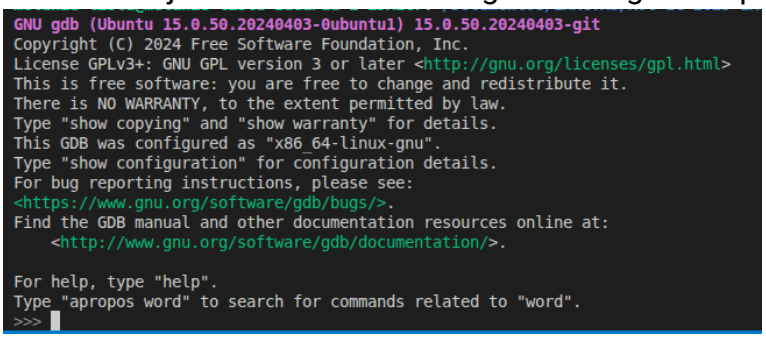
GLOSARIO GDB

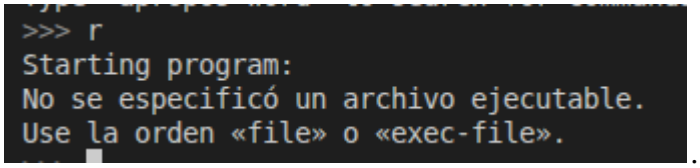
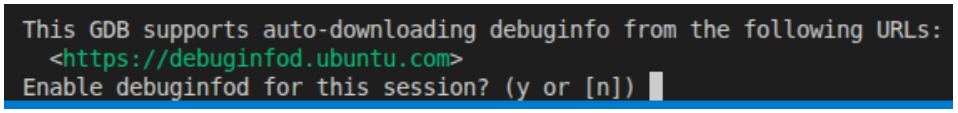
Presentamos un glosario que creemos que les será útil para aprender a utilizar GDB. Para más información vea la sección “**Un poco de GDB**”.
GDB es una herramienta que permite debugear y les puede ser de mucha ayuda en momentos críticos. Una vez que sepan utilizarlo, codear/programar puede volverse una tarea muy ágil. Durante toda la primera parte de la materia irán encontrándose con ejercicios complejos que necesitarán un seguimiento muy (**muy**) de cerca. Por esta razón, encontrar un error dentro del código puede tomar 2 horas o 5 minutos dependiendo de qué tan buen uso le den a debugger.
En principio puede parecerles difícil de entender, o incluso puede asustar un poquito la interfaz que tiene, pero confíen en que cuánto más uso le den, la navegación será más amena y llegará un punto donde no podrán programar sin utilizar esta herramienta. A continuación les brindamos los comandos que según nuestro criterio son los más utilizados y que necesitarán utilizar mucho. Los comandos deben escribirse en la terminal con el formato que indicamos.

COMANDOS

---

gdb	Nos permite abrir el debugger. Para esto debemos estar posicionados en un directorio que contenga el archivo que queramos correr. Cuando lo ejecutamos veremos algo de la siguiente pinta:  Con este comando no indicamos el archivo a debugear. Se necesita hacer un paso más (véase **run *file***).

gdb *file*	Es el primer paso para comenzar a trabajar. Nos permite debugear el ejecutable llamado *file*. Por ejemplo, si deseamos debugear los tests, y el ejecutable se llama ‘*tester*’, debemos escribir “*gdb tester*”. De este modo le indicamos a la herramienta el archivo que deseamos que cargue.

run	Comando equivalente: **r**. Nos permite correr el programa. Si no le indicamos a la herramienta previamente qué programa ejecutar, saldrá un aviso del estilo:  . Cuando lo ejecutamos suele requerir que respondamos:  Una vez indicada su respuesta, el programa se ejecutará.

quit	Nos permite cerrar el debugger

b *nombre_de_funcion*	Nos permite colocar un breakpoint. El breakpoint es para indicar dónde frenar la ejecución. Acá le pedimos que frene en la función llamada "*nombre_de_funcion*"

b *nombre_de_archivo:numero_de_linea*	Nos permite colocar un breakpoint en el archivo *"nombre_del_archivo"* en la línea "*numero_de_linea*". (No olvidarse de los dos puntos en el medio.)

info b	Nos permite conocer todos los breakpoints que tenemos colocados en el programa. Nos imprime una lista con la información de cada uno de ellos.

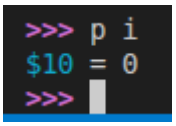
delete *número_de_breakpoint*	Nos permite eliminar un breakpoint. Cada breakpoint está identificado por un número id (véase **info b**).

delete breakpoints	Nos permite eliminar todos los breakpoints.

