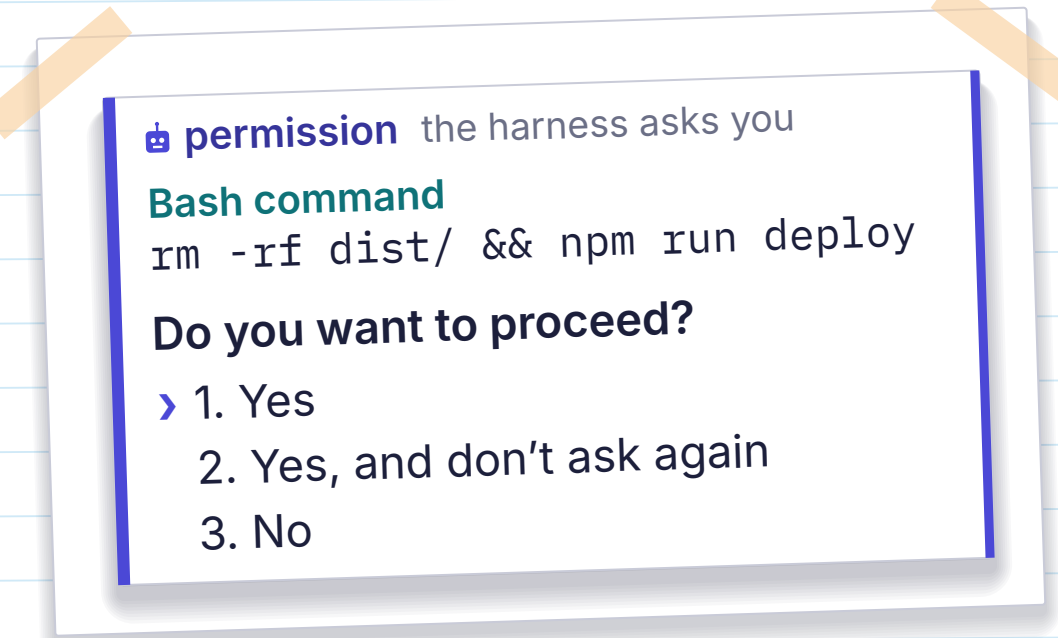
Prof. Aviv Zohar

Semester A, 2026–7

Claude Code: Permissions

Permission modes, standing rules, and how far to loosen them



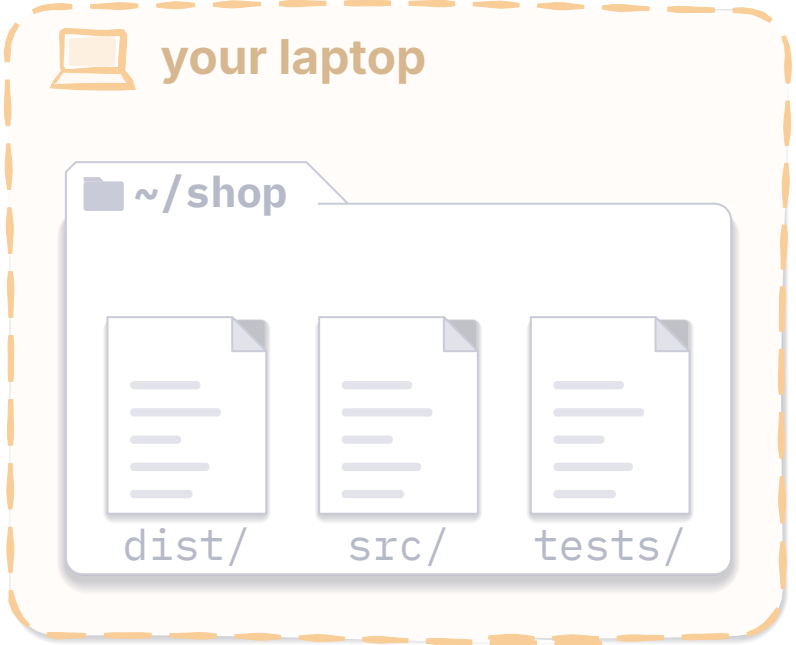
Before a command runs, the agent asks you

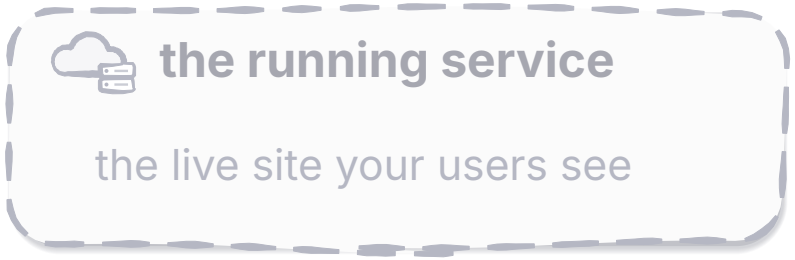
agent session ~/shop

you typed by you

Clean up the build and redeploy.

What you asked for, in words.





Before a command runs, the agent asks you

agent session ~/shop

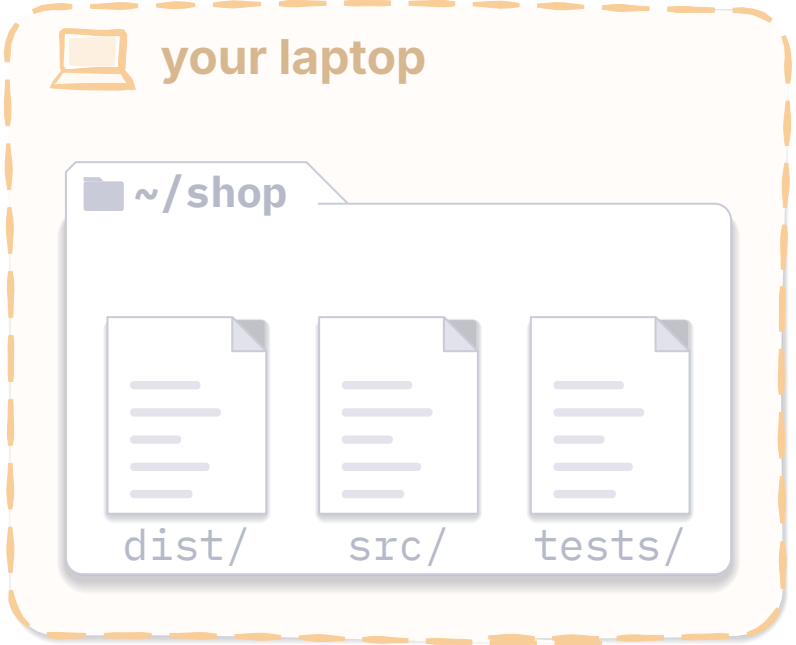
you typed by you

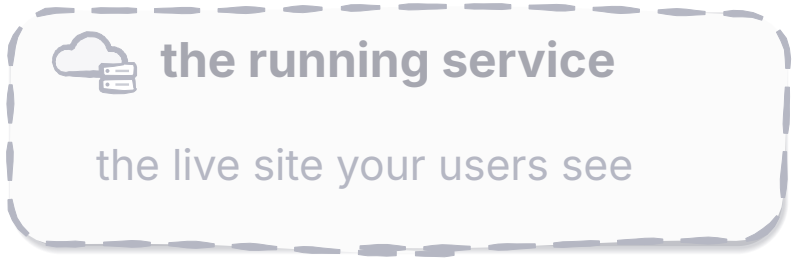
Clean up the build and redeploy.

tool call asked by the model

Bash `rm -rf dist/ && npm run deploy`

`rm -rf dist/` removes the dist folder,
then `npm run deploy` runs the deploy script.





