

# Reversing with Radare2

## Starting Radare

The basic usage is **radare2** *executable* (on some systems you can use **r2** instead of **radare2**); if you want to run radare2 without opening any file, you can use **--** instead of an executable name.

Some command-line options are:

```
-d file|pid      debug executable file or process pid
-A              analyze all referenced code (aaa command)
-R profile.rr2   specifies r2run2 profile (same as
                -e dbg.profile=profile.rr2)
-w             open file in write mode
-p prj          use project prj
-p             list projects
-h             show help message (-hh the verbose one)
```

Example: **r2 -dA /bin/ls**

## General information

The command **?** prints the help. Command names are hierarchically defined; for instance, all **p**rinting commands start with **p**. So, to understand what a command does, you can append **?** to a *prefix* of such a command; for instance, to learn what **pdf** does, you can first try **pd?**, then the more general **p?**.

Single-line comments can be entered using **#**; e.g. **s # where R we?**.

Command **?** can also be used to evaluate an expression and print its result in various format; e.g. **? 5 \* 8 + 2** (note the space between **?** and the expression). There are also some special **\$**-variables (list all of them with: **?\$?**):

```
$$  current virtual seek
$b  block size
```

Where an address *addr* is expected, you can provide any expression that evaluates to an address, e.g. a function name or a register name. In this cheatsheet we sometimes use *fn-name*, instead of *addr*, to emphasize that the argument is supposed to be a function starting address. As default address is (usually?) used the current seek: **\$\$**.

All commands that:

- accept an optional size (e.g. **pd**), use the current block size by default (see: **b**)
- accept an optional address (e.g., **pdf**), use the current position by default (see: **s**)

## Internal grep-like filtering

You can filter command output by appending **~[!]str**, to display only rows [not] containing string *str*; e.g. **pdf~rdx** and **pdf~!rdx**. You can further filter by appending

```
:r      to display row r (0 ≤ r < #rows or, backwards
        with: -#rows ≤ r ≤ -1)
[c]     to display column c (0 ≤ c < #cols)
:r[c]   to display column c of row r
```

Examples: **af1~[0]**, **af1~malloc[0]**, **pdf~:2** and **pdf~mov:2**

## Shell interaction

Command output can be redirected to a file by appending **>filename** or piped to an external command with **|progname [args]**. Examples: **af1 > all.functions** and **af1 | wc -l**.

External commands can be run with **!!progname [args]**. Note: if a command starts with a single **!**, the rest of the string is passed to currently loaded IO plugin (only if no plugin can handle the command, it is passed to the shell).

The output of external programs can be used as arguments for internal commands by using back-ticks to enclose the invocation of external commands; e.g. **pdf `echo 3` @ `echo entry0`**.

## Python scripting

Assuming that Python extension has been installed (**#!** lists installed extensions) an, interactive Python interpreter can be spawned with **#!python** and a script can be run with **#!python script-filename**.

Inside the spawned interpreter **r2** is an *r2pipe* object that can be used to interact with the same instance of Radare, by invoking method **cmd**; e.g. **print(r2.cmd('pdf @ entry0'))**.

In a script, and inside any Python interpreter (in)directly run with **#!pipe cmd**, the same behaviour can be obtained by **importing r2pipe** and initializing **r2** with **r2pipe.open("#!pipe")**.

You can make most Radare2 commands output in JSON format by appending a **j**; e.g. **pdfj** (instead of **pdf**).

Method **cmdj** can de-serialize JSON output into Python objects; e.g. **f = r2.cmdj('pdfj @ entry0')**  
**print f['name'], f['addr'], f['ops'][0]['opcode']**

## Configuration

```
e??      list all variable names and descriptions
e?[?] var-name show description of var-name
e var-name show the value of var-name
e        show the value of all variables
eco theme-name select theme; eg. eco solarized
eco      list available themes
b        display current block size
b size   set block size
env [name [=value]] get/set environment variables
```

## Some variables

```
asm.pseudo  enable pseudo-code syntax
             (in visual mode, toggle with: $)
asm.bytes   display bytes of each instruction
asm.cmtright comments at right of disassembly if they fit
asm.emu     run ESIL emulation analysis on disasm
asm.demangle Show demangled symbols in disasm

bin.demangle Import demangled symbols from RBin

cmd.stack   command to display the stack in visual
             debug mode (Eg: px 32)

dbg.follow.child continue tracing the child process on fork

io.cache    enable cache for IO changes
             (AKA non-persistent write-mode)

scr.utf8    show nice UTF-8 chars instead of ANSI
             (Windows: switch code-page with chcp 65001)
scr.nkey    select seek mode (fun, hit, flag); affects commands
             n and N during visual mode
scr.wheel   enables mouse-wheel in visual mode
```

Example: **my ~/.radare2rc**

```
e asm.bytes=0
e scr.utf8=true
e asm.cmtright=true
e cmd.stack=px 32
e scr.wheel=false
eco solarized
```

## Searching: /

```
/ str      search for string str
/x hstr    search for hex-string hstr
/a asm-instr assemble instruction and search for its bytes
/R opcode  find ROP gadgets containing opcode;
           see: http://radare.today/posts/ropnroll/
```

