

# Déboguer confortablement en ligne de commande

GDB, Valgrind, LLDB, PDB, IPDB, JDB



2<sup>ème</sup> session LIFTech'

**Auteur:** Florent Jaillet

**Date:** 9 juin 2016

# Plan

- Motivation
- Intérêt du débogage en ligne de commande
- GDB (*C, C++, Objective-C, Fortran, Go, Rust...*)
- Valgrind + GDB
- LLDB (*C, C++, Objective-C*)
- PDB et IPDB (*Python*)
- JDB (*Java*)

# Motivation

- Présentation à CppCon 2015 de Greg Law : "Give me 15 minutes & I'll change your view of GDB"
- Outils en ligne de commande difficiles à apprendre et peu intuitifs, mais plus pratiques que laisse paraître leur première utilisation
- Objectif ici n'est pas de présenter en détail le fonctionnement des outils, mais plutôt d'identifier les fonctionnalités clefs qui rendent l'utilisation plus facile et pratique
- Utiliser ce support de présentation comme mémo pour creuser les points intéressants par la suite

# Intérêt du débogage en ligne de commande

- Ligne de commande disponible partout (par défaut sous GNU/Linux, BSD, MAC OS X, [Windows Subsystem for Linux](#) sous Windows 10, [installable](#) sous android)
- Parfois la seule solution (machine distante, grappe de calcul, système embarqué...)
- Puissance des outils
- Éventuelle préférence pour la ligne de commande

# GDB : Exemple d'utilisation

- Compilation :

```
g++ -g example.cpp -o example
```

- Lancement du débogage :

```
gdb ./example
```

# GDB : Utilisation basique

Aide	<b>help</b>
Lancement	start, run
Points d'arrêt	<b>break</b> , <b>disable</b> , <b>enable</b> , <b>clear</b> , <b>delete</b>
Flot d'exécution	<b>step</b> , <b>next</b> , <b>finish</b> , <b>continue</b>
Affichage informations	<b>print</b> , <b>backtrace</b> ( <i>ou where</i> ), <b>list</b> , <b>info</b>
Arrêt et fin	<b>kill</b> , <b>quit</b>

# GDB : Plus de confort avec tui

- Interface avec fenêtres textuelles multiples via le raccourci Ctrl+X+A ou la commande `tui enable`
- Informations affichables en plus de la ligne de commande de GDB :
  - code source
  - code assembleur
  - registres
- Choix de la disposition des fenêtres avec les raccourcis Ctrl+X+2 et Ctrl+X+1 ou la commande `layout`
- Raccourcis :
  - Ctrl+L : rafraichissement interface
  - Ctrl+P, Ctrl+N, Ctrl+B, Ctrl+F : remplacement des touches flêches

# GDB : Plus de puissance

Points d'arrêt	<b>watch</b> , <b>catch</b> , <b>break</b> <where> if <condition>, <b>condition</b> , <b>commands</b> , <i>sauvegarde</i> (save, <b>source</b> )
Affichage	<b>dprintf</b>
Appel à la volée	<b>call</b>
Rétro-débugage	<b>record</b> , <b>reverse-step</b> , <b>reverse-next</b> , <b>reverse-finish</b> , <b>reverse-continue</b>
Script et extension	<b>python</b> , <b>python-interactive</b>
Historique	set history save ( <i>à mettre dans \$HOME/.GDBinit</i> )
Commandes système	<b>!</b> ( <i>ou shell</i> )



# Valgrind

Suite d'outils pour l'analyse dynamique (en particulier mémoire), notamment :

- détection de l'utilisation de mémoire non allouée ou non initialisée

```
Invalid read of size 1
```

```
at 0x4005C8: main (example_valgrind.c:18)
```

```
Address 0x5202100 is 0 bytes after a block of size 16 alloc'd
```

```
at 0x4C2BBCF: malloc (vg_replace_malloc.c:299)
```

```
by 0x4005BF: main (example_valgrind.c:15)
```

- détection de fuites de mémoire

```
LEAK SUMMARY:
```

```
definitely lost: 35 bytes in 2 blocks
```

```
indirectly lost: 0 bytes in 0 blocks
```

```
possibly lost: 0 bytes in 0 blocks
```

```
still reachable: 0 bytes in 0 blocks
```

```
suppressed: 0 bytes in 0 blocks
```

- profilage de l'utilisation mémoire (`--tool=massif`)

# Valgrind + GDB

Utilisation interactive de Valgrind possible via GDB :

- Lancement de Valgrind : `valgrind --vgdb-error=0 ./example`
- Lancement de GDB dans une autre invite de commande : `gdb ./example`
- Puis dans GDB : `(gdb) target remote | vgdb`

