# Study of diff algorithms for CI pipeline logs

In this study, you will be facing various CI logs annotated by two different algorithms. These logs come from the execution of CI workflows from different open-source projects on GitHub. The scenario in which you find yourself is as follows:

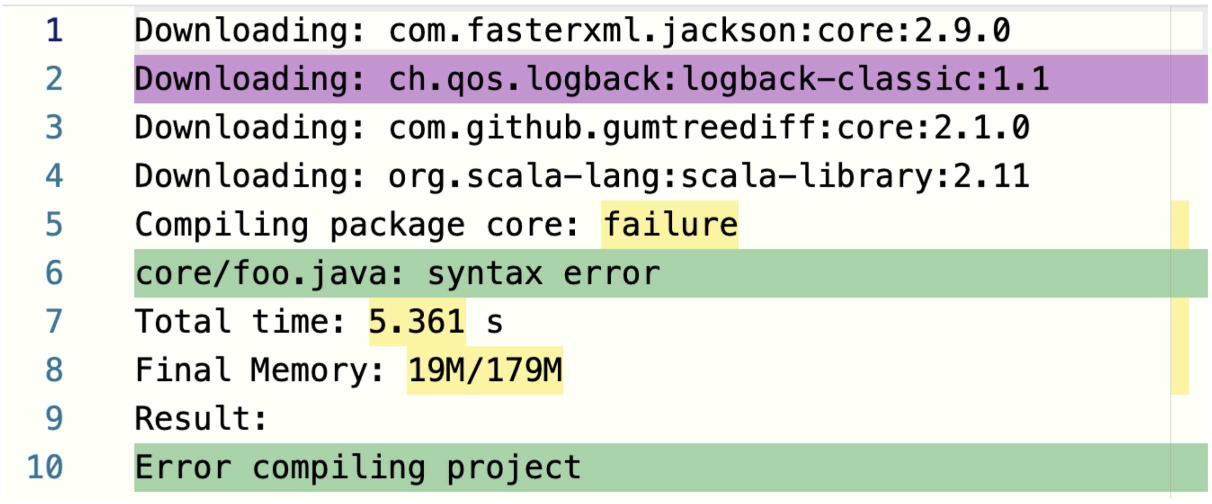
> *You are a developer of a project for which a continuous integration pipeline has been set up. That pipeline just failed. Your goal is to find error messages relevant to understanding the failure of the pipeline.*

To help you in detecting these relevant error messages, the log of the failing pipeline (called *failing log*) is annotated by comparison against the log of the last known passing pipeline (called *passing log*). The lines can be of four different colors:

- White line: line that is identically present in the passing log. In general these lines do not help in finding relevant error messages as they were already present in the passing log.
- Green line: line added with respect to the passing log. For the most part, relevant error messages are contained in these lines, as error messages are not present in the passing log.
- Orange line: line already present in the passing log but with a partial modification of content (only the modification is in orange). These lines can sometimes give context to the failure (like a test status changing from *success* to *failure*).
- Purple line: line already present in the passing log but appearing in a different order in the failing log. These lines present operations that have not been carried out in the same order as in the passing log (like a test not executed at the same moment). In really rare cases failures can be provoked by these changes of order.

To make your life easier, the default view shows only lines annotated in green with a context of three lines above and below, given that these green lines are the ones with the highest probability to contain relevant error messages explaining the source failure. If needed, you can unfold the hidden zones to show more context. To unfold a hidden zone, simply click the angle brackets near the line numbers.

Here is an example of an annotated failing log:



```
 1   Downloading: com.fasterxml.jackson:core:2.9.0
 2   Downloading: ch.qos.logback:logback-classic:1.1
 3   Downloading: com.github.gumtreediff:core:2.1.0
 4   Downloading: org.scala-lang:scala-library:2.11
 5   Compiling package core: failure
 6   core/foo.java: syntax error
 7   Total time: 5.361 s
 8   Final Memory: 19M/179M
 9   Result:
10   Error compiling project
```

In this example, one line was moved (in purple), a few lines were updated (with the differences in orange), and two lines were added (in green). The identical lines remained in white. The most interesting line to understand the failure is line 6 (in green) which displays an error message. Furthermore, lines 5 (in orange) and 10 (in green) give more context to the failure. Finally, the other lines are not related to the failure. Of course, this example is very small, but in real cases logs are much larger.

The goal of this experiment is to indicate which annotation algorithm you prefer to complete the above scenario on a sample of cases. In the different cases you will see it is therefore **the same failing log** but **annotated by two different algorithms**. You must indicate “**alpha**” when you prefer the result from the alpha algorithm, “**beta**” when you prefer the result from the beta algorithm, or “**none**” when you have no preference between the results of the two algorithms.