Genspio: Generate Your POSIX Shell Garbage

Sebastien Mondet (@smondet)

OCaml 2017 Workshop, Sep 8, 2017.
We're a team of software developers and data scientists working to understand and improve how the immune system battles cancer.

We occasionally blog about our work. Please contact us if you're interested in one of the jobs we have available!

We are grateful to the Icahn School of Medicine at Mount Sinai, the Parker Institute for Cancer Immunotherapy, and Neon Therapeutics for funding our work.
Work

Papers

- Contribution of systemic and somatic factors to clinical response and resistance to PD-L1 blockade in urothelial cancer: An exploratory multi-omic analysis
  published in PLOS Medicine
- Somatic Mutations and Neoepitope Homology in Melanomas Treated with CTLA-4 Blockade
  published in Cancer Immunology Research
- pileup.js: a JavaScript library for interactive and in-browser visualization of genomic data
  published in Bioinformatics
- Using a Machine Learning Approach to Predict Outcomes after Radiosurgery for Cerebral Arteriovenous Malformations
  published in Nature Scientific Reports
- How Will Big Data Improve Clinical and Basic Research in Radiation Therapy?
  published in the International Journal of Radiation Oncology, Biology, Physics
- Mutation-Derived Tumor Antigens: Novel Targets in Cancer Immunotherapy
  published in ONCOLOGY
Computational Cancer Immunotherapy

- Run big computational pipelines.
  - Servers with WebUIs, databases.
  - HPC scheduling (Torque, YARN, Google Cloud, AWS, …).
- Deal with precious human data.
  - HDFS, (broken) disks, S3, Gcloud Buckets, NFSs.
- Interactive exploration.
  - Direct access for the users (IPython, R, `awk | wc`, …).
Need to setup local/cloud/datacenter-ish infrastructure for the lab.

It’s nobody’s job.

Nothing seems there for the “long term.”

Make composable tools that allow people to setup/monitor/clean-up their own infrastructure.

(and it’s more fun, and a better use of software people’s time)
It always looks simple at first …

```ocaml
Unix.execv "/usr/bin/apt-get" [] "apt-get";"install"; "-y"; "postgresql" []

let cmd =
["apt-get";"install"; "-y"; "postgresql"]
|> List.map ~f:Filename.quote
|> String.concat ~sep:" 

in
Unix.execv "/usr/bin/ssh" [] "ssh"; host_info ; cmd []
```

Who failed? ssh or apt-get?
Ketrew’s SSH Call

```haskell
(** Strong version of an SSH call, trying to be like [Unix.exec].
It “stores” the value of ["$?"], in the stderr channel
enclosing the error log of the actual command between (hopefully) unique
strings.

It calls the command (list of strings, [argv]-like) with [exec]
inside a sub-shell, and escapes all the arguments with [Filename.quote].

Then it forces the “script” to return ['0'], if the overall execution of
the whole SSH command does not return ['0'], we know that the problem
is with the SSH call, not the command.

')

let generic_ssh_exec ssh command =
    let unique_tag = Unique_id.create () in
    let spiced_cmd =
        fmt "echo -n %s >&2 ; \
        (exec %s) ;
        echo -n %s$ >&2 ;
        exit 0"
        unique_tag
        (List.map command -f:(Filename.quote) |> String.concat ~sep:" ")
    unique_tag
    in
    let ssh_exec = do_ssh ssh spiced_cmd in
    let parse_error_log out err =
        let fail_parsing msg = fail (`Ssh_failure (`Wrong_log msg, err)) in
        let pieces = String.split ~on:`String unique_tag err in
        match pieces with
        | "" ; actual_stderr ; return_value ; [] ->
            begin match Int.of_string (String.strip return_value) with
            | Some r -> return (out, actual_stderr, r)
            | None ->
                fail_parsing (fmt "Return value not an integer: %S" return_value)
            end
        | something_else -> fail_parsing "Cannot parse error log"
```
:facepalm: after :facepalm:

FastQC makes the OpenJDK9 seg-fault #283

smondet opened this issue on Jun 3, 2016 · 1 comment

smondet commented on Jun 3, 2016

By default on Ubuntu we get the OpenJDK runtime, and FastQC makes it segfault:

