SAND2022-6764C

Sandia National Laboratories

ShellLogger

Keeping Track of Python's Interactions with the Shell

Jason M. Gates, David Collins, Josh Braun

May 24, 2022

U.S. DEPARTMENT OF ENERGY | NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2022-#### #

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Motivation

```
curl -L http://github.com/Kitware/CMake/releases/download/v3.20.2/cmake-3.20.2.tar.gz | tar xzf -
cd cmake-3.20.2
./bootstrap --no-qt-gui --parallel=8 --prefix=$HOME/cmake-3.20.2
make -j8
make install
```

Motivation

```
curl -L http://github.com/Kitware/CMake/releases/download/v3.20.2/cmake-3.20.2.tar.gz > run.txt
cd cmake-3.20.2
./bootstrap --no-qt-gui --parallel=8 --prefix=$HOME/cmake-3.20.2
make -j8
make install
```

Could get you this:

```
 % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:02:06 --:--:--     0
curl: (7) Failed connect to github.com:80; Connection timed out
```

Because of this:

```
$ printenv http_proxy
$
```

Motivation

```
curl -L http://github.com/Kitware/CMake/releases/download/v3.20.2/cmake-3.20.2.tar.gz | tar xzf -
cd cmake-3.20.2
./bootstrap --no-qt-gui --parallel=8 --prefix=$HOME/cmake-3.20.2
make -j8
make install
```

Could get you this:

```
Error when bootstrapping CMake:
Cannot find a C++ compiler that supports both C++11 and the specified C++ flags.
Please specify one using environment variable CXX.
The C++ flags are "".
They can be changed using the environment variable CXXFLAGS.
See cmake_bootstrap.log for compilers attempted.
-----------------------------------------
Log of errors: /tmp/cmake-3.20.2/Bootstrap.cmk/cmake_bootstrap.log
-----------------------------------------
```

Because of this:

```
$ printenv CC
$ printenv CXX
icpc
```

Motivation

```
curl -L http://github.com/Kitware/CMake/releases/download/v3.20.2/cmake-3.20.2.tar.gz | tar xzf -
cd cmake-3.20.2
./bootstrap --no-qt-gui --parallel=8 --prefix=$HOME/cmake-3.20.2
make -j8
make install
```

What do we need?

- Environment variables

What else would be nice?

- Working directory
- Hostname
- User and group
- `umask`
- Return code
- `ulimit`
- Duration
- Resource usage (CPU, memory, disk)

Enter ShellLogger

Python module that keeps track of context in a HTML log file

- Command
- Description
- `stdout` and `stderr`
- Environment variables
- Working directory
- `hostname`
- User and group
- `umask`
- Return code
- `ulimit`
- Start time & duration
- (Optional) CPU, memory, and disk usage
- (Optional) `strace` or `ltrace`

Similar in principle to Unix `script` command

- with substantially more functionality

Motivation similar to Python's `logging` module

- but captures what's happening *in the shell* rather than in Python

Enter ShellLogger

Why Python? Why not bash?

- Community standards
  - Style: PEP8 (flake8, YAPF)
  - Documentation: Sphinx
  - Testing: pytest
- Easier to write, maintain, reuse, etc.
- Future developers likely more proficient with it
- Near-ubiquitous

Enter ShellLogger

Shell:

```
# Description of cmd1
cmd1

# Description of cmd2
cd /different/dir
cmd2
cd -
```

Python + ShellLogger:

(stdout and stderr to HTML only)

```
from shelllogger import ShellLogger
from Pathlib import Path
sl = ShellLogger("Name of Script", Path.cwd()/"log")
sl.log("Description of cmd1", cmd1)
sl.log("Description of cmd2", cmd2, Path("/different/dir"))
sl.finalize()
```

See `log/Name_of_Script.html` for results.

Enter ShellLogger

Shell:

```
# Description of cmd1
cmd1

# Description of cmd2
cd /different/dir
cmd2
cd -
```

Python + ShellLogger:

(stdout and stderr to HTML and console)

```
from shelllogger import ShellLogger
from Pathlib import Path
sl = ShellLogger("Name of Script", Path.cwd()/"log")
sl.log("Description of cmd1", cmd1, live_stdout=True, live_stderr=True)
sl.log("Description of cmd2", cmd2, Path("/different/dir"),
       live_stdout=True, live_stderr=True)
sl.finalize()
```

See log/Name_of_Script.html for results.

Enter ShellLogger

Shell:

```
# Description of cmd1
cmd1

# Description of cmd2
cd /different/dir
cmd2
cd -
```

Python + ShellLogger:

(stdout and stderr to HTML, capture resource usage)

```
from shelllogger import ShellLogger
from Pathlib import Path
sl = ShellLogger("Name of Script", Path.cwd()/"log")
sl.log("Description of cmd1", cmd1, measure=["cpu", "memory", "disk"])
sl.log("Description of cmd2", cmd2, Path("/different/dir"),
       measure=["cpu", "memory", "disk"])
sl.finalize()
```