Before a command runs, the agent asks you

agent session ~/shop

you typed by you

Clean up the build and redeploy.

tool call asked by the model

Bash `rm -rf dist/ && npm run deploy`

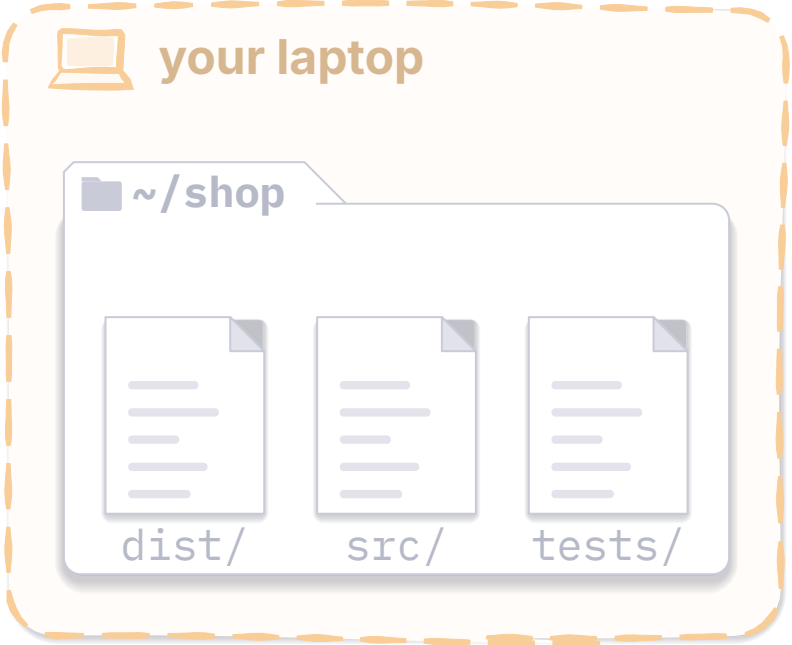
permission the harness asks you

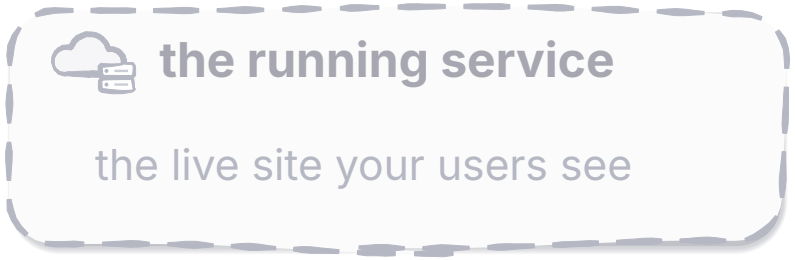
Do you want to proceed?

› 1. Yes
  2. Yes, and don't ask again for similar commands
  3. No

Yes once, a saved rule, or no.

Read what always allow covers first.





Before a command runs, the agent asks you

agent session ~/shop

you typed by you

Clean up the build and redeploy.

tool call asked by the model

```
Bash rm -rf dist/ && npm run deploy
```

permission the harness asks you

Do you want to proceed?
› 1. Yes
2. Yes, and don't ask again for similar commands
3. No

If you say yes, both happen for real.



Before a command runs, the agent asks you

agent session ~/shop

you typed by you

Clean up the build and redeploy.

tool call asked by the model

Bash `rm -rf dist/ && npm run deploy`

permission the harness asks you

Do you want to proceed?

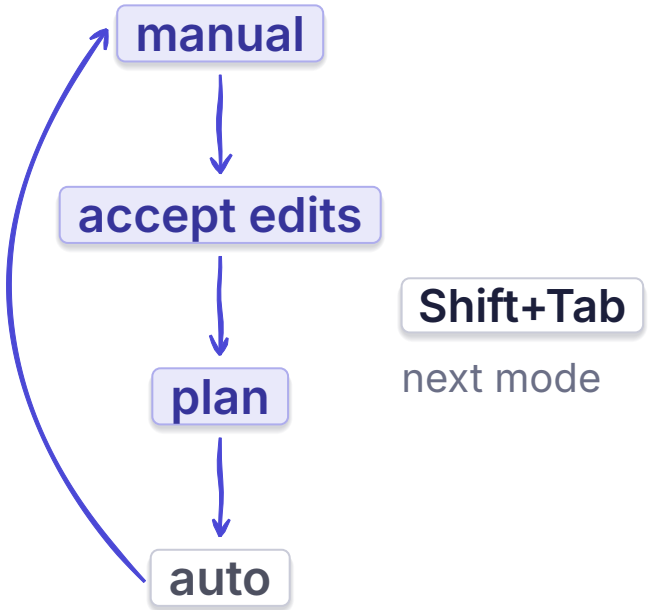
› 1. Yes
2. Yes, and don't ask again for similar commands
3. No

Read the action before you approve it. This is your chance to catch it.



The mode sets the default; rules decide recurring cases

Permission mode · this session



The session's general approval behaviour.

Standing rules · a settings file

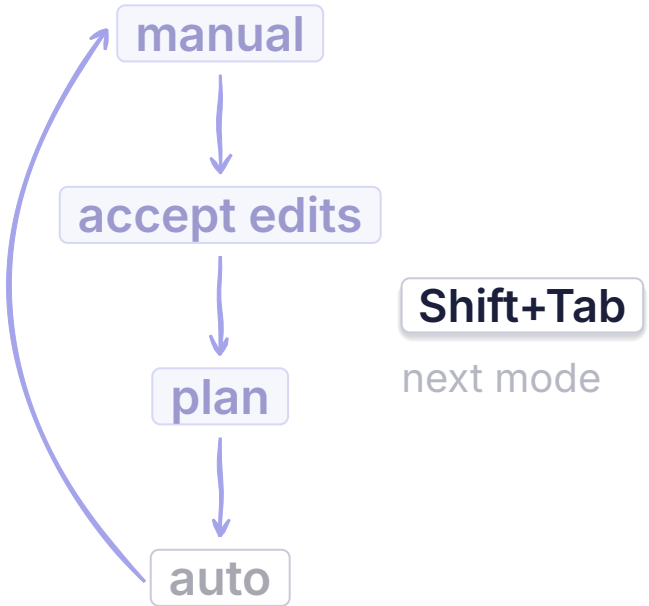
.claude/settings.json

```
{
  "permissions": {
    "allow": [
      "Bash(npm test)",
      "Bash(git status)"
    ],
    "deny": [
      "Bash(rm -rf *)",
      "Bash(git push --force)"
    ]
  }
}
```

Decided in advance, saved for every session this file covers.

The mode sets the default; rules decide recurring cases

Permission mode · this session



Standing rules · a settings file

.claude/settings.json

```
{
  "permissions": {
    "allow": [
      "Bash(npm test)",
      "Bash(git status)"
    ],
    "deny": [
      "Bash(rm -rf *)",
      "Bash(git push --force)"
    ]
  }
}
```

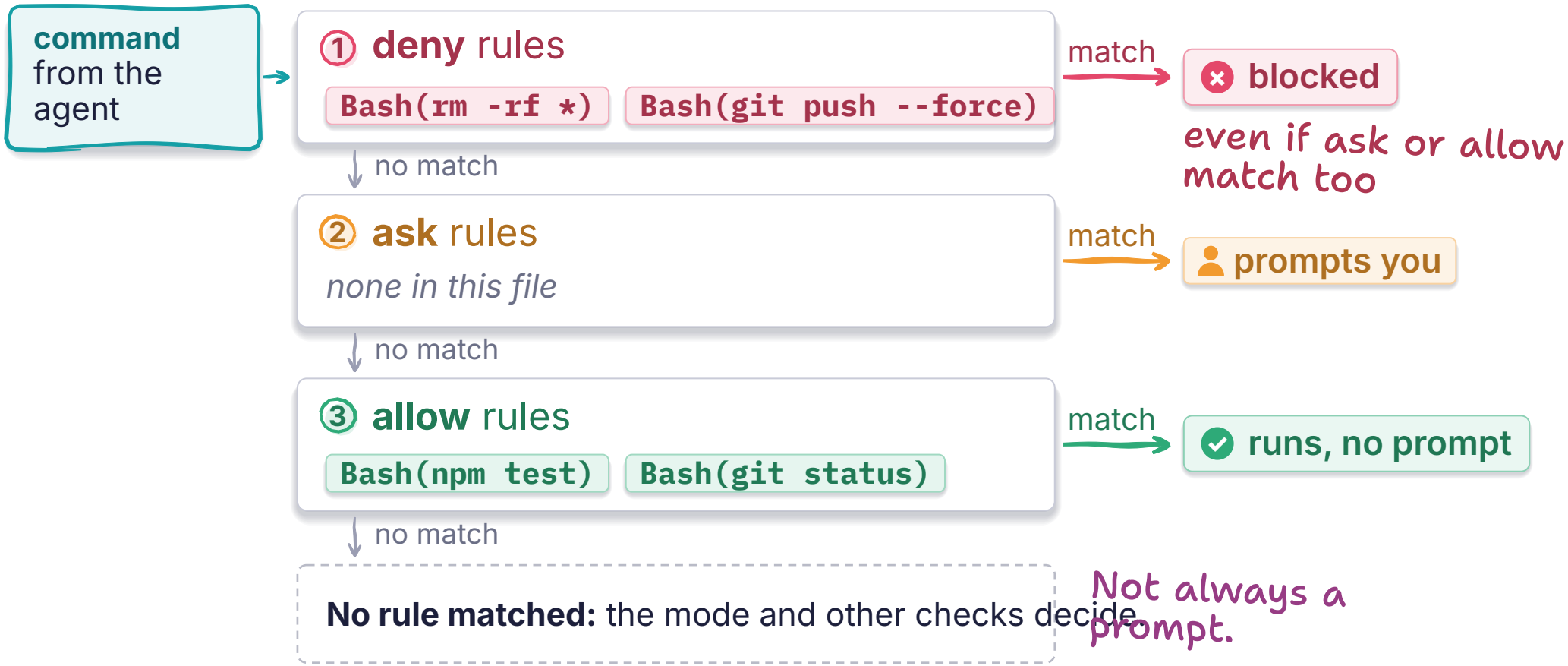
allow: runs with no prompt. Check what `npm test` runs.

