Usage

- Sign in
- Start, stop, and remove
  - Remove a sandbox when the agent exits
- Choose a workspace
- Reconnect and name sandboxes
- Create without attaching
  - Keep a sandbox running in the background
- Set environment variables
- Run commands inside a sandbox
- Interactive mode
- Git workspace modes
  - Clone mode
- Multiple workspaces
- Copy files between host and sandbox
- Publish ports
- What persists
- Saving a sandbox as a template
  - Save and reuse
  - List and remove templates
  - Export and import
  - Limitations
- Load a template
  - Template caching
  - Updating agents

This page describes local sandboxes. For cloud commands, file transfers, ports, and expiration, see Use cloud sandboxes.

Use this page as a command-oriented guide to day-to-day `sbx` operations. For scenario-based recommendations, see Workflow patterns. For script output and the v0.48.0 CLI migration, see CI and headless use.

Sign in

Sign in from a terminal:

```
$ sbx login
```

For scripts or CI runners where a browser isn't available, see CI and headless use.

Start, stop, and remove

The basic workflow is `run` to start, `ls` to check status, `stop` to pause, and `rm` to clean up:

```
$ sbx run claude                    # start an agent in the current directory
$ sbx ls                            # see what's running
$ sbx stop my-sandbox               # pause it
$ sbx rm my-sandbox                 # delete it entirely
```

`sbx rm` asks for confirmation before deleting a sandbox. Use `--force` to skip the prompt. This flag also permits removal when the sandbox has an active session — an open attach, SSH connection, or in-flight SFTP transfer:

```
$ sbx rm --force my-sandbox
```

If you need a clean slate, remove the sandbox and run it again:

```
$ sbx stop my-sandbox
$ sbx rm my-sandbox
$ sbx run claude
```

To remove all stopped local sandboxes, use `sbx prune`. Running sandboxes are never removed. Preview the sandboxes that would be removed, or filter out sandboxes stopped within the last week:

```
$ sbx prune --dry-run
$ sbx prune --filter until=168h
```

The `until` filter uses the time the sandbox stopped. It accepts a duration such as `168h`, an RFC 3339 timestamp, or a Unix timestamp. The older `since=<duration>` filter remains supported.

Run `sbx prune` without flags to confirm and remove all stopped sandboxes.

Remove a sandbox when the agent exits

Pass `--rm` to `sbx run` for a throwaway session. The sandbox is removed when the agent exits, without a confirmation prompt:

```
$ sbx run --rm claude
```

The sandbox is removed however the agent exits, and `sbx run` exits with the agent's exit status. If `sbx run` created the sandbox and the agent fails to start, the sandbox is also removed. If you reattach to an existing sandbox with `--rm`, that sandbox is removed only after its agent session finishes.

`--rm` needs an attached agent session to know when to remove the sandbox, so you can't combine it with `--detach` or `--detach-keys`.

Choose a workspace

`sbx run` mounts the current directory when you don't pass a workspace path. Pass a path to mount another directory instead:

```
$ sbx run claude
$ sbx run claude ~/my-project
```

The first workspace path is the primary workspace. The agent starts there, and `sbx exec` uses it as the default working directory. The host directory is mounted at the same absolute path inside the sandbox. When you don't pass a path to `sbx run`, the current directory is the primary workspace.

Starting with `sbx` version 0.42.0, workspace paths are optional for `sbx create`. Omit them to create a mountless sandbox without a host workspace bind mount, then attach to the sandbox by name:

```
$ sbx create --name scratch claude
$ sbx run --name scratch
```

In a mountless sandbox, the agent starts in the template image's working directory. Docker-provided templates use `/home/agent/workspace`. Files there persist across stops and restarts but are deleted when you remove the sandbox. Assign the sandbox a name so you can reconnect to it, and use `sbx cp` to transfer files between the sandbox and the host.

Reconnect and name sandboxes

Sandboxes persist after the agent exits. Running the same workspace path again reconnects to the existing sandbox rather than creating another sandbox:

```
$ sbx run claude ~/my-project  # creates sandbox
$ sbx run claude ~/my-project  # reconnects to same sandbox
```

Use `--name` to give a sandbox an explicit identity:

```
$ sbx run --name my-project claude
```

Once a named sandbox exists, reattach from any working directory with `sbx run --name`. You can omit the agent name when reattaching:

```
$ sbx run --name my-project        # re-attaches from anywhere
$ sbx run claude --name my-project # same, with agent confirmed
```

To run multiple sandboxes against the same workspace, give each a distinct name:

```
$ sbx run claude --name feature ~/my-project
$ sbx run claude --name spike ~/my-project
```

Create without attaching

`sbx run` creates the sandbox and attaches you to the agent. To create a sandbox with the current directory mounted, without attaching to it:

```
$ sbx create --name my-project claude .
```

Omit the path to create a mountless sandbox instead. Attach later with `sbx run --name`:

```
$ sbx create --name scratch claude
$ sbx run --name scratch
```

After `sbx create` finishes, the local sandbox stops automatically when no sessions keep it running. Its files and configuration persist. Running `sbx run --name <sandbox-name>` starts it again and attaches you to the agent.