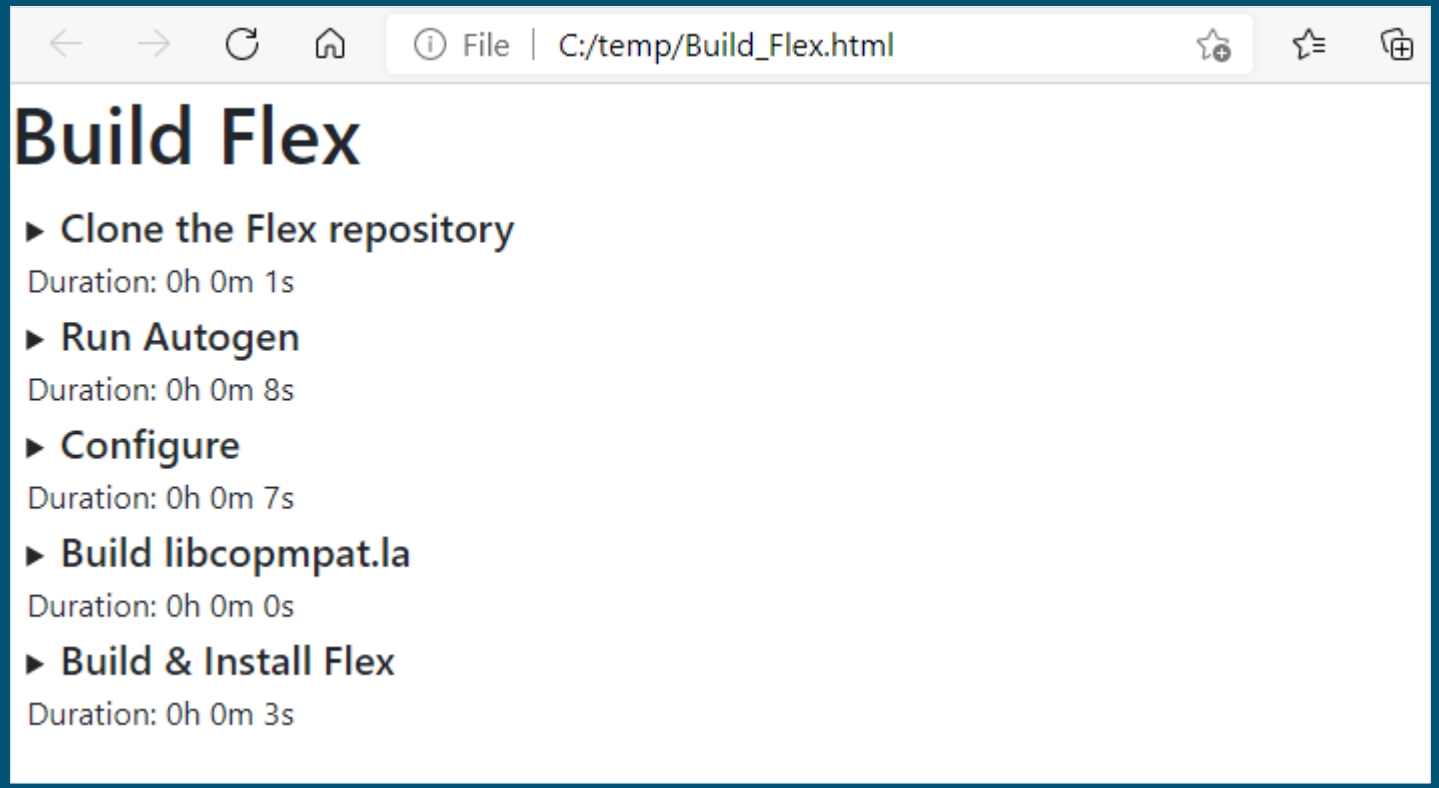
See log/Name_of_Script.html for results.