Analysis complete for normal.chr28-b2Fq-PE_R1.Tastq
#
# A fatal error has been detected by the Java Runtime Environment:
#
# SIGSEGV (0xb) at pc=9x00007ff1d24b7009, pid=26995, tid=27037
#
# JRE version: OpenJDK Runtime Environment (9.0) (build 9-internal-9-2016-04-14-195246.build6)
# Java VM: OpenJDK 64-Bit Server VM (9-internal-9-2016-04-14-195246.build6.src, mixed mode, t
# Problematic frame:
#  C [libjava.so=0x1d000] JNU_GetEnv+6x19
#
# Core dump will be written. Default location: Core dumps may be processed with "/usr/share/a
# An error report file with more information is saved as:
# /home/sebastienmondet/hw.err.pid26995.log
#
# If you would like to submit a bug report, please visit:
# http://bugreport.java.com/bugreport/crash.jsp
# The crash happened outside the Java Virtual Machine in native code.
# See problematic frame for where to report the bug.
#
So I go Oracle's JRE:
(cf. this doc):
DevOps 101: Install The Oracle JDK

Everybody ends-up reading some Stack-overflow answer

If OpenJDK/OpenJRE works fine for you, I recommend using that package instead as suggested by @SAM. However, some software really requires Oracle’s JDK/JRE. This answer is how to silence the license question with the Oracle package from the PPA.

First, let’s recognize the question asked is a *feature* of the package, created by the developer.

```
 oracle-java7-installer (7u7-0-webupd8-4) maverick; urgency=medium
     * removed cookie file use or *else* the PPA stays disabled
     * request the user to accept the Oracle license before installation
     -- Alin Andrei <webupd8@gmail.com> Tue, 04 Sep 2012 14:18:29 +0200
```

As @Nate indicated in his answer, there should be a silent option. And there is. Do this before installing it:

```
$ echo debconf shared/accepted-oracle-license-v1-1 select true | \
  sudo debconf-set-selections
$ echo debconf shared/accepted-oracle-license-v1-1 seen true | \
  sudo debconf-set-selections
```

This sets the value of the debconf key to true, but also marks it as seen by the user. Now this question should not appear!

**How did I find this?**

In the source of the package, I tracked this down in the `oracle-java7-installer.preinst` file:

```
license=oracle-license-v1-1

# snip

db_get shared/accepted-$license
if [ "$RET" = "true" ]; then
    echo "$license license has already been accepted" >&2
    exit 0
fi
```
It’s all strings after all:

```bash
# The hard-one Oracle's Java 7
RUN sudo add-apt-repository --yes ppa:webupd8team/java
RUN sudo apt-get update
RUN http://askubuntu.com/questions/190582/installing-java-automatically-with-silent-option
RUN sudo bash -c "echo debconf shared/accepted-oracle-license-v1-1 select true | debconf-set-selections"
RUN sudo bash -c "echo debconf shared/accepted-oracle-license-v1-1 seen true | debconf-set-selections"
RUN sudo bash -c "DEBIAN_FRONTEND=noninteractive apt-get install --yes --allow-unauthenticated oracle-java7-installer"
```
What Could Go Wrong?

gcloud compute create deprecates the already dysfunctional --wait option

```python
module Shell_commands = struct

    let wait_until_ok ?(attempts = 10) ?(sleep = 10) cmd =
    (* Hackish way of waiting for an SSH server to be ready: *)
    sprintf "for count in $(seq 1 \$d); do\n\n    sleep \$d\n\n    echo "Attempt $count"\n\n    \%s && break || echo 'Attempt FAILED'\n\n    done"
    attempts sleep cmd

end
```
Write Once – Debug Everywhere™

`sudo` in some Debian version **erases** new lines …
Typed/Functional Step Back

1. Start writing simple combinators.
2. Add more typing info.
3. Hit portability / representation problems.
4. Go full-blown EDSL that compiles to pure POSIX shell.
Genspio 0.0.0

- Simple, typed EDSL
- `Language.t` is a 30+ entry GADT.
  - Boolean, Integer arithmetic + `to_string/of_string` + (very) basic lists.
  - `if-then-else`, loops.
  - `exec`.
  - Redirects, pipes, and captures.
  - Basic exception-like jumping.