Keep a sandbox running in the background

To keep a sandbox running after every session ends, for example to serve an application on a published port, start it with `sbx run --detach` (`-d`). The command starts the sandbox, prints its ID, and returns without opening an agent session:

```
$ sbx run -d --name my-project claude .
```

A detached sandbox keeps running until you stop it with `sbx stop` or remove it with `sbx rm`. You can attach to it with `sbx run --name` or run commands with `sbx exec`, and it keeps running after those sessions end.

Running `sbx run -d --name <sandbox-name>` against an existing sandbox, such as one created with `sbx create`, switches it to detached mode permanently. To return to the default behavior, remove the sandbox and create it again.

Set environment variables

> [!NOTE] The `-e`/`--env` and `--env-file` flags require `sbx` version 0.39.0 or later.

Pass `-e` or `--env` to `sbx run` or `sbx create` to set an environment variable in the sandbox:

```
$ sbx run -e LOG_LEVEL=debug claude
```

Specify a variable name without a value to copy its value from the host environment:

```
$ export API_URL=https://api.example.com
$ sbx run -e API_URL claude
```

To load multiple variables, pass one or more environment files:

```
$ sbx create --name my-project --env-file .env.sandbox claude .
```

The flags follow `docker run` precedence rules. Values passed with `-e` override values from environment files. When you pass multiple environment files, a value in a later file overrides the same variable in an earlier file.

When either command creates a sandbox, the variables are stored with the sandbox. They are also available to the agent session started by `sbx run`. When `sbx run` re-attaches to an existing sandbox, the variables apply to that agent session without changing the sandbox's stored environment. To set variables for one command instead, use `sbx exec -e` or `sbx exec --env-file`.

To persist a variable across future sessions of an existing sandbox, append an export to `/etc/sandbox-persistent.sh`:

```
$ sbx exec <sandbox-name> bash -c "echo 'export INTERNAL_API_URL=https://api.example.com' >>
/etc/sandbox-persistent.sh"
```

The `bash -c` wrapper ensures the `>>` redirect runs inside the sandbox instead of on your host. The file is sourced when Bash starts inside the sandbox, including for interactive sessions and agents started with `sbx run`. A command passed directly to `sbx exec` doesn't start a shell. Wrap that command in `bash -c` if it needs variables from the persistent environment file.

A variable added to the file only takes effect for sessions and agents started afterward. Restart a running agent, or stop and start the sandbox, to pick up the new value.

Environment variables are readable by processes inside the sandbox. For API keys and other credentials, use `sbx secret set` for a supported service or the experimental `sbx secret set-custom` for a credential sent to known hosts. The host-side proxy can then inject the real value without exposing it to the agent.

Run commands inside a sandbox

To get a shell inside a running sandbox, use `sbx exec`:

```
$ sbx exec -it <sandbox-name> bash
```

Without `--workdir`, the command starts in the sandbox's primary workspace. In a mountless sandbox, it starts in the container image's working directory.

`sbx exec` runs commands in the foreground. Detached execution (`-d` or `--detach`) isn't supported.

Interactive mode

Running `sbx` with no subcommands opens an interactive terminal dashboard:

```
$ sbx
```

The dashboard shows all your sandboxes as cards with live status, CPU, and memory usage. From here you can:

- **Create** a sandbox (`c`).
- **Start or stop** a sandbox (`s`).
- **Attach** to an agent session (`Enter`), same as `sbx run`.
- **Open a shell** inside the sandbox (`x`), same as `sbx exec`.
- **Remove** a sandbox (`r`).

The dashboard also includes a network governance panel where you can monitor outbound connections made by your sandboxes and manage network rules. Use `tab` to switch between the sandboxes panel and the network panel.

From the network panel you can browse connection logs, allow or block specific hosts, and add custom network rules. Press `?` to see all keyboard shortcuts.

At selection and text prompts, Enter accepts the default. Ctrl-D cancels the prompt. Declining a required confirmation returns a non-zero exit status.

