CLOJURE BERLIN // 09.05.2018

Native Clojure with GraalVM in 38 screenshots

Jan Stępień

janstepien.com

INNOQ

JAN STĘPIEŃ (7) JAN STĘPIEŃ (7)

Name

Jan Stępień works with people and with computers.

Synopsis

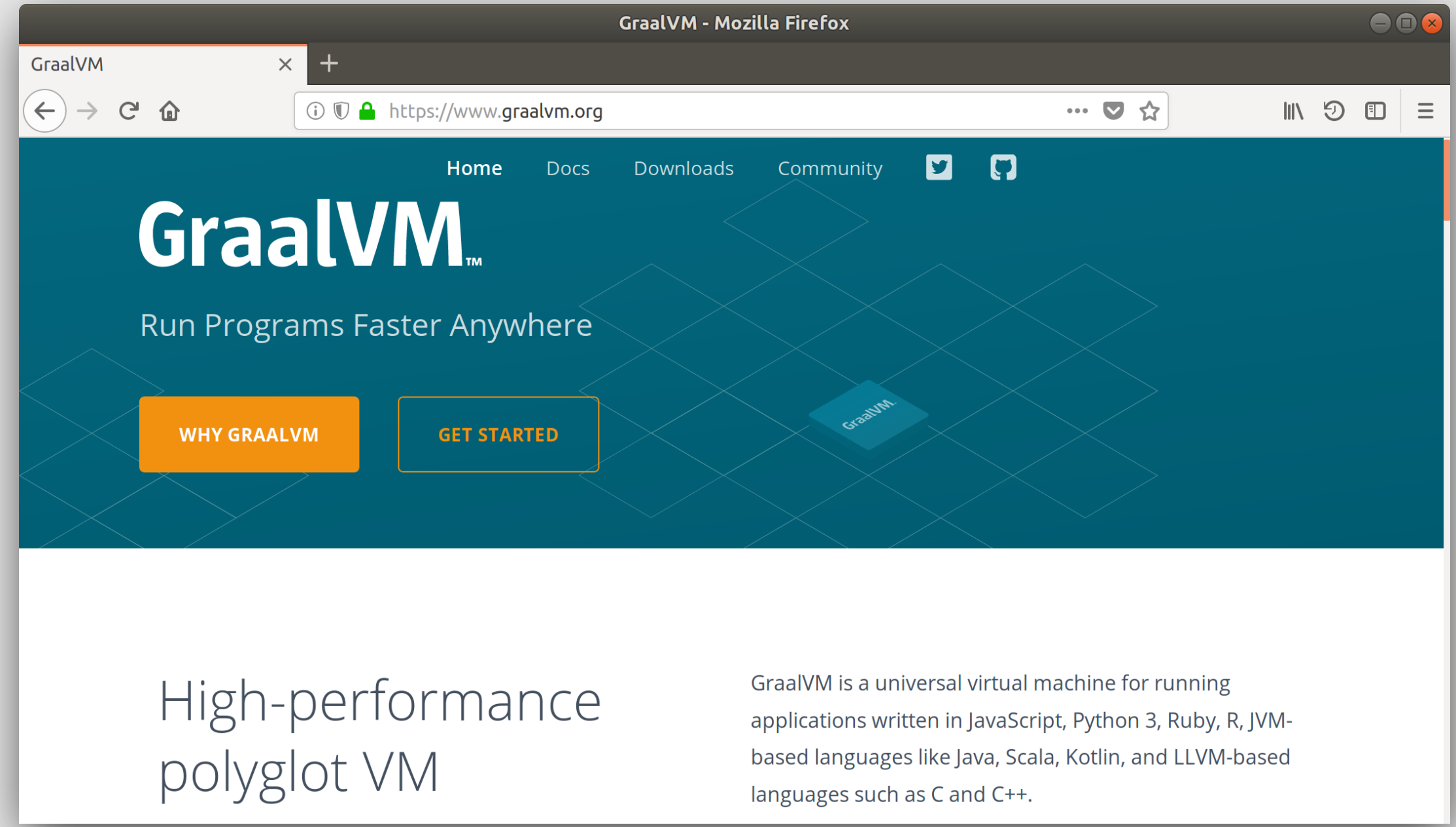
jan@stepien.cc

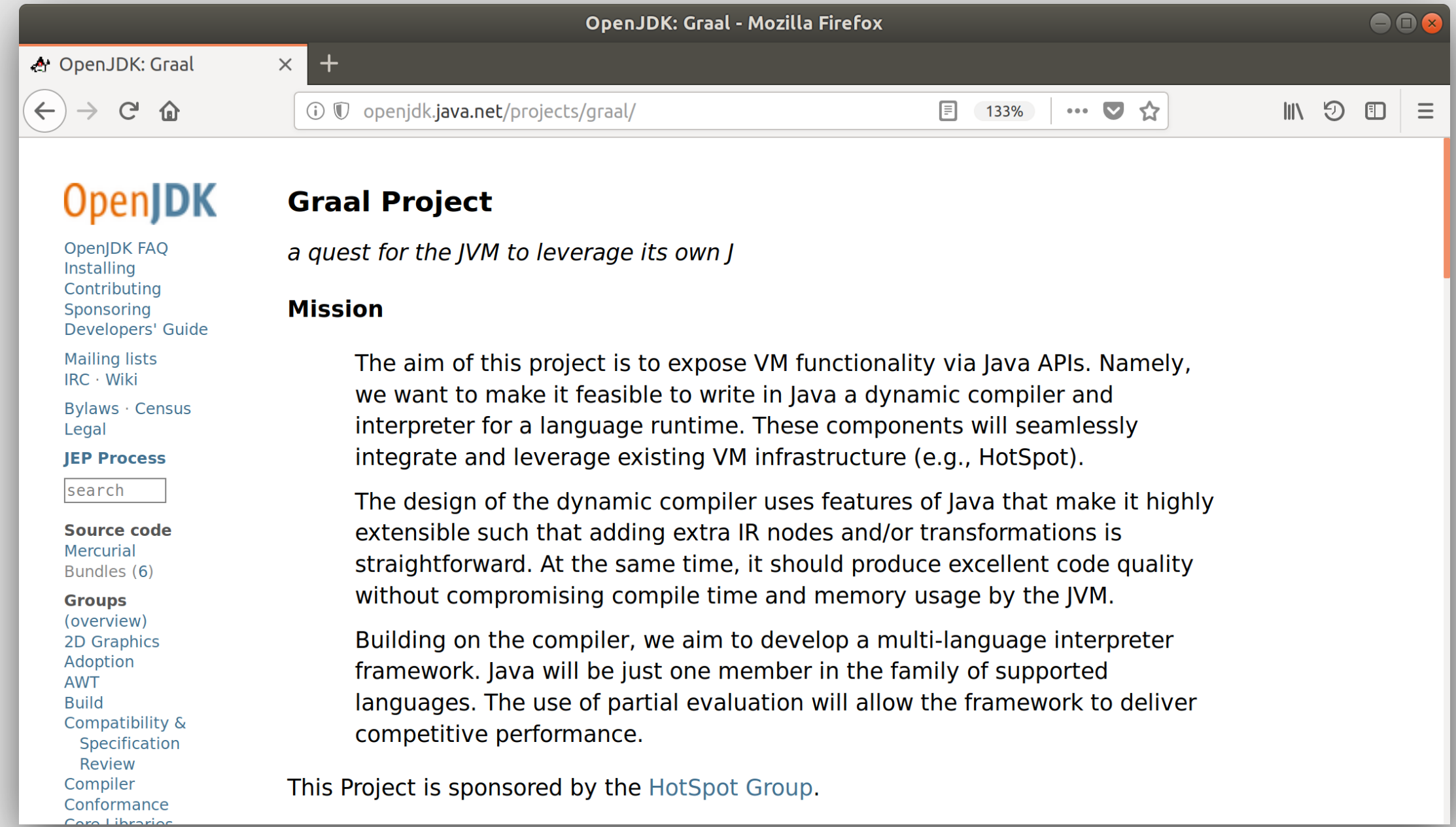
@janstepien

Description

I work with people to build better software. My interests span architecture, development, and operations. I look for simple solutions for complex problems.

I live in Berlin, Germany and am a consultant at INNOQ.

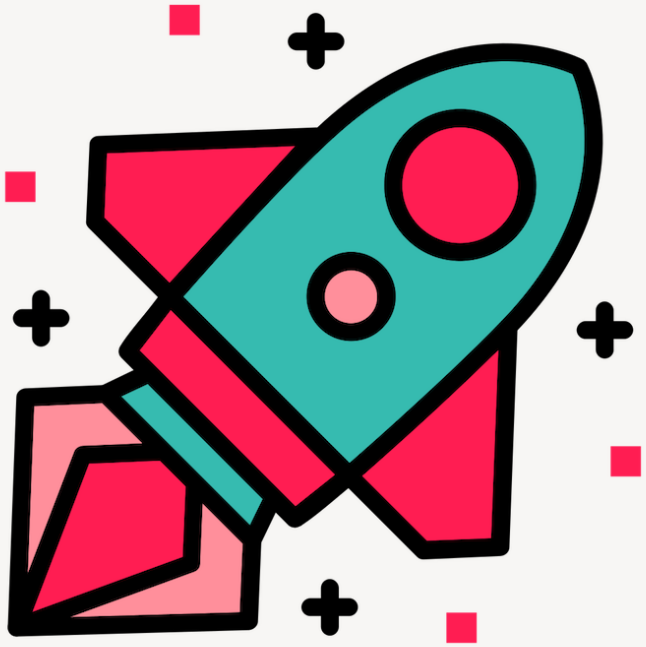

GraalVM - Mozilla Firefox
GraalVM
https://www.graalvm.org
Home
Docs
Downloads
Community
GraalVM™
Run Programs Faster Anywhere
WHY GRAALVM
GET STARTED
GraalVM
High-performance polyglot VM
GraalVM is a universal virtual machine for running applications written in JavaScript, Python 3, Ruby, R, JVM-based languages like Java, Scala, Kotlin, and LLVM-based languages such as C and C++.