deny: blocked. Only these forms, not every way to delete.

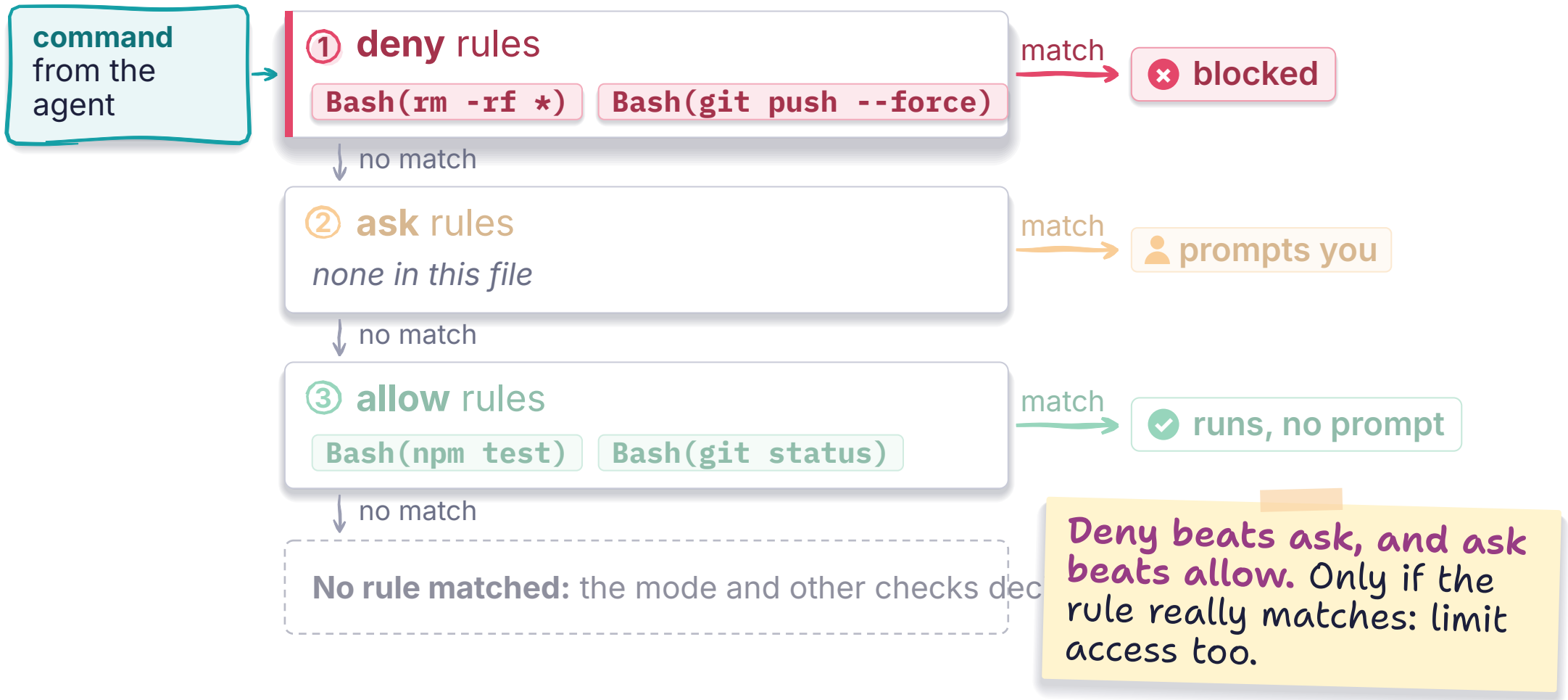
This file: one project. Rules apply where you save them.

Your user settings: **`~/.claude/settings.json`**

Deny is checked first, then ask, then allow



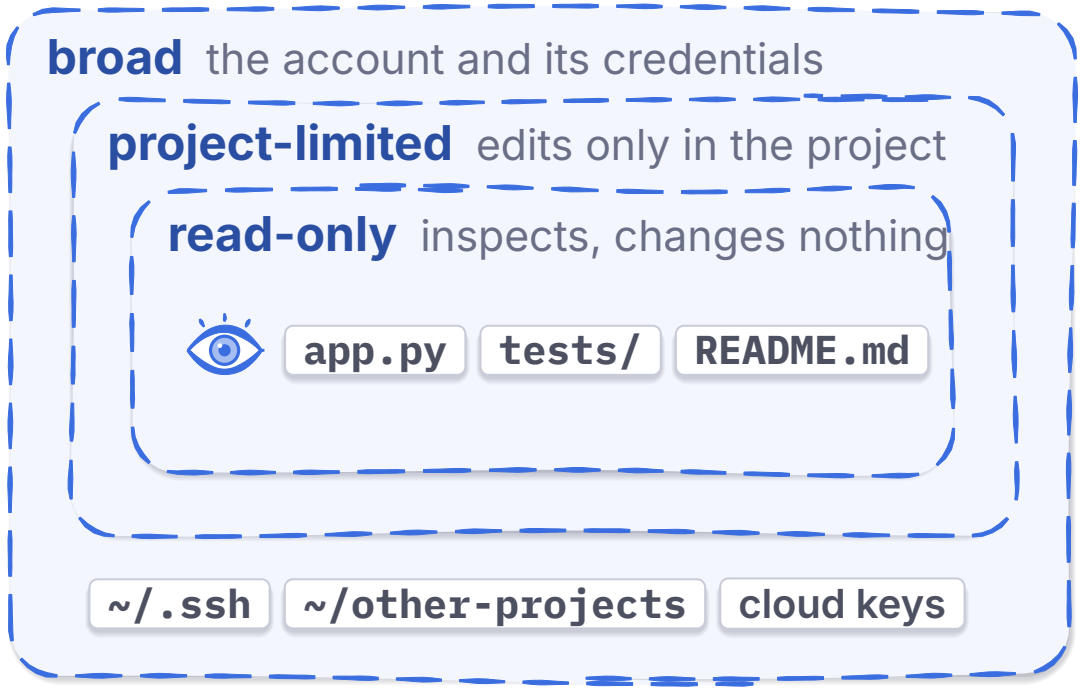
Deny is checked first, then ask, then allow



