hook

ifdef::git-hook[] :see-git-hook: endif::git-hook[] ifndef::git-hook[] :see-git-hook: See linkgit:git-hook[1]. endif::git-hook[]

hook.<friendly-name>.command
The command to execute for `hook.<friendly-name>`. `<friendly-name>` is a unique name that identifies this hook. The hook events that trigger the command are configured with `hook.<friendly-name>.event`. The value can be an executable path or a shell oneliner. If more than one value is specified for the same `<friendly-name>`, only the last value parsed is used. {see-git-hook}

hook.<friendly-name>.event
The hook events that trigger `hook.<friendly-name>`. The value is the name of a hook event, like "pre-commit" or "update". (See linkgit:githooks[5] for a complete list of hook events.) On the specified event, the associated `hook.<friendly-name>.command` is executed. This is a multi-valued key. To run `hook.<friendly-name>` on multiple events, specify the key more than once. An empty value resets the list of events, clearing any previously defined events for `hook.<friendly-name>`. {see-git-hook}

The `<friendly-name>` must not be the same as a known hook event name (e.g. do not use `hook.pre-commit.event`). Using a known event name as a friendly-name is a fatal error because it creates an ambiguity with `hook.<event>.enabled` and `hook.<event>.jobs`. For unknown event names, a warning is issued when `<friendly-name>` matches the event value.

hook.<friendly-name>.enabled
Whether the hook `hook.<friendly-name>` is enabled. Defaults to `true`. Set to `false` to disable the hook without removing its configuration. This is particularly useful when a hook is defined in a system or global config file and needs to be disabled for a specific repository. {see-git-hook}

hook.<friendly-name>.parallel
Whether the hook `hook.<friendly-name>` may run in parallel with other hooks for the same event. Defaults to `false`. Set to `true` only when the hook script is safe to run concurrently with other hooks for the same event. If any hook for an event does not have this set to `true`, all hooks for that event run sequentially regardless of `hook.jobs`. Only configured (named) hooks need to declare this. Traditional hooks found in the hooks directory do not need to, and run in parallel when the effective job count is greater than 1. {see-git-hook}

hook.<event>.enabled
Switch to enable or disable all hooks for the `<event>` hook event. When set to `false`, no hooks fire for that event, regardless of any per-hook `hook.<friendly-name>.enabled` settings. Defaults to `true`. {see-git-hook}

Note on naming: `<event>` must be the event name (e.g. `pre-commit`), not a hook friendly-name. Since using a known event name as a friendly-name is disallowed (see `hook.<friendly-name>.event` above), there is no ambiguity between event-level and per-hook `.enabled` settings for known events. For unknown events, if a friendly-name matches the event name despite the warning, `.enabled` is treated as per-hook only.

hook.<event>.jobs
Specifies how many hooks can be run simultaneously for the `<event>` hook event (e.g. `hook.post-receive.jobs = 4`). Overrides `hook.jobs` for this specific event. The same parallelism restrictions apply: this setting has no effect unless all configured hooks for the event have `hook.<friendly-name>.parallel` set to `true`. Set to `-1` to use the number of available CPU cores. Must be a positive integer or `-1`; zero is rejected with a warning. {see-git-hook}

Note on naming: although this key resembles `hook.<friendly-name>.*` (a per-hook setting), `<event>` must be the event name, not a hook friendly name. The key component is stored literally and looked up by event name at runtime with no translation between the two namespaces. A key like `hook.my-hook.jobs` is stored under `"my-hook"` but the lookup at runtime uses the event name (e.g. `"post-receive"`), so `hook.my-hook.jobs` is silently ignored even when `my-hook` is registered for that event. Use `hook.post-receive.jobs` or any other valid event name when setting `hook.<event>.jobs`.

hook.jobs
Specifies how many hooks can be run simultaneously during parallelized hook execution. If unspecified, defaults to 1 (serial execution). Set to `-1` to use the number of available CPU cores. Can be overridden on a per-event basis with `hook.<event>.jobs`. Some hooks always run sequentially regardless of this setting because they operate on shared data and cannot safely be parallelized:

`applypatch-msg`;; `prepare-commit-msg`;; `commit-msg`;; Receive a commit message file and may rewrite it in place. `pre-commit`;; `post-checkout`;; `push-to-checkout`;; `post-commit`;; Access the working tree, index, or repository state.

This setting has no effect unless all configured hooks for the event have `hook.<friendly-name>.parallel` set to `true`.

For `pre-push` hooks, which normally keep stdout and stderr separate, setting this to a value greater than 1 (or passing `-j`) will merge stdout into stderr to allow correct de-interleaving of parallel output.

□□□ #2
□ □□□ □□□ 2026-10-31 20:07:17 UTC
□ □□□ □□□ 2026-10-31 20:07:18 UTC