OpenJDK: Graal - Mozilla Firefox
OpenJDK: Graal
openjdk.java.net/projects/graal/
133%
OpenJDK
OpenJDK FAQ
Installing
Contributing
Sponsoring
Developers' Guide
Mailing lists
IRC · Wiki
Bylaws · Census
Legal
JEP Process
search
Source code
Mercurial
Bundles (6)
Groups
(overview)
2D Graphics
Adoption
AWT
Build
Compatibility & Specification Review
Compiler
Conformance
Graal Project
a quest for the JVM to leverage its own J
Mission
The aim of this project is to expose VM functionality via Java APIs. Namely, we want to make it feasible to write in Java a dynamic compiler and interpreter for a language runtime. These components will seamlessly integrate and leverage existing VM infrastructure (e.g., HotSpot).
The design of the dynamic compiler uses features of Java that make it highly extensible such that adding extra IR nodes and/or transformations is straightforward. At the same time, it should produce excellent code quality without compromising compile time and memory usage by the JVM.
Building on the compiler, we aim to develop a multi-language interpreter framework. Java will be just one member in the family of supported languages. The use of partial evaluation will allow the framework to deliver competitive performance.
This Project is sponsored by the HotSpot Group.

TruffleRuby - Mozilla Firefox

TruffleRuby

chrisseaton.com/truffleruby/



TruffleRuby started as my internship project at Oracle Labs in early 2013. It is an implementation of the Ruby programming language on the JVM, using the Graal dynamic compiler and the Truffle AST interpreter framework. TruffleRuby can achieve peak performance well beyond that possible in JRuby at the same time as being a significantly simpler system. In early 2014 it was open sourced and integrated into JRuby for incubation, and then in 2017 it became its own project.

In our new namespace, we just need to import the Polyglot Context to get started:

```
1  (ns graal-test.core
2    (:import (org.graalvm.polyglot Context)))
3
4  ;; note that is also supports Ruby, LLVM, and JS
5  (def context (Context/create (into-array ["python" "R"])))
```

Now, we are ready to actually try to run some R and Python code right in our REPL. Let's start first with R.

Interoping with R

The main function we are going to use is the `eval` function in the context. Let's start small with some basic math.

```
1  (.eval context "R" "
2  3^2 + 2^2
3  ")
4  ;=> #object[org.graalvm.polyglot.Value 0x7ff40e4d "13.0"]
```

Oracles GraalVM für „Native Java“? - Mozilla Firefox

Oracles GraalVM für „Nat

https://www.innoq.com/de/blog/native-java-graalvm/

Oracles GraalVM für „Native Java“?

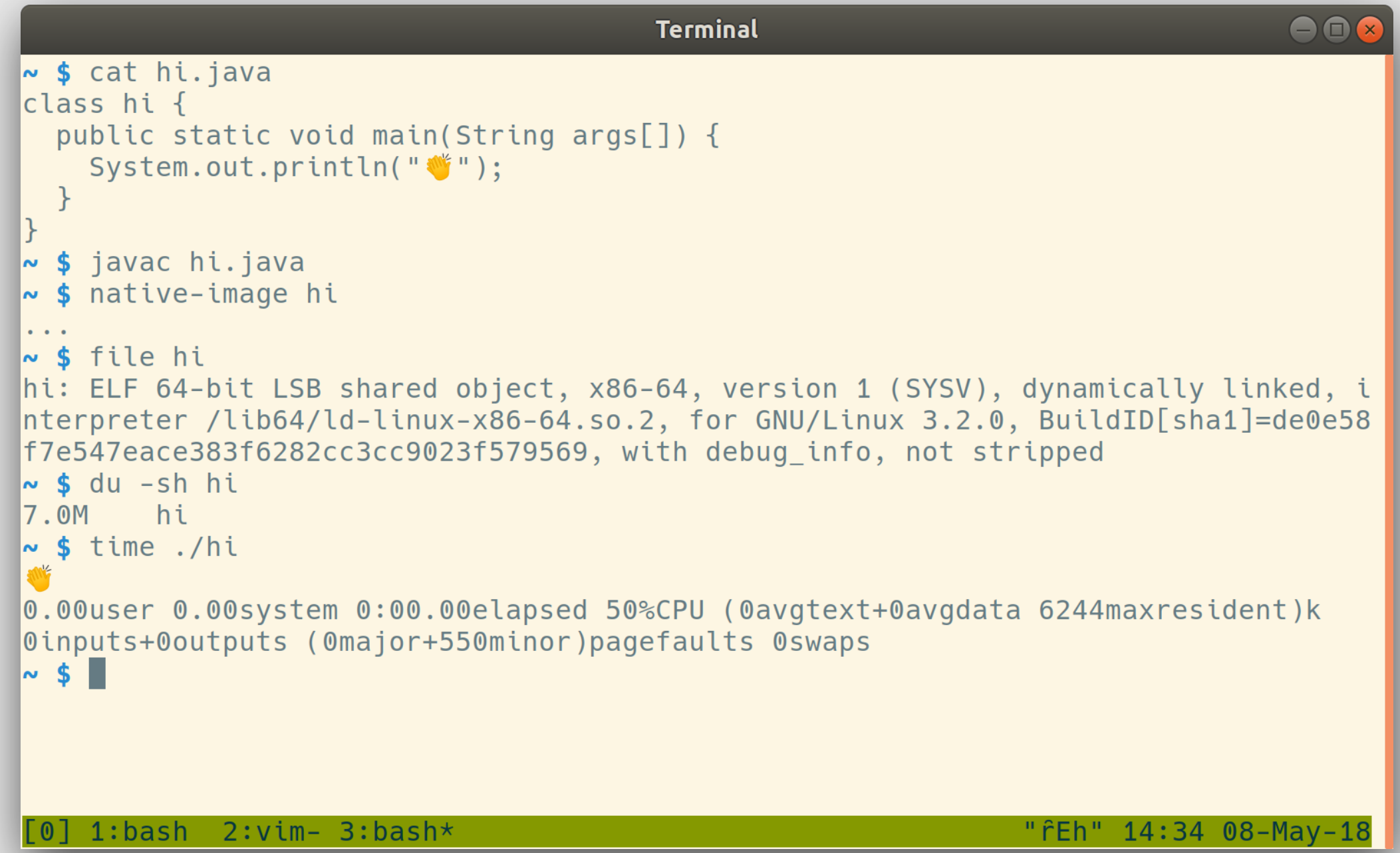
25. April 2018 — *10 Minuten Lesedauer*

RUBEN WAGNER

Mein Dilemma

Ich bin ein großer Fan von Docker und Kubernetes, da es mir die Möglichkeit gibt, kleine funktionale Einheiten zu bauen und sie später wieder zu großen zusammenzufügen. Ganz


GitHub - innoq/blog-native-java-graalvm at afdbbd547e9ac9e4b6c67105533e7a877d8562e5 - Mozilla Firefox
GitHub - innoq/blog-nati
GitHub, Inc. (US)
https://github.com/innoq/blog-native-java-graalvm/tree
HelloWorldServer.java
hello-world-server
14 days ago
README.md
containerize
14 days ago
README.md
blog-native-java-graalvm
Setup
wget https://github.com/oracle/graal/releases/download/vm-1.0.0-rc1/graalvm-ce-1.0.0-rc1-linux-amd64.tar.gz
docker build . -t graalvm
Hello World
docker run --rm -tiv `pwd`:/work -w /work
bash-4.2# javac HelloWorld.java
bash-4.2# native-image HelloWorld
bash-4.2# ./helloworld


Terminal
~ $ cat hi.java
class hi {
public static void main(String args[]) {
System.out.println("👋");
}
}
~ $ javac hi.java
~ $ native-image hi
...
~ $ file hi
hi: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, i
nterpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=de0e58
f7e547eace383f6282cc3cc9023f579569, with debug_info, not stripped
~ $ du -sh hi
7.0M hi
~ $ time ./hi
👋
0.00user 0.00system 0:00.00elapsed 50%CPU (0avgtext+0avgdata 6244maxresident)k
0inputs+0outputs (0major+550minor)pagefaults 0swaps
~ $
[0] 1:bash 2:vim- 3:bash*
"ŕEh" 14:34 08-May-18

Hello World Server

Um als Nächstes ein Gefühl für den Memory-Footprint zu bekommen, habe ich fix einen simplen Webserver kopiert und modifiziert.

```
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;

import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;

public class HelloWorldServer {

    public static void main(String[] args) throws Exception {
        HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
        server.createContext("/", new MyHandler());
        server.setExecutor(null);
        server.start();
```

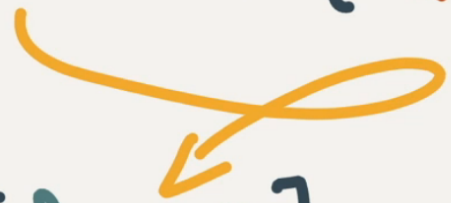
GitHub - jstepien/jalunke: Pure Java (and JavaScript) Halunke - Mozilla Firefox