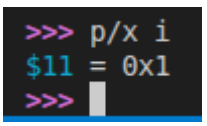
next	Comando similar: **n**. Nos permite avanzar de línea. No ingresa a las funciones, solo avanza a la siguiente línea.

step	Comando similar: **s**. Nos permite ingresar a una función.

continue	Comando similar: **c**. Nos permite avanzar al siguiente breakpoint. Si no hay un breakpoint, llega hasta el final del programa.

print *nombre_de_variable*	Comando similar: **p *nombre_de_variable***. Nos permite imprimir en consola el contenido de la variable "*nombre_de_variable*". Formato: Imprime según el tipo de dato de la variable. Ejemplo: En un ciclo for, si queremos conocer el valor del iterador i, podemos hacer: 

print $*nombre_de_registro*	Comando similar: **p $*nombre_de_registro***. Nos permite imprimir en consola el contenido del registro "*nombre_de_registro*". Formato: decimal. *TIP:* Utilicen los diferentes tamaños de los registros. Por ejemplo: **p $rdx** o también **p $dl**.

p/x *nombre_de_variable* **p/x $*nombre_de_registro***	Nos permite imprimir en consola el contenido del registro "*nombre_de_registro*" o la variable "*nombre_de_variable*". Formato: hexadecimal. Ejemplo: Si queremos conocer el valor de la variable i, podemos hacer: 

p/u *nombre_de_variable* **p/u $*nombre_de_registro***	Nos permite imprimir en consola el contenido del registro "*nombre_de_registro*"o la variable "*nombre_de_variable*". Formato: decimal unsigned.

p/c *nombre_de_variable* **p/c $*nombre_de_registro***	Nos permite imprimir en consola el contenido del registro "*nombre_de_registro*"o la variable "*nombre_de_variable*". Formato: char. Carácter ASCII.

p/f *nombre_de_variable* **p/f $*nombre_de_registro***	Nos permite imprimir en consola el contenido del registro "*nombre_de_registro*"o la variable "*nombre_de_variable*". Formato: float.

p (uint32_t*) *nombre_de_variable* **p (uint32_t*) $*nombre_de_registro***	Nos permite imprimir en consola el contenido del registro "*nombre_de_registro*" o la variable "*nombre_de_variable*". Formato: puntero a uint32_t. *Obs: No imprime el contenido en memoria. Solo imprime el puntero (dirección de memoria). Con **uint32_t** indicamos al tipo de dato al que apunta.*

p *(*uint32_t) *nombre_de_variable*** **p *(*uint32_t**) $*nombre_de_registro***	Comando similar: **x/(uint32_t) $*nombre_de_registro*** . Nos permite imprimir en consola la memoria apuntada por el puntero contenido en el registro ""*nombre_de_registro*" o la variable "*nombre_de_variable*". Formato: Lee la memoria como un dato de tipo *uint32_t*, entonces imprime en ese formato.

p *(*uint32_t) *nombre_de_variable*** **p *(*char**) $*nombre_de_registro***	Comando similar: **x/c $*nombre_de_registro*.** Nos permite imprimir en consola la memoria apuntada por el puntero contenido en el registro "*nombre_de_registro*" o la variable "*nombre_de_variable*". Formato: Lee la memoria como un carácter, entonces imprime en ese formato.

p ({char} $*nombre_de_registro*)@*numero_de_bytes*	Nos permite imprimir una cantidad (***numero_de_bytes***) de

	elementos de un tipo determinado (**{char}**) a partir de una dirección (***$nombre_de_registro***). *TIP: Útil cuando conocemos la dirección y el tipo de una variable y queremos ver su contenido.* *TIP2: en lugar de poner **$nombre_de_registro** podemos escribir directamente de la dirección de memoria.*

x *nombre_de_variable* **x $*nombre_de_registro***	Nos permite imprimir la memoria apuntada por el puntero contenido en el registro "*nombre_de_registro*" o la variable "*nombre_de_variable*".

Nota: El comando x está explicado en profundidad en la sección "Un poco de GDB".

x/4 *nombre_de_variable* **x/4 $*nombre_de_registro***	Nos permite imprimir 4 bytes de memoria apuntada por el puntero contenido en el registro "*nombre_de_registro*" o la variable "*nombre_de_variable*". *TIP: Útil para conocer el contenido de un array.* *TIP2: En lugar del 4 se coloca el número de bytes deseado e imprime dicha cantidad.*

x/4c *nombre_de_variable* **x/4c $*nombre_de_registro***	Nos permite imprimir 4 bytes de memoria apuntada por el puntero contenido en el registro "*nombre_de_registro*" o la variable "*nombre_de_variable*". Formato: 4 bytes como char. *TIP: Útil para conocer el contenido de un array.* *TIP2: En lugar del 4 se coloca el número de bytes deseado e imprime dicha cantidad.* *TIP3: En lugar de c se coloca el tipo de dato deseado.*