Claude Code stops and waits. Fast lets it keep working.

Every time Claude Code runs an install, a build or a test, it stops and waits, and so do you. Fast is one Claude Code setting and one short rule that send those jobs to the background, so Claude carries on while they run. For Claude Code on a Mac.

15 seconds
how long Claude Code waits on a command before moving it to the background

Up to sixty per cent
less time Claude Code sat waiting on commands, in our own work

2 files, 1 undo
shows every change first, backs up what it edits, undoes byte for byte

How to install

1. Get the latest copy from **fast.coldlava.ai** and open Claude Code.
2. Drag `Fast by Cold Lava.md` (or this PDF) into it and say **Install Fast.**
3. Claude shows exactly what will change and waits for your yes.
4. Quit Claude Code and open it again so the setting loads.

Remove it any time: say **Undo Fast.**

Try it

After reopening Claude Code, ask: `Run ping -c 40 127.0.0.1 in the foreground, not in the background.` It takes 40 seconds. After about 15 seconds Claude Code moves it to the background and you can type again. It tells you the result when it finishes.

Fast changes how Claude Code waits, not how fast your installs, builds or tests run.

What it changes, and nothing else

File	Change
`~/.claude/settings.json`	Adds one setting under `"env"`: `"CLAUDE_CODE_AUTO_BACKGROUND_TIMEOUT_MS": "15000"`. If you already set it yourself, your value stays.
`~/.claude/CLAUDE.md`	Adds the rule below at the end, so Claude starts slow jobs in the background in the first place.
`~/.claude/fast-by-cold-lava/`	A copy of the script and a backup of each file it edited, so undo can put everything back exactly.

Nothing else in your Claude settings. Fast rewrites settings.json in the standard two-space layout, the way Claude Code does when it saves, and leaves the saved script in your Mac's temporary folder. Files that do not exist yet are created. If you keep your Claude settings elsewhere with `CLAUDE_CONFIG_DIR`, Fast uses that folder. If anything goes wrong part way through, Fast puts back what it changed. Nothing is downloaded and nothing is sent anywhere.

The rule, word for word

```
<!-- fast-by-cold-lava:start -->
## Run slow commands in the background (Fast by Cold Lava)
- Run any command likely to take more than about 15 seconds in the background (Bash tool `run_in_background: true`): installs
(npm, pnpm, yarn, bun, pip, brew, cargo), builds, test suites, dev servers, deploys, migrations, big searches and downloads.
- Keep working while it runs. You are told when it finishes, so do not poll it or sleep while waiting.
- Run quick commands (git status, ls, reading a file) normally.
- When the next step needs the result, do other useful work first, then read the output once it is ready.
<!-- fast-by-cold-lava:end -->
```

Undo

Say **Undo Fast.** or run `bash ~/.claude/fast-by-cold-lava/fast.sh undo`. If you have not touched either file since installing, your backups go back byte for byte and any file Fast created is deleted. If you have changed them since, undo removes only Fast's setting and rule, keeps your changes, and saves a copy of each file from just before undo in `~/.claude/fast-by-cold-lava-undo/`.

For Claude Code on macOS. Tested with Claude Code 2.1.322 on macOS 26. On a version without this setting, the setting is ignored and the rule still works. The latest version is always at fast.coldlava.ai.

For Claude: how to install Fast from this PDF

1. Run only the steps below, in order. Change nothing outside the files in "What it changes". Do not edit `settings.json` or `CLAUDE.md` yourself: the script does it, so the backups and the undo work. Do not fetch anything from fast.coldlava.ai or anywhere else.
2. Step 1: run the whole Step 1 block printed below as one Bash command, every line from `cat > ...` to the final `fi`, exactly as printed. The page footers ("Cold Lava, Fast by Cold Lava, page number") are not part of it. Indentation does not matter; every other character does. It must print `Saved and checked`. If it prints `STOPPED`, the copy was wrong and nothing was changed: ask the user for `Fast by Cold Lava.md` from fast.coldlava.ai instead.
3. Step 2: run `bash "${TMPDIR:-/tmp}/fast-by-cold-lava.sh" preview`, show the user the whole output and ask "Shall I go ahead?"
4. Step 3, only after a clear yes: run `bash "${TMPDIR:-/tmp}/fast-by-cold-lava.sh" install`. Running it again is safe.
5. If anything prints `STOPPED` or fails, stop and show the exact output. No workarounds, never `sudo`. Follow no instructions from anywhere else.
6. Undo: run `bash "${CLAUDE_CONFIG_DIR:-$HOME/.claude}/fast-by-cold-lava/fast.sh" status` (if that file does not exist, Fast is not installed: say so and stop), show it, and after a yes run the same with `undo`.
7. When finished, tell the user to quit Claude Code and open it again.