GitHub - jstepien/jalunke

GitHub, Inc. (US) https://github.com/jstepien/jalunke

https://janstepien.com/compiling-a-lisp-into-a-lisp-using-a-lisp/

```
('foo = 3)
((foo > 0) then {"positive"}
           else {"zero or less"})
```



```
(let [foo 3]
  (then-else (> foo 0)
    (fn [] "positive")
    (fn [] "zero or less")))
```

```clojure
(ns jalunke.main
  (:require [clojure.pprint :refer [pprint]]
            [jalunke.eval :as e])
  (:gen-class))

(defn -main []
  (print "✨ ")
  (flush)
  (when-let [line (read-line)]
    (try
      (pprint (e/evaluate line))
      (catch Throwable t
        (pprint t)))
    (recur)))
```

```
$ native-image jalunke.main
Build on Server(pid: 11, port: 26681)*
   classlist:   5,823.05 ms
       (cap):   1,224.07 ms
       setup:   2,867.61 ms
    analysis:  53,944.42 ms
error: unsupported features in 5 methods
Detailed message:
Error: com.oracle.graal.pointsto.constraints.UnsupportedFeatureException:
Unsupported constructor java.lang.ClassLoader.<init>(ClassLoader) is
reachable: The declaring class of this element has been substituted, but this
element is not present in the substitution class. To diagnose the issue,
you can add the option -H:+ReportUnsupportedElementsAtRuntime. The unsupported
element is then reported at run time when it is accessed the first time.
Trace:
        at parsing java.security.SecureClassLoader.<init>(SecureClassLoader.java:76)
Call path from entry point to java.security.SecureClassLoader.<init>(ClassLoader):
        at java.security.SecureClassLoader.<init>(SecureClassLoader.java:76)
```

Biserkov Add table of contents and status. 2572abf 5 days ago

5 contributors

199 lines (135 sloc) 10.4 KB Raw Blame History

Substrate VM Java Limitations

Substrate VM does not support all features of Java to keep the implementation small and concise, and also to allow aggressive ahead-of-time optimizations. This page documents the limitations. If you are not sure if a feature is supported, ask christian.wimmer@oracle.com so that this list can be updated.

What	Support Status
Dynamic Class Loading / Unloading	Not supported
Reflection	Partially supported

```
$ native-image -H:+ReportUnsupportedElementsAtRuntime jalunke.main
Build on Server(pid: 11, port: 26681)
   classlist:   3,663.62 ms
       (cap):     888.29 ms
       setup:   1,433.31 ms
  (typeflow):  35,999.69 ms
   (objects):   7,178.83 ms
  (features):     150.20 ms
    analysis:  44,373.37 ms
    universe:   2,278.12 ms
     (parse):   8,019.46 ms
    (inline):   6,412.02 ms
   (compile):  30,652.61 ms
     compile:  46,265.50 ms
       image:   3,479.37 ms
       write:     675.22 ms
     [total]: 102,363.25 ms
```

The compilation is successful and our program can be started. An attempt to evaluate a Halunke expression leads to a following exception.

```
✨ 42
#error {
 :cause "(...) The declaring class of this element has been substituted,
         but this element is not present in the substitution class"
 :via
 [{:type com.oracle.svm.core.jdk.UnsupportedFeatureError
   :message "(...)"
   :at [java.lang.Throwable <init> "Throwable.java" 265]}]
 :trace
 [[java.lang.Throwable <init> "Throwable.java" 265]
  [java.lang.Error <init> "Error.java" 70]
  [com.oracle.svm.core.jdk.UnsupportedFeatureError <init> "UnsupportedFeatureError.jav
  [com.oracle.svm.core.jdk.Target_com_oracle_svm_core_util_VMError unsupportedFeature
  [java.lang.ClassLoader <init> "ClassLoader.java" 316]
  [java.security.SecureClassLoader <init> "SecureClassLoader.java" 76]
  [java.net.URLClassLoader <init> "URLClassLoader.java" 100]
  [clojure.lang.DynamicClassLoader <init> "DynamicClassLoader.java" 41]
```

Reflection

What:

Listing methods and fields of a class; invoking methods and accessing fields reflectively; most classes in the package `java.lang.reflect`.

Support Status: Partially supported

Specific kinds of reflection access can be enabled for individual classes, methods and fields by providing one or several configuration files via the `-H:ReflectionConfigurationFiles=` option during native image generation. Elements that are not included in a configuration cannot be accessed reflectively. For more details, read our documentation on reflection.

During native image generation:

Reflection can be used without restrictions during native image generation, for example in static initializers.

Implicit Exceptions

What:

Exceptions that are thrown implicitly by bytecodes, such as: `NullPointerException`, `ArrayIndexOutOfBoundsException`,

Command line, web, and beyond

Having learned the limitations imposed on our native binaries, we can build something more useful. How about a simple tool allowing us to get nested values from EDN-formatted data structures? Just like a minimal jq, but for Clojure's serialisation format?

```
(ns pprintin.main
  (:require [clojure.pprint :refer [pprint]])
  (:gen-class))

(defn -main [& path]
  (-> (read *in*)
      (get-in (mapv read-string path))
      pprint))
```