Example

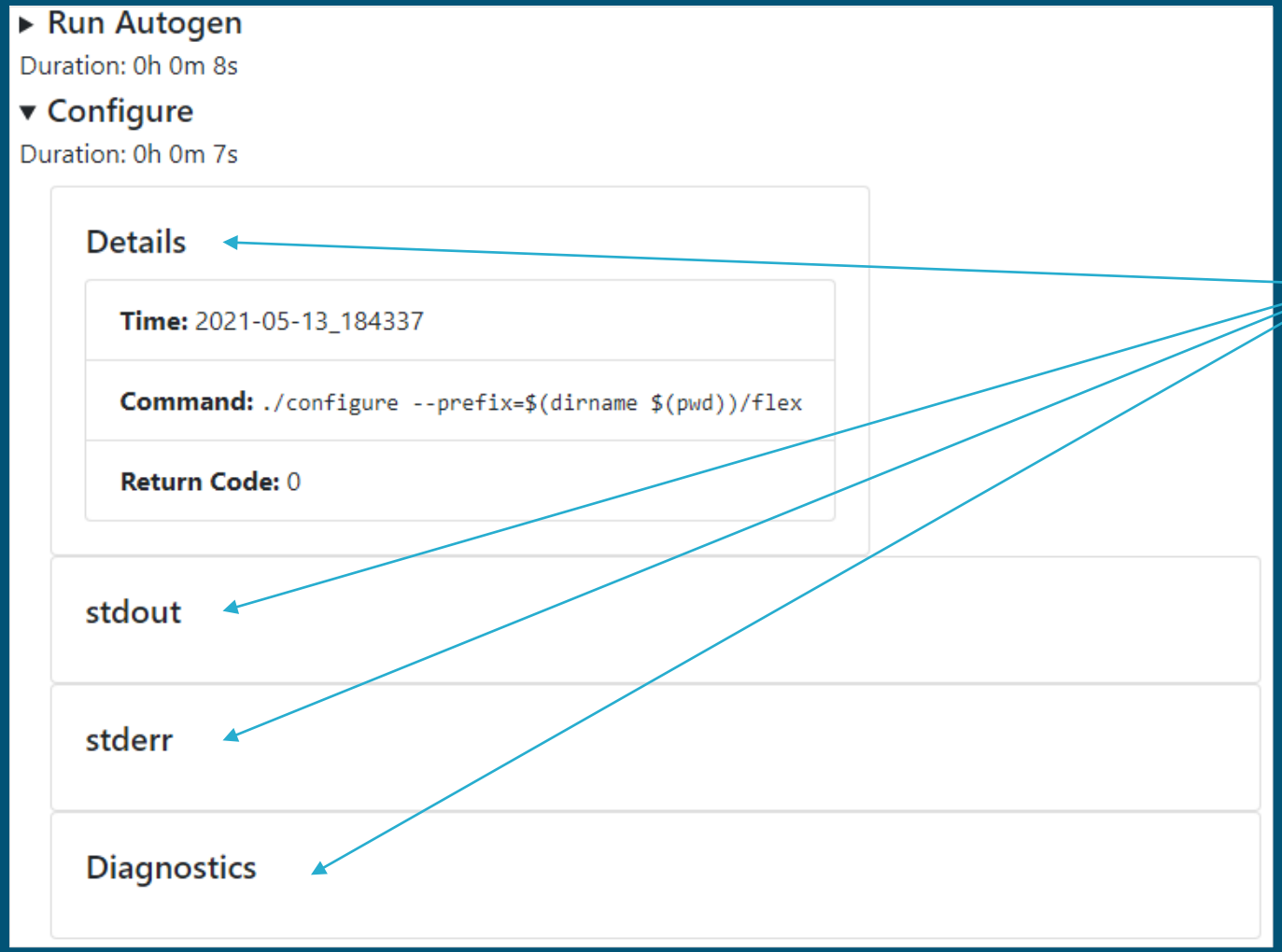
```
git clone --depth 1 --branch flex-2.5.39 https://github.com/westes/flex.git flex-2.5.39
cd flex-2.5.39 ; ./autogen.sh
./configure --prefix=$(dirname $(pwd))/flex
cd ./lib ; make libcompat.la
cd .. ; make install-exec
```

```
from shelllogger import ShellLogger
from pathlib import Path

sl = ShellLogger("Build Flex", Path.cwd()/"log")

sl.log("Clone the Flex repository",
       "git clone --depth 1 --branch flex-2.5.39 https://github.com/westes/flex.git flex-2.5.39",
       live_stdout=True, live_stderr=True)
sl.log("Run Autogen", "./autogen.sh", Path.cwd()/"flex-2.5.39",
       live_stdout=True, live_stderr=True)
sl.log("Configure", "./configure --prefix=$(dirname $(pwd))/flex", Path.cwd()/"flex-2.5.39",
       live_stdout=True, live_stderr=True, measure=["cpu", "memory", "disk"])
sl.log("Build libcopmpat.la", "make libcompat.la", Path.cwd()/"flex-2.5.39/lib",
       live_stdout=True, live_stderr=True, measure=["cpu", "memory", "disk"])
sl.log("Build & Install Flex", "make install-exec", Path.cwd()/"flex-2.5.39",
       live_stdout=True, live_stderr=True, measure=["cpu", "memory", "disk"])

sl.finalize()
```

What Do You Get?