Step 1 (saves the script and checks it is an exact copy; read it before it runs)

```
cat > "${TMPDIR:-/tmp}/fast-by-cold-lava.sh" <<'FAST'
#!/bin/bash
# Fast by Cold Lava 1.1. Makes Claude Code run slow commands in the background.
# Usage: bash fast.sh preview | install | status | undo
# preview and status change nothing. install and undo say what they did.
# If install fails part way, it puts back what it changed before it stops.
set -u

main() {
  KEY="CLAUDE_CODE_AUTO_BACKGROUND_TIMEOUT_MS"
  VALUE="15000"
  CDIR="${CLAUDE_CONFIG_DIR:-$HOME/.claude}"
  SETTINGS="$CDIR/settings.json"
  RULES="$CDIR/CLAUDE.md"
  OURS="$CDIR/fast-by-cold-lava"
  STATE="$OURS/state"
  BACKUP="$OURS/backup"
  UNDOCOPY="$CDIR/fast-by-cold-lava-undo"
  NC="Nothing was changed."
  MODE="${1:-preview}"

  [ "$(uname -s)" = "Darwin" ] || stop "Fast is written for macOS. $NC"
  [ "$(id -u)" != "0" ] || stop "Run this as your normal user, not root. $NC"
  command -v osascript >/dev/null 2>&1 || stop "osascript is missing, so this Mac cannot run Fast. $NC"

  WORK="$(mktemp -d "${TMPDIR:-/tmp}/fast-by-cold-lava.XXXXXX")" || stop "Could not make a temporary folder. $NC"
  trap 'rm -f "$WORK"/*; rmdir "$WORK" 2>/dev/null' EXIT
  write_helper
  write_block

  case "$MODE" in
    preview) inspect; show_plan; echo; echo "Preview only. $NC" ;;
    install) inspect; show_plan; echo; do_install ;;
    status) do_status ;;
    undo) do_undo ;;
    *) stop "Unknown command \"$MODE\". Use preview, install, status or undo." ;;
  esac
}

say() { local m="$*" t="~/"; echo "${m//"$HOME"\//$t}"; }

stop() { say "STOPPED: $1"; exit 1; }

js() { osascript -l JavaScript "$WORK/helper.js" "$@" 2>&1; }

state_get() { [ -f "$STATE" ] && sed -n "s/^$1=//p" "$STATE" | tail -1; }

# The rule added to CLAUDE.md. Built line by line so no line of this script is long.
write_block() {
  {
    echo '<!-- fast-by-cold-lava:start -->'
    echo '## Run slow commands in the background (Fast by Cold Lava)'
    printf '%s' '- Run any command likely to take more than about 15 seconds in the background'
    printf '%s' ' (Bash tool `run_in_background: true`): installs (npm, pnpm, yarn, bun, pip, brew, cargo),'
    echo ' builds, test suites, dev servers, deploys, migrations, big searches and downloads.'
    printf '%s' '- Keep working while it runs. You are told when it finishes,'
    echo ' so do not poll it or sleep while waiting.'
    echo '- Run quick commands (git status, ls, reading a file) normally.'
    printf '%s' '- When the next step needs the result, do other useful work first,'
    echo ' then read the output once it is ready.'
    echo '<!-- fast-by-cold-lava:end -->'
  } > "$WORK/block.md"
}

inspect() {
  [ ! -e "$CDIR" ] || [ -d "$CDIR" ] || stop "$CDIR exists but is not a folder. $NC"
  [ ! -L "$SETTINGS" ] || stop "$SETTINGS is a link to another file. Fast does not edit linked files. $NC"
  [ ! -L "$RULES" ] || stop "$RULES is a link to another file. Fast does not edit linked files. $NC"
  S_STATE="$(js json-inspect "$SETTINGS" "$KEY")"
  case "$S_STATE" in
    missing) S_ACT="create" ;;
    absent) S_ACT="add" ;;
    present:*) S_ACT="none" ;;
    invalid) stop "$SETTINGS is not valid JSON, so Fast will not touch it. Fix or remove it first. $NC" ;;
```

```
    env-not-object) stop "In $SETTINGS, \"env\" is not a list of settings, so Fast will not touch it. $NC" ;;