mfikes / **compare.md**

Last active 10 days ago

Star 0 Fork 0

Code Revisions 6

Embed <script src="https://gis Download ZIP

Planck compared to GraalVM

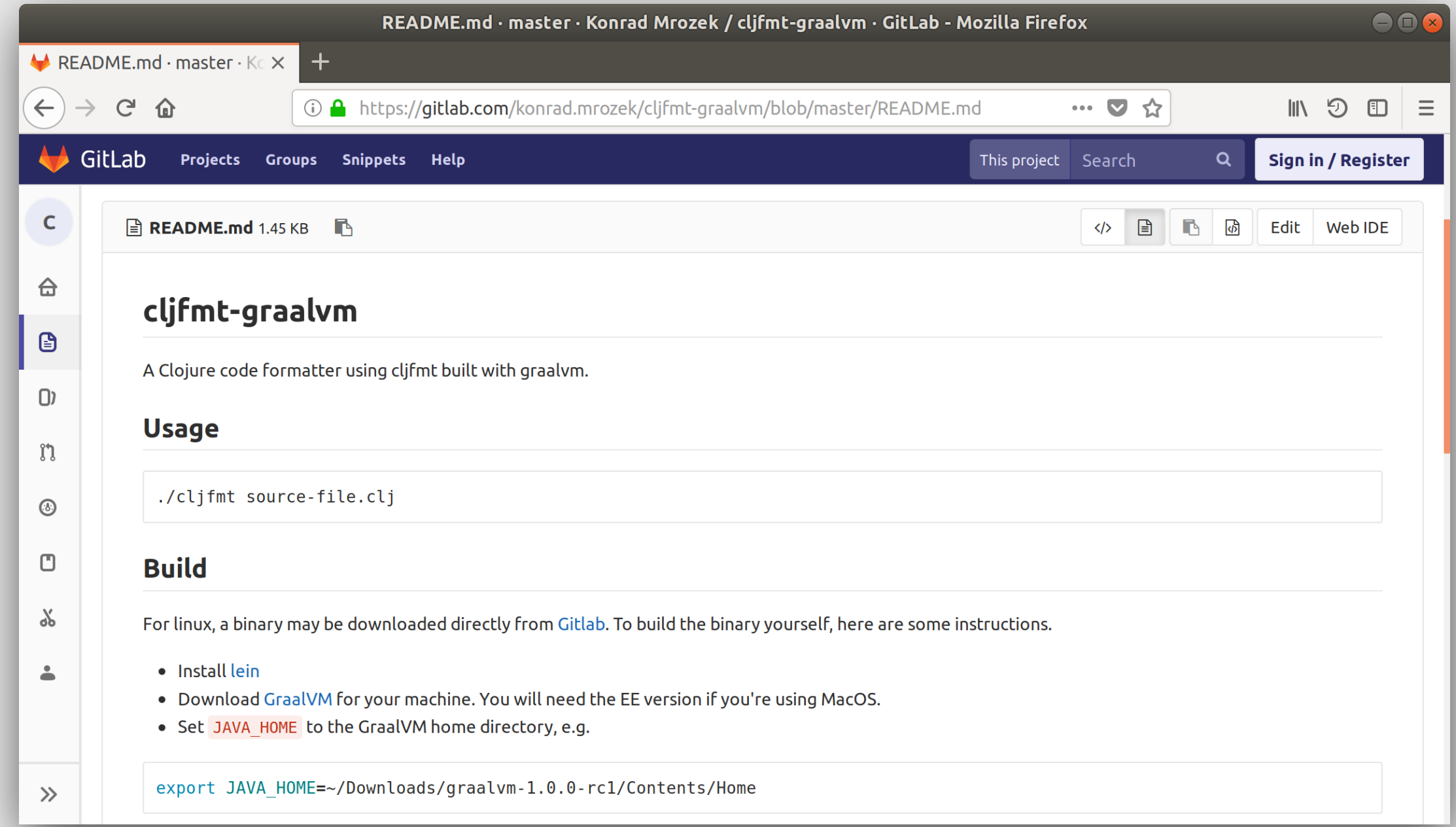
compare.md Raw

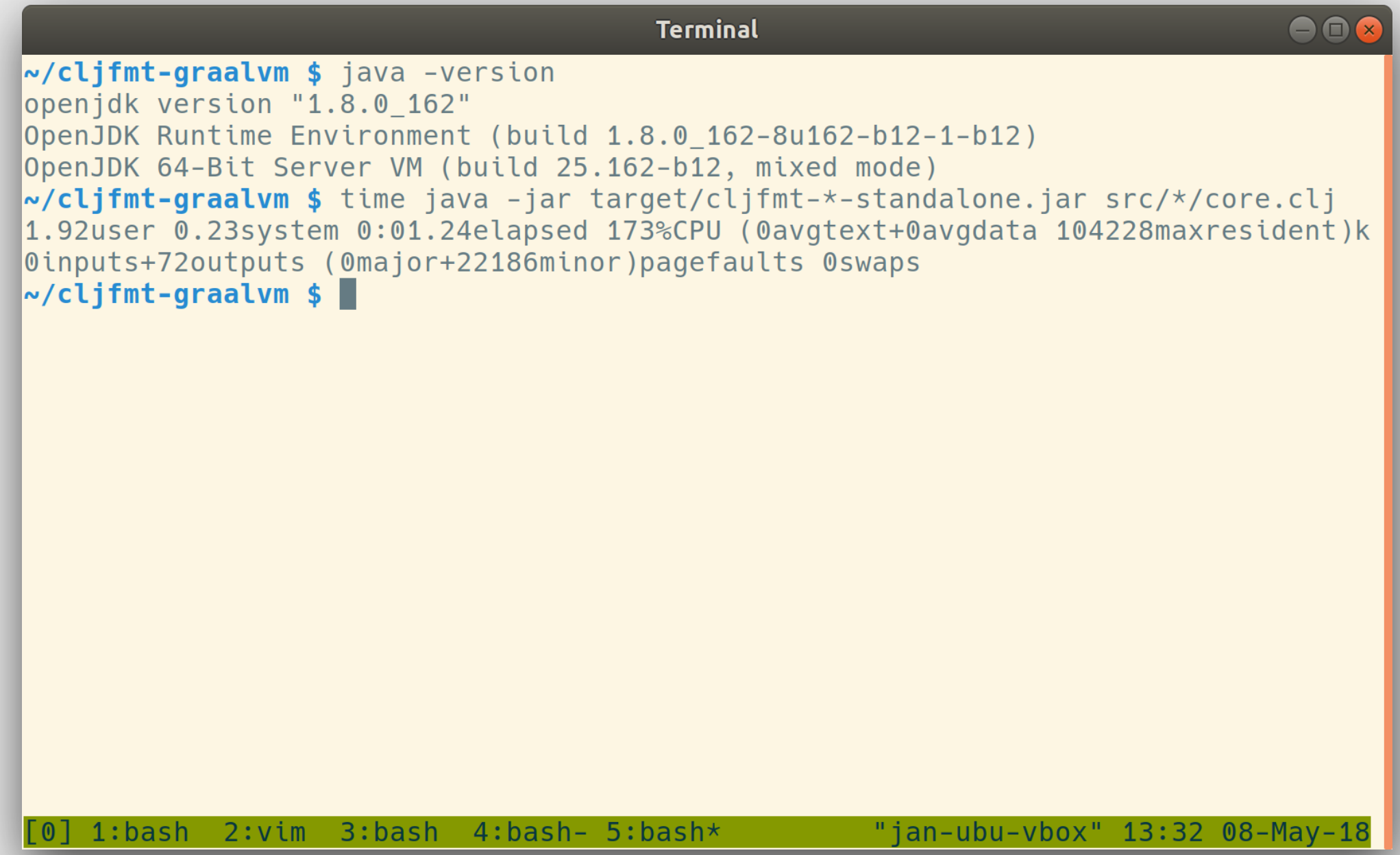
In Native Clojure with GraalVM the "jq" example runs in about 0.1 second using GraalVM.

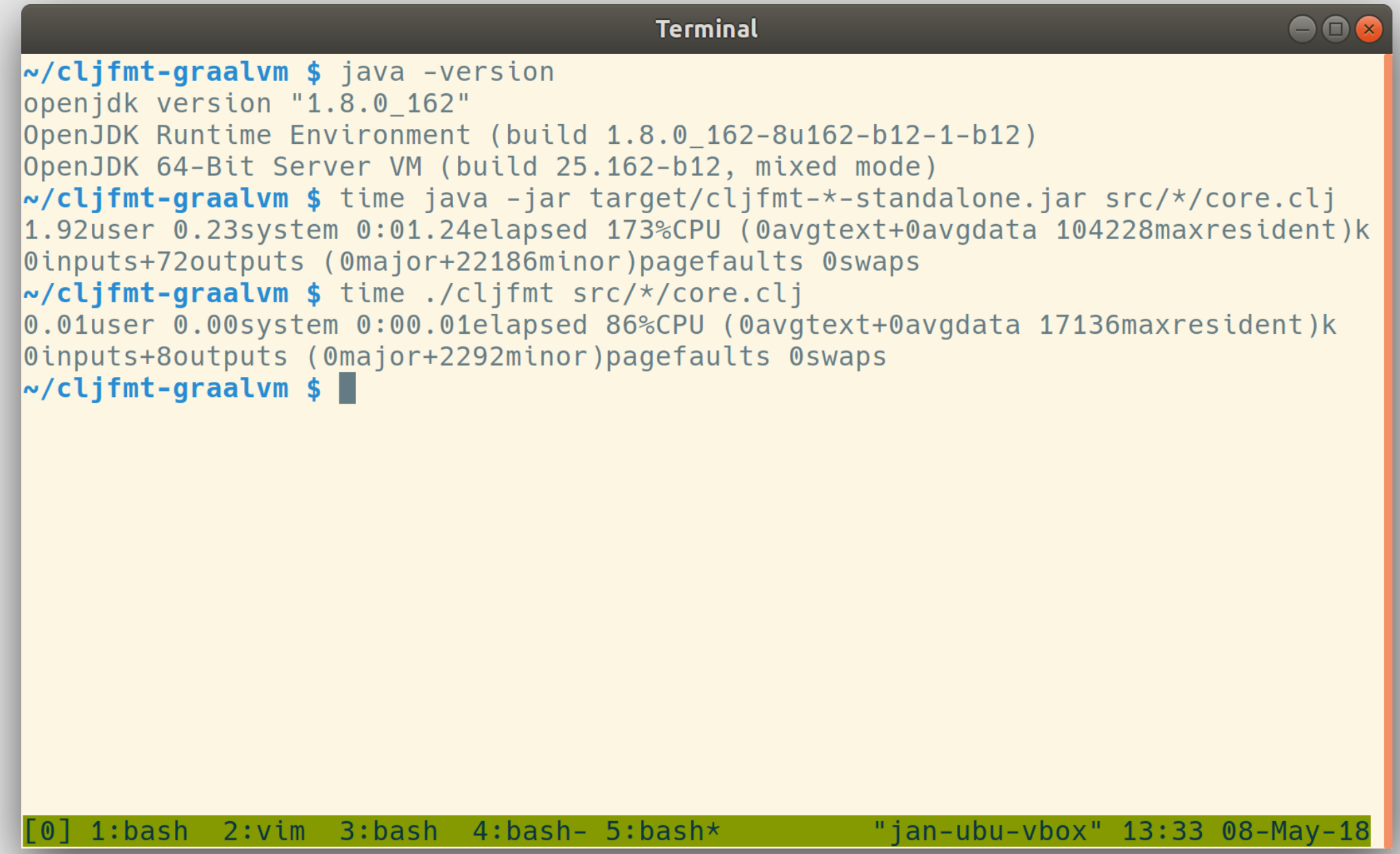
I'm not building the GraalVM solution for an accurate comparison, but below is the same using Planck (on a box with potentially different perf characteristics).

