1h1 Outil GDB

Lancement de GDB

On compile avec le **flag -g**
(pour ajouter les balises de lignes)

On le lance comme ceci
gdb --tui nom_du_programme
OU
gdb nom_du_programme
Puis tui enable ou CTRL+X+A

ex : gdb --tui a.out

Petit tips les options --silent, -quiet, -q permettent de retirer les CopyRight de Gdb

File, Run and Start

file : Dit / change quel exécutable exécuter *(utile si on ne lui en a pas déjà donné au lancement)*

Attention bug d'affichage si on change d'exécutable : faire directory puis refaire la commande file.

run (r): Exécute le programme normalement jusqu'à ce que l'exécution se termine ou rencontre un breakpoint.

start : Exécute le programme en s'arrêtant au tout début de celui-ci . *(Option la plus intéressante pour le débogage)*

Arguments

Si votre programme prend des paramètres :

start arg1 arg2 ... ou **run** arg1 arg2 ... permet de donner des arguments à l'exécutable.

Si on relance start et run sans argument, gdb gardera les arguments donnés la première fois.

Pour voir les arguments donnés à l'exécutable on peut faire **show arg**.

Pour changer les arguments on peut refaire **start** arg1 ... ou **run** arg1 ... ou alors **set arg** arg1 arg2.

Quit, Kill

quit (q) : Quitte GDB.

CTRL + D : Quitte GDB.

kill : Arrête la session de debug sans quitter GDB.

Tips

Petits Tips avant de rentrer dans le vrai débogage:

focus (fs) : Change le focus entre les différentes fenêtres

Ex : **fs** cmd *permet de focus le terminal et donc de naviguer à l'aide du clavier dans celui-ci.*

help : Donne de la documentation sur une commande comme un man.

shell (!): Exécute une commande shell.

Ex : **!** gcc fichier.c *va exécuter gcc comme si c'était un terminal Bash.*

Ex : make *va faire fonctionner le makefile (note qu'il n'y a pas besoin de mettre **shell** ou **!** devant).*

CTRL + L : Rafraîchit l'affichage.

TAB : Fonctionne comme sur bash.

Pas à Pas

next (n) : Exécute une instruction (ne rentre pas dans les fonctions) peut être suivi du nombre de lignes à exécuter.

step (s) : Exécute une instruction (rentre potentiellement dans les fonctions). *(Attention à ne pas step dans les fonctions des librairies standard, si cela arrive faites un finish)*

finish (fin) : Exécute les instructions jusqu'à la fin de la fonction.

backtrace (bt) : Permet de voir le chemin d'exécution jusqu'à la ligne actuelle. *(Quelles fonctions ont été appelées et avec quels paramètres)*

Imprimer / Surveiller / Changer des valeurs

print (p) : Affiche la valeur d'une variable, sortie de fonction et toutes autres expressions évaluables.

Ex : **p** i *imprime la valeur de i*, **p** *ptr *imprime la valeur du pointeur*, **p** struct *imprime toute la structure*, **p** tab[0]@size *imprime la valeur des cases du tableau,* **p** ft_print::ind_number *imprime la valeur de ind_number contenue dans la fonction ft_print*.

set : Change la valeur d'une variable.

Ex : **set** VAR = EXPR *donne la valeur de l'expression à VAR (VAR est une variable de GDB).*
Ex : **set** variable VAR = EXPR *donne la valeur de l'expression à VAR (VAR est une variable du code).*

watch : Ajoute un watchpoint à la variable qui affichera la variable à chaque modification de celle-ci. *(Regarder aussi rwatch, awatch).*

display : Imprime la variable à chaque arrêt de l'exécution.

Point d’arrêts and Co

break (b, br) : Introduit un breakpoint à l’endroit (fonction, n° de ligne) spécifié. *tbreak.*

Ex : **break** 65, **br** ft_putstr, **b** monficher.c:15 .

*On peut ajouter une condition par exemple **br** 123 **if** i > 12, **br** 15 **if** str[i + 3] == ‘\0’*

clear (cl): Efface le breakpoint se trouvant à l’endroit (fonction, n° de ligne) spécifié.

delete (del, d): Efface le breakpoint, watchpoint, display spécifié (n° d’identifiant) *(Regarder aussi disable, enable, enable … once, enable …. delete)*.

info (inf, i): Donne des informations tel que le n° d’identifiant, adresse, expression, … sur ce qu’on lui demande.

Ex : **info** breakpoint, **inf** watchpoint, **info** locals, **inf** args, **i** display

Tips++

command : Permet de donner un set de commandes à exécuter à un breakpoint précis. *Silent.*

Ex : **command** 65, puis echo “Valeur de i\n” (entrée), print i (entrée), end.

call : Appelle la fonction donnée en paramètre et affiche le retour (si la fonction n’est pas void).

Ex : **call** ft_strlen(“Test”)

show : Commande similaire à info mais permettant d’avoir des informations sur d’autres choses que info.

Ex : **show** arg montre les arguments passés au programme

Pimp My GDB

Possibilité de rajouter des extensions pour GDB :

- *en modifiant ~/.gdbinit (pour les alias)*
- *avec des surcouches python*

Plus d'info pour pimp son GDB-> Link

Plus d'info sur GDB -> *Link*

Pimp My gdbinit

Pour ajouter des alias à GDB :

Dans le fichier ~/.gdbinit (si il n'existe pas il faut le créer)

```
define <alias_name>
      commande gdb à exécuter
end
```

Les alias peuvent permettre de faire des fonctions pour print certains type de donnés, automatiser des tâches, faciliter l'utilisation de gdb.

Penser à faire la documentation de vos alias pour pouvoir faire (help <alias name>)

```
document <alias_name>
      Documentation de l'alias
end
```

Pour ajouter un historique à GDB :

Dans le fichier ~/.gdbinit (si il n'existe pas il faut le créer)

```
set history save on
set history size 10000 (nombre de commande que GDB retiendrai)
set history filename ~/.gdb_history
```

Merci pour votre attention !

C 4 A

Code For All, école 42