SIDEBAR Operating Systems

WinDbg Basics Cheat Sheet

Sidebar

Operating Systems · WinDbg Basics Cheat Sheet

Opening a crash dump in WinDbg, setting symbols, and the handful of commands that answer what crashed?. None of this was run: WinDbg is not installed on the PC this was written on, and its minidumps cannot be read without administrator rights. The commands come from Microsoft Learn's WinDbg documentation (the symbol-path, `!analyze` and kernel-dump pages) where marked, otherwise from my own knowledge. The newer Store version of WinDbg may label menus differently from the classic one described by Microsoft.

INSTALL AND OPEN

`winget install Microsoft.WinDbg`	Package id from `winget show` on this PC (version 1.2610.1001.0, msix, publisher Microsoft); not installed here. Or the Microsoft Store listing
`windbg -y <symbols> -z <dump>`	Start with a symbol path and open a dump (Microsoft)
`windbg -i <imagepath>`	Add the executable image path; kernel and small dumps may need it (Microsoft)
`File > Open crash dump (Ctrl+D)`	Classic UI. Dump files end `.dmp` or `.mdmp`
`.opendump <file>`	Open a dump from the command window
`Open a copy`	Copy the dump first (administrator PowerShell); never open or edit the original

SYMBOLS

`.symfix C:\Symbols`	Set the path to Microsoft's public symbol server, with a local cache (Microsoft)
`.sympath srv*C:\Symbols*https://msdl.microsoft.com/download/symbols`	The explicit form (Microsoft)
`.sympath`	Show the current path
`.reload /f`	Reload symbols for every module, forcing it
`_NT_SYMBOL_PATH`	Environment variable that sets the path before WinDbg starts (Microsoft)
`!sym noisy`	Verbose symbol loading: turn it on when symbols fail to load (Microsoft)
`Third-party drivers`	Microsoft's server returned HTTP 404 for the Killer and NordVPN drivers on this PC: expect module names and offsets, not function names

FIRST COMMANDS

`.bugcheck`	The stop code and its four parameters. Check them against event 1001 / event 41: if they differ, wrong dump
`!analyze -v`	Microsoft: begin with this. Bug check, *Probably caused by*, failure bucket and stack
`!analyze -show 13A`	The debugger's own description of a bug check code and its parameters (Microsoft documents `-show BugCheckCode`)
`lm`	List loaded modules (drivers) with load addresses and symbol state
`lmvm <module>`	Detail of one module: version, company name and timestamp (from knowledge)
`k / kv`	Stack of the current thread (`kv` adds arguments) (from knowledge)
`.lastevent`	What the debugger last saw: the exception or bug check (from knowledge)
`q`	Quit

LOOKING FURTHER (MICROSOFT LISTS THESE)

`!process 0 0`	List processes
`!locks`	Kernel resource locks
`!memusage`	Physical memory usage
`!vm`	Virtual memory counters
`!errlog`	The WHEA/system error log in the dump
`!verifier`	Driver Verifier statistics, if it was enabled
`!heap / !pool <address>`	Heap and pool detail: Microsoft names `!heap` for bug check 0x13A. `!pool` is from knowledge

READING !ANALYZE -V

`BUGCHECK_CODE / BUGCHECK_P1–P4`	The code and parameters; decode with Microsoft's page for that code
`MODULE_NAME / IMAGE_NAME`	The module the analyzer blames: a suspect, not a verdict
`FAILURE_BUCKET_ID`	A grouping label; useful when searching or reporting
`STACK_TEXT`	The call stack at the crash; heap corruption is often only noticed here, not caused here
`SYMBOL_NAME / FAILURE_ID_HASH`	More labels for reports and searches
`DEFAULT_BUCKET_ID`	A coarse category (for example a driver-fault bucket)

Field names are from my knowledge of typical `!analyze -v` output; I have not seen output for this PC's dumps.

WHAT A DUMP DOES AND DOES NOT PROVE

`Stop code and parameters`	What kind of failure was detected, and where. Not whose code did it
`Probably caused by`	A suspect. For heap corruption the code that noticed may not be the code that caused it
`Module list`	What was present. Not that any of it was involved
`One dump`	One data point. The same module in several dumps is far stronger
`Small dump`	Microsoft: errors not directly caused by the running thread may not be discoverable. Consider a larger dump type
`Sending dumps`	Send the text output first; a dump itself only when asked, and privately: it can contain more than you expect

THIS PC'S TWO BUGCHECKS (FOR CHECKING)

`101926-16187-01.dmp`	19 Oct 2026: 0x13A, P1 0x11, P2 0xffff840729100140, P3 0xffff8407770608e0, P4 0
`102426-15109-01.dmp`	24 Oct 2026: 0x13A, P1 0x11, P2 0xffffb00ee3100140, P3 0xffffb00f1e44e350, P4 0
`Expect from .bugcheck`	`BugCheck 13A, {11, ffffb00ee3100140, ffffb00f1e44e350, 0}` for the second dump (from the event log, not from the dump)
`P1 0x11`	Microsoft: the heap detected invalid internal state; can be a double-free or heap corruption

NOT RUN HERE

Every command on this sheet is unrun on this PC. Where a line says *Microsoft* it is from Microsoft Learn's WinDbg pages; otherwise it is from my own knowledge of WinDbg and may differ in the Store version.

OPEN A COPY, KEEP THE ORIGINAL

Dumps are evidence. Copy them first, record their hashes, and open only the copies.