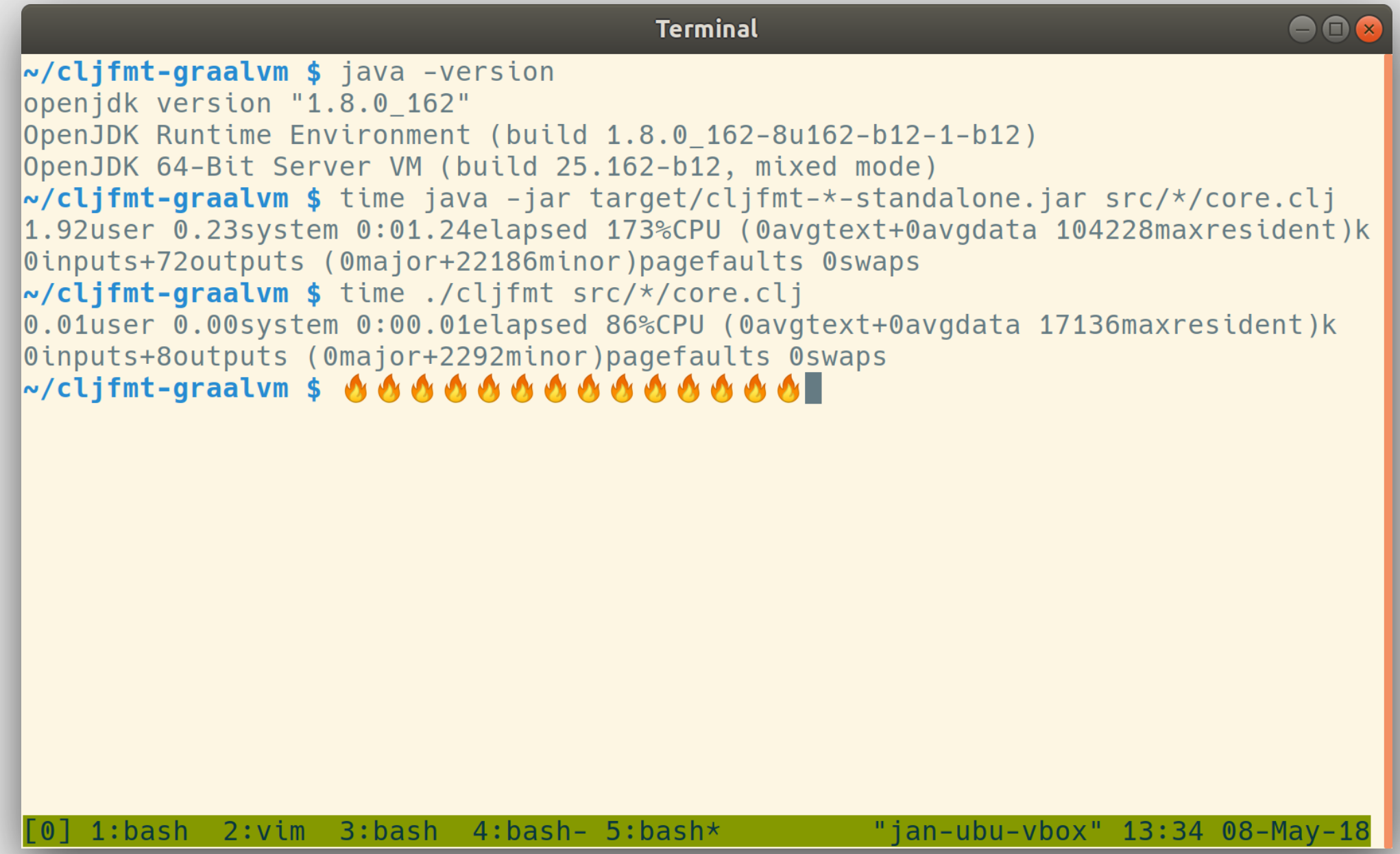
Porting to Planck (by adding `[planck.core :refer [*in* read read-string]]`) shows that the GraalVM approach is much faster. Planck is taking about 0.85 seconds and even around 0.68 seconds if you cache simple-optimized code:

`src/pprintin/main.cljs`:


README.md · master · Konrad Mrozek / cljfmt-graalvm · GitLab - Mozilla Firefox
README.md · master · K
https://gitlab.com/konrad.mrozek/cljfmt-graalvm/blob/master/README.md
GitLab
Projects
Groups
Snippets
Help
This project
Search
Sign in / Register
README.md 1.45 KB
Edit
Web IDE
cljfmt-graalvm
A Clojure code formatter using cljfmt built with graalvm.
Usage
./cljfmt source-file.clj
Build
For linux, a binary may be downloaded directly from Gitlab. To build the binary yourself, here are some instructions.
Install lein
Download GraalVM for your machine. You will need the EE version if you're using MacOS.
Set JAVA_HOME to the GraalVM home directory, e.g.
export JAVA_HOME=~/Downloads/graalvm-1.0.0-rc1/Contents/Home


Terminal
~/cljfmt-graalvm $ java -version
openjdk version "1.8.0_162"
OpenJDK Runtime Environment (build 1.8.0_162-8u162-b12-1-b12)
OpenJDK 64-Bit Server VM (build 25.162-b12, mixed mode)
~/cljfmt-graalvm $ time java -jar target/cljfmt-*-standalone.jar src/*/core.clj
1.92user 0.23system 0:01.24elapsed 173%CPU (0avgtext+0avgdata 104228maxresident)k
0inputs+72outputs (0major+22186minor)pagefaults 0swaps
~/cljfmt-graalvm $
[0] 1:bash 2:vim 3:bash 4:bash- 5:bash* "jan-ubu-vbox" 13:32 08-May-18


Terminal
~/cljfmt-graalvm $ java -version
openjdk version "1.8.0_162"
OpenJDK Runtime Environment (build 1.8.0_162-8u162-b12-1-b12)
OpenJDK 64-Bit Server VM (build 25.162-b12, mixed mode)
~/cljfmt-graalvm $ time java -jar target/cljfmt-*-standalone.jar src/*/core.clj
1.92user 0.23system 0:01.24elapsed 173%CPU (0avgtext+0avgdata 104228maxresident)k
0inputs+72outputs (0major+22186minor)pagefaults 0swaps
~/cljfmt-graalvm $ time ./cljfmt src/*/core.clj
0.01user 0.00system 0:00.01elapsed 86%CPU (0avgtext+0avgdata 17136maxresident)k
0inputs+8outputs (0major+2292minor)pagefaults 0swaps
~/cljfmt-graalvm $
[0] 1:bash 2:vim 3:bash 4:bash- 5:bash* "jan-ubu-vbox" 13:33 08-May-18


Terminal
~/cljfmt-graalvm $ java -version
openjdk version "1.8.0_162"
OpenJDK Runtime Environment (build 1.8.0_162-8u162-b12-1-b12)
OpenJDK 64-Bit Server VM (build 25.162-b12, mixed mode)
~/cljfmt-graalvm $ time java -jar target/cljfmt-*-standalone.jar src/*/core.clj
1.92user 0.23system 0:01.24elapsed 173%CPU (0avgtext+0avgdata 104228maxresident)k
0inputs+72outputs (0major+22186minor)pagefaults 0swaps
~/cljfmt-graalvm $ time ./cljfmt src/*/core.clj
0.01user 0.00system 0:00.01elapsed 86%CPU (0avgtext+0avgdata 17136maxresident)k
0inputs+8outputs (0major+2292minor)pagefaults 0swaps
~/cljfmt-graalvm $ 🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥🔥
[0] 1:bash 2:vim 3:bash 4:bash- 5:bash* "jan-ubu-vbox" 13:34 08-May-18

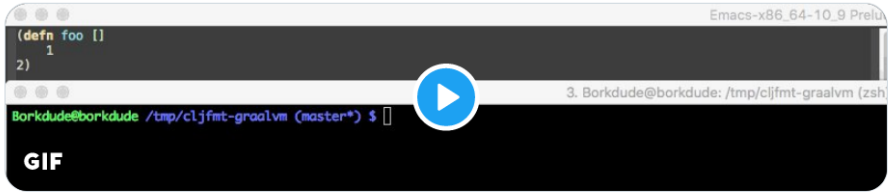
(λ. borkdude) on Twitter: "cljfmt using #graalvm, really fast! https://t.co/84J7uMteVO #cl...

(λ. borkdude) on Twitter:

Twitter, Inc. (US) https://twit 80%

(λ. borkdude)
@borkdude

cljfmt using #graalvm, really fast!
gitlab.com/konrad.mrozek/…
#clojure pic.twitter.com/sLhM1MuiXy



10:13 PM · May 3, 2018

21 Retweets **61** Likes

(λ. borkdude) @borkdude · May 3
Replying to @borkdude
Integrated with Emacs now!

2 13

cljtree-graalvm

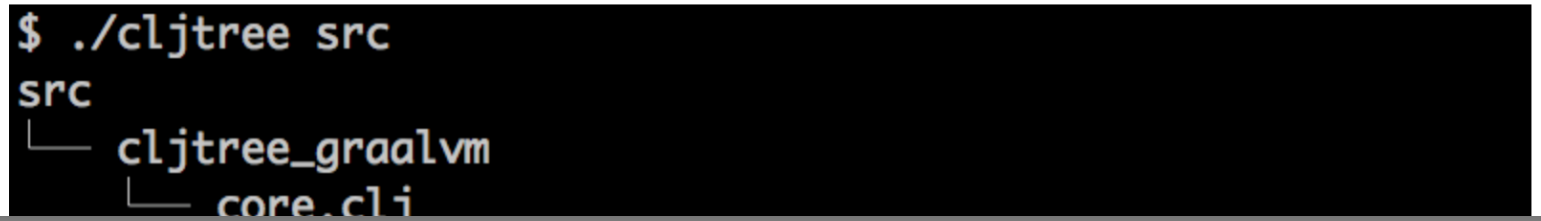
A Clojure version of `tree` built with GraalVM.

