

# Comma-shell, interaktivní debugger shellu

Tomáš Nesrovnal

Fakulta informačních technologií  
České vysoké učení technické v Praze  
Obor: Systémové programování  
Vedoucí práce: Ing. Jan Baier

4. června 2017

## Zadání práce

- ▶ Proveďte rešerši existujících nástrojů pro statickou analýzu, krokování a hledání chyb v BASH skriptech.
- ▶ Navrhněte a implementujte nástroj, který umožní psát uživatelské skripty pro analýzu příkazů a ovlivňování jejich spouštění a vykonávání. Nástroj musí umožňovat krokovat složitější skripty po jednotlivých příkazech. Pro analýzu spouštěných skriptů využijte vhodný nástroj z rešeršní části.
- ▶ Vytvořte ukázkové skripty pro zvrácení, či zamezení efektů základních příkazů z GNU coreutils. Zaznamenávejte jejich spouštění a umožněte jimi provedené změny vrátit do původního stavu.

# Obsah prezentace FIXME: chci vubec obsah?

Cíl práce

Existující řešení

- Bash Debugger

- ShellCheck

Comma-shell

- Hooks

- Debugger

- Bezpečné příkazy

Ukázky

Instalace

Otázky: oponent

# Cíl práce

- ▶ Vytvořit nástroj usnadňující výuku v shellu Bash
- ▶ Analýza příkazů před jejich spuštěním
- ▶ Jednoduchý, interaktivní debugger příkazů v příkazové řádce
- ▶ Sada bezpečných příkazů

# Existující řešení

- ▶ Debugování skriptů: Bash Debugger
- ▶ Statická analýza skriptů: ShellCheck

# Existující řešení: Bash Debugger

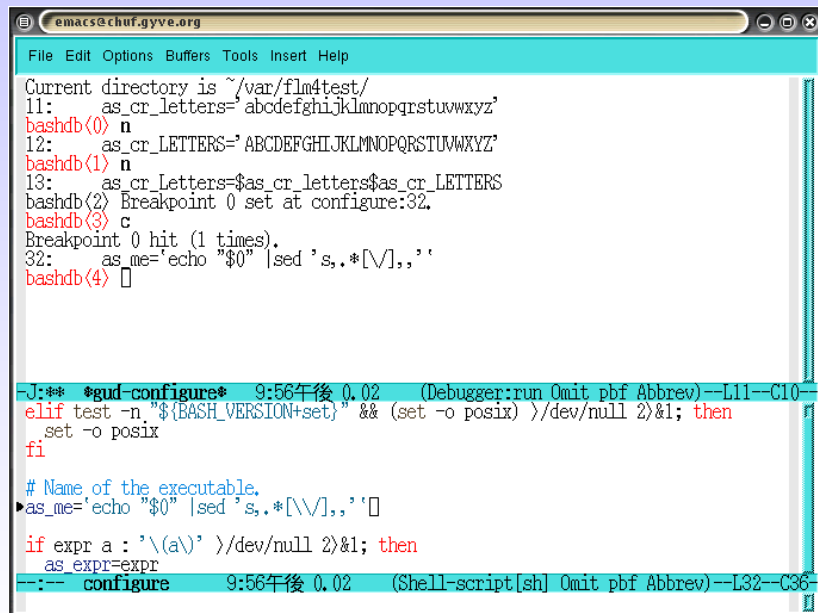
## Výhody

- ▶ Ovládání jako GNU GDB

## Nevýhody

- ▶ Pouze pro skripty

# Existující řešení: Bash Debugger



```
emacs@chuf.gyve.org
File Edit Options Buffers Tools Insert Help

Current directory is ~/var/flm4test/
11: as_cr_letters='abcdefghijklmnopqrstuvwxyz'
bashdb<0> n
12: as_cr_LETTERS='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
bashdb<1> n
13: as_cr_Letters=$as_cr_letters$as_cr_LETTERS
bashdb<2> Breakpoint 0 set at configure:32.
bashdb<3> c
Breakpoint 0 hit (1 times).
32: as_me='echo "$0" |sed 's,.*[\\],,,'
bashdb<4> [

J:** *gud-configure* 9:56午後 0.02 (Debugger:run Omit pbf Abbrev)--L11--C10--
elif test -n "$${BASH_VERSION+set}" && (set -o posix) >/dev/null 2>&1; then
set -o posix
fi

# Name of the executable.
as_me='echo "$0" |sed 's,.*[\\],,,'

if expr a : '\\(a\\)' >/dev/null 2>&1; then
as_expr=expr

--:-- configure 9:56午後 0.02 (Shell-script[sh] Omit pbf Abbrev)--L32--C36--
```