Aide	<b>monitor help</b>
Fuite mémoire	<b>monitor leak-check</b>
Validité mémoire	<b>monitor get-vbits</b> , <b>monitor check_memory</b>
Occupation mémoire	<b>monitor snapshot</b> <i>(nécessite l'option --tool=massif de Valgrind)</i>

Plus de détails dans la [documentation de Valgrind](#) et ce [tutoriel](#)

# LLDB : Utilisation basique

	GDB	LLDB
Aide	<b>help</b>	<b>help</b>
Lancement	start, run	run -s, run
Points d'arrêt	<b>break</b> , <b>disable</b> , <b>enable</b> , <b>clear</b> , <b>delete</b>	<b>breakpoint set</b> , <b>breakpoint disable</b> , <b>breakpoint enable</b> , <b>breakpoint clear</b> , <b>breakpoint delete</b>
Flot d'exécution	<b>step</b> , <b>next</b> , <b>finish</b> , <b>continue</b>	<b>step</b> , <b>next</b> , <b>finish</b> , <b>continue</b>
Affichage informations	<b>print</b> , <b>backtrace</b> ( <i>ou where</i> ), <b>list</b> , <b>info</b>	<b>print</b> , <b>bt</b> , <b>list</b> , ( <b>frame variable</b> , <b>breakpoint list</b> , <b>watchpoint list</b> ,...)
Arrêt et fin	<b>kill</b> , <b>quit</b>	<b>kill</b> , <b>quit</b>

# LLDB : Plus de puissance 1/2

	GDB	LLDB
Multifenêtre	tui	gui
Points d'arrêt	<b>watch</b> , <b>catch</b> , <b>break</b> <where> if <condition>, <b>condition</b> , <b>commands</b> , <i>sauvegarde</i> (save, <b>source</b> )	<b>watchpoint set variable</b> , <b>breakpoint set -F</b> , <b>breakpoint set -c</b> , <b>breakpoint modify -c</b> , <b>breakpoint command add</b> , $\emptyset$
Affichage	<b>dprintf</b>	$\emptyset$
Appel à la volée	<b>call</b>	<b>expression</b>
Rétro-débugage	<b>record</b> ,...	$\emptyset$

## LLDB : Plus de puissance 2/2

	GDB	LLDB
Script et extension	<b>python</b> , <b>python-interactive</b>	<b>script</b> , <b>script</b>
Historique	set history save ( <i>à mettre dans \$HOME/.GDBinit</i> )	<i>disponible par défaut</i>
Commandes système	<b>!</b> ( <i>ou shell</i> )	<b>platform shell</b>

[Page avec plus d'infos sur les correspondances GDB / LLDB](#)

# PDB et IPDB

	GDB	PDB et IPDB
Aide	<b>help</b>	<b>help</b>
Lancement	start, run	ø, run
Points d'arrêt	<b>break</b> , <b>disable</b> , <b>enable</b> , <b>clear</b> , <b>delete</b> , <b>condition</b> , <b>commands</b>	<b>break</b> , <b>disable</b> , <b>enable</b> , <b>clear</b> , <b>clear</b> , <b>condition</b> , <b>commands</b>
Flot	<b>step</b> , <b>next</b> , <b>finish</b> , <b>continue</b>	<b>step</b> , <b>next</b> , <b>return</b> , <b>continue</b>
Affichage informations	<b>print</b> , <b>backtrace</b> ( <i>ou</i> where), <b>list</b> , <b>info</b>	(p, pp, print), <b>where</b> , <b>list</b> , (dir(), locals(), globals(), <b>break</b> ,...)
Arrêt et fin	<b>kill</b> , <b>quit</b>	ø, <b>quit</b>
Appel, script	<b>call</b> , <b>python</b> , <b>python-interactive</b>	[!]
Système	! ( <i>ou</i> shell)	os.system("<cmd>")

Voir [pdb++](#) ou [pudb](#) pour des fonctionnalités plus avancées

# JDB

Fonctionnalités limitées, pas de fonctionnalités avancées :

	GDB	JDB
Aide	<b>help</b>	help
Lancement	start, run	ø, run
Points d'arrêt	<b>break, disable, enable, clear, delete, watch, catch</b>	stop, ø, ø, clear, clear, watch, catch
Flot d'exécution	<b>step, next, finish, continue</b>	step, next, step up, cont
Affichage informations	<b>print, backtrace</b> ( <i>ou</i> where), list, info	(print, dump), where, list, (locals, clear,...)
Appel à la volée	<b>call</b>	eval ( <i>ou</i> print)
Arrêt et fin	<b>kill, quit</b>	kill, quit