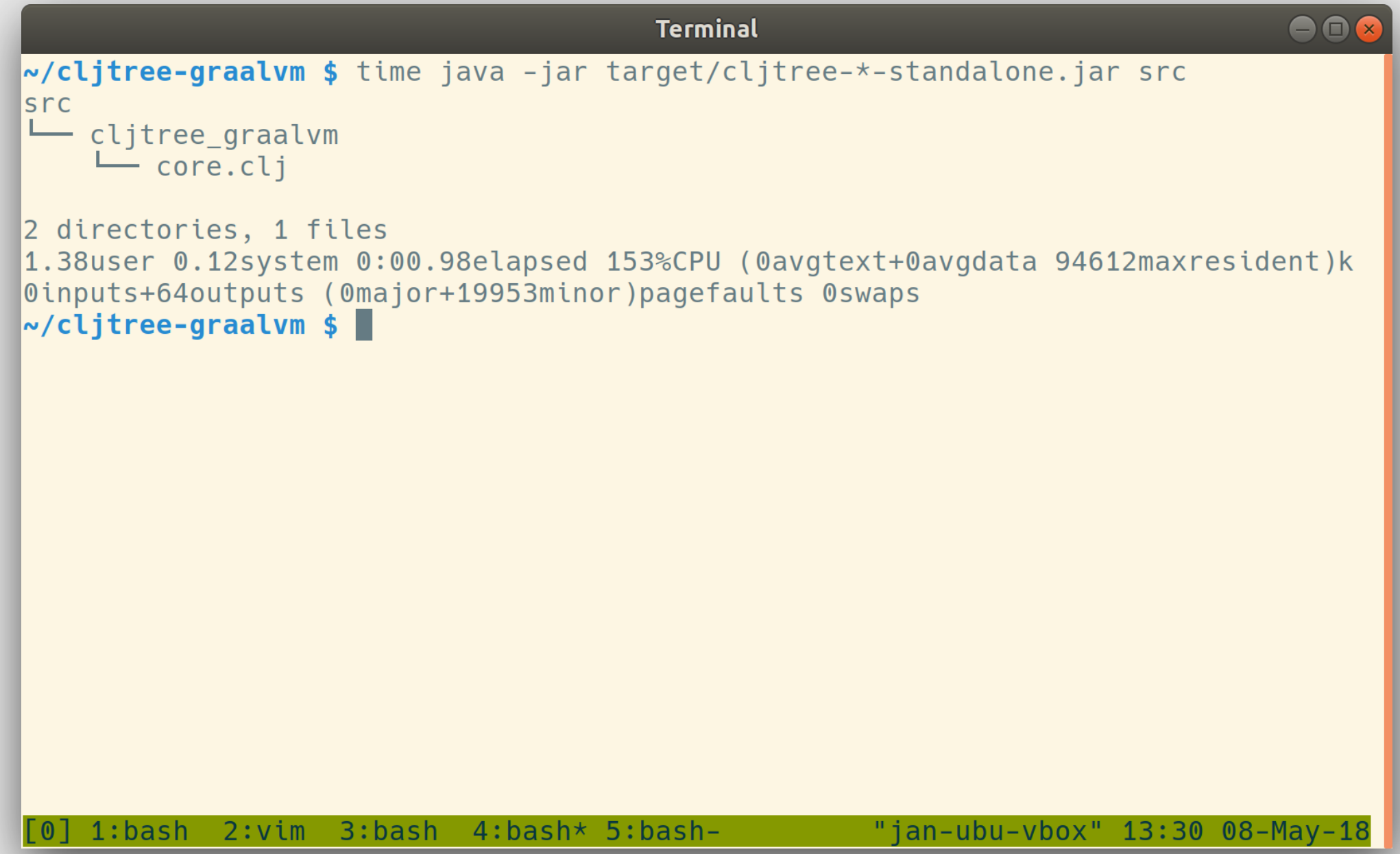
Credits

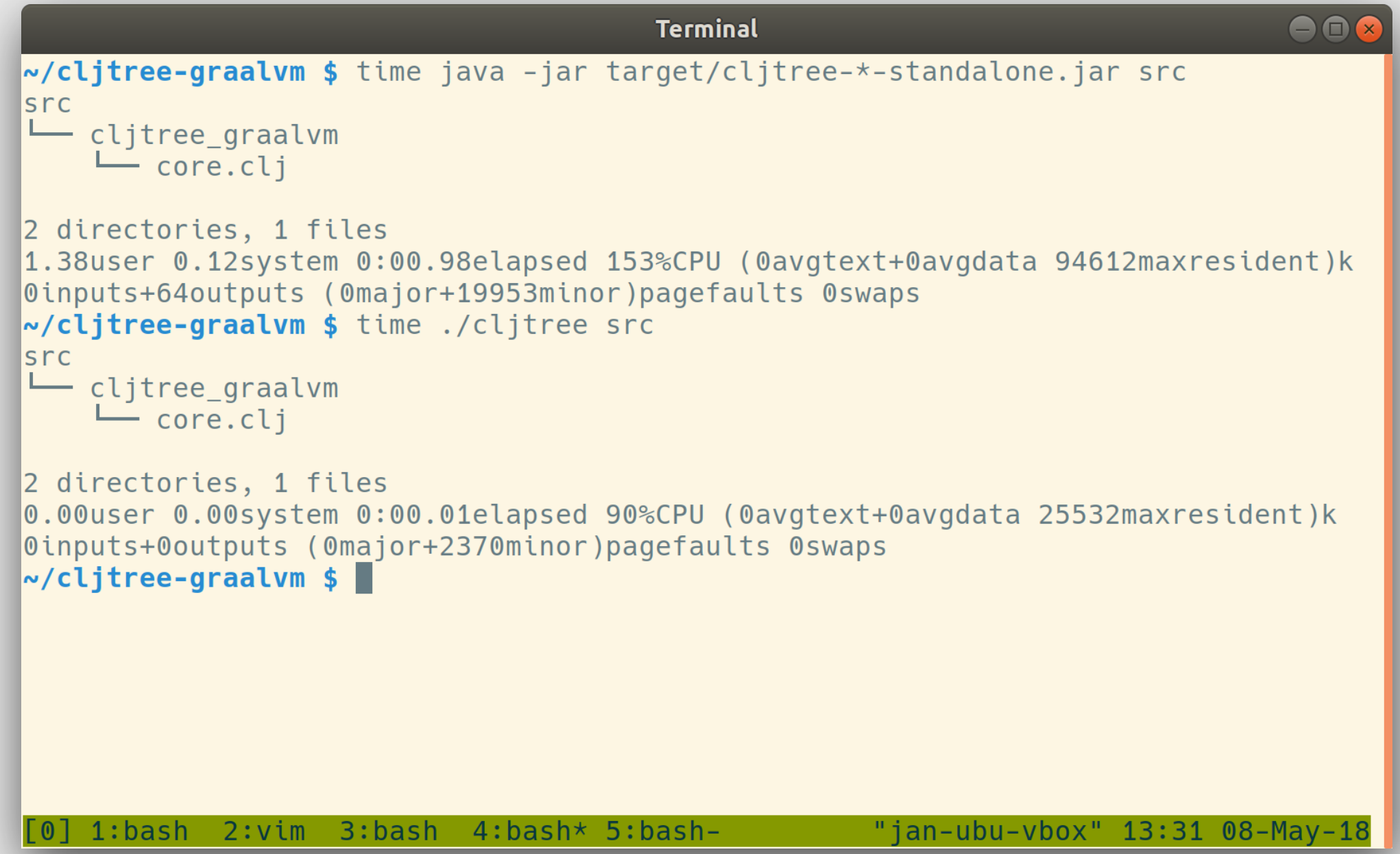
This repo is inspired by:

- https://gitlab.com/konrad.mrozek/cljfmt-graalvm
- https://github.com/lambdaisland/birch

Usage




Terminal
~/cljtree-graalvm $ time java -jar target/cljtree-*-standalone.jar src
src
└── cljtree_graalvm
    └── core.clj
2 directories, 1 files
1.38user 0.12system 0:00.98elapsed 153%CPU (0avgtext+0avgdata 94612maxresident)k
0inputs+64outputs (0major+19953minor)pagefaults 0swaps
~/cljtree-graalvm $
[0] 1:bash 2:vim 3:bash 4:bash* 5:bash- "jan-ubu-vbox" 13:30 08-May-18


Terminal
~/cljtree-graalvm $ time java -jar target/cljtree-*-standalone.jar src
src
└── cljtree_graalvm
    └── core.clj
2 directories, 1 files
1.38user 0.12system 0:00.98elapsed 153%CPU (0avgtext+0avgdata 94612maxresident)k
0inputs+64outputs (0major+19953minor)pagefaults 0swaps
~/cljtree-graalvm $ time ./cljtree src
src
└── cljtree_graalvm
    └── core.clj
2 directories, 1 files
0.00user 0.00system 0:00.01elapsed 90%CPU (0avgtext+0avgdata 25532maxresident)k
0inputs+0outputs (0major+2370minor)pagefaults 0swaps
~/cljtree-graalvm $
[0] 1:bash 2:vim 3:bash 4:bash* 5:bash- "jan-ubu-vbox" 13:31 08-May-18

GitHub - nicokosi/hubstats: Command-line for GitHub pull request statistics - Mozilla Firefox

GitHub - nicokosi/hubsta

GitHub, Inc. (US) https://github.com/nicokosi/hubstats

hubstats build passing

`hubstats` is a command line tool to compute statistics for GitHub pull requests.

Prerequisite

Install leiningen.