Git workspace modes

When your primary workspace is a Git repository, choose how the sandbox receives it when you create the sandbox:

- Direct mode is the default for `sbx run`. It also applies when you pass a workspace path to `sbx create`. The agent has read-write access to your working tree, and changes appear on your host immediately.
- Clone mode uses `--clone`. The agent edits a separate Git clone inside the sandbox. Its changes stay there until you fetch them or the agent pushes them. Your host repository is also available at `/run/sandbox/source`, but only with read access.

For guidance on branch strategy, fetching work from a sandbox, and parallel agent workflows, see Git workflows. For the security model behind each mode, see Workspace isolation.

Clone mode

To create a clone-mode sandbox, pass `--clone` when you run or create it:

```
$ sbx run --clone claude .
```

You can also create the sandbox in the background and attach later:

```
$ sbx create --clone --name my-sandbox claude .
$ sbx run --name my-sandbox
```

Clone mode has a few create-time constraints:

- Clone mode is fixed at create time. To switch an existing sandbox to clone mode, remove it and recreate it with `sbx create --clone`.
- The clone follows whichever ref your host repository has checked out at create time. No branch is created automatically.
- The primary workspace must be a Git repository. Omit `--clone` for non-Git workspaces.
- Clone mode is rejected from inside a Git worktree other than the main one. The read-only bind mount can't resolve the worktree's `.git` pointer file. Run `sbx create --clone <agent> .` from the main repository checkout instead.
- Removing a clone-mode sandbox drops the in-sandbox clone. Fetch or push any commits you want to keep before you remove it.

Multiple workspaces

You can mount extra directories into a sandbox alongside the main workspace. The first path is the primary workspace — the agent starts here, and the sandbox's in-container Git clone is populated from this directory if you use `--clone`. Extra workspaces are always mounted directly.

Each workspace path appears inside the sandbox at the same absolute path as on the host. Append `:ro` to mount an extra workspace read-only — useful for reference material or shared libraries the agent shouldn't modify:

```
$ sbx run claude ~/project-a ~/shared-libs:ro ~/docs:ro
```

You can also run separate projects side-by-side. Remove unused sandboxes when you're done to reclaim disk space:

```
$ sbx run claude ~/project-a
$ sbx run claude ~/project-b
$ sbx rm <sandbox-name>       # when finished
```

Copy files between host and sandbox

Use `sbx cp` to copy files or directories between your host and a sandbox. This is useful for one-off files that aren't part of a mounted workspace, such as generated output, logs, or setup files. The sandbox path must be absolute. `sbx cp` doesn't resolve relative paths such as `.` against the sandbox's default working directory.

For example, copy files to or from the default working directory used by a Docker-provided agent template:

```
$ sbx cp ./config.json my-sandbox:/home/agent/workspace/
$ sbx cp my-sandbox:/home/agent/workspace/output.log ./
$ sbx cp ./src/ my-sandbox:/home/agent/workspace/src
```

One side of the copy must use `SANDBOX:PATH`. Copying directly between two sandboxes isn't supported.

Publish ports

Sandboxes are network-isolated — your browser or local tools can't reach a server running inside one by default. A port mapping of `8080:3000` publishes sandbox port 3000 on host port 8080.

If you know which ports you need, publish them when you create the sandbox:

```
$ sbx run --publish 8080:3000 --name my-sandbox claude
```

For an existing sandbox, use `sbx ports` to forward traffic from your host. Publishing a port on a stopped local sandbox starts it first:

```
$ sbx ports my-sandbox --publish 8080:3000
$ open http://localhost:8080
```

To let the OS pick a free host port instead of choosing one yourself, specify only the sandbox port. Then use `sbx ports` to check which host port was assigned:

```
$ sbx ports my-sandbox --publish 3000
$ sbx ports my-sandbox
```

`sbx ls` shows active port mappings alongside each sandbox. `sbx ports` lists them in detail. The `CREATED` column shows the sandbox age, and the `STATUS` column identifies sandboxes kept running in detached mode with `(detached)`.

```
$ sbx ls
SANDBOX      AGENT    STATUS      PORTS                          WORKSPACE          CREATED
my-sandbox   claude   ● running   127.0.0.1:8080->3000/tcp4      /home/user/proj    2m
```

To stop forwarding a port:

```
$ sbx ports my-sandbox --unpublish 8080:3000
```

When `sbx run` re-attaches to an existing sandbox, it ignores `--publish`. Use `sbx ports` to publish ports on that sandbox. For dev server and host-service recipes, see Local services.