Also: **e??search** for options

## Seeking: s

```
s        print current position/address
s addr   seek to addr
s+ n      seek n bytes forward
s++       seek block-size bytes forward
s- n      seek n bytes backward
s--       seek block-size bytes backward
s-        undo seek
s+        redo seek
s=        list seek history
s*        list seek history as r2-commands
```

## Writing: w

<b>wa</b> <i>asm-instr</i>	assemble and write opcodes; for more instructions the whole command must be quoted: "wa <i>asm-instr</i> <sub>1</sub> ; <i>asm-instr</i> <sub>2</sub> ; ..."
<b>w</b> <i>str</i>	write string <i>str</i>
<b>wz</b> <i>str</i>	write string <i>str</i> and append byte \x00
<b>wx</b> <i>hex-pairs</i>	write hex-pairs

## Analysis (functions and syscalls): a

<b>aaa</b>	analyze ( <b>aa</b> ) and auto-name all functions
<b>afl</b>	list functions
<b>afl1</b>	list functions with details
<b>afi</b> <i>fn-name</i>	show verbose info for <i>fn-name</i>
<b>afn</b> <i>new-name addr</i>	name function at address <i>addr</i>
<b>afn</b> <i>new-name old-name</i>	rename function
<b>asl</b>	list syscalls
<b>asl</b> <i>name</i>	display syscall-number for <i>name</i>
<b>asl</b> <i>n</i>	display name of syscall number <i>n</i>
<b>afvd</b> <i>var-name</i>	output r2 command for displaying the address and value of arg/local <i>var-name</i>
<b>.afvd</b> <i>var-name</i>	display address and value of <i>var-name</i>
<b>afvn</b> <i>name new-name</i>	rename argument/local variable
<b>afvt</b> <i>name type</i>	change type for given argument/local
<b>axt</b> <i>addr</i>	find data/code references to <i>addr</i>

## Graphviz/graph code: ag

<b>ag</b> <i>addr</i>	output graphviz code (BB at <i>addr</i> and children) E.g. view the function graph with: <b>ag \$\$   xdot -</b>
<b>agc</b> <i>addr</i>	callgraph of function at <i>addr</i>
<b>agC</b>	full program callgraph

## Information: i

<b>i</b>	show info of current file
<b>ie</b>	entrypoint
<b>iz</b>	strings in data sections
<b>izz</b>	strings in the whole binary
<b>il</b>	libraries
<b>ii</b>	imports
<b>iS</b>	sections

## Printing: p

<b>ps</b> [ <b>@</b> <i>addr</i> ]	print C-string at <i>addr</i> (or current position)
<b>pxr</b> [ <i>n</i> ] [ <b>@</b> <i>addr</i> ]	print <i>n</i> bytes (or block-size), as words, with references to flags and code (telescoping) at <i>addr</i> (or current position)
<b>px</b> [ <i>n</i> ] [ <b>@</b> <i>addr</i> ]	hexdump
<b>pxh</b> ...	hexdump half-words (16 bits)
<b>pxw</b> ...	hexdump words (32 bits)
<b>pxq</b> ...	hexdump quad-words (64 bits)
<b>pxl</b> [ <i>n</i> ] [ <b>@</b> <i>addr</i> ]	display <i>n</i> rows of hexdump
<b>px/fmt</b> [ <b>@</b> <i>addr</i> ]	gdb-style printing <i>fmt</i> (in gdb see: <b>help x</b> from r2: <b>!!gdb -q -ex 'help x' -ex quit</b> )
<b>pd</b> [ <i>n</i> ] [ <b>@</b> <i>addr</i> ]	disassemble <i>n</i> instructions
<b>pD</b> [ <i>n</i> ] [ <b>@</b> <i>addr</i> ]	disassemble <i>n</i> bytes
<b>pd -n</b> [ <b>@</b> <i>addr</i> ]	disassemble <i>n</i> instructions backwards
<b>pdf</b> [ <b>@</b> <i>fn-name</i> ]	disassemble function <i>fn-name</i>
<b>pdC</b> [ <b>@</b> <i>fn-name</i> ]	pseudo-disassemble in C-like syntax

## Debugging: d

<b>?d</b> <i>opcode</i>	description of <i>opcode</i> (eg. <b>?d jle</b> )
<b>dc</b>	continue (or start) execution
<b>dcu</b> <i>addr</i>	continue until <i>addr</i> is reached
<b>dcs</b> [ <i>name</i> ]	continue until the next syscall (named <i>name</i> , if specified)
<b>dcr</b>	continue until ret (uses step over)
<b>dr=</b>	show general-purpose regs and their values
<b>dro</b>	show previous (old) values of registers
<b>drr</b>	show register references (telescoping)
<b>dr</b> <i>reg-name</i> = <i>value</i>	set register value
<b>drt</b>	list register types
<b>drt</b> <i>type</i>	list registers of type <i>type</i> and their values
<b>db</b>	list breakpoints
<b>db</b> <i>addr</i>	add breakpoint
<b>db -addr</b>	remove breakpoint
<b>doo</b> <i>args</i>	(re)start debugging
<b>ood</b>	synonym for <b>doo</b>
<b>ds</b>	step into
<b>dso</b>	step over
<b>dbt</b>	display backtrace
<b>drx</b>	hardware breakpoints
<b>dm</b>	list memory maps; the asterisk shows where the current offset is
<b>dmp</b>	change page permissions (see: <b>dmp?</b> )

