Advanced command-line reverse engineering framework

Radare2 Manual

Made by Moeez Javed

Radare2 Manual

> **Ethics first** — Use radare2 only on software you own or have explicit permission to analyze. Reverse engineering can be restricted by law or license. This manual is for defensible, educational use.

1) What is radare2 & why it matters

radare2 (r2) is a free, open-source framework for reverse engineering and binary analysis. It runs on Linux (including Kali), Windows, and macOS, and supports many file formats and CPU architectures. Unlike GUI-heavy tools, r2 is *terminal-first*, scriptable, and ideal for automation and CTFs.

Why teach r2 to beginners - Learn fundamentals of assembly, control flow, and program structure. - Perform safe *static* and *dynamic* (debug) analysis on local binaries. - Automate tasks and produce reproducible, graded lab outputs. - Free, fast, and available by default on many security distributions.

Learning outcomes - Install radare2 on Kali Linux and verify setup. - Load a binary, run analysis, and navigate code/data. - Find strings, imports, exports, sections, and functions. - Use x-refs, graphs, and visual mode for comprehension. - (Optional) Debug a program, set breakpoints, step, and inspect memory. - (Optional) Patch bytes/assembly in a *safe* toy binary.

2) Install & set up on Kali Linux (beginner-friendly)

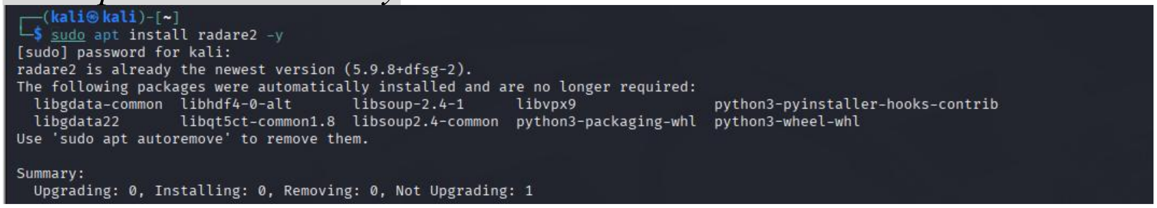
Kali generally packages an up-to-date radare2.

1. ***Update your system***

   ```
   sudo apt update && sudo apt upgrade -y
   ```

2. ***Install radare2***

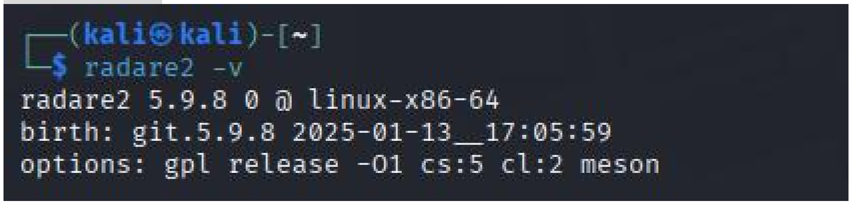
   ```
   sudo apt install radare2 -y
   ```



```
┌──(kali㉿kali)-[~]
└─$ sudo apt install radare2 -y
[sudo] password for kali:
radare2 is already the newest version (5.9.8+dfsg-2).
The following packages were automatically installed and are no longer required:
  libgdata-common  libhdf4-0-alt       libsoup-2.4-1      libvpx9               python3-pyinstaller-hooks-contrib
  libgdata22       libqt5ct-common1.8  libsoup2.4-common  python3-packaging-whl  python3-wheel-whl
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 1
```

3. ***Verify installation***

   radare2 -v

   ```
   ┌──(kali㉿kali)-[~]
   └─$ radare2 -v
   radare2 5.9.8 0 @ linux-x86-64
   birth: git.5.9.8 2025-01-13__17:05:59
   options: gpl release -O1 cs:5 cl:2 meson
   ```

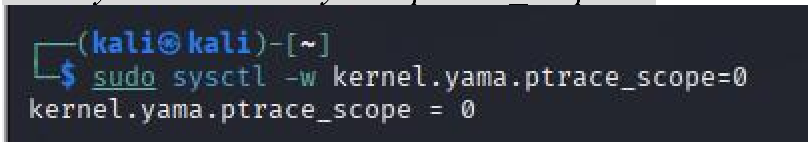
   You should see a version string (e.g., radare2 5.x.y or later).

4. ***(Optional) Install Cutter GUI*** — helpful for visual graphs while still using r2 underneath.

   sudo apt install cutter -y

5. ***(Optional) Enable debugging of child processes*** (some distros restrict ptrace). If needed:

   sudo sysctl -w kernel.yama.ptrace_scope=0

   ```
   ┌──(kali㉿kali)-[~]
   └─$ sudo sysctl -w kernel.yama.ptrace_scope=0
   kernel.yama.ptrace_scope = 0
   ```

   This setting resets on reboot; do not lower it on shared or production machines without approval.

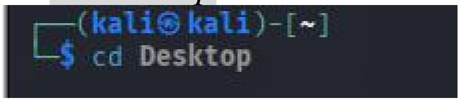
You're ready.

3) First run (static analysis quick start)

We'll use a harmless demo program compiled by you or provided in class (e.g., hello or sample.exe). Commands after the $ are run in your shell; everything else is inside r2.