- Compiler to POSIX shell.
  - Either one-liners, or multi-line scripts.
  - Unreadable output *by default*, but tries to do better when it statically knows.
Examples

```ocaml
let username_trimmed : string t =
    (* The usual shell-pipe operator is ||>,
     output_as_string takes stdout from a unit t as a string t. *)
    (exec ["whoami"] ||> exec ["tr"; "-d"; \"\n\"] |> output_as_string)
```
with_failwith (fun error_function ->
    let get_user = (* the contents of `$USER`: *)
        getenv (string "USER")
    in
    (* The operator `=$=` is `string t` equality, it returns a `bool t` that we can use with `if_seq`: *)
    if_seq
        (get_user =$= username_trimmed)
    ~t:[ (* more commands *) ]
    ~e:[
        (* `$USER` is different from `whoami`, system is broken, we exit using the failwith function: *)
        error_function
            ~message:(string "I'm dying") ~return:(int 1)
    ])

Now Jump!
let cli_spec = Command_line.Arg.
  (string ~doc:"The URL to the stuff" [-u]; "--url"
   ~default:no_value
  & flag ["-c"; "--all-in-tmp"] ~doc:"Do everything in the temp-dir"
  & string ["-f"; "--local-filename"]
   ~doc:"Override the downloaded file-name"
   ~default:no_value
  & string ["-t"; "--tmp-dir"]
   ~doc:"Use <dir> as temp-dir"
   ~default:(Genspio.EDSL.string "/tmp/genspio-downloader-tmpdir")
  & usage "Download archives and decrypt/unarchive them.
   ./downloader -u URL [-c] [-f <file>] [-t <tmpdir>]
  ) in
Command_line.parse cli_spec
begin fun ~anon url all_in_tmp filename_ov tmp_dir ->
let on_stdin_lines ~body =
    let fresh =
        sprintf "var_%d_%s" Random.(int 10_000)
        (Genspio.Language.to_one_liner (body (string "bouh"))
            |> Digest.string |> Digest.to_hex)
    in
    loop_while (exec ["read"; "-r"; fresh] |> succeeds)
    ~body:(seq [
        exec ["export"; fresh];
        body (getenv (string fresh));
    ])

smondet/habust/.../main.ml#L29-38
Nice Call

(* ... *)

```ml
exec ["ldd"; exe]
||> exec ["awk"; 
{ if ( $2 ~ /=>/ ) { print $3 } else { print $1 } }]
||> on_stdin_lines begin fun line ->
    seq [
        call [string "printf"; string "Line %s\n"; line];
        call [string "cp"; line; string ("/tmp" // basename)];
    ]
end
```

smondet/habust/.../main.ml#L196-203
That’s when “crazy” really means “insane.”

```
| Output_as_string e ->
  sprintf "\"$( { %s ; } | od -t o1 -An -v | tr -d ' \n' )\"" (continue e)
```

Vs

```
let expand_octal s =
  sprintf
    {sh| printf -- "$(printf -- '%%s' %s | sed -e 's/\(.*\)
 experimented/\1' | sed -e 's/\(.*\)/\3/g')" |sh}
  s in
```
let to_argument varprefix =
let argument ?declaration ?variable_name argument =
(* ... *)

function
| `String (Literal (Literal.String s)) when Literal.String.easy_to_escape s ->
  argument (Filename.quote s)
| `String (Literal (Literal.String s)) when
  Literal.String.impossible_to_escape_for_variable s ->
  ksprintf failwith "to_shell: sorry literal %S is impossible to escape as `exec` argument" s
| `String v ->
  let variable_name = Unique_name.variable varprefix in
  let declaration =
    sprintf "%s=$(%s; printf 'x')" variable_name (continue v |> expand_octal) in
  argument ~variable_name ~declaration
  (sprintf "" variable_name)

Future work: 2 string types …
In the beginning there was UNIX …

```c
#include <stdio.h>

int main (int argc, char *argv[])
{
    /* Insert VULN Here */
}
```
Testing, Locally

Test tries all the shells it knows about on the current host:

Summary:

* Test "dash" (`'dash' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 0 / 190 failures
  - time: 13.31 s.
  - version: "Version: 0.5.8-2.1ubuntu2".

