Delve Go debugger

CONCEPTS

dlv debug and dlv test build with optimizations off; for dlv exec build with -gcflags="all=-N -l" yourself.

Launch debug, test, exec
↓ `break + continue`
Stopped goroutine, frame
↓ `inspect`
Inspect print, locals, stack
↓ `next / step`
Resume continue, restart

Enter repeats the last command; short aliases: b, c, n, s, so, p, bt, grs, bp.

Pass program arguments after --, e.g. dlv debug -- -port 8080.

START A SESSION

`dlv debug [<pkg>]`
Build and debug a main package

`dlv debug -- <args>`
Debug with program arguments

`dlv test [<pkg>]`
Build and debug a test binary

`dlv test -- -test.run <TestName>`
Debug a single test

`dlv exec <binary> -- <args>`
Debug a prebuilt binary

`go build -gcflags="all=-N -l" -o <bin>`
Build a debug-friendly binary

`dlv attach <pid>`
Attach to a running process

`dlv core <binary> <core>`
Examine a core dump

`dlv debug --build-flags="-tags=<tag>"`
Pass flags to go build

`dlv trace <regex>`
Log calls of matching functions

REMOTE AND HEADLESS

`dlv debug --headless --listen=:2345`
Run only the debug server

`dlv connect <host>:<port>`
Attach the terminal client

`--accept-multiclient`
Let clients connect and reconnect

`exit -c`
Resume target, then disconnect

BREAKPOINTS

`break <func>`
Break at a function (b)

`break <file>:<line>`
Break at a line

`break <loc> if <cond>`
Break when a condition holds

`breakpoints`
List breakpoints (bp)

`clear <id>`
Delete one breakpoint

`clearall [<loc>]`
Delete all (or matching) breakpoints

`toggle <id>`
Disable or enable a breakpoint

`condition <id> <expr>`
Add a condition later (cond)

`watch -w <expr>`
Stop when memory is written

`trace <loc>`
Tracepoint: log a hit, keep running

RUN CONTROL

`continue`
Run to the next breakpoint (c)

`next`
Step over the next line (n)

`step`
Step into calls (s)

`stepout`
Run until the function returns (so)

`continue <loc>`
Run to a location

RUN CONTROL (cont.)

`restart`
Restart with the same arguments (r)

`rebuild`
Rebuild the binary and restart

`exit`
Quit the debugger (q)

INSPECT DATA

`print <expr>`
Evaluate an expression (p)

`locals`
Show local variables

`args`
Show function arguments

`vars <regex>`
Show matching package variables

`display -a <expr>`
Print on every stop

`set <var> = <value>`
Change a variable

GOROUTINES

`goroutines`
List goroutines (grs)

`goroutine <id>`
Switch to a goroutine (gr)

`goroutine <id> stack`
Run a command on a goroutine

`goroutines -t`
List with stack traces

STACK AND SOURCE

`stack`
Show the call stack (bt)

`up / down`
Move one frame up / down

`frame <n>`
Select a frame

`frame <n> locals`
Run a command in a frame

`list`
Show source around current line (l)