Two separate questions: reach and approval

Access

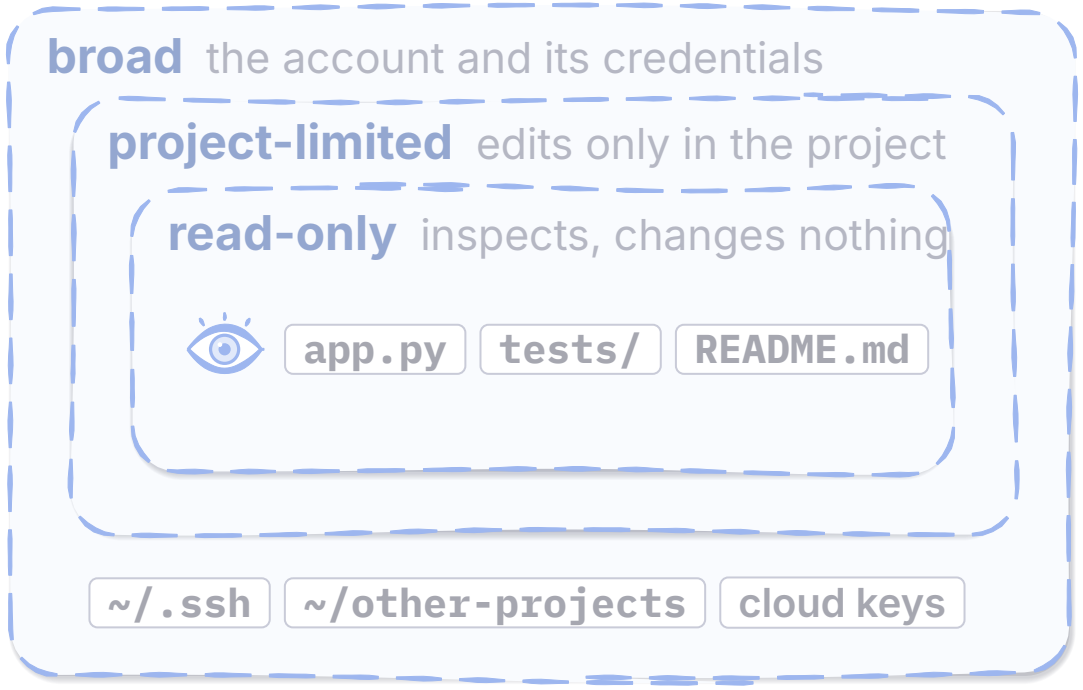
what an action can reach



Two separate questions: reach and approval

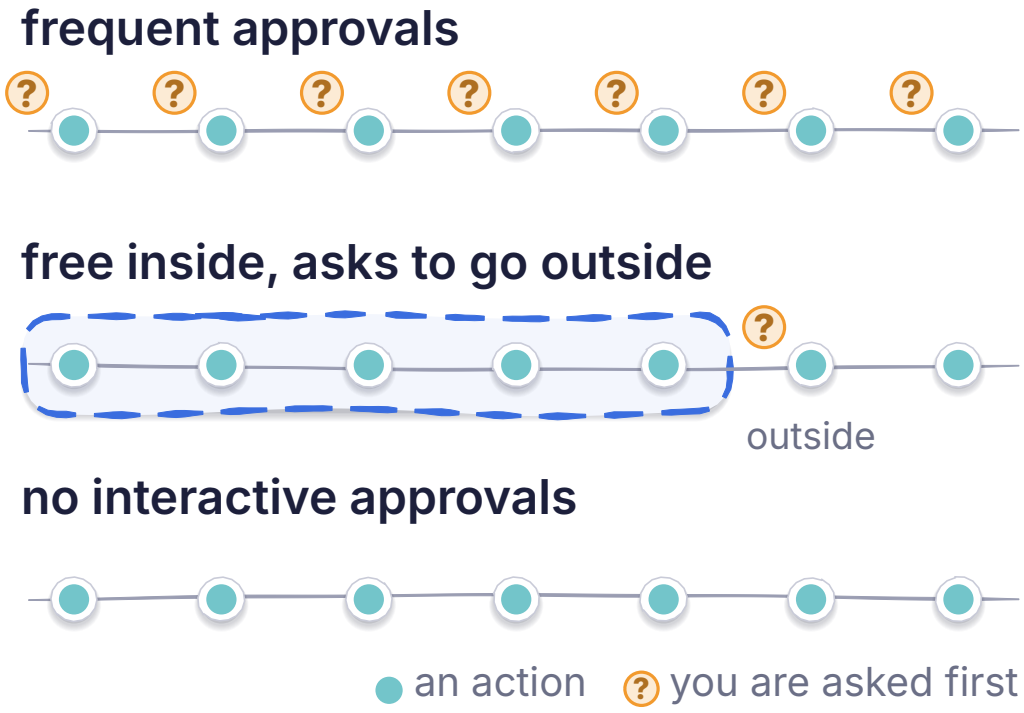
Access

what an action can reach



Oversight

when it asks first

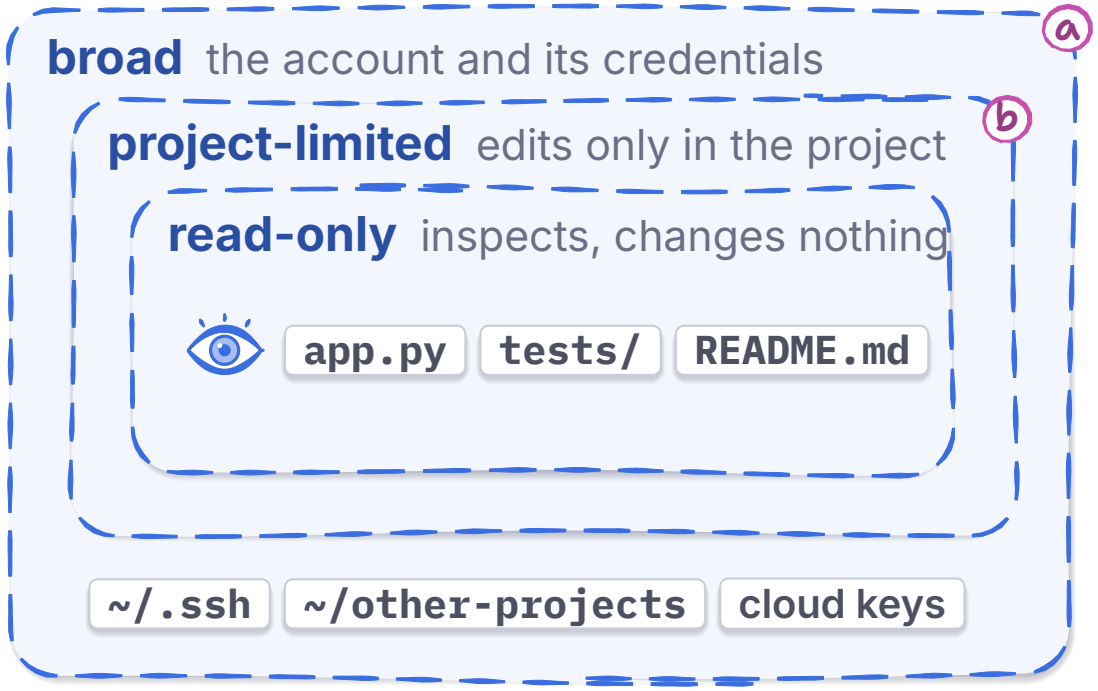


Two separate questions: reach and approval

Access

Claude Code: sandbox settings

what an action can reach

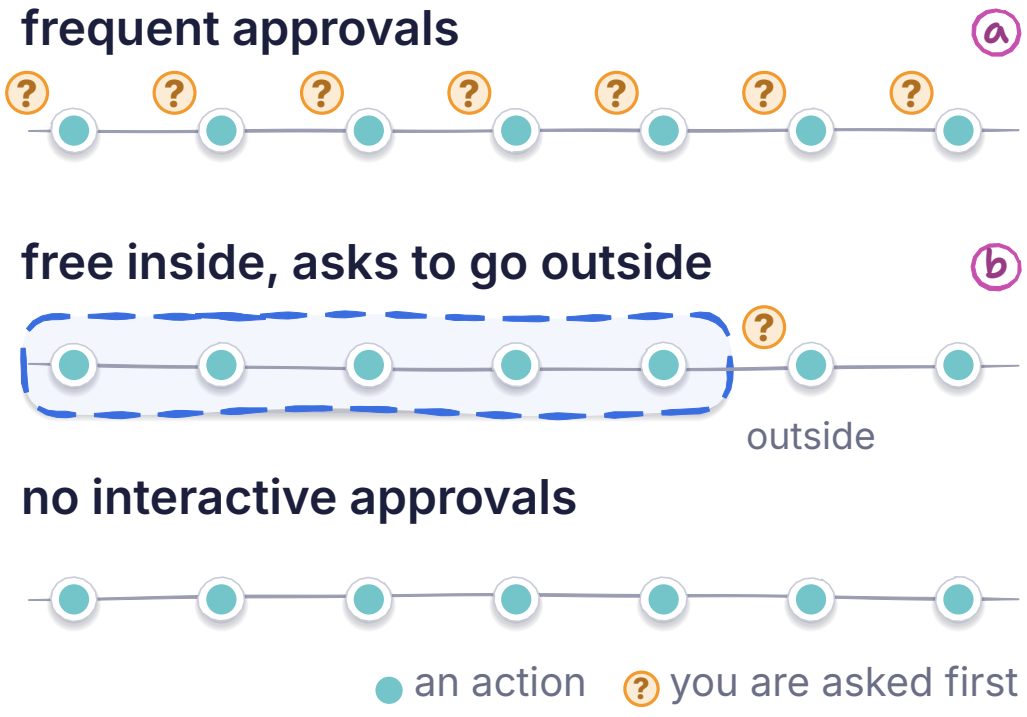


ⓐ Broad access, frequent prompts: depends on you catching a bad action.