## Types: t

<b>"td</b> <i>C-type-def</i> "	define a new type
<b>t</b> <i>t-name</i>	show type <i>t-name</i> in <b>pf</b> syntax
<b>.t</b> <i>t-name</i> <b>@</b> <i>addr</i>	display the value (of type <i>t-name</i> ) at <i>addr</i>
<b>t</b>	list (base?) types
<b>te</b>	list enums
<b>ts</b>	list structs
<b>tu</b>	list unions
<b>to</b> <i>file</i>	parse type information from C header file
<b>t1</b> <i>t-name</i>	link <i>t-name</i> to current address
<b>t1</b> <i>t-name</i> = <i>addr</i>	link <i>t-name</i> to address <i>addr</i>
<b>t1</b>	list all links in readable format

## Visual mode: V

Command <b>V</b>	enters <i>visual mode</i> .
<b>q</b>	exit visual-mode
<b>c</b>	cursor-mode, <i>tab</i> switches among stack/regs/disassembly
<b>:</b>	execute a normal-mode command; e.g. <b>:dm</b>
<b>p</b> and <b>P</b>	rotate forward/backward print modes
<b>/str</b>	highlight occurrences of string <i>str</i>
<b>\$</b>	toggle pseudo-syntax
<b>O</b>	toggle ESIL-asm
<b>;</b>	add/remove comments (to current offset)
<b>x</b>	browse xrefs-to current offset
<b>X</b>	browse xrefs-from current function
<b>-</b>	browse flags
<b>d</b>	define function, end-function, rename, ...
<b>V</b>	enter block-graph viewer
<b>A</b>	enter visual-assembler
<b>n</b> / <b>N</b>	seek next/previous function/flag/hit (see <b>scr.nkey</b> )

## Seeking (in Visual Mode)

<b>.</b>	seeks to program counter
<b>Enter</b>	on jump/call instructions, follow target address
<b>u</b>	undo
<b>U</b>	redo
<b>o</b>	go/seek to given offset
<b>d</b> (a digit)	jump to the target marked [ <i>d</i> ]
<b>m/l</b> (a letter)	mark the spot with letter <i>l</i>
<b>'l</b>	jump to mark <i>l</i>
<b>n</b>	jump to next function
<b>N</b>	jump to previous function

## Debugging (in Visual Mode)

<b>b</b> or <b>F2</b>	toggle breakpoint
<b>F4</b>	run to cursor
<b>s</b> or <b>F7</b>	step-into
<b>S</b> or <b>F8</b>	step-over
<b>F9</b>	continue

## Flags (AKA “bookmarks”): f

Note: in order to get your defined *names* appear in disassembly, you must include a prefix (**fun**, **sub**, **obj**, ...); e.g. **f obj.foo @ 0x1234**

<b>f</b> <i>name</i> <b>@</b> <i>addr</i>	or	
<b>f</b> <i>name</i> = <i>addr</i>		associate name <i>name</i> to address <i>addr</i>
<b>f-</b> <b>@</b> <i>addr</i>		remove the association at address <i>addr</i>
<b>f-</b> <i>name</i>		remove the association with name <i>name</i>

## Comments: C

<b>CC</b>	list all comments in human friendly form
<b>CCu</b> <i>text</i> [ <b>@</b> <i>addr</i> ]	set (update?) comment <i>text</i> at <i>addr</i>
<b>CC</b> <i>text</i> [ <b>@</b> <i>addr</i> ]	append comment <i>text</i> at <i>addr</i>
<b>CC-</b> [ <b>@</b> <i>addr</i> ]	remove comment at <i>addr</i>
<b>CC.</b> [ <b>@</b> <i>addr</i> ]	show comment at <i>addr</i>
<b>CC!</b> [ <b>@</b> <i>addr</i> ]	edit comment using <b>cfg.editor</b> (vim, ...)

## Projects: P

<b>P1</b>	list all projects
<b>Ps</b> [ <i>prj-name</i> ]	save project <i>prj-name</i>
<b>Po</b> <i>prj-name</i>	open project <i>prj-name</i>
<b>Pd</b> <i>prj-name</i>	delete project <i>prj-name</i>

## Running in different environments: rarun2

**rarun2** is used as a launcher for running programs with different environment, arguments, permissions, directories and overridden default file-descriptors. Usage:  
**rarun2** [-t|*script-name.rr2*] [*directives*] [--] [*prog-name*] [*args*]  
**rarun2 -t** shows the terminal name, say  $\alpha$ , and wait for a connection from another process. For instance, from another terminal, you can execute **rarun2 stdio= $\alpha$  program=/bin/sh** (use **stdin/stdout** to redirect one stream only).  
**rarun2** supports *a lot* of directives, see the man page.