Cargo Rust package manager

START HERE

`cargo new <name>`
Create a project

`cargo add <crate>`
Add a dependency

`cargo run`
Build and run

`cargo build --release`
Optimised build

`cargo check`
Type-check fast

`cargo fmt`
Format the code

`cargo test`
Run all tests

`cargo clippy`
Lint the code

Workspace root Cargo.toml, Cargo.lock, target/
Package one Cargo.toml, a crate or more
Target lib, bin, example, test, bench

A workspace shares one Cargo.lock and one target/ directory

CONCEPTS

Use cargo check while editing: it skips code generation and is much faster than build.

Cargo.toml is what you edit; Cargo.lock records the exact resolved versions. Commit the lock for binaries.

Arguments after -- go to the program or test harness, not to Cargo.

PROJECTS

`cargo new <name>`
Create a binary package
Add --lib for a library.

`cargo init`
Make the current directory a package

`cargo run`
Build and run the binary
Program arguments go after --.

`cargo run --bin <name>`
Run a specific binary

`cargo run --example <name>`
Run an example

`cargo clean`
Delete the target directory

`cargo doc --open`
Build docs and open them
Add --no-deps to skip dependency docs.

`cargo install <crate>`
Install a binary crate
Lands in ~/.cargo/bin by default.

`cargo install --path .`
Install the local package

`cargo uninstall <crate>`
Remove an installed binary

BUILD & CHECK

`cargo build`
Compile in the dev profile

`cargo build --release`
Compile with optimisations
Output goes to target/release/.

`cargo check`
Type-check without building

`cargo check --all-targets`
Also check tests and examples

`cargo build --target <triple>`
Cross-compile
The target's std must be installed first.

`cargo build --timings`
HTML report of build times

BUILD & CHECK (cont.)

`cargo fix`
Apply compiler-suggested fixes
Refuses on a dirty tree unless --allow-dirty.

TEST & LINT

`cargo test`
Run unit, integration and doc tests

`cargo test <filter>`
Run tests whose name contains text

`cargo test -- --nocapture`
Show output of passing tests

`cargo test --lib`
Unit tests only

`cargo test --doc`
Doc tests only

`cargo test --test <name>`
One integration test file

`cargo test --no-fail-fast`
Keep going after a failure
By default Cargo stops after the first failing test binary.

`cargo test -- --exact <path>`
Match the full test name

`cargo test -- --test-threads=1`
Run tests serially

`cargo test --no-run`
Compile tests without running

`cargo bench`
Run benchmarks

`cargo fmt`
Format with rustfmt

`cargo fmt --check`
Fail if code is unformatted

`cargo clippy`
Lint with Clippy

`cargo clippy -- -D warnings`
Treat lints as errors

DEPENDENCIES

`cargo add <crate>`
Add a dependency

`cargo add <crate>@<ver>`
Add with a version requirement

`cargo add --dev <crate>`
Add a dev-dependency

`cargo add <crate> -F <feat>`
Add with a feature enabled

`cargo remove <crate>`
Remove a dependency

DEPENDENCIES (cont.)

`cargo update`
Update Cargo.lock within semver

`cargo update <crate>`
Update only one crate

`cargo update <crate> --precise <ver>`
Pin a crate to a version

`cargo tree`
Show the dependency tree

`cargo tree -i <crate>`
Who depends on this crate?

`cargo tree -d`
Show duplicate versions

`cargo tree -e features -i <crate>`
Why is a feature enabled?

`cargo metadata --format-version 1`
Package info as JSON

FEATURES

`cargo build --features <list>`
Enable features
Comma- or space-separated; quote a space-separated list.

`cargo build --no-default-features`
Turn off the default feature set

`cargo build --all-features`
Enable every feature

`cargo build -F <pkg>/<feat>`
Enable a feature of a member

Cargo.toml features

```
[features]
default = ["json"]
json = ["dep:serde_json"]
fast = []

[dependencies]
serde_json = { version = "1", optional = true }
```

Features are additive: a crate gets the union of everything requested by anyone in the build.

PROFILES

dev vs release defaults

Setting	dev	release
opt-level	0	3
debug	true	false
debug-assertions	true	false
overflow-checks	true	false
incremental	true	false

`cargo build --profile <name>`
Build with a custom profile
Output goes to target/<name>/.

PROFILES (cont.)

Release tuning

```
[profile.release]
lto = true
codegen-units = 1
panic = "abort"
strip = "symbols"

[profile.release-debug]
inherits = "release"
debug = true
```

WORKSPACES

`cargo build --workspace`
Build every member

`cargo test -p <pkg>`
Test one package

`cargo build -p <pkg>`
Build one package

`cargo run -p <pkg>`
Run a member's binary

Root Cargo.toml

```
[workspace]
members = ["crates/*"]
resolver = "3"

[workspace.dependencies]
serde = "1"
```

PUBLISHING

`cargo login`
Store a registry token

`cargo package --list`
Show files that would be packaged

`cargo publish --dry-run`
Run all checks, upload nothing

`cargo publish`
Upload to crates.io
Published versions cannot be deleted, only yanked.

RECIPES

New project to first run

```
cargo new hello && cd hello
cargo add clap -F derive
cargo run
```

CI gate

```
cargo fmt --check
cargo clippy --all-targets -- -D warnings
cargo test --workspace --locked
```

Why is this crate built twice?

```
cargo tree -d
cargo tree -i <crate>@<ver>
```

Release a crate

```
cargo test
cargo publish --dry-run
git tag v<ver> && cargo publish
```

HELP

`cargo help <command>`
Manual for a command