Delve Go debugger

START HERE

`dlv debug`
Build and debug main package

`dlv test`
Debug the package tests

`break <func>`
Stop at a function

`continue`
Run to the next stop

`next / step`
Over / into the next line

`print <expr>`
Show a value

`stack`
Show the call stack

`goroutines`
List all goroutines

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

CONCEPTS

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

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
Without a package it builds the one in the current directory.

`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
Test flags need the -test. prefix after --.

`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
Needs ptrace permission (root or kernel.yama.ptrace_scope).

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

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

`dlv debug --wd <dir>`
Set the program's working dir

`dlv exec <bin> -r stdin:<file>`
Redirect a standard stream

`dlv attach --waitfor <name>`
Attach once a process starts

`dlv trace <regex>`
Log calls of matching functions
Trace output goes to stderr.

`dlv trace --stack <n> <regex>`
Log with a stack trace

`dlv trace -t <regex>`
Trace in a test binary

REMOTE AND HEADLESS

`dlv debug --headless --listen=:2345`
Run only the debug server
No auth: bind to 127.0.0.1 or tunnel over SSH.

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

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

`exit -c`
Resume target, then disconnect

`--listen=unix:<socket>`
Listen on a Unix socket

`dlv dap --listen=<addr>`
DAP server for editors

BREAKPOINTS

`break <func>`
Break at a function (b)
*pkg.(*Type).Method; any unambiguous suffix works.*

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

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

`break <name> <loc>`
Create a named breakpoint

`break /<regex>/`
Break on all matching functions
Also matches runtime and stdlib functions.

`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)

`cond -hitcount <id> > <n>`
Stop only after n hits

`cond -clear <id>`
Remove a condition

`on <id> print <expr>`
Print a value on each hit

`watch -w <expr>`
Stop when memory is written
-r for reads, -rw for both.

`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

`next <count>`
Step over several lines

`restart`
Restart with the same arguments (r)

`rebuild`
Rebuild the binary and restart
Only for binaries Delve built itself.

`step-instruction`
Step one CPU instruction (si)

`call <func>(<args>)`
Call a function (experimental)

`exit`
Quit the debugger (q)

`exit -d`
Detach and leave the process running

INSPECT DATA

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

`locals`
Show local variables

`args`
Show function arguments

`vars <regex>`
Show matching package variables

`print %x <expr>`
Print with a fmt verb

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

`set <var> = <value>`
Change a variable
Only numbers and pointers can be set.

`config max-string-len <n>`
Show longer strings

GOROUTINES

`goroutines`
List goroutines (grs)

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

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

GOROUTINES (cont.)

`goroutines -t`
List with stack traces

`goroutines -group userloc`
Group by user-code location

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

`stack -full`
Stack with locals and arguments

`list`
Show source around current line (l)

`list <loc>`
Show source at a location

CONFIG AND SCRIPTS

`help <cmd>`
Help for a command

`config -list`
Show all settings

`config substitute-path <from> <to>`
Map build paths to local source
Needed for binaries built in CI or containers.

RECIPES

Debug one failing test

```
dlv test ./<pkg> -- -test.run
<TestName>
(dlv) break
<file>_test.go:<line>
(dlv) continue
(dlv) locals
```

Remote: server in a container

```
dlv exec ./<bin> --headless --
listen=:2345 \
    --accept-multiclient
# on your machine:
dlv connect <host>:2345
(dlv) config substitute-path
<from> <to>
```

Stop on the 100th loop pass

```
(dlv) break loop <file>:<line>
(dlv) cond -hitcount loop ==
100
(dlv) continue
(dlv) display -a <var>
```

Find stuck goroutines

```
dlv attach <pid>
(dlv) goroutines -group userloc
(dlv) goroutine <id> stack
(dlv) exit -d    # leave it
running
```