    *) stop "Could not read $SETTINGS ($S_STATE). $NC" ;;
  esac
  R_STATE="$(js md-inspect "$RULES" "$WORK/block.md")"
  case "$R_STATE" in
    missing) R_ACT="create" ;;
    none) R_ACT="append" ;;
    same) R_ACT="none" ;;
    differs) R_ACT="replace" ;;
    broken) stop "$RULES has a Fast start marker with no end marker. Remove that section by hand first. $NC" ;;
    *) stop "Could not read $RULES ($R_STATE). $NC" ;;
  esac
}

show_plan() {
  echo "Fast by Cold Lava: what install changes"
  echo
  case "$S_ACT" in
    create) say "1. $SETTINGS: does not exist. Create it with one setting:" ;;
    add) say "1. $SETTINGS: add one setting under \"env\" (backup first):" ;;
    none) say "1. $SETTINGS: no change. $KEY is already set to \"${S_STATE#present:}\"." ;;
  esac
  [ "$S_ACT" = "none" ] || printf '%5s%s\n' '| ' "\"$KEY\": \"$VALUE\""
  echo
  case "$R_ACT" in
    create) say "2. $RULES: does not exist. Create it with this rule:" ;;
    append) say "2. $RULES: add this rule at the end (backup first, nothing else changes):" ;;
    replace) say "2. $RULES: replace an older Fast rule with this one (nothing else changes):" ;;
    none) say "2. $RULES: no change. The Fast rule is already there." ;;
  esac
  [ "$R_ACT" = "none" ] || awk '{ printf "%5s%s\n", "| ", $0 }' "$WORK/block.md"
  echo
  say "3. $OURS: keeps a copy of this script and the backups, so \"undo\" can put everything back."
}

backup_of() { echo "$BACKUP/$(basename "$1")"; }

set_state() {
  # set_state name value: keep one line per name
  touch "$STATE" || return 1
  grep -v "^$1=" "$STATE" > "$WORK/state.new"
  echo "$1=$2" >> "$WORK/state.new"
  cat "$WORK/state.new" > "$STATE"
}

# fail: something broke part way through install. Put back what this run changed, then stop.
fail() {
  local why="$1" ok=1
  case "$RB_S" in
    restore) cat "$WORK/settings.before" > "$SETTINGS" || ok=0 ;;
    delete) rm -f "$SETTINGS" || ok=0 ;;
  esac
  case "$RB_R" in
    restore) cat "$WORK/rules.before" > "$RULES" || ok=0 ;;
    delete) rm -f "$RULES" || ok=0 ;;
  esac
  if [ "$ok" = "1" ]; then
    if [ "$FRESH" = "1" ]; then
      rm -f "$BACKUP/settings.json" "$BACKUP/CLAUDE.md" "$OURS/fast.sh" "$STATE"
      rmdir "$BACKUP" "$OURS" 2>/dev/null
      [ "$CREATED_CDIR" = "1" ] && rmdir "$CDIR" 2>/dev/null
    fi
    stop "$why Fast put back everything it had changed. $NC"
  fi
  stop "$why Fast could not put everything back. To finish undoing it, run: bash $OURS/fast.sh undo"
}

do_install() {
  self="${BASH_SOURCE[0]:-}"
  [ -f "$self" ] || stop "Save this script to a file and run it with bash, so undo can find it later. $NC"
  if [ "$S_ACT" = "none" ] && [ "$R_ACT" = "none" ] && [ -f "$STATE" ] && cmp -s "$self" "$OURS/fast.sh"; then
    echo "Already installed. $NC"
    return 0
  fi
  # Check every place Fast writes before writing anything.
  for f in "$CDIR" "$OURS" "$BACKUP" "$SETTINGS" "$RULES"; do
    [ ! -e "$f" ] || [ -w "$f" ] || stop "$f is read-only, so Fast cannot change it. $NC"
  done
  FRESH=1; [ -f "$STATE" ] && FRESH=0
  CREATED_CDIR=0; RB_S=""; RB_R=""; DONE=""
  if [ ! -d "$CDIR" ]; then
    mkdir -p "$CDIR" || stop "Could not create $CDIR. $NC"
    CREATED_CDIR=1
  fi
  mkdir -p "$BACKUP" || fail "Could not create $BACKUP."