Oversight

Claude Code: modes and rules

when it asks first



ⓑ Limited access, free inside: limits what an allowed action can affect.

One flag turns the approval prompts off

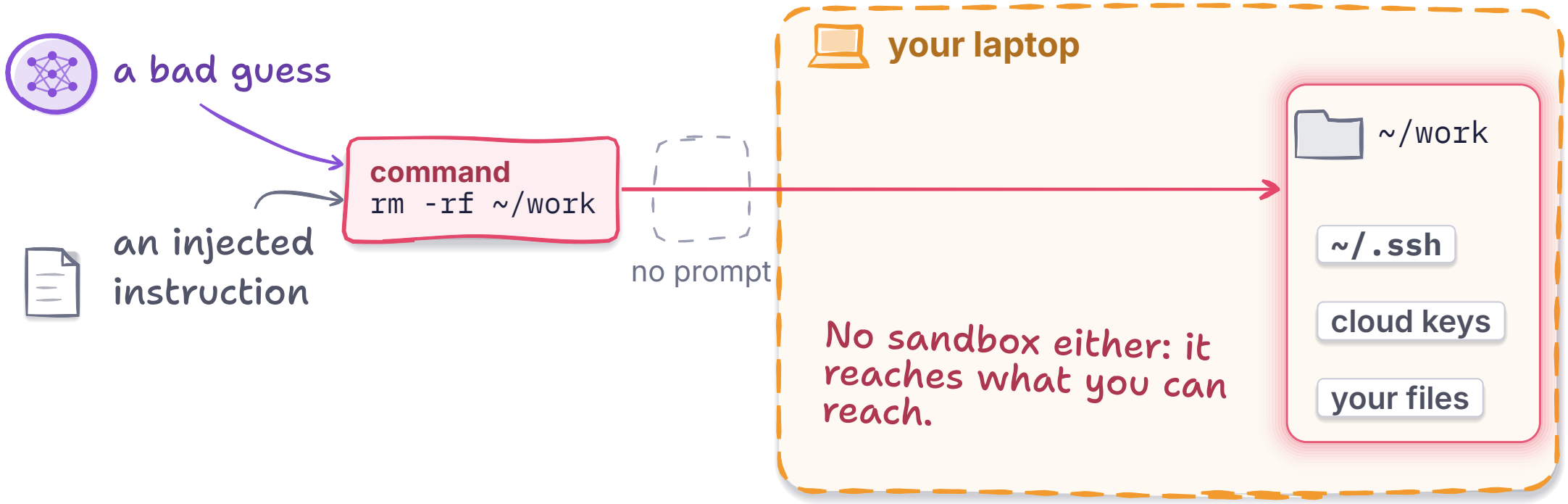
```shell
~/shop $ claude --dangerously-skip-permissions
```

bypasses the usual approval prompts and checks

One flag turns the approval prompts off

```shell
~/shop $ claude --dangerously-skip-permissions
```

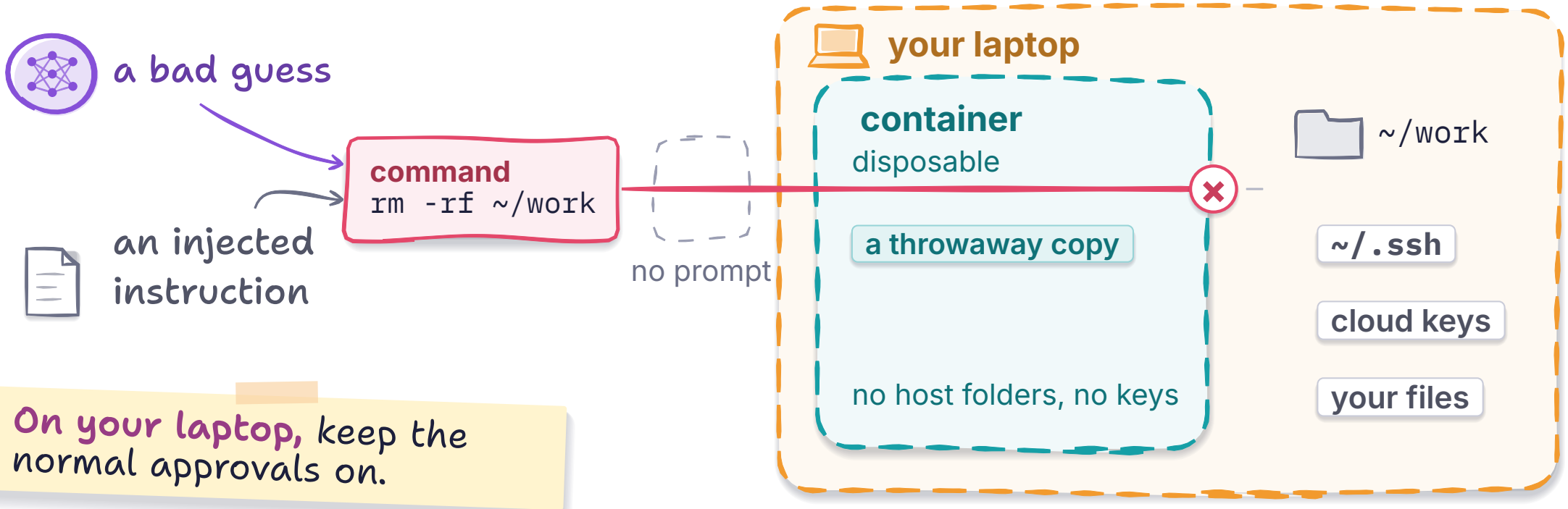
bypasses the usual approval prompts and checks



One flag turns the approval prompts off

```
shell
~/shop $ claude --dangerously-skip-permissions
```

bypasses the usual approval prompts and checks



On your laptop, keep the normal approvals on.

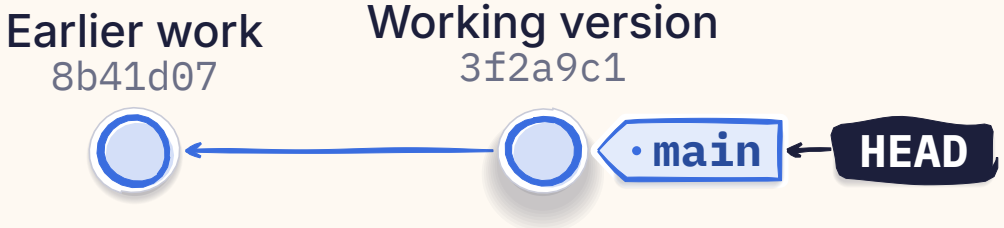
Make mistakes recoverable before you loosen the limits

your laptop

mode **manual**

shell

```
~/shop $ git add src/ tests/
~/shop $ git commit -m "Working version"
[main 3f2a9c1] Working version
```

