first is that you might have noticed that it took like not very much time uh for the miscellaneous folder to get created, but then it took a while for these other um sort of uh um more overlappy uh folders to be created. And the reason for that is it takes a while for Claude to you know, uh to reason about these folders, think about it, uh and and process the tokens, right? And so hooks are great, especially if they're deterministic code, but

anything that involves more subjective thinking on the part of an LLM.

You want to be very selective about it because it's going to add latency to your to your workflow. I actually had created a bunch of different hooks thinking that it was going to be really good for me and for my productivity, but I noticed that I was getting actually quite frustrated in terms of how long everything was taking that I just removed all the hooks. And now I only have a few hooks and even those hooks are almost all deterministic code versus passing anything to Claude cuz it just takes too long.

Um So, that's one. The second thing is the way you set up a hook you can see that this fires pre-tool use and it's going to match and you have these tool names, right? So, uh the directory check fires before every tool use and then checks if the tool name is right, edit, or bash. And if yes, then this will fire, right? But, remember that this is only really this hook is only relevant when we're trying to create new directories, right? And so, what we do then is we further filter that out to say right here.

If the commands pattern is MKDIR, which is I think it stands for make directory, so create a new directory, then proceed with this. So, there's like this pattern of filtering because Claude doesn't let you filter at this level of the pattern. So, you have to sort of have this catch-all um setting where any pre-tool use for right, edit, bash will be routed to this directory check and inside of directory check will further filter that out out or filter that down to only when uh there is a an MKDIR command because we're only interested in commands that are going to create new directories. So, if you wanted to create another hook that also looks at pre-tool use for write edit and bash, but you only want to look at markdown files for example, right? Then what you might do is pass it to another pass it to another file and and filter that down to say if the file's extension is .md, then you know, do this and this and this.

So, that's just like a a pattern that, you know, you would establish depending on how many hooks you have and the how many different paths you have, but something worth keeping in mind. Great. So, we walked through hooks and the importance of being selective, especially when you're writing hooks that um that go to Claude or Nala LM and how to design it so that you can sort of filter and route different different commands.

So, hopefully you found this helpful. As always, if you have any questions, please let me know. Thanks.