Help

`lb`
every command, and what is usually run

`lb help release check`
a group's commands, or one command's options and examples; `--help` works at every level

Each command prints the script it runs before running it; the scripts stay usable on their own. Every environment variable a script reads is also an option, and the option's help names it.

Every day

`lb payload check`
does the patch queue still apply to the pinned SHAs? (~20 s)

`lb payload status`
which payload this checkout runs, and where from

`lb payload test`
the converter: fonts, docx → odt → docx, docx → pdf

`lb linux test`
the suite on Linux, in the test container

`lb linux test -k bridge`
anything after it goes to pytest

`lb linux test --gui`
the gui tier, a real GTK window under Xvfb

`lb linux lint`
the type checkers with the Linux branches live (`make lint-linux`)

`make verify`: lint, the whole suite, docs links, `lb payload check`, `lb payload test`. `make test`, `make lint`, `make test-gui`: as in any Python project. `make payload` is `lb payload build`; `make ship` is `lb release check`, then `lb release publish`.

The payload

`lb payload build`
every stage, then the test: hours, V8 about one

`lb payload build configure`
one stage: `fetch`, `configure`, `core`, `assemble`, `dist`; several run in that order

`lb payload build fetch --repo core`
one repository, at its pin, the queue applied; `--mirror DIR` for local clones

`lb payload build core --v8-jobs 4 --core-jobs 8`
x2t and the native libraries, the long one

`lb payload build dist --fonts full`
the tarballs and their manifest; `core` is the set that ships

`lb components build`
this machine's `libera-core` wheel, from its sdist, and `libera-web`'s; `--out DIR`

`--build-root DIR` (`BUILD_ROOT`): `~/euro-office-build` on Linux, the external case-sensitive volume on macOS. `uv sync` builds the two components itself, from the pins.

Toolchain

`lb toolchain check`
what is missing, in a second rather than an hour into a build

`lb toolchain list`
the packages this system needs, installing nothing

`lb toolchain install --dry-run`
what `install` would do; without `--dry-run`, it does it

Applications

`lb app build macos`
the `.app`; `--standalone` puts Python, `libera` and the payload inside

`lb app build flatpak --arch arm64`
also `macos-launcher`, `flatpak-wheels`, `windows`

`lb app check flatpak`
the built Flatpak installed, `--diagnose` run in the sandbox

`lb app test flatpak`
the Flatpak converting a document under the GNOME runtime

`lb app test windows`
the gui tier, in a Windows window

`lb linux screenshot --dialog`
a GTK window under Xvfb, screenshotted; `--lang fr_FR.UTF-8`

The `linux` commands install the component wheels of their architecture (`--wheels DIR`), or mount a payload as is (`--payload DIR`). `--arch amd64|arm64` defaults to this machine's.

Containers

`lb container image payload`
build or rebuild an image: `payload`, `flatpak` or `linux`

`lb container versions payload`
what that image is pinned to

`lb container shell linux`
a shell in one

`lb container run payload`
the whole Linux payload build in its container

`lb container run payload dist.sh --fonts core`
or one `build/` script in it

`lb container export payload`
the artifacts, copied out; where to on stdout

`lb container clean flatpak`
the Flatpak builder's runtime volume, dropped (asks)

`ENGINE=podman` or `docker`. A docker context that is a daemon over ssh cannot mount this tree, and `lb` says so.

On a builder

`lb on linux-x86_64 app test flatpak`
any `lb` command, in that builder's checkout; output streams back

`lb on windows-x86_64 --update toolchain check`
this commit sent there first

Builders: `build/builders.toml` (gitignored; copy `builders.toml.example`). Platforms: `macos-arm64`, `linux-x86_64`, `linux-arm64`, `windows-x86_64`. Nothing interactive works through `lb on`.

A release

`lb release bump 0.8.0`
the version of `libera`, `libera-core` and `libera-web`, in all six places; `0.8.0rc1` is a pre-release

`lb release check --dry-run`
the plan, running nothing

`lb release check`
build and install everything as users will; publishes nothing

`lb release check stage`
one phase, or several; they always run in their order

`lb release check components bundles --only linux-x86_64`
one builder: narrows `update`, `components`, `bundles`

`lb release status`
where the release has got to: the tree, the run directory, PyPI

`lb release publish`
the stage, as stamped, to PyPI and the CDN, then read back. No undo: asks on the terminal unless `--yes`

`lb release publish downloads`
what users download, installed and photographed on the builders

check `update` `components` `bundles` `wheel` `stage` `distros`
publish `publish` `verify` `downloads` `tap` `docs`

Run directory `build/out/release/<commit>/`: `pypi/`, `bundles/`, `staged/` with `STAGED.json`. A new commit is a new, empty run: run the phases again. **Don't commit during a run**: it stops after the phase that saw HEAD move.
Late Windows build: `check bundles --only windows-x86_64`, `check stage`, then `publish`; PyPI skips what it has.
Logs: `tmp/remote/<platform>.log`. A builder's job runs detached (`~/.lb/runs/` there); the same phase run again watches it.

Housekeeping

`lb tidy --dry-run`
every build on this machine, with its size; `lb tidy` asks before removing, `--yes` does not

`lb bench --runs 5`
a document to the screen, timed stage by stage; `--idle 20` starts cold

`lb fidelity --runs 2`
the same pages rendered twice and scored; `--ref DIR` against reference PDFs

Environment

`BUILD_ROOT`	sources and builds (`--build-root`)
`MIRROR`	local clones to fetch from (`--mirror`)
`V8_BUILD_JOBS`	V8's parallel jobs; `CORE_BUILD_JOBS` for core's ninja
`ARCH`	`amd64` or `arm64` (`--arch`)
`ENGINE`	`podman` or `docker`
`LIBERA_PAYLOAD`	the payload `libera` and the tests use, instead of the components
`CDN_TOKEN`	`publish`'s token for the CDN zone