Bazel Fast, hermetic builds

START HERE

`bazel build //...`
Build everything

`bazel test //...`
Run all tests

`bazel run //<pkg>:<target>`
Build and run a binary

`bazel test --test_output=errors //...`
Show failing test logs

`bazel query "deps(//<pkg>:<target>)"`
What does it depend on?

`bazel build -c opt //<pkg>:<target>`
Optimized build

`bazel clean`
Delete build outputs

`bazel info`
Paths and server state

Load read BUILD files → packages → **Analyze** build the action graph → actions → **Execute** run actions, use cache → outputs → **bazel-bin/** results

CONCEPTS

A label is //package:target. The package is a directory with a BUILD file; //pkg alone means //pkg:pkg.

Builds are hermetic and cached: a rerun with no input changes does nothing. --nobuild stops after analysis.

Bazel 9 uses Bzlmod only: dependencies are declared in MODULE.bazel, WORKSPACE is gone.

BUILD

`bazel build <targets>`
Build the given targets

`bazel build //...`
Build every target in the main repo

`bazel build //<pkg>/...`
Build a package and all below it

`bazel build //<pkg>:all`
Build all rule targets in one package

`bazel build -- //... -//<pkg>/...`
Build all except a subtree
The -- is required before patterns that start with -.

`bazel build -c <mode>`
fastbuild, dbg or opt

`bazel build -k`
Keep going after errors

`bazel build --config=<name>`
Apply a named .bazelrc config

`bazel build --nobuild`
Load and analyze only

TEST

`bazel test <targets>`
Build and run tests

`bazel test --test_output=errors`
Print output of failed tests only

`bazel test --test_output=all`
Print all test output

`bazel test --test_filter=<expr>`
Select tests inside a target
How the filter is read depends on the test framework.

TEST (cont.)

`bazel test --test_arg=<arg>`
Pass an argument to the test binary

`bazel test --runs_per_test=<N>`
Run each test N times (flaky hunt)

`bazel test --cache_test_results=no`
Force tests to rerun

`bazel coverage <targets>`
Run tests with coverage

RUN

`bazel run <target>`
Build then run a target

`bazel run <target> -- <args>`
Pass args to the program

`bazel-bin/<pkg>/<target>`
Run the built binary directly

QUERY

`bazel query "deps(<t>)"`
All dependencies of a target

`bazel query "rdeps(//..., <t>)"`
Who depends on this target

`bazel query "somepath(<a>, <b>)"`
One dependency path from a to b

`bazel query "allpaths(<a>, <b>)"`
All dependency paths from a to b

`bazel query "kind(<regex>, //...)"`
Targets by rule kind

`bazel query //<pkg>/... --output=label_kind`
List targets with their rule

`bazel query <expr> --output=graph`
Graphviz dot output

`bazel query <expr> --noimplicit_deps`
Hide implicit dependencies

`bazel cquery <expr>`
Query with configurations applied

`bazel cquery <expr> --output=files`
Show output files of targets

QUERY (cont.)

`bazel aquery <expr>`
Show the actions behind targets

`bazel aquery "mnemonic('<regex>', <t>)"`
Filter actions by mnemonic

`bazel aquery <expr> --output=commands`
One command line per action

DEPENDENCIES (BZLMOD)

`bazel mod graph`
Show the module dependency graph

`bazel mod explain <module>`
Why is this module here?

`bazel mod path <module> --from <root>`
One path from a module to another

`bazel mod show_repo @<repo>`
Show a repo's definition

`bazel mod tidy`
Format MODULE.bazel, fix use_repo

`bazel fetch //...`
Pre-fetch external repos

`--lockfile_mode=error`
Fail if MODULE.bazel.lock is stale

SERVER AND CLEANUP

`bazel clean`
Delete build outputs

`bazel clean --expunge`
Delete the whole output base

`bazel info`
Show workspace and output paths

`bazel info <key>`
One value (bazel-bin, output_base)

`bazel version`
Print Bazel version

`bazel help <command>`
Help for one command

DEBUGGING BUILDS

`bazel build --verbose_failures`
Print the failing command in full

DEBUGGING BUILDS (cont.)

`bazel build -s`
Print every command as it runs

`bazel build --sandbox_debug`
Keep sandbox dirs for inspection

`bazel build --profile=<file>`
Write a performance profile

`bazel build --announce_rc`
Show which rc options apply

.BAZELRC

`build --<flag>`
Default flag for build and its children

`common --<flag>`
Flag for every command that supports it

`build:<name> --<flag>`
Flag group used by --config=<name>

`try-import %workspace%/user.bazelrc`
Optional personal overrides

`build:linux --<flag>`
Per-OS flags (with platform config on)
Add common --enable_platform_specific_config so build:linux, build:macos and build:windows apply automatically.

Command-line flags beat rc files; more specific commands (test) beat build, which beats common.

RECIPES

Test one target and see why it fails

```
bazel test //<pkg>:<test> \
  --test_output=errors --
test_filter=<name> \
  --cache_test_results=no --
verbose_failures
```

Why does A depend on B?

```
bazel query "somepath(//
<pkg>:<a>, //<pkg>:<b>)"
bazel query "rdeps(//..., //
<pkg>:<b>)" --output=label_kind
```

Release and debug configs

```
# .bazelrc
build --keep_going
build:release -c opt
build:debug -c dbg --strip=never
# bazel build --
config=release //...
```