# Existující řešení: ShellCheck

## Výhody

- ▶ Databáze "špatného kódu"

## Nevýhody

- ▶ Napsaný v jazyce Haskell
- ▶ Pouze pro skripty



# Comma-shell: Hooks

## Využití

- ▶ Spouštění skriptů před vykonáním příkazů z příkazové řádky
- ▶ Příkazy k vykonání je možné měnit, nebo nevykonávat

## Implementace

- ▶ Zastavení všech příkazů pomocí DEBUG trap v módu extdebug
- ▶ Příkaz k vykonání se zjišťuje pomocí příkazu history
- ▶ Vykonávání příkazů probíhá pomocí příkazu eval

# Comma-shell: Debugger

## Využití

- ▶ Spouštění částí příkazů v příkazové řádce

## Implementace

- ▶ Jedná se o hook, původní příkaz se nevykoná
- ▶ Pomocí knihovny bashlex se prochází AST příkazu
- ▶ K částem příkazu se přidá další kód a v Bashi je vykonán pomocí příkazu eval

# Comma-shell: Safe commands

## Využití

- ▶ Wrapper základních příkazů
- ▶ Změny provedené těmito příkazy lze vrátit zpět

## Implementace

- ▶ Funkce se stejným jménem jako příkaz z GNU coreutils
- ▶ Logování příkazů a jejich efektů
- ▶ rm implementováno pomocí FreeDesktop.org koše

# Ukázky: ShellCheck hook

```
n@:~$ printf "$test" | grep '*test*'
,: ShellCheck:
printf "$test" | grep '*test*'
    ^-- SC1015: This is a unicode double quote. Delete and retype it.
    ^-- SC2059: Don't use variables in the printf format string. Use printf "..%s.." "$foo".
        ^-- SC1015: This is a unicode double quote. Delete and retype it.
            ^-- SC2063: Grep uses regex, but this looks like a glob.
,: Now what? [r]un, [s]top, [i]gnore, [p]retype: |
```

# Ukázky: Comma-shell debugger

```
net:~$ echo "the year is $(date +%Y)" | grep 2017
,: commash debugger:

echo "the year is $(date +%Y)" | grep 2017
      ^-- [1] show substitution: $(date +%Y)
              ^-- [2] show pipe flow: echo "the year is $(date +%Y)"

,: Select your option, [0] debug whole cmd, [q]uit, [r]un normally █
```

## Ukázky: Comma-shell safe

```
n@t:/tmp/rm$ touch a/b.c {d,e}.f g
n@t:/tmp/rm$ ,rm *
,rm: Files to remove:
,rm:     Directory: "a" (with 1 files)
,rm:     Removing file: "/tmp/rm/g"
,rm:     2 files with extension "f"
,rm: Choose:
,rm:     [r]emove files
,rm:     [q]uit
,rm:     [t]rash files
,rm:     [s]how more files
```



# Instalace

## Git repozitář

- ▶ [github.com/nesro/commash](https://github.com/nesro/commash)

## Instalace

- ▶ `sudo apt update`
- ▶ `sudo apt install git shellcheck python-pip xdotool -y`
- ▶ `sudo pip install bashlex`
- ▶ `git clone https://github.com/nesro/commash ~/.commash`
- ▶ `source ~/.commash/comma.sh`

## Otázky: oponent (1/2)

- ▶ Comma-shell používá pro vlastní spuštění trasovaných a debugovaných příkazů příkaz eval. Jak se vypořádá s použitím tohoto příkazu?
- ▶ Příkaz eval je možné spouštět rekurzivně.
- ▶ Comma-shell příkazem eval spustí více příkazů, ale na rekurzivní spouštění to nemá vliv.



## Otázky: oponent (2/2)

- ▶ V práci jsem nenašel uživatelskou příručku s výčtem debugovatelných konstrukcí shellu. Můžete doplnit, které konstrukce Comma-shell podporuje a které ne?
- ▶ Konstrukce, s kterými umí Comma-shell debugger (v interaktivním shellu) pracovat jsou: roura, subshell, cykly for a while
- ▶ Přidání dalších konstrukcí, se kterými umí pracovat knihovna bashlex je snadné