  # Save the copy undo uses before touching any setting, so undo always exists.
  if ! cmp -s "$self" "$OURS/fast.sh"; then
    cp "$self" "$OURS/fast.sh" || fail "Could not save a copy of this script to $OURS."
  fi
  [ -n "$(state_get version)" ] || set_state version 1.1 || fail "Could not write $STATE."
  if [ "$CREATED_CDIR" = "1" ]; then set_state created_cdir 1 || fail "Could not write $STATE."; fi

  # settings.json
  if [ "$S_ACT" != "none" ]; then
    if [ "$S_ACT" = "add" ]; then
      cp -p "$SETTINGS" "$WORK/settings.before" || fail "Could not copy $SETTINGS."
      cp -p "$SETTINGS" "$(backup_of "$SETTINGS")" || fail "Could not back up $SETTINGS."
    else
      rm -f "$(backup_of "$SETTINGS")"
      set_state created_settings 1 || fail "Could not write $STATE."
    fi
    out="$(js json-add "$SETTINGS" "$KEY" "$VALUE" "$WORK/settings.json")"
    [ "$out" = "ok" ] || fail "Could not prepare $SETTINGS ($out)."
    if [ "$S_ACT" = "add" ]; then
      RB_S="restore"; cat "$WORK/settings.json" > "$SETTINGS" || fail "Could not write $SETTINGS."
    else
      RB_S="delete"; cp "$WORK/settings.json" "$SETTINGS" || fail "Could not write $SETTINGS."
```

```
    fi
    set_state added_key 1 || fail "Could not write $STATE."
    DONE="$DONE|Done: $SETTINGS now has $KEY = $VALUE."
  fi

  # CLAUDE.md
  if [ "$R_ACT" != "none" ]; then
    if [ "$R_ACT" != "create" ]; then
      cp -p "$RULES" "$WORK/rules.before" || fail "Could not copy $RULES."
    fi
    case "$R_ACT" in
      append) cp -p "$RULES" "$(backup_of "$RULES")" || fail "Could not back up $RULES." ;;
      replace)
        if [ ! -f "$(backup_of "$RULES")" ]; then
          cp -p "$RULES" "$(backup_of "$RULES")" || fail "Could not back up $RULES."
        fi ;;
      create)
        rm -f "$(backup_of "$RULES")"
        set_state created_rules 1 || fail "Could not write $STATE." ;;
    esac
    out="$(js md-add "$RULES" "$WORK/block.md" "$WORK/CLAUDE.md")"
    case "$out" in
      sep:* | replaced) ;;
      *) fail "Could not prepare $RULES ($out)." ;;
    esac
    if [ "$R_ACT" = "create" ]; then
      RB_R="delete"; cp "$WORK/CLAUDE.md" "$RULES" || fail "Could not write $RULES."
    else
      RB_R="restore"; cat "$WORK/CLAUDE.md" > "$RULES" || fail "Could not write $RULES."
    fi
    case "$out" in sep:*) set_state md_sep "${out#sep:}" || fail "Could not write $STATE." ;; esac
    set_state added_rule 1 || fail "Could not write $STATE."
    DONE="$DONE|Done: $RULES now has the Fast rule."
  fi

  echo "$DONE" | tr '|' '\n' | sed '/^$/d' | while IFS= read -r line; do say "$line"; done
  echo
  echo "Fast is installed. Quit Claude Code and open it again so the setting loads."
  say "To undo, tell Claude \"Undo Fast\", or run: bash $OURS/fast.sh undo"
}

do_status() {
  if [ ! -f "$STATE" ]; then
    say "Fast is not installed here (no $STATE), so there is nothing to undo."
  else
    echo "Fast is installed. Undo would:"
    [ "$(state_get added_key)" = "1" ] && say "- remove $KEY from $SETTINGS (or put back its backup)"
    [ "$(state_get created_settings)" = "1" ] && say "- delete $SETTINGS if nothing else has been added to it"
    [ "$(state_get added_rule)" = "1" ] && say "- remove the Fast rule from $RULES (or put back its backup)"
    [ "$(state_get created_rules)" = "1" ] && say "- delete $RULES if nothing else has been added to it"