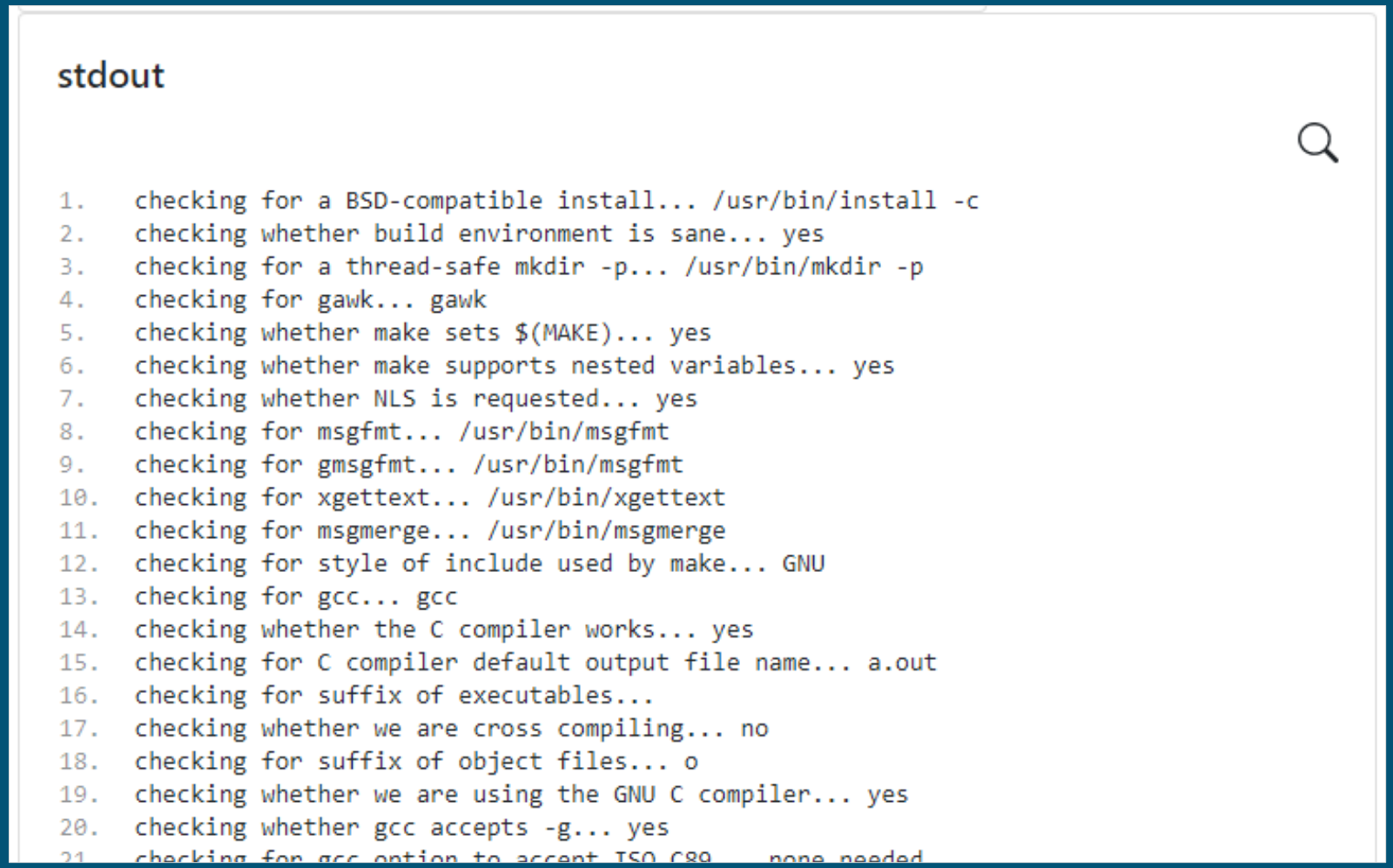


What Do You Get?



Click to expand for more details

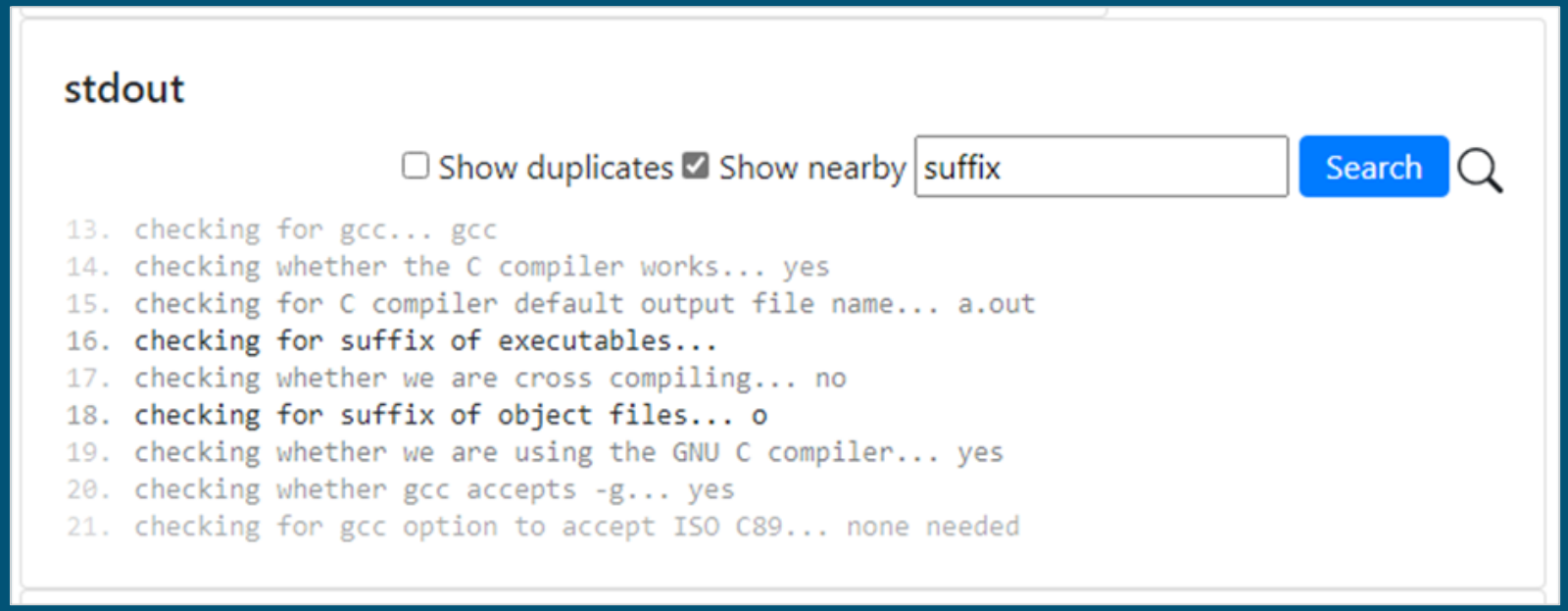
What Do You Get?



