gdb Debugging Cheatsheet

running gdb

Remember to compile your program with debugging symbols enabled! That's usually the `-g` or `-ggdb` flag.

Running your executable with `gdb`:

```
$ gdb /path/to/your/executable
```

```
$ gdb --args /path/to/your/executable
   --arg1 --arg2
```

If your executable requires command line arguments (`--arg1 --arg2`)

gdb basic commands

`r (run)`	run the program `gdb` loaded
`c (continue)`	continue run
`n (next)`	execute next line
`s (step)`	execute next step
`finish`	after a breakpoint, run function until end and halt there
`l (list)`	show source code at current loc
`l <lnr>`	show source code at line ¡lnr¿
`tui enable`	start a fancy Text User Interface
`p (print) <var>`	print variable ¡var¿
`p *<var_p>`	print value of pointer ¡var_p¿ instead of address
`display <var>`	print variable ¡var¿ every time it is touched throughout the run
`info locals`	print all local variables
`b (break) <loc>`	set a breakpoint at ¡loc¿
`tbreak <loc>`	set a temporary breakpt at ¡loc¿
`watch <var>`	set a watchpoint at var ¡var¿
`bt (backtrace)`	show stack trace
`where`	show current location in trace
`frame`	show current location in trace
`frame <nr>`	change into frame ¡nr¿

gdb and MPI

```
$ mpirun -n 4 xterm -e gdb -ex run
   your_program
```

Runs 4 `xterm` terminals with `MPI` and executes your program through `gdb`

```
$ mpirun -n 4 xterm -e gdb -ex
    run --args your_program --arg1 --
   arg2
```

same as above, but allows your program to read in command line arguments

gdb and core dumps

```
$ ulimit -S -c unlimited
```

To enable core dumps on linux

```
$ coredump
```

Command shows you where cores will be dumped on your system

```
$ gdb -c core.XXXX path/to/executable
```

Load core dump with `gdb`

gdb breakpoints

```
break <line number in main file>
```

To set a breakpoint (*before* you execute `run`)

```
break path/to/file.c:<line_nr>
```

To set a breakpoint on ¡line_nr¿ in specific file

```
break file.c:function_name
```

To break when a function is called in `file.c`

```
tbreak <loc>
```

Make temporary breakpoint that deletes itself after it gets hit once

```
info break
```

show info on currently set `breakpoints`

```
del <brnr>
```

delete `breakpoint` ¡brnr¿ (find ¡brnr¿ using `info break`)

```
disable/enable <brnr>
```

disable/enable `breakpoint` ¡brnr¿ (skip or don't skip it without deleting it)

gdb and "value has been optimized out"

Either recompile your program without optimization, or tell compiler not to optimize specific function you're looking at. Doing that depends on the compiler.

For `GCC`:

```
#pragma GCC push_options
#pragma GCC optimize ("O0")

  void your_function(){...}

#pragma GCC pop_options
```

For `intel`:

```
#pragma optimize( "", off )

  void your_function() {...}

#pragma optimize( "", on )
```

or

```
#pragma intel optimization_level 0
  void your_funtction(){...}
```

For `clang`:

```
__attribute__((optnone))
  void your_function()
    {...}
```