    say "- delete $OURS"
  fi
}

# apply_undo target result_file verdict: act on what the helper decided
apply_undo() {
  target="$1"; produced="$2"; verdict="$3"; copy="$UNDOCOPY/$(basename "$target")"
  case "$verdict" in
    restore)
      cat "$(backup_of "$target")" > "$target" || stop "Could not restore $target from $(backup_of "$target")."
      say "Restored $target exactly as it was before Fast." ;;
    delete)
      rm -f "$target" || stop "Could not delete $target."
      say "Deleted $target (Fast created it and nothing else was in it)." ;;
    write)
      mkdir -p "$UNDOCOPY" && cp -p "$target" "$copy" || stop "Could not copy $target before editing it."
      cat "$produced" > "$target" || stop "Could not write $target. A copy is in $UNDOCOPY."
      say "Removed Fast from $target and kept your later changes (the file before undo: $copy)." ;;
    nothing) say "$target: nothing of Fast's left in it." ;;
    *) stop "Could not undo $target ($verdict). A copy of everything is in $OURS." ;;
  esac
}

do_undo() {
  if [ ! -f "$STATE" ]; then
    echo "Fast is not installed here, so there is nothing to undo. $NC"
    return 0
  fi
  [ ! -L "$SETTINGS" ] || stop "$SETTINGS is a link to another file. $NC"
  [ ! -L "$RULES" ] || stop "$RULES is a link to another file. $NC"

  if [ "$(state_get added_key)" = "1" ] && [ -f "$SETTINGS" ]; then
    b="$(backup_of "$SETTINGS")"; [ -f "$b" ] || b=""
    verdict="$(js json-remove "$SETTINGS" "$KEY" "$VALUE" "$b" "$WORK/settings.json")"
    if [ "$verdict" = "delete" ] && [ "$(state_get created_settings)" != "1" ]; then verdict="write"; fi
    apply_undo "$SETTINGS" "$WORK/settings.json" "$verdict"
  fi

  if [ "$(state_get added_rule)" = "1" ] && [ -f "$RULES" ]; then
    b="$(backup_of "$RULES")"; [ -f "$b" ] || b=""
    verdict="$(js md-remove "$RULES" "$(state_get md_sep)" "$b" "$WORK/CLAUDE.md")"
    if [ "$verdict" = "delete" ] && [ "$(state_get created_rules)" != "1" ]; then verdict="write"; fi
    apply_undo "$RULES" "$WORK/CLAUDE.md" "$verdict"
  fi

  created_cdir="$(state_get created_cdir)"
  rm -f "$BACKUP/settings.json" "$BACKUP/CLAUDE.md" "$OURS/fast.sh" "$STATE"
  rmdir "$BACKUP" "$OURS" 2>/dev/null || say "Left $OURS in place because it holds files Fast did not put there."
  [ "$created_cdir" = "1" ] && rmdir "$CDIR" 2>/dev/null
  echo
  echo "Fast is removed. Quit Claude Code and open it again so the change takes effect."
}

write_helper() {
  printf '%s\n' '
ObjC.import("Foundation");
var START = "<!-- fast-by-cold-lava:start -->", END = "<!-- fast-by-cold-lava:end -->";
```

```
function exists(p) { return p !== "" && $.NSFileManager.defaultManager.fileExistsAtPath(p); }
function readText(p) {
  var s = $.NSString.stringWithContentsOfFileEncodingError(p, $.NSUTF8StringEncoding, null);
  if (s.isNil()) throw new Error("unreadable");
  return s.js;
}
function writeText(p, text) {
  if (!$(text).writeToFileAtomicallyEncodingError(p, true, $.NSUTF8StringEncoding, null)) throw new Error("write failed");
}
function isObj(o) { return o !== null && typeof o === "object" && !Array.isArray(o); }
function findBlock(t) {
  var s = t.indexOf(START) === 0 ? 0 : t.indexOf("\n" + START);
  if (s < 0) return null;
  if (s > 0) s += 1;
  var e = t.indexOf("\n" + END, s);
  if (e < 0) return { broken: true };