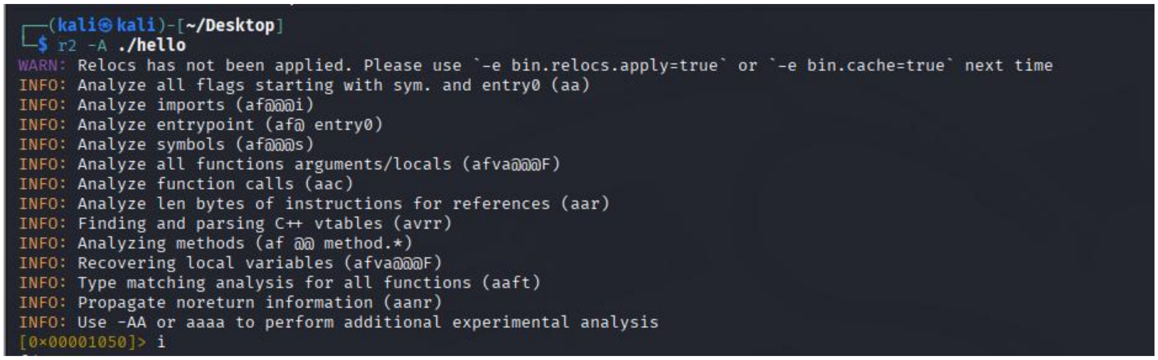
1. ***Open the file with auto-analysis***

   cd Desktop



```
┌──(kali㉿kali)-[~]
└─$ cd Desktop
```

   r2 -A ./hello

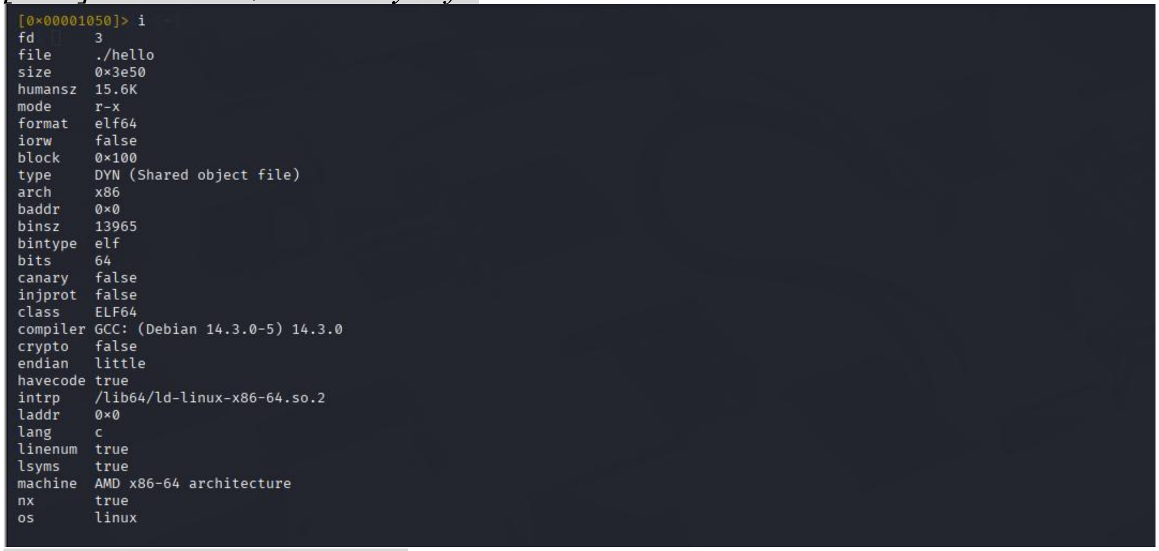


```
┌──(kali㉿kali)-[~/Desktop]
└─$ r2 -A ./hello
WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next time
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods (af @@ method.*)
INFO: Recovering local variables (afva@@@F)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
[0x00001050]> i
```

-A runs analysis (equivalent to aaa after opening). You land at the r2 prompt ([0x0000....]>).

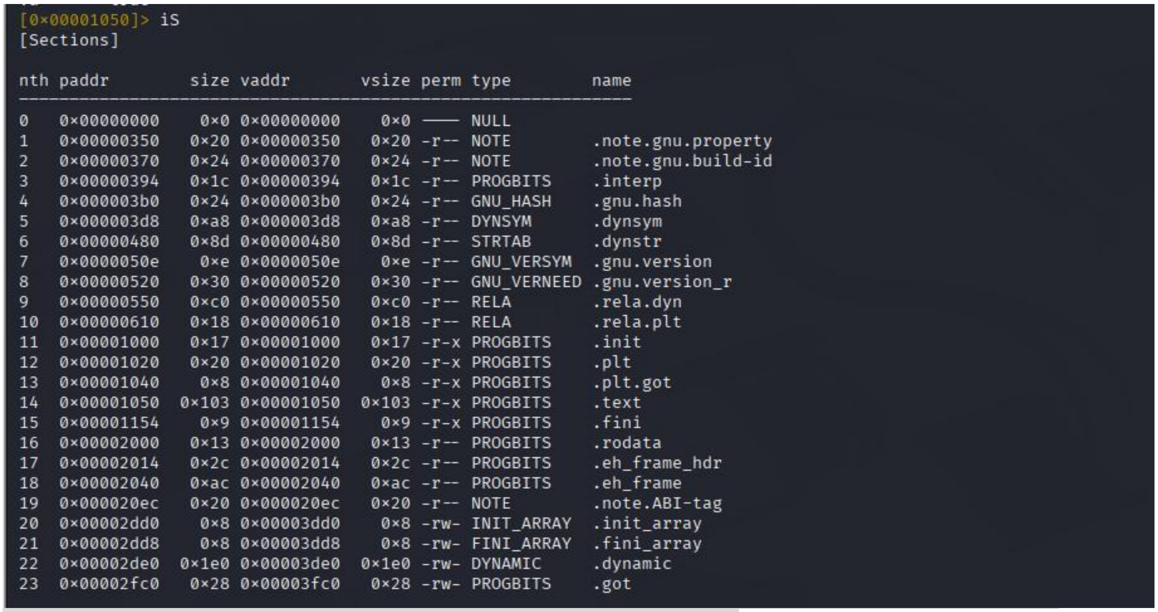
2. ***Basic file information** (inside r2):*