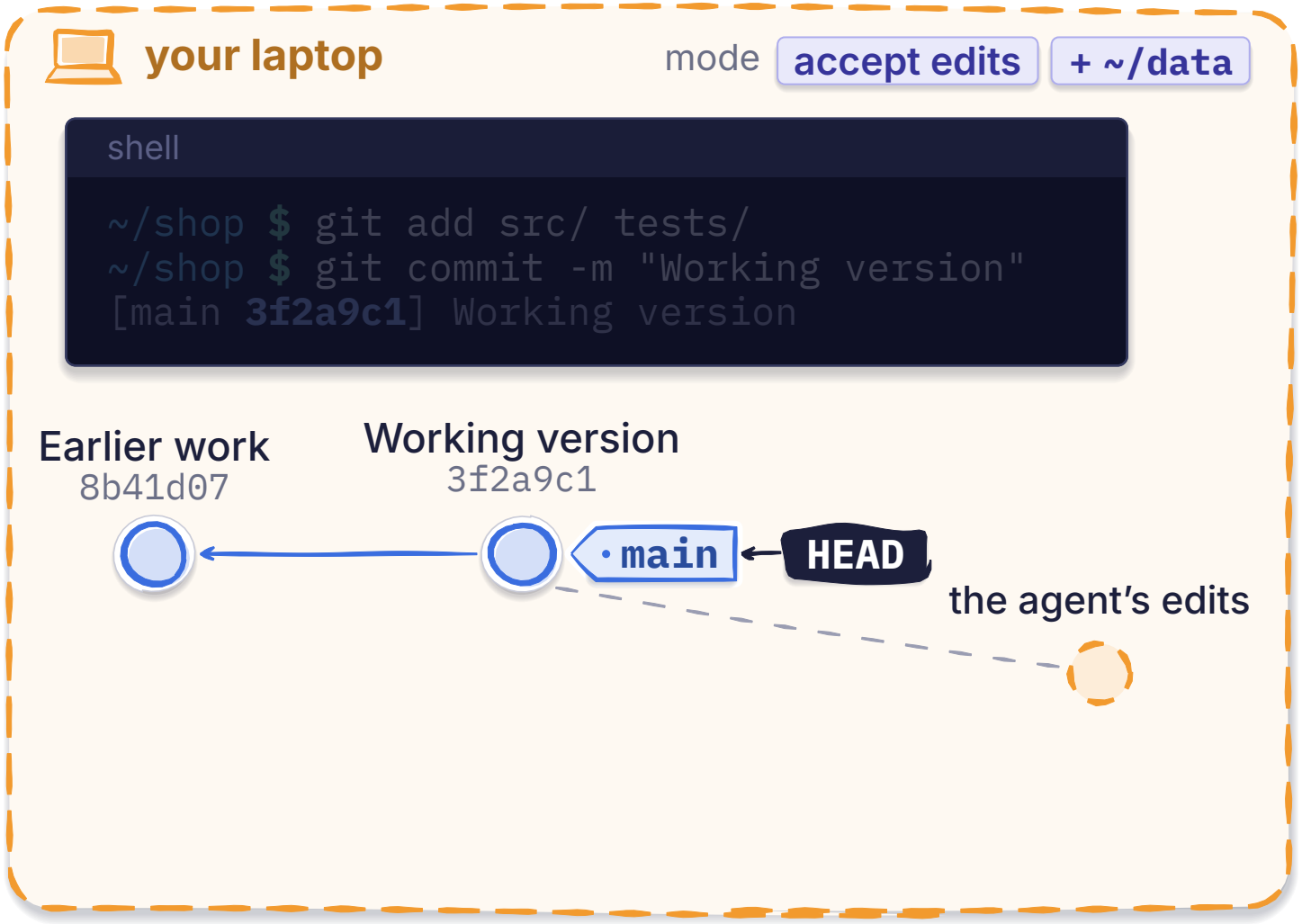


Start strict. Before you loosen, save the version you want to keep.

Select the files that belong in it.

Keep its id, to compare later changes.

Make mistakes recoverable before you loosen the limits



Then allow more routine work, where a mistake is easy to undo.

Keep reviewing the result.

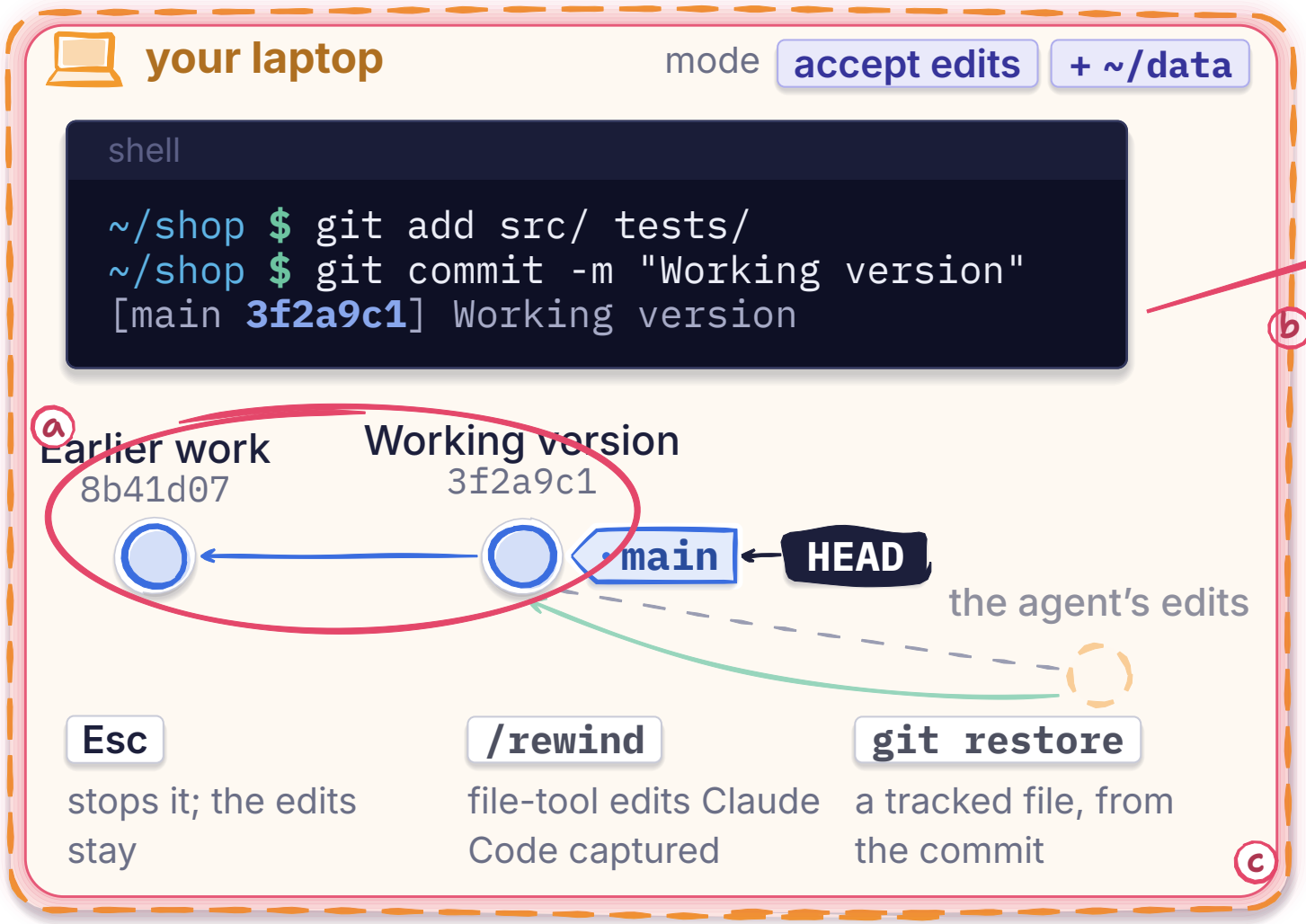
Make mistakes recoverable before you loosen the limits

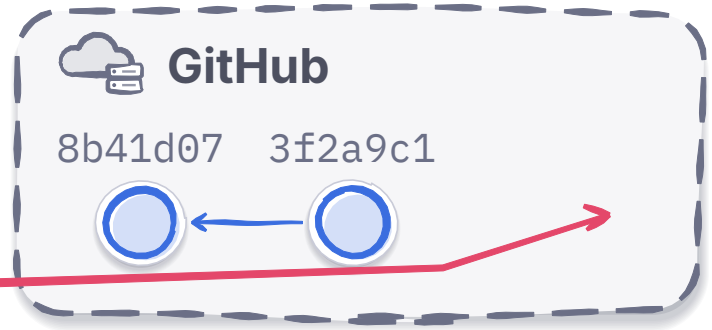


Stopping does not undo.

Anything else: decide what to keep, then ask the agent to explain the operation before it runs.