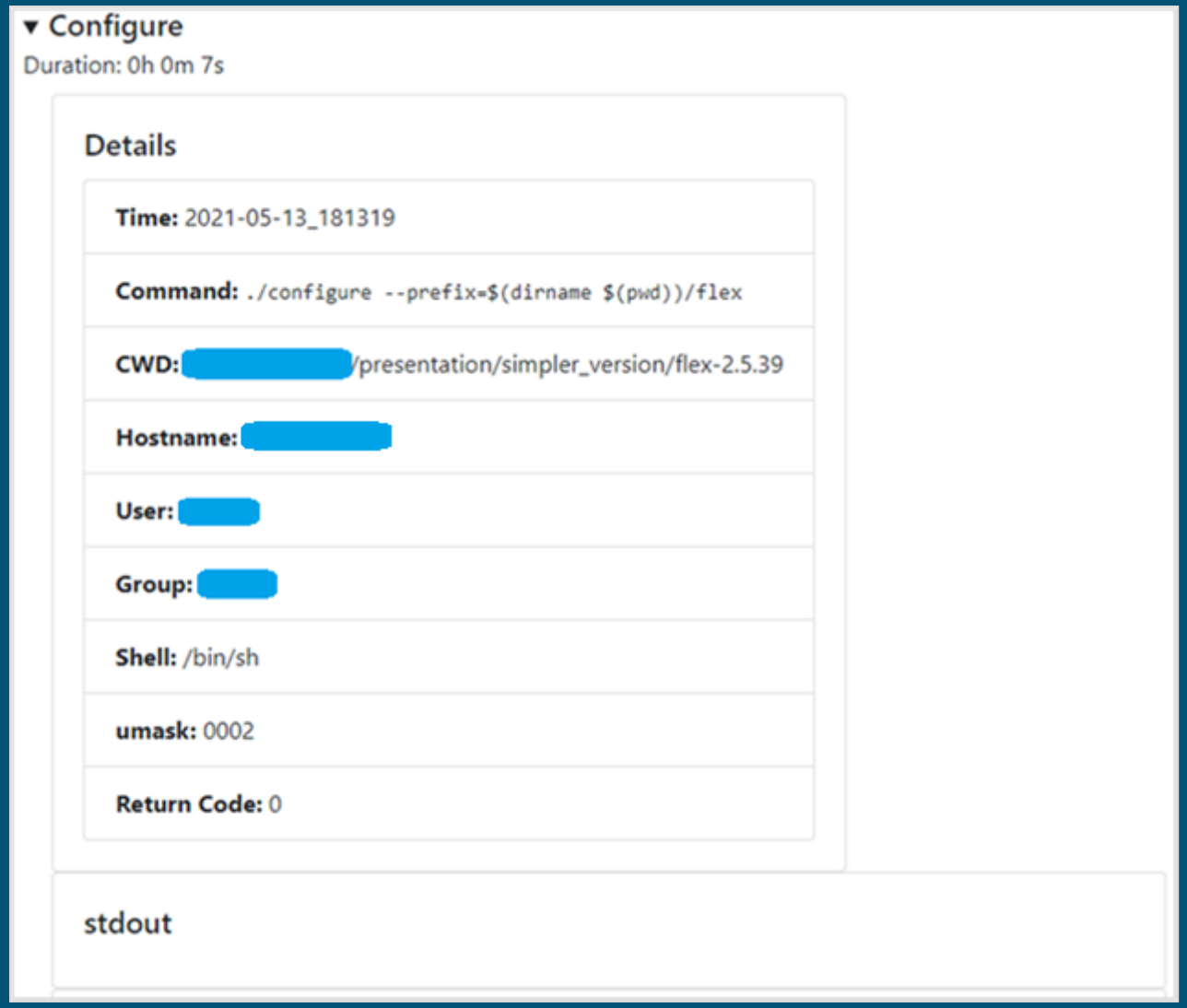
stdout

```
1.  checking for a BSD-compatible install... /usr/bin/install -c
2.  checking whether build environment is sane... yes
3.  checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
4.  checking for gawk... gawk
5.  checking whether make sets $(MAKE)... yes
6.  checking whether make supports nested variables... yes
7.  checking whether NLS is requested... yes
8.  checking for msgfmt... /usr/bin/msgfmt
9.  checking for gmsgfmt... /usr/bin/msgfmt
10. checking for xgettext... /usr/bin/xgettext
11. checking for msgmerge... /usr/bin/msgmerge
12. checking for style of include used by make... GNU
13. checking for gcc... gcc
14. checking whether the C compiler works... yes
15. checking for C compiler default output file name... a.out
16. checking for suffix of executables...
17. checking whether we are cross compiling... no
18. checking for suffix of object files... o
19. checking whether we are using the GNU C compiler... yes
20. checking whether gcc accepts -g... yes
21. checking for gcc option to accept ISO C89... none needed
```