What persists

While a sandbox exists, installed packages, Docker images, configuration changes, command history, and mountless workspace files all persist across stops and restarts. Removing a sandbox deletes its container filesystem. Host workspace files, including repositories used as clone sources, and the shared agent skills store remain on your host. Persistent kit volumes in local sandboxes also survive removal. See Keep kit data when recreating a sandbox for how to reattach them.

To capture changes in the container filesystem, save a template. For a reproducible environment defined in source, author a kit.

Saving a sandbox as a template

Save a sandbox's container filesystem as a reusable template image after setting up tools or configuration interactively. A template contains image content; the agent kit still supplies runtime settings such as credentials and network rules. The examples here reuse templates with built-in agents.

A saved template isn't a backup of the whole sandbox. Mounted filesystems, including host workspaces and the Docker store at `/var/lib/docker`, aren't included. Save any data from those mounts separately.

> [!WARNING] Saving a sandbox captures files in its container filesystem, including any secrets stored there. If you manually added API keys, tokens, or other credentials to the sandbox, they're embedded in the saved template and shared with anyone you distribute it to. To keep credentials out of templates, manage them with `sbx secret set` instead — the proxy injects them at runtime so they're never written to the filesystem. For more information, see Manage credentials.

Save and reuse

Stop the sandbox (or let the CLI prompt you), then save it with a name and tag:

```
$ sbx template save my-sandbox my-template:v1
```

The image is stored in the sandbox runtime's local image store. Create a new sandbox from it with the `-t` flag:

```
$ sbx run -t my-template:v1 claude
```

List and remove templates

List all saved templates:

```
$ sbx template ls
```

Remove a template you no longer need:

```
$ sbx template rm my-template:v1
```

Export and import

To share a saved template or move it to another machine, export it as a tar file:

```
$ sbx template save my-sandbox my-template:v1 --output my-template.tar
```

On the other machine, load the tar file and use it:

```
$ sbx template load my-template.tar
$ sbx run -t my-template:v1 claude
```

Limitations

Agent configuration files are always recreated when a sandbox is created. Changes to user-level agent configuration files, such as `/home/agent/.claude/settings.json` and `/home/agent/.claude.json`, do not persist in saved templates.

If the saved template was built for a different agent than the one you specify in `sbx run`, the CLI prints a warning to stderr with a suggested command. For example, saving a Claude sandbox and running it with `codex` suggests `sbx run -t my-template:v1 claude`.

Load a template

To create a sandbox from a template image in a registry, pass its full image reference to `--template`. Use the agent the image was prepared for:

```
$ sbx run --template docker.io/my-org/my-template:v1 claude
```

Unlike Docker commands, `sbx` doesn't automatically add the Docker Hub domain (`docker.io`) to image references. For available images and the built-in agent workflow, see Base images.

> [!NOTE] The Docker daemon used by Docker Sandboxes pulls templates from a registry directly; it doesn't share the image store of your local Docker daemon on the host. To route Docker Hub image pulls through your organization's registry infrastructure, configure a registry mirror.

> [!IMPORTANT] For Docker Hub, `sbx` reuses your `sbx login` session to pull private images. For other registries (GitHub Container Registry, ECR, ACR, a self-hosted Nexus, and so on), store pull credentials with `sbx secret set --registry` before running the sandbox:
>
> ```
> $ gh auth token | sbx secret set --registry ghcr.io --password-stdin
> ```

> Without stored credentials, pulls from non-Docker Hub registries are anonymous and private images fail to pull.

For locally-built images, save the image to a tar and load it directly into the sandbox runtime instead of pulling from a registry:

```
$ docker image save my-org/my-template:v1 -o my-template.tar
$ sbx template load my-template.tar
$ sbx run --template my-org/my-template:v1 claude
```

`sbx template load` imports the tar into the sandbox runtime's image store, so the image doesn't need to be reachable from a registry at sandbox creation time.

Template caching

When creating a sandbox, `sbx` checks the registry for the template image by default and downloads missing or updated layers. If the pull fails and the image is cached locally, it can use the cached image. Cached images persist across sandbox creation and deletion, and are cleared when you run `sbx reset`.

Updating agents

Agent templates include an agent version, which can differ from the agent's latest release. Updating the `sbx` CLI or pulling an updated template doesn't update agents inside existing sandboxes.

To update an installed agent, run its documented update command inside the sandbox, either from a sandbox shell or with `sbx exec`. Restart the agent session to use the updated version. The update persists across sandbox stops and starts, but is deleted when you remove the sandbox. To reuse the updated agent in other sandboxes, save a template.

□□□ #2
□ □□□ □□□ 2026-10-31 19:02:24 UTC
□ □□□ □□□ 2026-10-31 19:02:27 UTC