Make mistakes recoverable before you loosen the limits





Broad access can damage the recovery itself:

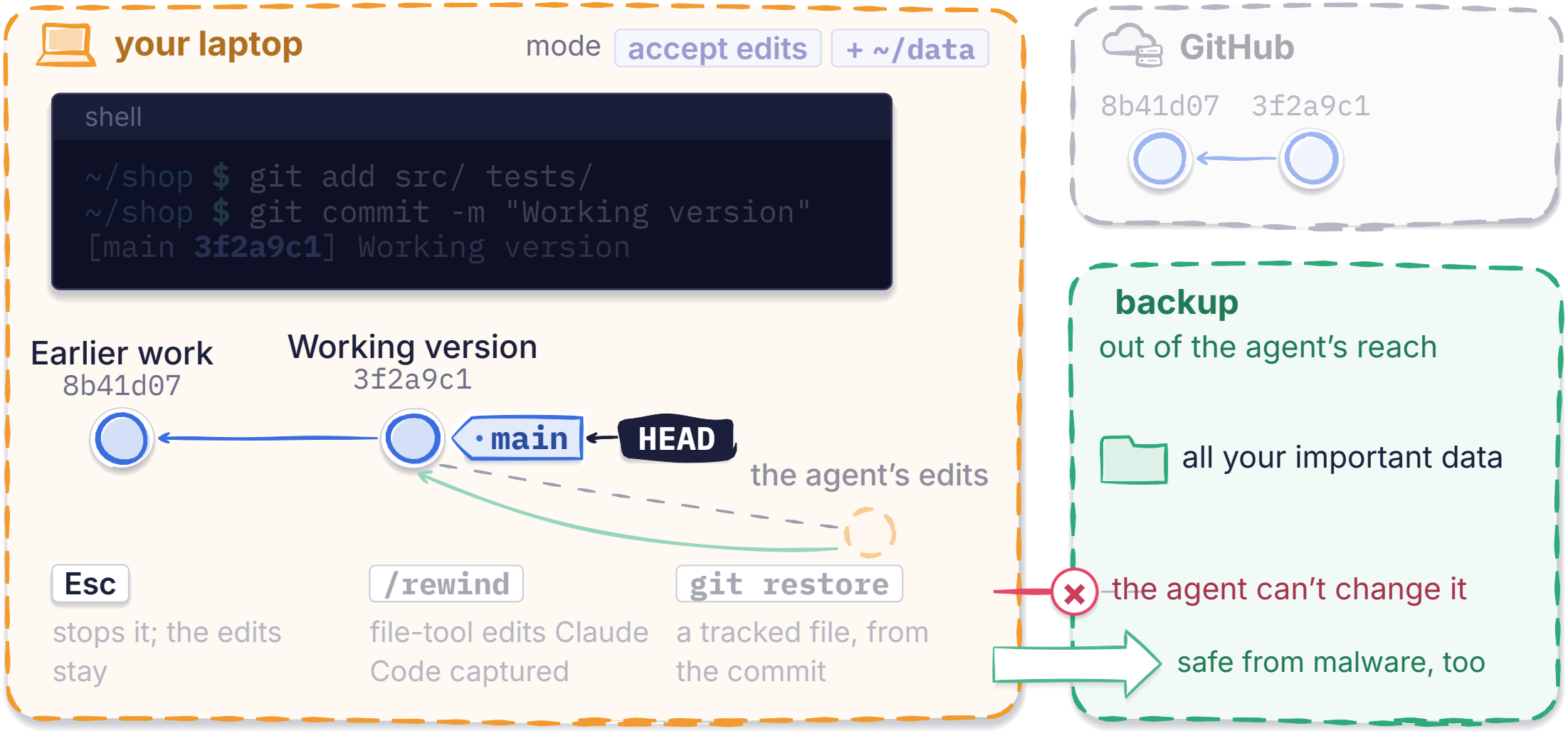
ⓐ rewrite or delete history

ⓑ push harmful changes, if its credentials allow

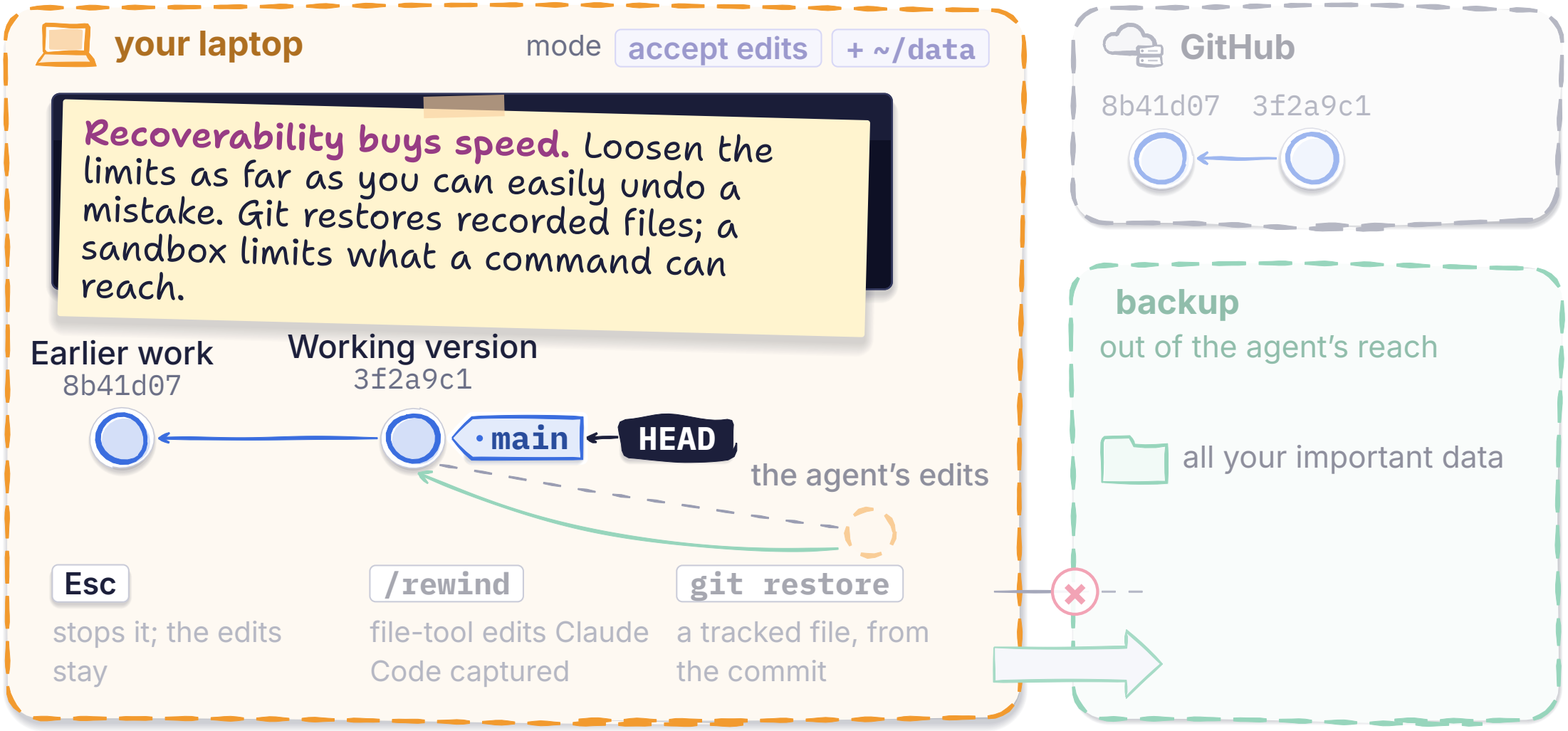
ⓒ delete the whole project

Recovery tools do not undo every shell command or outside effect.