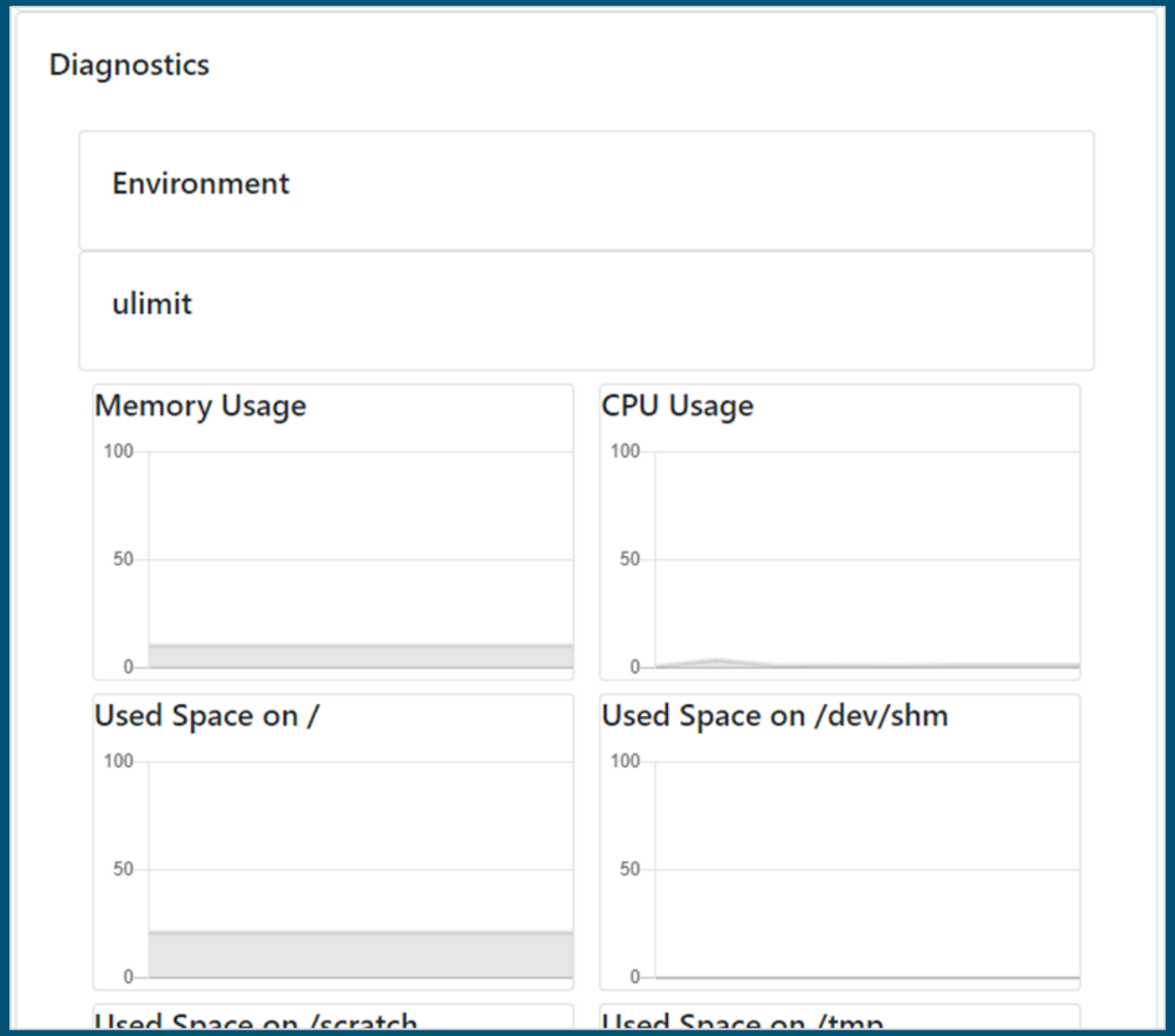
What Do You Get?