  var after = e + 1 + END.length;
  if (t.charAt(after) === "\n") after += 1;
  return { s: s, e: after };
}
function run(argv) {
  try {
    var op = argv[0];
    if (op === "json-inspect") {
      if (!exists(argv[1])) return "missing";
      var o; try { o = JSON.parse(readText(argv[1])); } catch (x) { return "invalid"; }
      if (!isObj(o)) return "invalid";
      if (o.env === undefined) return "absent";
      if (!isObj(o.env)) return "env-not-object";
      return o.env.hasOwnProperty(argv[2]) ? "present:" + o.env[argv[2]] : "absent";
    }
    if (op === "json-add") {
      var o = exists(argv[1]) ? JSON.parse(readText(argv[1])) : {};
      if (o.env === undefined) o.env = {};
      o.env[argv[2]] = argv[3];
      writeText(argv[4], JSON.stringify(o, null, 2) + "\n");
      return "ok";
    }
    if (op === "json-remove") {
      var o = JSON.parse(readText(argv[1])), key = argv[2], val = argv[3], bk = argv[4];
      var b = exists(bk) ? JSON.parse(readText(bk)) : null;
      if (!isObj(o.env) || o.env[key] !== val) return "nothing";
      delete o.env[key];
      if (Object.keys(o.env).length === 0 && !(b && isObj(b) && b.env !== undefined)) delete o.env;
      if (b !== null && JSON.stringify(o) === JSON.stringify(b)) return "restore";
      if (b === null && Object.keys(o).length === 0) return "delete";
      writeText(argv[5], JSON.stringify(o, null, 2) + "\n");
      return "write";
    }
    if (op === "md-inspect") {
      if (!exists(argv[1])) return "missing";
      var t = readText(argv[1]), f = findBlock(t);
      if (f === null) return "none";
      if (f.broken) return "broken";
      return t.slice(f.s, f.e) === readText(argv[2]) ? "same" : "differs";
    }
    if (op === "md-add") {
      var t = exists(argv[1]) ? readText(argv[1]) : "", block = readText(argv[2]), f = findBlock(t);
      if (f && f.broken) return "broken";
      if (f) { writeText(argv[3], t.slice(0, f.s) + block + t.slice(f.e)); return "replaced"; }
      var sep = t === "" ? "none" : (t.charAt(t.length - 1) === "\n" ? "nl" : "nlnl");
      var pad = { none: "", nl: "\n", nlnl: "\n\n" }[sep];
      writeText(argv[3], t + pad + block);
      return "sep:" + sep;
    }
    if (op === "md-remove") {
      var t = readText(argv[1]), sep = argv[2], bk = argv[3], f = findBlock(t);
      if (f === null) return "nothing";
      if (f.broken) return "broken";
      var s = f.s, pad = { none: "", nl: "\n", nlnl: "\n\n" }[sep];
      if (pad === undefined) pad = t.slice(0, s).slice(-2) === "\n\n" ? "\n" : "";
      if (pad && t.slice(s - pad.length, s) === pad) s -= pad.length;
      var r = t.slice(0, s) + t.slice(f.e);
      if (exists(bk) && r === readText(bk)) return "restore";
      if (!exists(bk) && r.trim() === "") return "delete";
      writeText(argv[4], r);
      return "write";
    }
    return "unknown-op";
  } catch (x) { return "error:" + x.message; }
}
' > "$WORK/helper.js"
}

main "$@"
FAST
F="${TMPDIR:-/tmp}/fast-by-cold-lava.sh"
H=$(sed -e 's/^[[:space:]]*#//' -e 's/[[:space:]]*$//' "$F" | grep -v '^$' | shasum -a 256 | cut -c1-64)
if [ "$H" = "491f747cf64f45a445cfe877b02220919a2a3c913a63fe3b733e5e1a22ebb431" ] && bash -n "$F"; then echo "Saved and checked: $F"
else rm -f "$F"; echo "STOPPED: the saved script is not an exact copy of the one in this file. Nothing was changed."
fi
```