[0x...]> i ; summary info



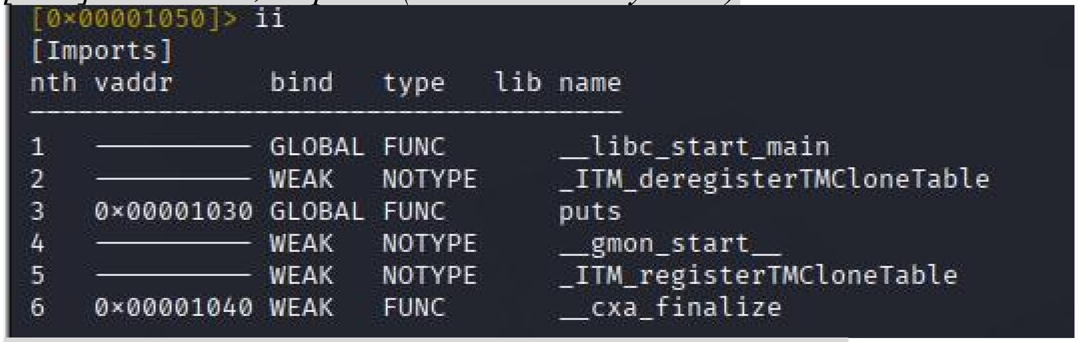
```
[0x00001050]> i
fd       3
file     ./hello
size     0x3e50
humansz  15.6K
mode     r-x
format   elf64
iorw     false
block    0x100
type     DYN (Shared object file)
arch     x86
baddr    0x0
binsz    13965
bintype  elf
bits     64
canary   false
injprot  false
class    ELF64
compiler GCC: (Debian 14.3.0-5) 14.3.0
crypto   false
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
laddr    0x0
lang     c
linenum  true
lsyms    true
machine  AMD x86-64 architecture
nx       true
os       linux
```

[0x...]> iS ; Sections



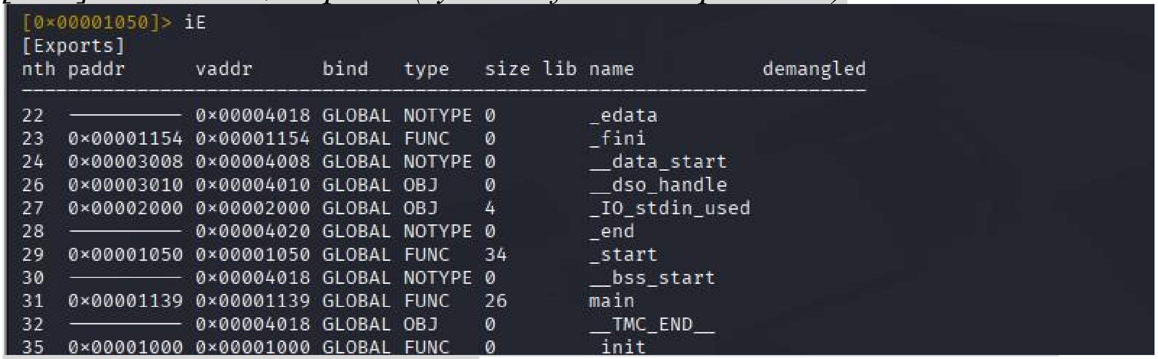
```
[0×00001050]> iS
[Sections]

nth paddr        size vaddr       vsize perm type        name
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――

0   0×00000000    0×0 0×00000000    0×0 ―――― NULL
1   0×00000350   0×20 0×00000350   0×20 -r-- NOTE        .note.gnu.property
2   0×00000370   0×24 0×00000370   0×24 -r-- NOTE        .note.gnu.build-id
3   0×00000394   0×1c 0×00000394   0×1c -r-- PROGBITS    .interp
4   0×000003b0   0×24 0×000003b0   0×24 -r-- GNU_HASH    .gnu.hash
5   0×000003d8   0×a8 0×000003d8   0×a8 -r-- DYNSYM      .dynsym
6   0×00000480   0×8d 0×00000480   0×8d -r-- STRTAB      .dynstr
7   0×0000050e    0×e 0×0000050e    0×e -r-- GNU_VERSYM  .gnu.version
8   0×00000520   0×30 0×00000520   0×30 -r-- GNU_VERNEED .gnu.version_r
9   0×00000550   0×c0 0×00000550   0×c0 -r-- RELA        .rela.dyn
10  0×00000610   0×18 0×00000610   0×18 -r-- RELA        .rela.plt
11  0×00001000   0×17 0×00001000   0×17 -r-x PROGBITS    .init
12  0×00001020   0×20 0×00001020   0×20 -r-x PROGBITS    .plt
13  0×00001040    0×8 0×00001040    0×8 -r-x PROGBITS    .plt.got
14  0×00001050  0×103 0×00001050  0×103 -r-x PROGBITS    .text
15  0×00001154    0×9 0×00001154    0×9 -r-x PROGBITS    .fini
16  0×00002000   0×13 0×00002000   0×13 -r-- PROGBITS    .rodata
17  0×00002014   0×2c 0×00002014   0×2c -r-- PROGBITS    .eh_frame_hdr
18  0×00002040   0×ac 0×00002040   0×ac -r-- PROGBITS    .eh_frame
19  0×000020ec   0×20 0×000020ec   0×20 -r-- NOTE        .note.ABI-tag
20  0×00002dd0    0×8 0×00003dd0    0×8 -rw- INIT_ARRAY  .init_array
21  0×00002dd8    0×8 0×00003dd8    0×8 -rw- FINI_ARRAY  .fini_array
22  0×00002de0  0×1e0 0×00003de0  0×1e0 -rw- DYNAMIC     .dynamic
23  0×00002fc0   0×28 0×00003fc0   0×28 -rw- PROGBITS    .got
```

