Dart Language and SDK tool

CONCEPTS

Everything goes through one dart command; dart help <command> shows the options of each subcommand.

pubspec.yaml declares deps
↓ dart pub get
pubspec.lock pinned versions
↓ dart run
bin/ + lib/ your code
↓ dart compile
exe / js / wasm output

Sound null safety: a type is non-nullable unless you add ?, and the analyzer enforces it.

PROJECT

dart create *<name>*
New console app

dart create -t package *<name>*
New library package

dart --version
Print SDK version

RUN AND COMPILE

dart run
Run the package's main

dart run *<file.dart>*
Run one file

dart run *<pkg>*:*<script>*
Run a package executable

dart compile exe *<file.dart>*
AOT native executable

dart compile js *<file.dart>*
Compile to JavaScript

dart compile exe -o *<out>* *<file>*
Name the output file

PACKAGES

dart pub add *<pkg>*
Add a dependency

dart pub add dev:*<pkg>*
Add a dev dependency

dart pub remove *<pkg>*
Remove a dependency

PACKAGES (cont.)

dart pub get
Fetch locked deps

dart pub upgrade
Upgrade within constraints

dart pub upgrade --major-versions
Raise constraints too

dart pub outdated
Show newer versions

dart pub deps
Print dependency tree

dart pub global activate *<pkg>*
Install a CLI tool

QUALITY

dart format .
Format all files

dart format --output=none --set-exit-code .
Check formatting (CI)

dart analyze
Analyze the package

dart fix --dry-run
Preview automated fixes

dart fix --apply
Apply automated fixes

dart test
Run all tests

dart test *<path>*
Run one test file

dart test -n *<name>*
Filter tests by name

LANGUAGE BASICS

```
var x = 1; final y = 2; const z = 3;
```
Mutable, final, compile-time

```
String? s;
```
Nullable type

```
s?.length ?? 0
```
Null-aware access with default

```
s!
```
Assert non-null

```
late final T x;
```
Initialize later

LANGUAGE BASICS (cont.)

```
'Hi $name, ${a + b}'
```
String interpolation

```
int add(int a, int b) => a + b;
```
Arrow function

```
void f({required int a, int b = 0})
```
Named parameters

```
void f(int a, [int? b])
```
Optional positional

```
class P { final int x; P(this.x); }
```
Class with ctor shorthand

```
class A extends B with M implements I
```
Inheritance, mixin, interface

```
enum Color { red, green }
```
Enum

```
switch (v) { case 1 => 'a', _ => 'b' }
```
Switch expression

COLLECTIONS AND ASYNC

```
[1, 2, 3]
```
List literal

```
{'a': 1}
```
Map literal

```
{1, 2}
```
Set literal

```
[for (var i in xs) i * 2]
```
Collection for

```
xs.map((x) => x * 2).toList()
```
Transform a list

```
xs.where((x) => x > 1)
```
Filter

```
xs.fold(0, (a, b) => a + b)
```
Reduce

```
Future<T> f() async { await g(); }
```
Async function

```
await Future.wait([a(), b()])
```
Run futures together

```
Stream<T> f() async* { yield 1; }
```
Stream generator