리눅스에서 valgrind -h 또는 valgrind --help  쳐보면, 다음과 같은 설명이 나온다.  
사용법은 간단함.

|  |
| --- |
| >valgrind -h  usage: valgrind [options] prog-and-args  common user options for all Valgrind tools, with defaults in [ ]:     --tool=<name>             use the Valgrind tool named <name> [memcheck]     -h --help                 show this message     --help-debug              show this message, plus debugging options     --version                 show version     -q --quiet                run silently; only print error msgs     -v --verbose              be more verbose, incl counts of errors     --trace-children=no|yes   Valgrind-ise child processes? [no]     --track-fds=no|yes        track open file descriptors? [no]     --time-stamp=no|yes       add timestamps to log messages? [no]     --log-fd=<number>         log messages to file descriptor [2=stderr]     --log-file=<file>         log messages to <file>.<pid>     --log-file-exactly=<file> log messages to <file>     --log-file-qualifier=<VAR> incorporate $VAR in logfile name [none]     --log-socket=ipaddr:port  log messages to socket ipaddr:port    uncommon user options for all Valgrind tools:     --run-libc-freeres=no|yes free up glibc memory at exit? [yes]     --sim-hints=hint1,hint2,...  known hints:                                  lax-ioctls, enable-outer [none]     --show-emwarns=no|yes     show warnings about emulation limits? [no]     --smc-check=none|stack|all  checks for self-modifying code: none,                               only for code found in stacks, or all [stack]     --kernel-variant=variant1,variant2,...  known variants: bproc [none]                               handle non-standard kernel variants    user options for Valgrind tools that report errors:     --xml=yes                 all output is in XML (some tools only)     --xml-user-comment=STR    copy STR verbatim to XML output     --demangle=no|yes         automatically demangle C++ names? [yes]     --num-callers=<number>    show <number> callers in stack traces [12]     --error-limit=no|yes      stop showing new errors if too many? [yes]     --error-exitcode=<number> exit code to return if errors found [0=disable]     --show-below-main=no|yes  continue stack traces below main() [no]     --suppressions=<filename> suppress errors described in <filename>     --gen-suppressions=no|yes|all    print suppressions for errors? [no]     --db-attach=no|yes        start debugger when errors detected? [no]     --db-command=<command>    command to start debugger [gdb -nw %f %p]     --input-fd=<number>       file descriptor for input [0=stdin]     --max-stackframe=<number> assume stack switch for SP changes larger                               than <number> bytes [2000000]    user options for Memcheck:     --leak-check=no|summary|full     search for memory leaks at exit?  [summary]     --leak-resolution=low|med|high   how much bt merging in leak check [low]     --show-reachable=no|yes          show reachable blocks in leak check? [no]     --undef-value-errors=no|yes      check for undefined value errors [yes]     --partial-loads-ok=no|yes        too hard to explain here; see manual [no]     --freelist-vol=<number>          volume of freed blocks queue [5000000]     --workaround-gcc296-bugs=no|yes  self explanatory [no]     --alignment=<number>      set minimum alignment of allocations [8]   ......이하생략 |

노랑색 하이라이트된것은 내가 자주 사용하는 애들..ㅎㅎㅎ  
보통 다음과 같이 사용한다.  
  
valgrind  프로그램 --tool=memcheck --leak-check=yes --show-reachable=yes --log-file="./valgrind\_log   
  
사용 예를 한번..들어보자면   
  
test.c 파일을 아래와 같이 작성(free (test) 빼고)후 gcc test.c 컴파일하면 a.out 파일이 생김

|  |
| --- |
| #include <stdio.h> #include <stdlib.h> typedef struct \_test\_t {     int data; }test\_t; int main(void) {     test\_t \*test;     test = malloc(sizeof(test\_t));     test->data = 3;     printf("test->data = %d\n", test->data);  **free(test); --> malloc에 대한 메모리 해제**     return 0; } |