[0x...]> ii ; Imports (APIs the binary calls)



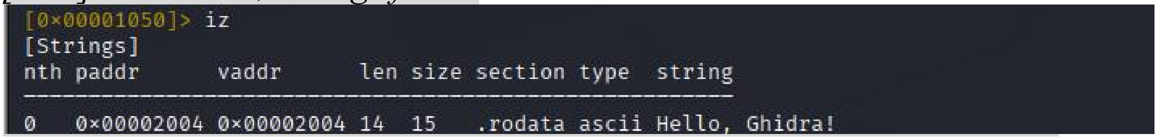
```
[0×00001050]> ii
[Imports]
nth vaddr      bind   type   lib name
―――――――――――――――――――――――――――――――――――――――――

1   ―――――――――― GLOBAL FUNC       __libc_start_main
2   ―――――――――― WEAK   NOTYPE     _ITM_deregisterTMCloneTable
3   0×00001030 GLOBAL FUNC       puts
4   ―――――――――― WEAK   NOTYPE     __gmon_start__
5   ―――――――――― WEAK   NOTYPE     _ITM_registerTMCloneTable
6   0×00001040 WEAK   FUNC       __cxa_finalize
```

[0x...]> iE ; Exports (symbols/functions provided)



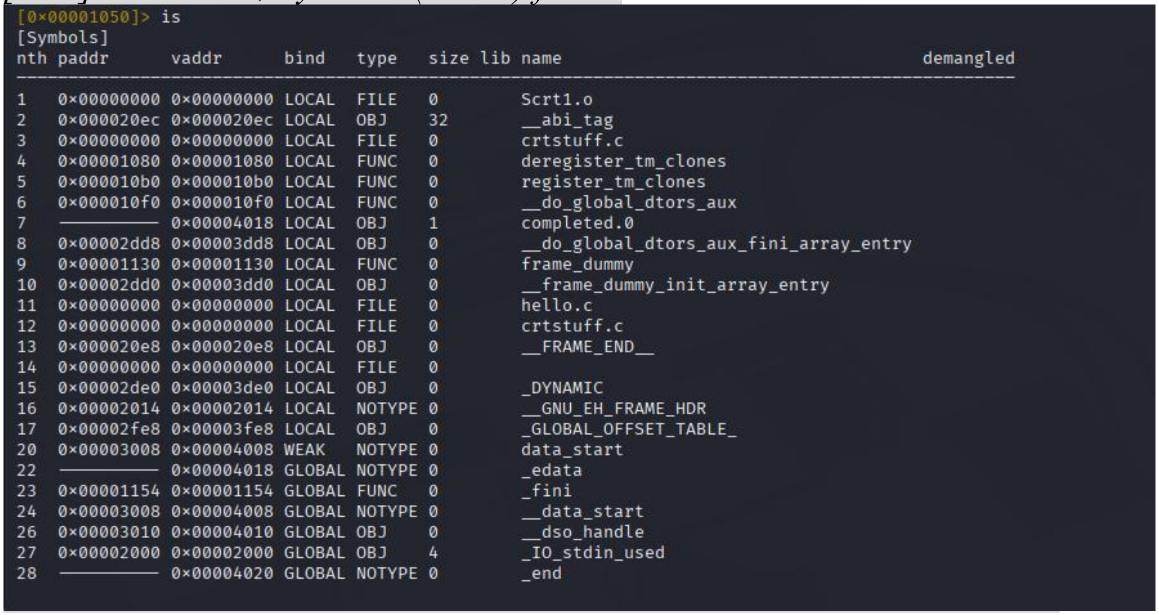
```
[0×00001050]> iE
[Exports]
nth paddr      vaddr      bind   type   size lib name           demangled
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――

22  ―――――――――― 0×00004018 GLOBAL NOTYPE 0        _edata
23  0×00001154 0×00001154 GLOBAL FUNC   0        _fini
24  0×00003008 0×00004008 GLOBAL NOTYPE 0        __data_start
26  0×00003010 0×00004010 GLOBAL OBJ    0        __dso_handle
27  0×00002000 0×00002000 GLOBAL OBJ    4        _IO_stdin_used
28  ―――――――――― 0×00004020 GLOBAL NOTYPE 0        _end
29  0×00001050 0×00001050 GLOBAL FUNC   34       _start
30  ―――――――――― 0×00004018 GLOBAL NOTYPE 0        __bss_start
31  0×00001139 0×00001139 GLOBAL FUNC   26       main
32  ―――――――――― 0×00004018 GLOBAL OBJ    0        __TMC_END__
35  0×00001000 0×00001000 GLOBAL FUNC   0         init
```