Make mistakes recoverable before you loosen the limits



Make mistakes recoverable before you loosen the limits



How much you approve is a dial, not a switch

oversight only · access is a separate decision



review often

+ more chances to catch a mistake
− it takes your time
• you can still approve a bad command

How much you approve is a dial, not a switch



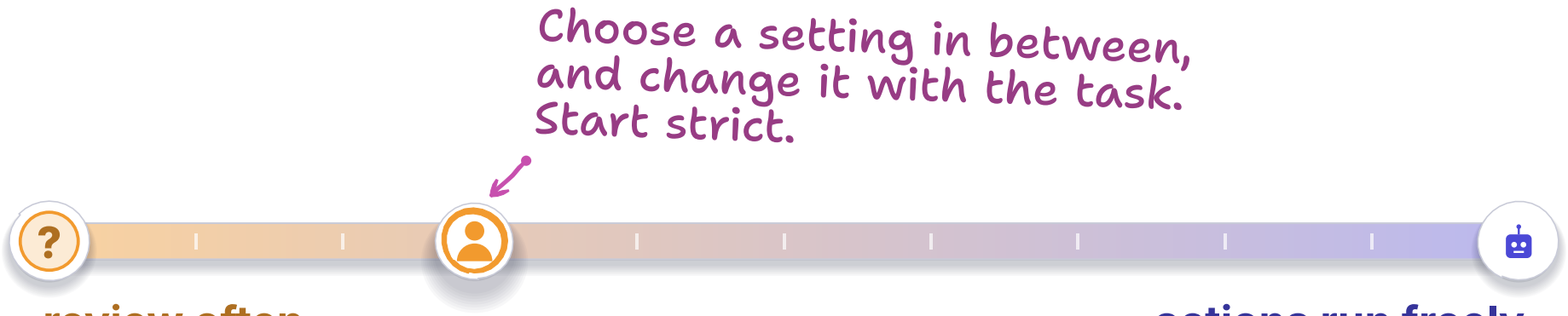
review often

+ more chances to catch a mistake
− it takes your time
• you can still approve a bad command

actions run freely

+ fewer interruptions
− a mistake can go further before you see it
• the damage still depends on its access

How much you approve is a dial, not a switch



review often

- \+ more chances to catch a mistake
- − it takes your time
- • you can still approve a bad command

actions run freely

- \+ fewer interruptions
- − a mistake can go further before you see it
- • the damage still depends on its access

How much you approve is a dial, not a switch

Loosen where recovery is easy or access is limited

Tighten for unfamiliar code and hard-to-undo actions

- a clean commit: tracked edits
- a sandbox: limits commands



review often

- \+ more chances to catch a mistake
- − it takes your time
- • you can still approve a bad command

actions run freely

- \+ fewer interruptions
- − a mistake can go further before you see it
- • the damage still depends on its access

How much you approve is a dial, not a switch

Tighten for unfamiliar code and hard-to-undo actions

Loosen where recovery is easy or access is limited

- a clean commit: tracked edits
- a sandbox: limits commands



review often

+ more chances to catch a mistake
− it takes your time
• you can still approve a bad command

actions run freely

+ fewer interruptions
− a mistake can go further before you see it
• the damage still depends on its access

To choose: what can its access damage, and what would recovery take?

a recorded file
restore it from the commit: may be simple

an exposed secret
may not be reversible

a changed remote service
may not be reversible

How much you approve is a dial, not a switch

Tighten for unfamiliar code and hard-to-undo actions

Loosen where recovery is easy or access is limited

- a clean commit: tracked edits
- a sandbox: limits commands



review often

- \+ more chances to catch a mistake
- − it takes your time
- • you can still approve a bad command

actions run freely

- \+ fewer interruptions
- − a mistake can go further before you see it
- • the damage still depends on its access

No one setting fits every task. Begin strict, then loosen for specific work you understand and can contain. Keep checking: a checkpoint helps only with what it records.

Real prompts are often long and chained

agent session ~/shop

tool call asked by the model

Bash

```
cd app && npm ci --silent 2>/dev/null && npx
tsc --noEmit -p . | grep -v TS6133 | head -40
&& rm -rf "$BUILD_DIR"/* && git stash && git
checkout -B release origin/main && git stash
pop && npm run build -- --prod > /tmp/build.log
2>&1; tail -5 /tmp/build.log
```

permission the harness asks you

Do you want to proceed?
› 1. Yes
  2. Yes, and don't ask again for similar commands
  3. No

One line, about ten steps, joined with && and |.

It looks like the hundred harmless prompts before it.

Real prompts are often long and chained

agent session ~/shop

tool call asked by the model

Bash

```
cd app && npm ci --silent 2>/dev/null && npx
tsc --noEmit -p . | grep -v TS6133 | head -40
&& rm -rf "$BUILD_DIR"/* && git stash && git
checkout -B release origin/main && git stash
pop && npm run build -- --prod > /tmp/build.log
2>&1; tail -5 /tmp/build.log
```

permission the harness asks you

Do you want to proceed?
› 1. Yes
2. Yes, and don't ask again for similar commands
3. No

`rm -rf "$BUILD_DIR"/*` deletes everything in that folder.

If `BUILD_DIR` is empty, the path becomes `/*`: everything it can reach from the root of the disk.

Approval fatigue: people stop reading

93%

of Claude Code permission prompts are approved

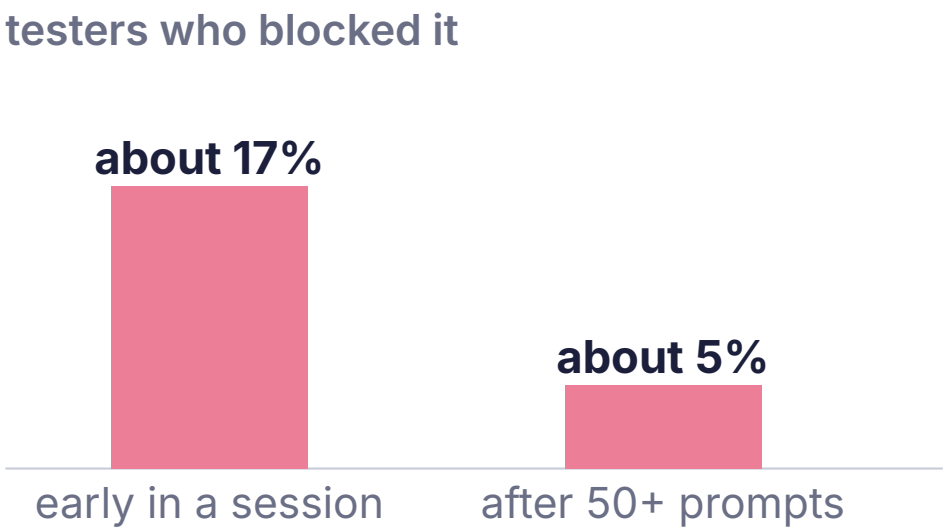
Anthropic, March 2026

13.6%

of 1,053 testers blocked a dangerous command swapped into one prompt

Anthropic study, August 2026

Almost every prompt is harmless, so a "yes" becomes a habit.



Auto mode hands the check to another model



89%

of the swapped dangerous commands blocked, in the same study

Anthropic study, August 2026

17%

of real cases where the agent went further than asked were missed

Anthropic, March 2026

A young field: hold this advice loosely

The tools, their defaults and the advice are changing fast. Keeping coding agents secure is still an open problem.

When a command is too long to check

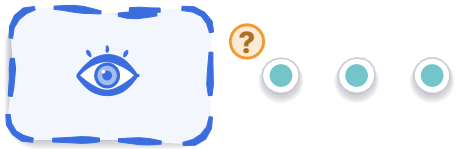
- deny it
- ask the agent to run the steps one at a time, or to explain each part

What holds up:

limit what the agent can reach

keep a way to recover

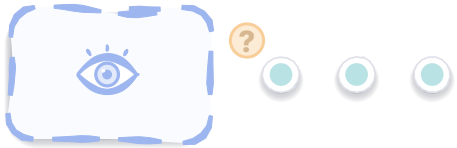
Recap



Two questions

Access: what an action can reach. **Oversight**: when it needs your approval. In Claude Code: modes and rules for approvals, sandbox settings and the environment for access.

Recap



Two questions

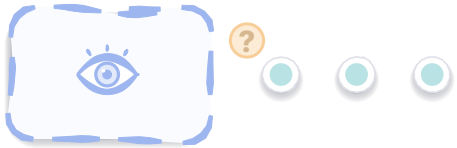
Access: what an action can reach. **Oversight**: when it needs your approval. In Claude Code: modes and rules for approvals, sandbox settings and the environment for access.



Checkpoint first

Start strict. Commit the version you want to keep, then loosen approvals only for work you can contain and recover from.

Recap



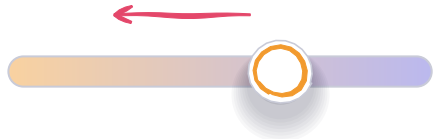
Two questions

Access: what an action can reach. **Oversight**: when it needs your approval. In Claude Code: modes and rules for approvals, sandbox settings and the environment for access.



Checkpoint first

Start strict. Commit the version you want to keep, then loosen approvals only for work you can contain and recover from.



Tighten again

For unfamiliar code, sensitive data and hard-to-undo actions. A saved version does not reverse everything a command can do.

Each new task: what can it reach, and what can it do without asking?