한눈에 봐도 메모리 누수가 일어날 것을 알수 있다...(free안했음)   
valgrind 를  실행시키면,

|  |
| --- |
| valgrind ./a.out --tool=memcheck --leak-check=summary ==19358== Memcheck, a memory error detector. ==19358== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al. ==19358== Using LibVEX rev 1732, a library for dynamic binary translation. ==19358== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP. ==19358== Using valgrind-3.2.3, a dynamic binary instrumentation framework. ==19358== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al. ==19358== For more details, rerun with: -v ==19358== test->data = 3 ==19358== ==19358== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 13 from 1) ==19358== malloc/free: in use at exit: 4 bytes in 1 blocks. ==19358== malloc/free: 1 allocs, 0 frees, 4 bytes allocated. ==19358== For counts of detected errors, rerun with: -v ==19358== searching for pointers to 1 not-freed blocks. ==19358== checked 48,380 bytes. ==19358== ==19358== LEAK SUMMARY: ==19358==    definitely lost: 4 bytes in 1 blocks. ==19358==      possibly lost: 0 bytes in 0 blocks. ==19358==    still reachable: 0 bytes in 0 blocks. ==19358==         suppressed: 0 bytes in 0 blocks. ==19358== Rerun with --leak-check=full to see details of leaked memory. |

definitely lost: 4 bytes in 1 blocks.  라고 메모리 누수가 발생했음을 알려준다.  
다시 free (test)루틴을 넣고 컴파일 한다음 valgrind를 돌리게 되면

|  |
| --- |
| [root@lkalove]# valgrind ./a.out --tool=memcheck --leak-check=summary ==19427== Memcheck, a memory error detector. ==19427== Copyright (C) 2002-2007, and GNU GPL'd, by Julian Seward et al. ==19427== Using LibVEX rev 1732, a library for dynamic binary translation.  ==19427== Copyright (C) 2004-2007, and GNU GPL'd, by OpenWorks LLP. ==19427== Using valgrind-3.2.3, a dynamic binary instrumentation framework. ==19427== Copyright (C) 2000-2007, and GNU GPL'd, by Julian Seward et al. ==19427== For more details, rerun with: -v ==19427== test->data = 3 ==19427== ==19427== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 13 from 1) ==19427== malloc/free: in use at exit: 0 bytes in 0 blocks. ==19427== malloc/free: 1 allocs, 1 frees, 4 bytes allocated. ==19427== For counts of detected errors, rerun with: -v ==19427== All heap blocks were freed -- no leaks are possible. |

<메모리 leak 실행 결과>

**[root@localhost Desktop]# valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all ./a.out**

==26764== Memcheck, a memory error detector

==26764== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.

==26764== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info

==26764== Command: ./a.out

==26764==

test->data = 3

==26764==

==26764== HEAP SUMMARY:

==26764== in use at exit: 4 bytes in 1 blocks

==26764== total heap usage: 1 allocs, 0 frees, 4 bytes allocated

==26764==

==26764== **4 bytes in 1 blocks are definitely lost in loss record 1 of 1**

==26764== at 0x4C29BE3: malloc (vg\_replace\_malloc.c:299)

==26764== **by 0x400591: main (in /root/Desktop/a.out)**

==26764==

==26764== LEAK SUMMARY:

==26764== **definitely lost: 4 bytes in 1 blocks**

==26764== indirectly lost: 0 bytes in 0 blocks

==26764== possibly lost: 0 bytes in 0 blocks

==26764== still reachable: 0 bytes in 0 blocks

==26764== suppressed: 0 bytes in 0 blocks

==26764==

==26764== For counts of detected and suppressed errors, rerun with: -v

==26764== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

<메모리 정상 실행 결과>

[root@localhost Desktop]# valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all ./a.out

==27255== Memcheck, a memory error detector

==27255== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.

==27255== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info

==27255== Command: ./a.out

==27255==

test->data = 3

==27255==

==27255== HEAP SUMMARY:

==27255== in use at exit: 0 bytes in 0 blocks

==27255== total heap usage: 1 allocs, 1 frees, 4 bytes allocated

==27255==

==27255== **All heap blocks were freed -- no leaks are possible**

==27255==

==27255== For counts of detected and suppressed errors, rerun with: -v

==27255== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)