[0x...]> iz ; Strings found



```
[0×00001050]> iz
[Strings]
nth paddr      vaddr      len size section type  string
―――――――――――――――――――――――――――――――――――――――――――――――――――――――

0   0×00002004 0×00002004 14  15   .rodata ascii Hello, Ghidra!
```

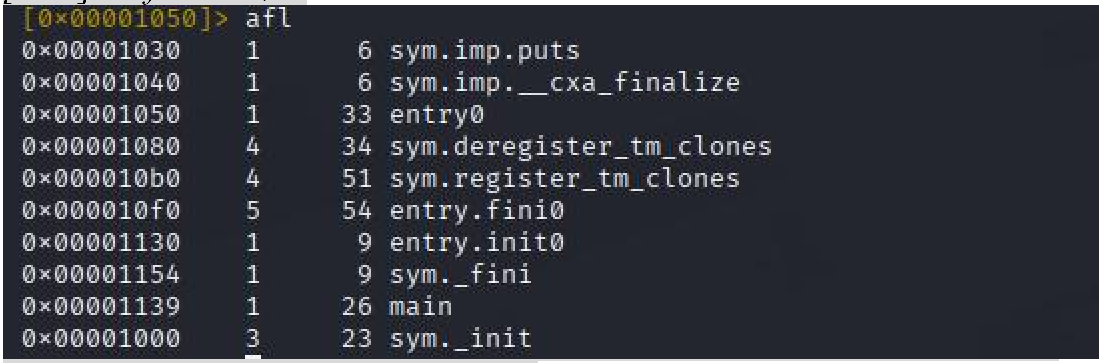
[0x...]> is ; Symbols (labels) found



```
[0x00001050]> is
[Symbols]
nth paddr      vaddr      bind   type   size lib name                                      demangled
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
1   0x00000000 0x00000000 LOCAL  FILE   0        Scrt1.o
2   0x000020ec 0x000020ec LOCAL  OBJ    32       __abi_tag
3   0x00000000 0x00000000 LOCAL  FILE   0        crtstuff.c
4   0x00001080 0x00001080 LOCAL  FUNC   0        deregister_tm_clones
5   0x000010b0 0x000010b0 LOCAL  FUNC   0        register_tm_clones
6   0x000010f0 0x000010f0 LOCAL  FUNC   0        __do_global_dtors_aux
7   ―――――――――― 0x00004018 LOCAL  OBJ    1        completed.0
8   0x00002dd8 0x00003dd8 LOCAL  OBJ    0        __do_global_dtors_aux_fini_array_entry
9   0x00001130 0x00001130 LOCAL  FUNC   0        frame_dummy
10  0x00002dd0 0x00003dd0 LOCAL  OBJ    0        __frame_dummy_init_array_entry
11  0x00000000 0x00000000 LOCAL  FILE   0        hello.c
12  0x00000000 0x00000000 LOCAL  FILE   0        crtstuff.c
13  0x000020e8 0x000020e8 LOCAL  OBJ    0        __FRAME_END__
14  0x00000000 0x00000000 LOCAL  FILE   0
15  0x00002de0 0x00003de0 LOCAL  OBJ    0        _DYNAMIC
16  0x00002014 0x00002014 LOCAL  NOTYPE 0        __GNU_EH_FRAME_HDR
17  0x00002fe8 0x00003fe8 LOCAL  OBJ    0        _GLOBAL_OFFSET_TABLE_
20  0x00003008 0x00004008 WEAK   NOTYPE 0        data_start
22  ―――――――――― 0x00004018 GLOBAL NOTYPE 0        _edata
23  0x00001154 0x00001154 GLOBAL FUNC   0        _fini
24  0x00003008 0x00004008 GLOBAL NOTYPE 0        __data_start
26  0x00003010 0x00004010 GLOBAL OBJ    0        __dso_handle
27  0x00002000 0x00002000 GLOBAL OBJ    4        _IO_stdin_used
28  ―――――――――― 0x00004020 GLOBAL NOTYPE 0        _end
```

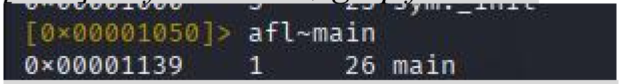
3. ***List functions, find main, and disassemble it***

[0x...]> afl ; A



```
[0x00001050]> afl
0x00001030    1      6 sym.imp.puts
0x00001040    1      6 sym.imp.__cxa_finalize
0x00001050    1     33 entry0
0x00001080    4     34 sym.deregister_tm_clones
0x000010b0    4     51 sym.register_tm_clones
0x000010f0    5     54 entry.fini0
0x00001130    1      9 entry.init0
0x00001154    1      9 sym._fini
0x00001139    1     26 main
0x00001000    3     23 sym._init
```

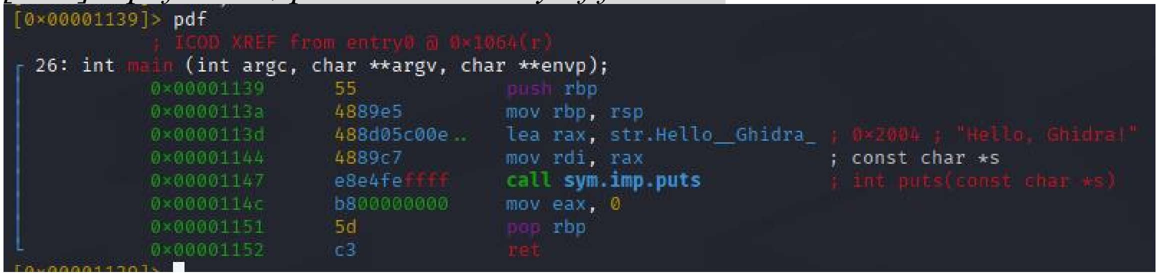
[0x...]> afl~main ; grep for main



```
[0x00001050]> afl~main
0x00001139    1     26 main
```