* Test "bash" (`'bash' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 0 / 190 failures
  - time: 23.37 s.
  - version: "GNU bash, version 4.3.46(1)-release (x86_64-pc-linux-gnu)".

* Test "sh" (`'sh' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 0 / 190 failures
  - time: 13.59 s.
  - version: "".

* Test "busybox" (`'busybox' 'ash' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 0 / 190 failures
  - time: 8.80 s.
  - version: "BusyBox v1.22.1 (Ubuntu 1:1.22.0-15ubuntu1) multi-call binary.".

* Test "ksh" (`'ksh' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 20 / 190 failures
  - time: 14.78 s.
  - version: "version sh (AT&T Research) 93u+ 2012-08-01".
  - Cf. `/tmp/genspio-test-ksh-failures.txt`.

* Test "mksh" (`'mksh' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 2 / 190 failures
  - time: 25.56 s.
  - version: "Version: 52c-2".
  - Cf. `/tmp/genspio-test-mksh-failures.txt`.

* Test "posh" (`'posh' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 2 / 190 failures
  - time: 24.40 s.
  - version: "Version: 0.12.6".
  - Cf. `/tmp/genspio-test-posh-failures.txt`.

* Test "zsh" (`'zsh' '-x' '-c' '<command>' ' -- ' '<arg1>' '<arg2>' '<arg-n>'
  - 20 / 190 failures
  - time: 17.94 s.
  - version: "zsh 5.1.1 (x86_64-ubuntu-linux-gnu)".
  - Cf. `/tmp/genspio-test-zsh-failures.txt`.

All "known" shells were tested ☺
Testing: FreeBSD/SSH

```
export add_shells="Freebsd-gcloud, escape, <cmd>, printf '%s' <cmd> | ssh -i ~/.ssh/google_compute_engine $(freebsd_ip_address) 'sh -x'"

export only_dash=true # We don't run all the other local tests this time
export single_test_timeout=50
_build/src/test/genspio-test.byte

We get the usual report:

* Test "Freebsd-gcloud" (\`printf '%s' 'askjdeidjiedjjdjekdijdeijjidejdlks ( ) { <command> ; } ; askj
  - 0 / 190 failures
  - time: 165.19 s.
  - version: \`"Command-line"\`.}
Testing: OpenWRT/Qemu/SSH

```bash
qemu-system-arm -M realview-pbx-a9 -m 1024M \\ -kernel openwrt-realview-vmlinux.elf \\ -net nic \\ -net user,hostfwd=tcp::10022-:22 \\ -nographic \\ -sd openwrt-realview-sdcard.img \\ -append "console=ttyAMA0 verbose debug root=/dev/mmcblk0p1"
```

root@OpenWrt:/# df -h

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>Size</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/root</td>
<td>46.5M</td>
<td>2.9M</td>
<td>42.7M</td>
<td>6%</td>
<td>/</td>
</tr>
<tr>
<td>tmpfs</td>
<td>378.1M</td>
<td>612.0K</td>
<td>377.5M</td>
<td>0%</td>
<td>/tmp</td>
</tr>
<tr>
<td>tmpfs</td>
<td>512.0K</td>
<td>0</td>
<td>512.0K</td>
<td>0%</td>
<td>/dev</td>
</tr>
</tbody>
</table>

* Test "OpenWRT-qemu-arm" (`printf '%s' 'askjdeidjjdjjjdejjkekjdeijjidejdejklksi () { <command> ; } ; as' |
  - 0 / 190 failures |
  - time: 800.90 s. |
  - version: "Command-line".`
Example of Rabbit Hole

For a given shell, trying:

```bash
$shell -c ' exec 4>&3 ; echo "Exec-returns: $?"' ; echo "Shell-returns: $?"
```

The POSIX ones:

- shell=dash, shell=sh, shell='busbox ash': Shell-returns: 2
- shell=ksh, shell=mksh: Shell-returns: 1

The non-POSIX ones:

- shell=bash, shell=zsh: Exec-returns: 1 Shell-returns: 0

→ even bash not always POSIX.
Secotrec

*Real-world* example.

▶ Library of Hammerlab-like deployment lego-bricks.
  ▶ Ketrew, Coclobas, NGinx auth, TLS tunnel
  ▶ Let’s Encrypt, GCloud DNS, …
  ▶ “Interactive exploration containers.”
  ▶ Kubernetes/AWS-Batch clusters.
  ▶ Take down everything, restart partially …

▶ With pre-assembled (but configurable) “examples” for GCloud, AWS, and “Local-docker” standard setups.

https://github.com/hammerlab/secotrec
Got to “scale” Genspio:

- Quickly hitting max length of command line argument.
- “Standard Library” that may merge into Genspio.
- Integration with docker-compose.

“GCPocalypse:”

- Too easy for users to setup their own infrastructure.
- Forgetful about cleaning up.
- → our benefactor said it’s too much
- Fast move of all ops back to local infrastructure.
Simple “build-stuff” EDSL, compiled to a Makefile + scripts:

- Download Qemu images.
- Setup/start qemu VM.
- Run recipe on the VM in a mostly restartable way.
- Grab artifacts from the VM into a .tgz (e.g. an executable + output of ldd).

#HackyExample #WIP https://gitlab.com/smondet/habust
"deb-arm-emacs", Build_definition.Construct.(within (qemu_arm debian_wheezy) [
  exec ["apt-get"; "update"];
  exec ["apt-get"; "install"; "--yes"; "emacs23"];
  get_executable "/usr/bin/emacs" ~dest:"emacs-armv7l-bin";
]
);
"deb-arm-ketrew", Build_definition.Construct.((* ... *)
  within (qemu_arm debian_wheezy) [
    ensure (executables_available ["unzip"; "gcc"; "make"; "git"])
    (* ... *)
  ];
  ensure (md5 opam_bin (Contains "46e25cc5b26"))
  [
    "wget"; opam_arm7l_url; "-O"; opam_bin];
  (* ... *)
  ensure (returns_zero @@ opam_exec ["vidimetro"; "--version"])
  (* opam_exec ["opam"; "remove"; "--yes"; "ocamlfind"]; *)
  pin_github "ketrew";
  opam_exec ["opam"; "depeext"; "--yes"; "ketrew"];
  opam_install ["ketrew"];
  get_executable (strf "/opam-root/%s/bin/ketrew" ocaml_version) ~dest:"ketrew-armv7l-bin";
  (* ... *)
Ketrew on ARM64

NAME
ketrew - A Workflow Engine for Complex Experimental Workflows

SYNOPSIS
ketrew COMMAND ...

SOME COMMAND ALIASES
pc  An alias for print-configuration.

COMMANDS
explore
  Run the interactive Target Explorer.

gui
  Get info about this instance.

initialize
  Initialize the application (create a config-directory)

interact
  Run the interactive menu.

kill
  Kill a target.

logs
  See the logs.

print-configuration
  Display current configuration.

start-server
  Start the server.

status
  Get info about this instance.

stop-server
  Stop the server.
Silence on ARM64

Could not get the graphical apps I wanted to show:

```bash
#=== ERROR while installing uri.1.9.4 ================================#
# opam-version 1.2.2
# os     linux
# command jbuilder build -p uri -j 4
# path /opam-root/4.03.0/build/uri.1.9.4
# compiler 4.03.0
# [...]
### stderr ###
# [...]
# /tmp/camlasm6e1b43.s:445651: Error: offset out of range
# /tmp/camlasm6e1b43.s:445679: Error: offset out of range
# /tmp/camlasm6e1b43.s:445687: Error: offset out of range
# File "etc/uri_services_full.ml", line 1:
# Error: Assembler error, input left in file /tmp/camlasm6e1b43.s

mantis#7608, mirage/ocaml-uri#106, janestreet/ppx_ast#3
```
Future Work

- Byte-array Vs C-String type.
- GADT Vs TTFI discussion (cf. this afternoon): we want to call the compiler within a “script” to use its output as a literal string
- More combinators (integration of Secotrec/Habust functions).
The End

Questions?