# Folding JMH into Continuous Integration

# Benchmarks in the Same Project

**No official documentation from OpenJDK, though:**

**Adds build semantics to benchmark execution**

**Lowers impedance when doing exploratory testing**

**Different hardware, kernels, or even JVM builds can wildly alter statistics at the micro scale**

**Clear goals and expectations are vital**

- Automated verdicts require more time
- Trending requires the data be freed

## Continuous Integration Requirements

Benchmarks go in the same project as the code

Benchmarks ship their output for later analysis

Benchmarks are asserted programmatically

# Add Benchmarks to Your Existing Project

```
project/
  src/
    main/
      java
    test/
      java
    benchmark/
      java
  pom.xml
```

Step One: **Tell Maven about your benchmark folder using** `build-helper-maven-plugin`

Step Two: **Tell Maven to use JMH's annotation processor when compiling tests in the** `maven-compiler-plugin` **configuration**

*(Optional):* **Tell Maven to ship the benchmarks in the same artifact as your code with the** `maven-shade-plugin`

Step Three: **Add** `jmh-core` **as a test dependency**

```
@Test
public void runBenchmarks() {
    Runner runner = new Runner();
    Collection<RunResult> results = runner.run();
    // analyze results
}
```

## Running JMH programmatically

**No-arg constructor configures JMH to run with the JMH command-line defaults, specifically:**

- Run all benchmarks on the classpath
- 10 JVM Forks, 20 measurement and warmup iterations each

```
Options opts =
    new OptionsBuilder()

        .shouldFailOnError(true)


        .include(“.*Test”)


        .output(“/dev/null”)
        .result(“/dev/null”)

        .build();

Runner runner =
    new Runner(opts);

runner.run();
```

- Configure JMH with options
- Fail the build if a benchmark fails to run
- Use regex to identify which benchmarks to run
- Redirect output appropriately

## Running Benchmarks in Your IDE

```
project/
  src/
    main/
      java
    test/
      java
    benchmark/
      java
  pom.xml
```

**Step One:** Add `jmh-generator-annprocessor` as a test dependency

**Step Two:** Tell IntelliJ/Eclipse to use annotation processing on your project

```java
@Test
public void runBenchmarks() {
    Runner runner = new Runner();
    Collection<RunResult> results = runner.run();
    for ( RunResult result : results ) {
        Result primary = result.getPrimaryResult();
        // assert with JUnit, ship to database, etc…
    }
}
```

## Programmatically analyze JMH results

**All data from the UI is available via RunResult**

**Not all data is available as the right datatype (e.g. units)**

## Congratulations!

**JShell**

A command-line REPL that saves you precious make time

**JMH**

A benchmark framework that saves you precious make time

Good luck!