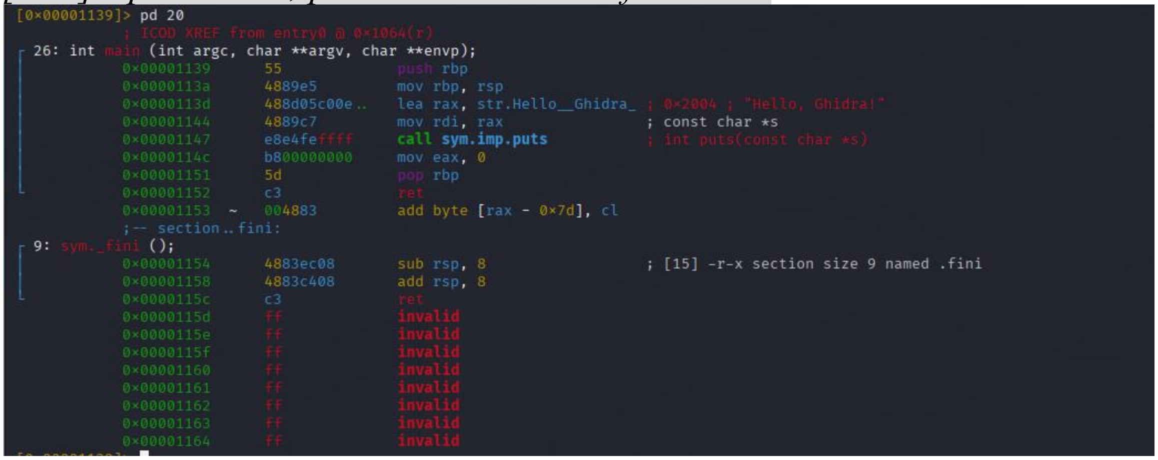
[0x...]> s sym.main ; seek/jump to main

[0x...]> pdf ; print disassembly of function



```
[0x00001139]> pdf
            ; ICOD XREF from entry0 @ 0x1064(r)
┌ 26: int main (int argc, char **argv, char **envp);
│           0x00001139      55             push rbp
│           0x0000113a      4889e5         mov rbp, rsp
│           0x0000113d      488d05c00e..   lea rax, str.Hello__Ghidra_ ; 0x2004 ; "Hello, Ghidra!"
│           0x00001144      4889c7         mov rdi, rax                ; const char *s
│           0x00001147      e8e4feffff     call sym.imp.puts           ; int puts(const char *s)
│           0x0000114c      b800000000     mov eax, 0
│           0x00001151      5d             pop rbp
└           0x00001152      c3             ret
```

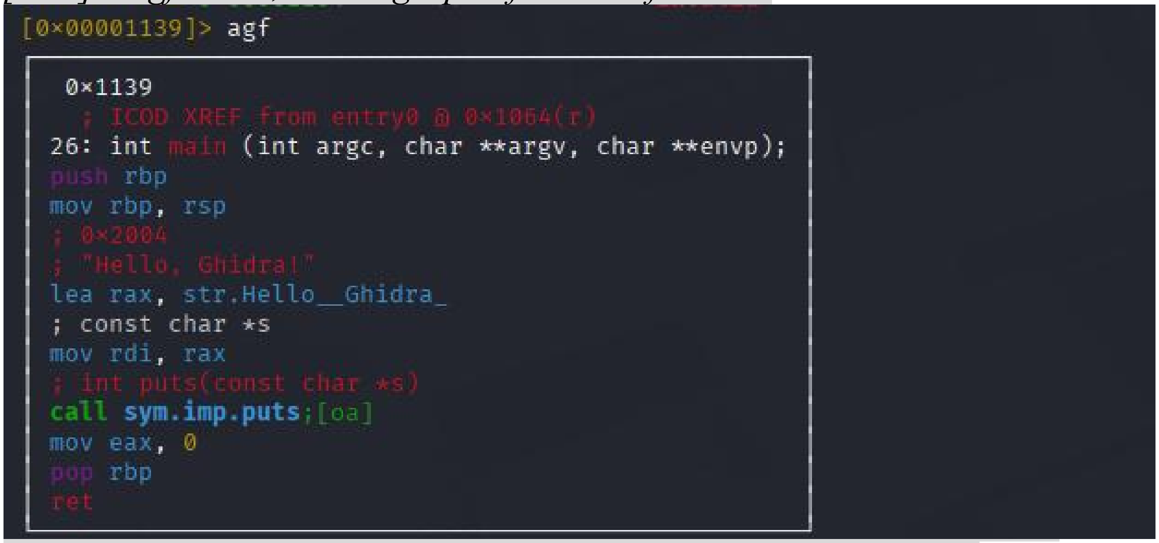
[0x...]> pd 20   ; print 20 instructions from here



```
[0x00001139]> pd 20
            ; ICOD XREF from entry0 @ 0x1064(r)
┌ 26: int main (int argc, char **argv, char **envp);
│           0x00001139      55             push rbp
│           0x0000113a      4889e5         mov rbp, rsp
│           0x0000113d      488d05c00e..   lea rax, str.Hello__Ghidra_ ; 0x2004 ; "Hello, Ghidra!"
│           0x00001144      4889c7         mov rdi, rax                ; const char *s
│           0x00001147      e8e4feffff     call sym.imp.puts           ; int puts(const char *s)
│           0x0000114c      b800000000     mov eax, 0
│           0x00001151      5d             pop rbp
└           0x00001152      c3             ret
            0x00001153 ~    004883         add byte [rax - 0x7d], cl
            ;-- section..fini:
┌ 9: sym._fini ();
│           0x00001154      4883ec08       sub rsp, 8                  ; [15] -r-x section size 9 named .fini
│           0x00001158      4883c408       add rsp, 8
└           0x0000115c      c3             ret
            0x0000115d      ff             invalid
            0x0000115e      ff             invalid
            0x0000115f      ff             invalid
            0x00001160      ff             invalid
            0x00001161      ff             invalid
            0x00001162      ff             invalid
            0x00001163      ff             invalid
            0x00001164      ff             invalid
```