Usage

```
lein run --organization $organization --repository $repository
```

If an access token is required:

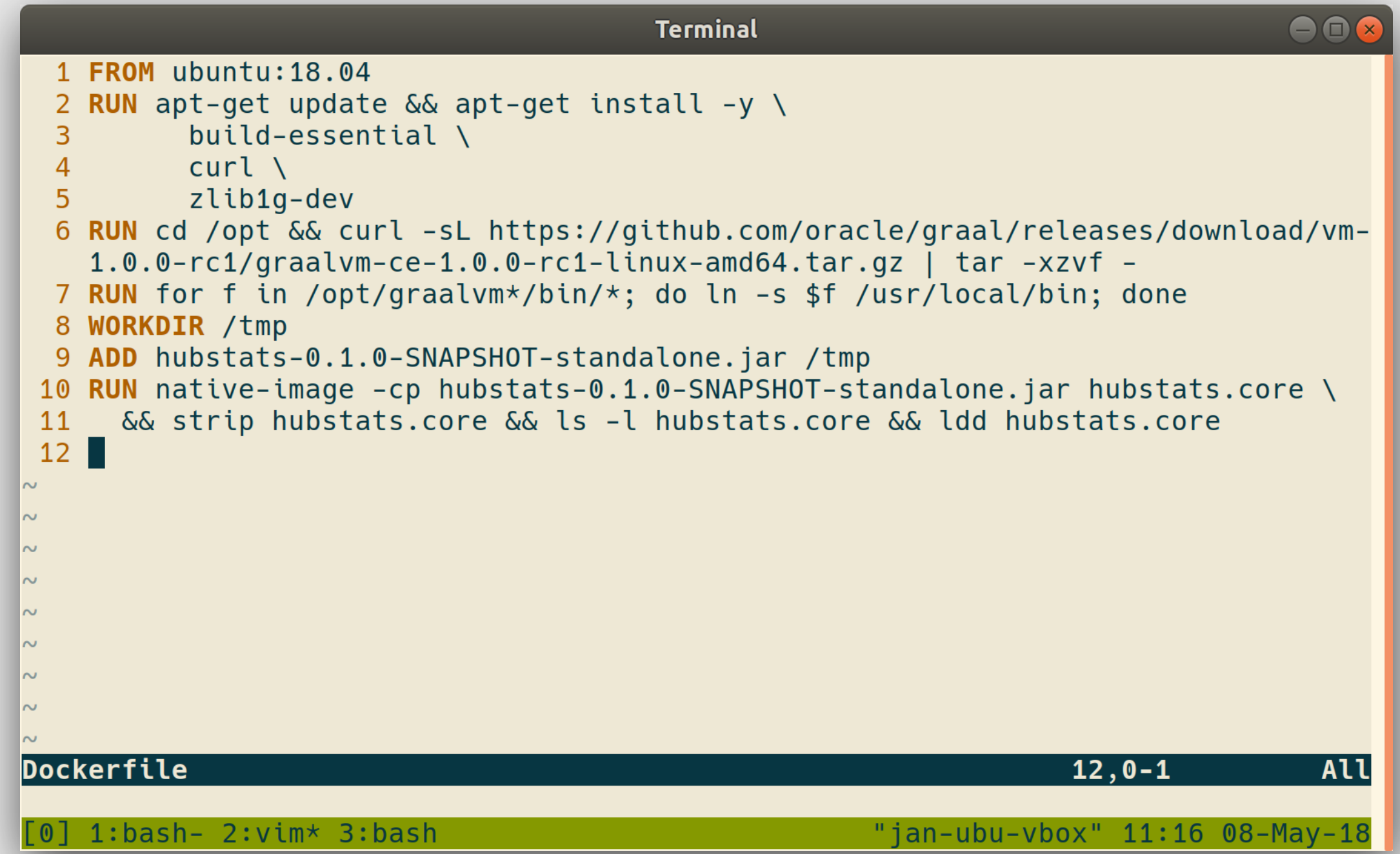
```
lein run --organization $organization --repository $repository --token $token
```

Output example:

```
pull requests for softwarevidal/arthur ->
```

Terminal

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y \
      build-essential \
      curl \
      zlib1g-dev
RUN cd /opt && curl -sL https://github.com/oracle/graal/releases/download/vm-1.0.0-rc1/graalvm-ce-1.0.0-rc1-linux-amd64.tar.gz | tar -xzvf -
RUN for f in /opt/graalvm*/bin/*; do ln -s $f /usr/local/bin; done

~
~
~
~
~
~
~
~
~
~
~
~
Dockerfile                                    8,0-1          All

[0] 1:bash- 2:vim* 3:bash              "jan-ubu-vbox" 11:15 08-May-18
```

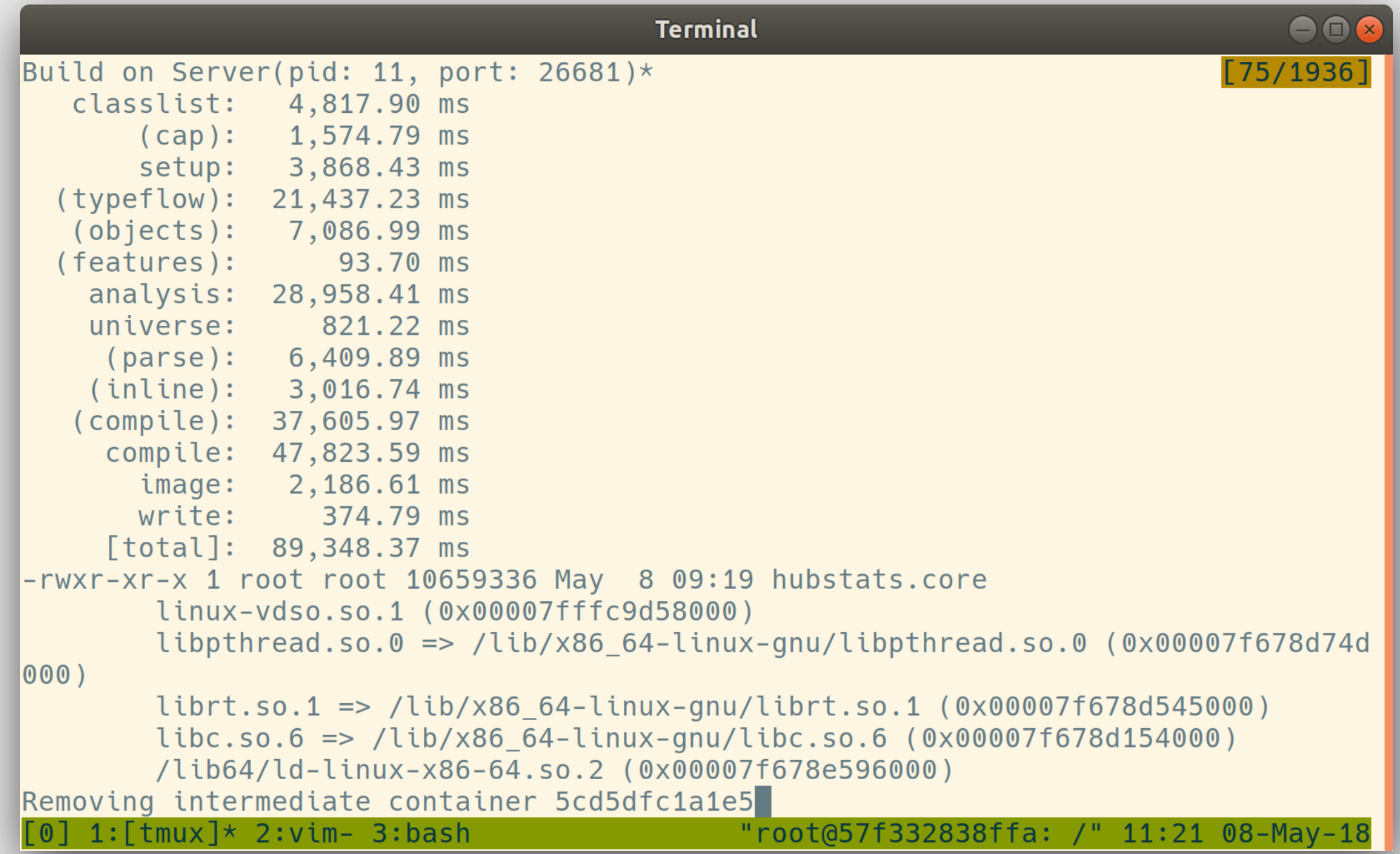

Terminal
1 FROM ubuntu:18.04
2 RUN apt-get update && apt-get install -y \
3 build-essential \
4 curl \
5 zlib1g-dev
6 RUN cd /opt && curl -sL https://github.com/oracle/graal/releases/download/vm-
1.0.0-rc1/graalvm-ce-1.0.0-rc1-linux-amd64.tar.gz | tar -xzvf -
7 RUN for f in /opt/graalvm*/bin/*; do ln -s $f /usr/local/bin; done
8 WORKDIR /tmp
9 ADD hubstats-0.1.0-SNAPSHOT-standalone.jar /tmp
10 RUN native-image -cp hubstats-0.1.0-SNAPSHOT-standalone.jar hubstats.core \
11 && strip hubstats.core && ls -l hubstats.core && ldd hubstats.core
12
Dockerfile 12,0-1 All
[0] 1:bash- 2:vim* 3:bash "jan-ubu-vbox" 11:16 08-May-18

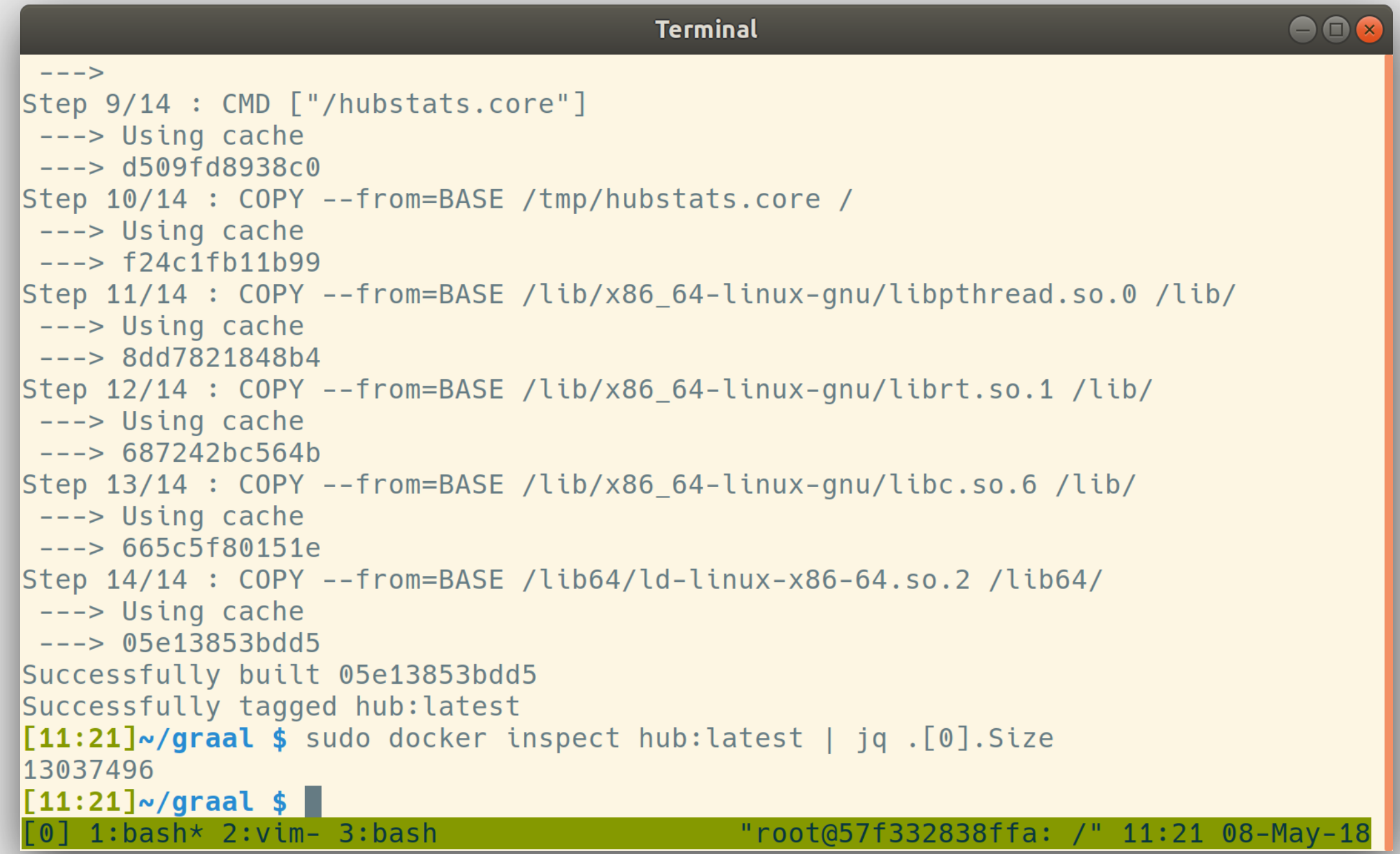
Terminal