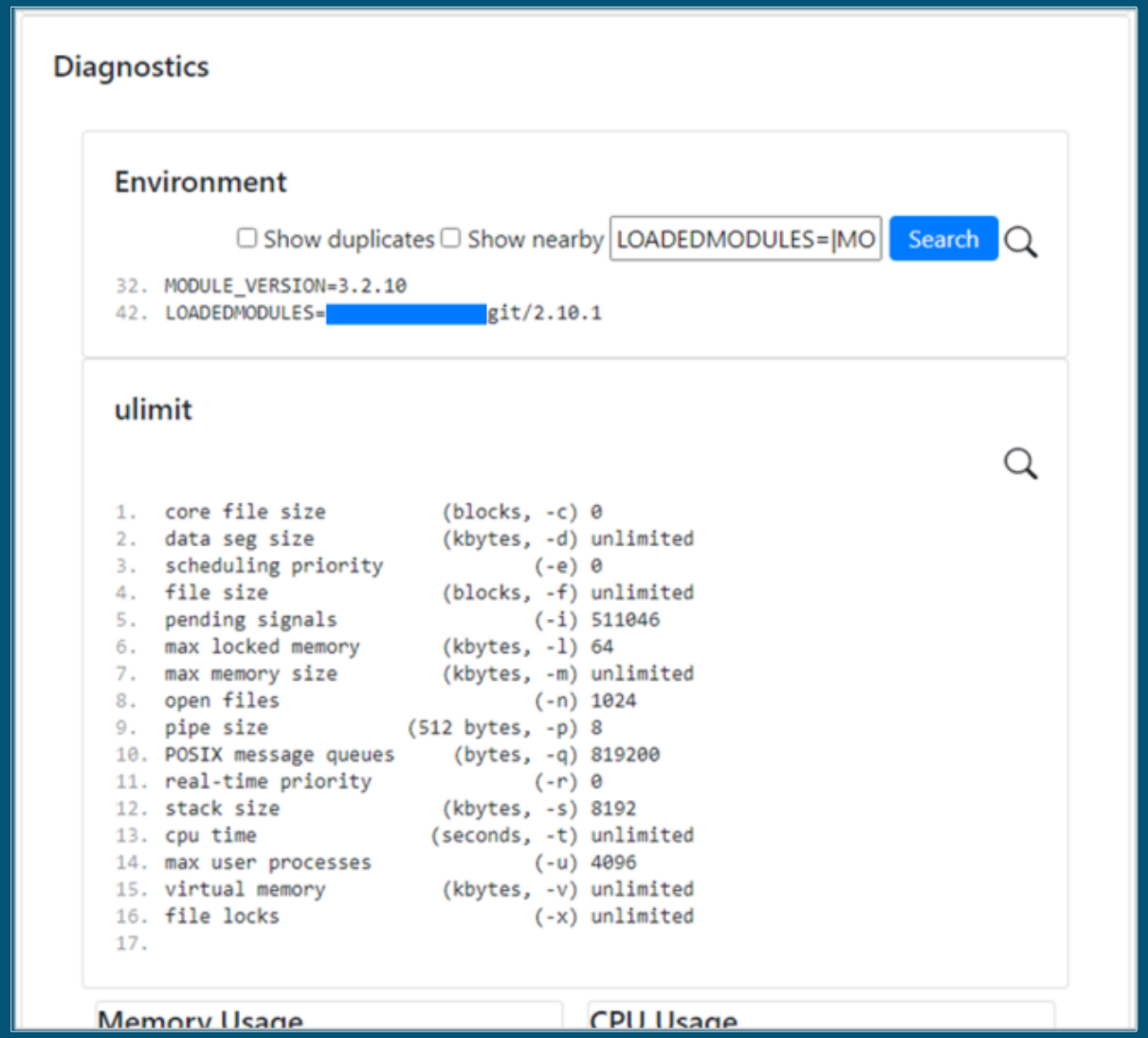
What Do You Get?



What Do You Get?



What Do You Get?



Diagnostics

Environment

Show duplicates Show nearby LOADEDMODULES=|MO Search

```
32. MODULE_VERSION=3.2.10
42. LOADEDMODULES=[redacted]git/2.10.1
```

ulimit

```
1.  core file size          (blocks, -c) 0
2.  data seg size           (kbytes, -d) unlimited
3.  scheduling priority             (-e) 0
4.  file size               (blocks, -f) unlimited
5.  pending signals                 (-i) 511046
6.  max locked memory       (kbytes, -l) 64
7.  max memory size         (kbytes, -m) unlimited
8.  open files                      (-n) 1024
9.  pipe size            (512 bytes, -p) 8
10. POSIX message queues     (bytes, -q) 819200
11. real-time priority              (-r) 0
12. stack size              (kbytes, -s) 8192
13. cpu time               (seconds, -t) unlimited
14. max user processes              (-u) 4096
15. virtual memory          (kbytes, -v) unlimited
16. file locks                      (-x) unlimited
17.
```

Memory Usage CPU Usage

What Do You Get?

In this example:

- Command
- Description
- Duration
- Return code, user, group, `umask`, working directory, `hostname`, shell
- Regex-searchable `stdout` and `stderr`
- Regex-searchable environment variable list
- `ulimit`
- CPU, memory, and disk usage

Other Features

`strace`/`ltrace` with expression filters

- e.g., `execve`, `getenv`

Statefulness

- `Logger.log` affects later `Logger.log` calls (e.g., `export VAR=VALUE`)
- Unlike Python's `os.system` or `subprocess.run`