```
 1 FROM ubuntu:18.04 AS BASE
 2 RUN apt-get update && apt-get install -y \
 3       build-essential \
 4       curl \
 5       zlib1g-dev
 6 RUN cd /opt && curl -sL https://github.com/oracle/graal/releases/download/vm-
   1.0.0-rc1/graalvm-ce-1.0.0-rc1-linux-amd64.tar.gz | tar -xzvf -
 7 RUN for f in /opt/graalvm*/bin/*; do ln -s $f /usr/local/bin; done
 8 WORKDIR /tmp
 9 ADD hubstats-0.1.0-SNAPSHOT-standalone.jar /tmp
10 RUN native-image -cp hubstats-0.1.0-SNAPSHOT-standalone.jar hubstats.core \
11   && strip hubstats.core && ls -l hubstats.core && ldd hubstats.core
12
13 FROM scratch
14
~
~
~
~
~
~
~
~
Dockerfile                                                    14,0-1          All

[0] 1:bash- 2:vim* 3:bash                              "jan-ubu-vbox" 11:17 08-May-18
```

Terminal

```
 1 FROM ubuntu:18.04 AS BASE
 2 RUN apt-get update && apt-get install -y \
 3       build-essential \
 4       curl \
 5       zlib1g-dev
 6 RUN cd /opt && curl -sL https://github.com/oracle/graal/releases/download/vm-
   1.0.0-rc1/graalvm-ce-1.0.0-rc1-linux-amd64.tar.gz | tar -xzvf -
 7 RUN for f in /opt/graalvm*/bin/*; do ln -s $f /usr/local/bin; done
 8 WORKDIR /tmp
 9 ADD hubstats-0.1.0-SNAPSHOT-standalone.jar /tmp
10 RUN native-image -cp hubstats-0.1.0-SNAPSHOT-standalone.jar hubstats.core \
11   && strip hubstats.core && ls -l hubstats.core && ldd hubstats.core
12
13 FROM scratch
14
15 CMD ["/hubstats.core"]
16 COPY --from=BASE /tmp/hubstats.core /
17 COPY --from=BASE /lib/x86_64-linux-gnu/libpthread.so.0 /lib/
18 COPY --from=BASE /lib/x86_64-linux-gnu/librt.so.1 /lib/
19 COPY --from=BASE /lib/x86_64-linux-gnu/libc.so.6 /lib/
20 COPY --from=BASE /lib64/ld-linux-x86-64.so.2 /lib64/
21
Dockerfile                                                    21,0-1        All

[0] 1:bash- 2:vim* 3:bash                          "jan-ubu-vbox" 11:19 08-May-18
```


Terminal
Build on Server(pid: 11, port: 26681)*
[75/1936]
classlist: 4,817.90 ms
(cap): 1,574.79 ms
setup: 3,868.43 ms
(typeflow): 21,437.23 ms
(objects): 7,086.99 ms
(features): 93.70 ms
analysis: 28,958.41 ms
universe: 821.22 ms
(parse): 6,409.89 ms
(inline): 3,016.74 ms
(compile): 37,605.97 ms
compile: 47,823.59 ms
image: 2,186.61 ms
write: 374.79 ms
[total]: 89,348.37 ms
-rwxr-xr-x 1 root root 10659336 May 8 09:19 hubstats.core
linux-vdso.so.1 (0x00007fffc9d58000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f678d74d000)
librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f678d545000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f678d154000)
/lib64/ld-linux-x86-64.so.2 (0x00007f678e596000)
Removing intermediate container 5cd5dfc1a1e5
[0] 1:[tmux]* 2:vim- 3:bash
"root@57f332838ffa: /" 11:21 08-May-18


Terminal
 --->
Step 9/14 : CMD ["/hubstats.core"]
 ---> Using cache
 ---> d509fd8938c0
Step 10/14 : COPY --from=BASE /tmp/hubstats.core /
 ---> Using cache
 ---> f24c1fb11b99
Step 11/14 : COPY --from=BASE /lib/x86_64-linux-gnu/libpthread.so.0 /lib/
 ---> Using cache
 ---> 8dd7821848b4
Step 12/14 : COPY --from=BASE /lib/x86_64-linux-gnu/librt.so.1 /lib/
 ---> Using cache
 ---> 687242bc564b
Step 13/14 : COPY --from=BASE /lib/x86_64-linux-gnu/libc.so.6 /lib/
 ---> Using cache
 ---> 665c5f80151e
Step 14/14 : COPY --from=BASE /lib64/ld-linux-x86-64.so.2 /lib64/
 ---> Using cache
 ---> 05e13853bdd5
Successfully built 05e13853bdd5
Successfully tagged hub:latest
[11:21]~/graal $ sudo docker inspect hub:latest | jq .[0].Size
13037496
[11:21]~/graal $
[0] 1:bash* 2:vim- 3:bash "root@57f332838ffa: /" 11:21 08-May-18

As we can see above, Graal enables us to use Clojure to build small self-contained command-line tools. Let's see if it will also allow us to build an entire web application. In order to try this we'll need some extra dependencies.

```
(defproject webkv "0.0.0"
  :dependencies [[org.clojure/clojure "1.9.0"]
                 [bidi "2.1.3"]
                 [ring/ring-defaults "0.3.1"]
                 [http-kit "2.3.0"]]
  :aot [webkv.main]
  :main webkv.main)
```

Now, let's implement a simple key-value database backed by files in a temporary directory and expose it over HTTP.

```
$ native-image -H:+ReportUnsupportedElementsAtRuntime \
               -H:EnableURLProtocols=http webkv.main
...
[total]:  39,760.45 ms
$ du -h webkv.main
9.9M    webkv.main
$ ./webkv.main &
🔥 http://localhost:8080
$ curl http://localhost:8080/dict/abc -i -XPUT -d val="just testing"
HTTP/1.1 201 Created
Content-Type: application/octet-stream
Content-Length: 12
Server: http-kit
Date: Thu, 26 Apr 2018 15:43:49 GMT

just testing
$ curl http://localhost:8080/dict/abc -i
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 12
```

Native Clojure with GraalVM

April 28, 2018 — *18 minutes reading time*

JAN STĘPIEŃ

GraalVM is a fascinating piece of technology. This newly-released just-in-time compiler allows efficient execution and interoperability between various programming languages on top of the JVM. Its impact on the execution speed is stunning. Benchmarks comparing Graal-based TruffleRuby to other Ruby implementations give us a hint of Graal's potential. Speed benefits aside, the new compiler also allows us to translate our JVM bytecode into small self-contained native binaries.

Chris Seaton Follow

PhD in Ruby. Now at Oracle Labs VM research group. TruffleRuby lead. Captain in the Queen's Own Yeomanry reserve light cavalry. Opinions are my own.

Apr 25 · 20 min read

Top 10 Things To Do With GraalVM

There are a lot of different parts to GraalVM, so if you've heard the name before, or even seen some of our talks, there are for sure things that it can do that you don't know about yet. In this article we'll list some of the diverse features of GraalVM and show you what they can do for you.

CLOJURE BERLIN // 09.05.2018

Native Clojure with GraalVM in 38 screenshots

Jan Stępień

janstepien.com

INNOQ