`Logger.log` returns things in Python

- A `dict` including `return_code`, `stdout`, `stderr`

Child loggers via `Logger.add_child`

- Independent shells
- Hierarchical HTML pages

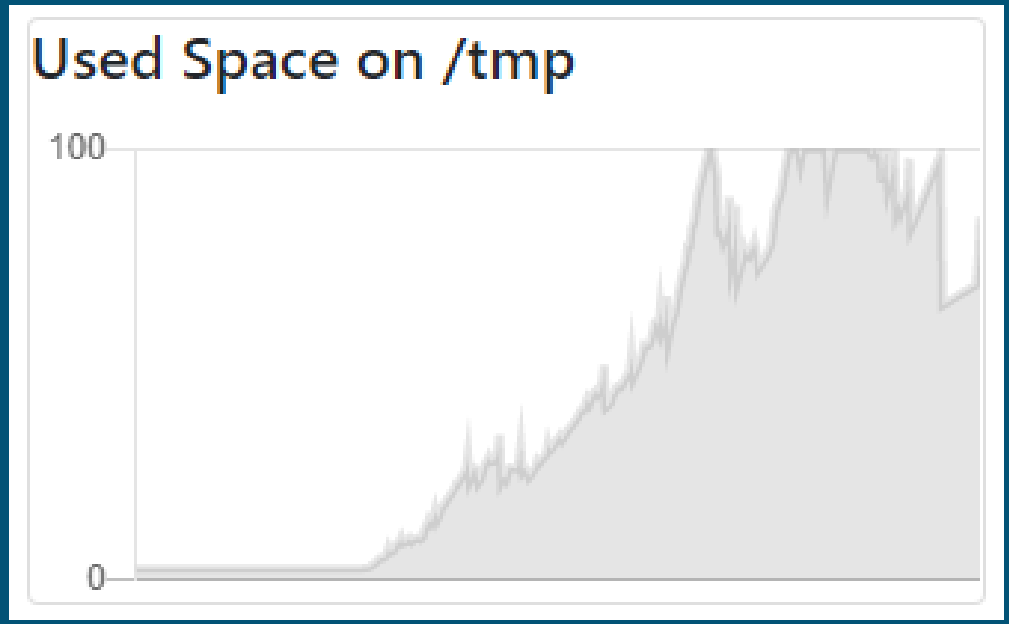
What Problems Have We Solved with ShellLogger?

Mysterious "No space left on device" in the middle of a make

```
$ make -k -j 32
...
No space left on device
FATAL ERROR: fwrite on file failed

$ df -h $(pwd)
Filesystem          Size  Used Avail Use%  Mounted on
srv3:/shares/home    25T   15T   10T  60%  /home
```

Why? Space used on /tmp

- Solved by setting TMPDIR before compiling with Intel or CUDA



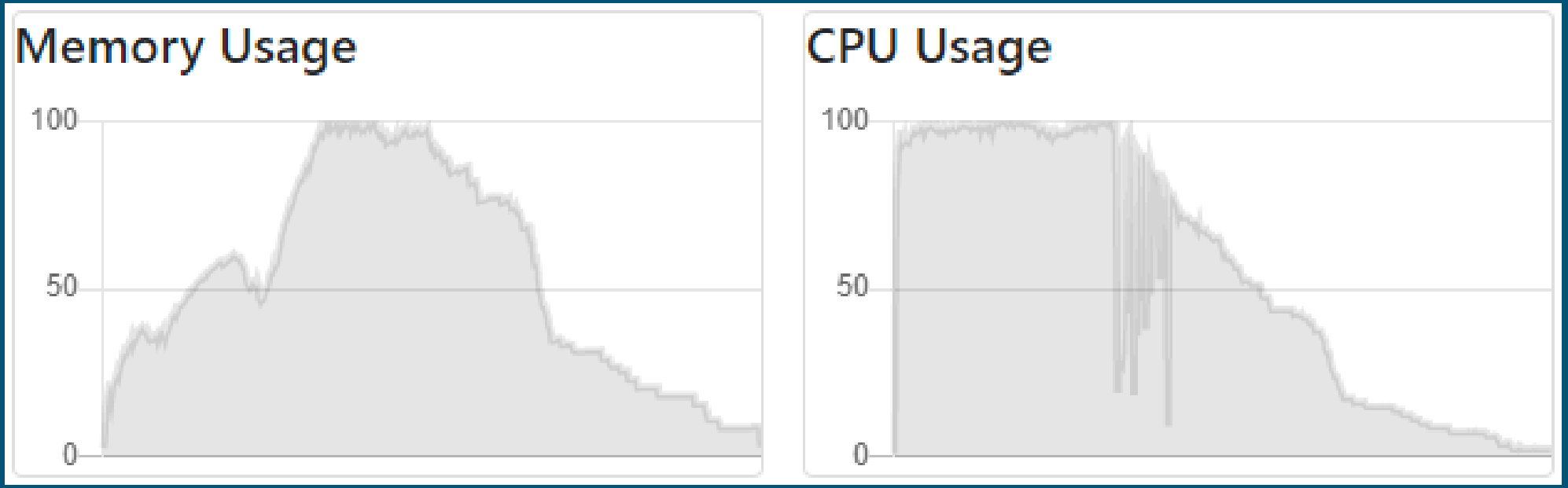
What Problems Have We Solved with ShellLogger?

Random terminations of the compiler in the middle of a make

```
make -k -j 80
...
icpc: error #10106: Fatal error in /path/to/mcpcom, terminated by kill signal
```

Why? Memory usage

- needed to decrease `-j`



Release

Where to find ShellLogger?

- Currently internal to Sandia National Laboratories
- Working on InnerSourcing
- Hope to open-source within the next year

Questions?