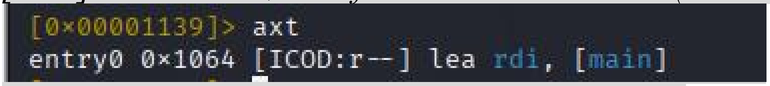
4. ***High-level view (graphs & x-refs)***

[0x...]> agf   ; ASCII graph of current function



```
[0x00001139]> agf

 ┌──────────────────────────────────────────────────────┐
 │  0x1139                                              │
 │   ; ICOD XREF from entry0 @ 0x1064(r)                │
 │ 26: int main (int argc, char **argv, char **envp);   │
 │ push rbp                                             │
 │ mov rbp, rsp                                         │
 │ ; 0x2004                                             │
 │ ; "Hello, Ghidra!"                                   │
 │ lea rax, str.Hello__Ghidra_                          │
 │ ; const char *s                                      │
 │ mov rdi, rax                                         │
 │ ; int puts(const char *s)                            │
 │ call sym.imp.puts;[oa]                               │
 │ mov eax, 0                                           │
 │ pop rbp                                              │
 │ ret                                                  │
 └──────────────────────────────────────────────────────┘
```

[0x...]> axt   ; X-refs to current address (who calls/uses this)



```
[0x00001139]> axt
entry0 0x1064 [ICOD:r--] lea rdi, [main]
```

[0x...]> axt sym.main ; X-refs to symbol main
[0x...]> axf sym.main ; X-refs from main (what it calls)

5. ***Quit r2***

[0x...]> q

4) r2 navigation & help (the essentials)

- ? or ?? — general help; ?cmd shows help for a command (e.g., ?afl).
- s <addr|sym> — **seek** to address or symbol (jump cursor).

- s+ 0x20 / s- 0x20 — move forward/backward.
- pd N @ addr — disassemble N instructions at address.
- V — enter **visual mode**. Press ? inside for keys. q to exit.
  - p cycles views (disasm/hex/bytes), g shows graph, ENTER to follow call/jump.
- af? / ax? / p? — topic-specific help.

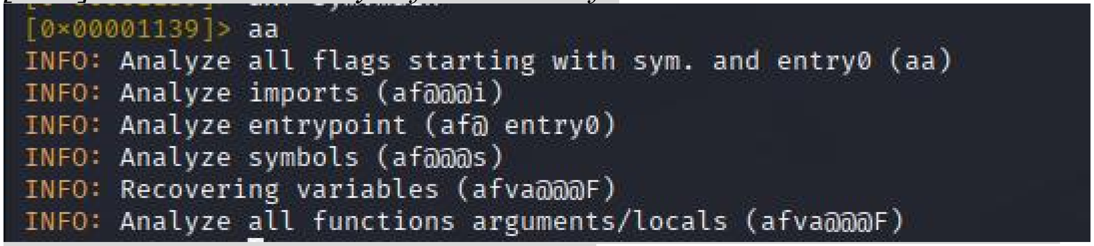
5) Deeper analysis workflow (step-by-step)

1. **Open without auto-analysis, then analyze manually**

   ```
   r2 ./hello
   ```

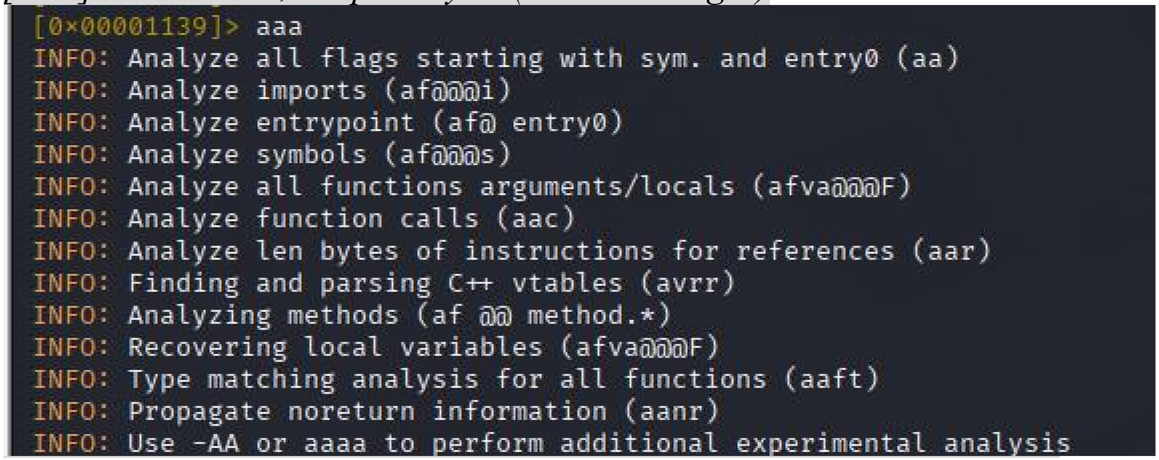
   Inside r2:

   ```
   [0x...]> aa          ; analyze functions/refs
   ```



```
[0x00001139]> aa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Recovering variables (afva@@@F)
INFO: Analyze all functions arguments/locals (afva@@@F)
```

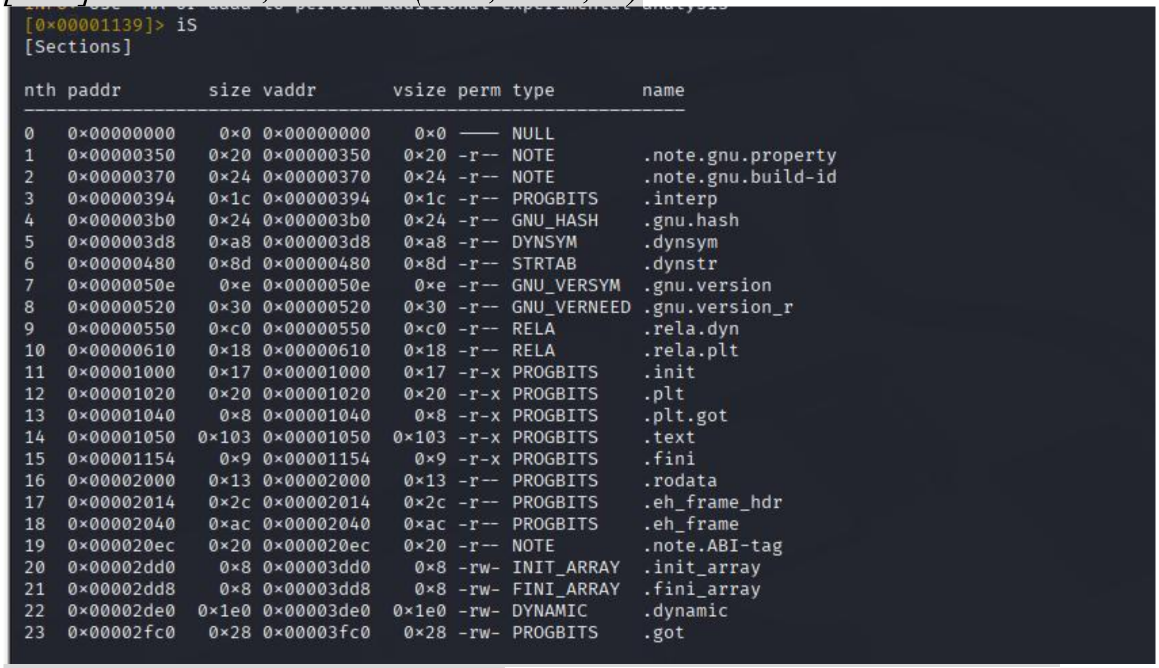
```
[0x...]> aac          ; analyze calls (light)
[0x...]> aae          ; analyze esil emulation hints
[0x...]> aaa          ; deep analysis (can take longer)
```



```
[0x00001139]> aaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods (af @@ method.*)
INFO: Recovering local variables (afva@@@F)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
```

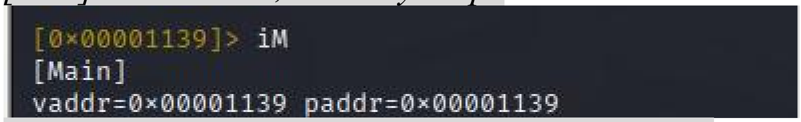
2. ***Explore program structure***

[0x...]> iS ; sections (.text, .data, ...)



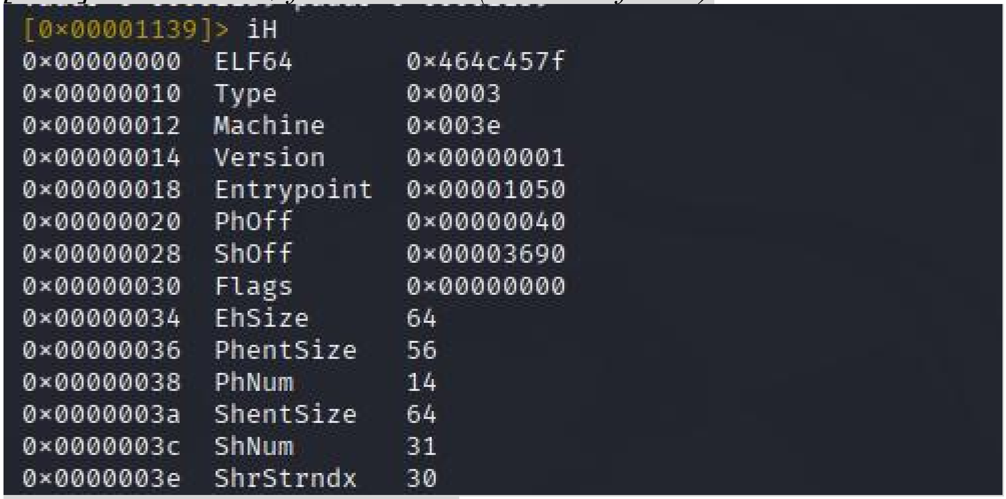
```
[0x00001139]> iS
[Sections]

nth paddr        size vaddr       vsize perm type        name
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――

0   0x00000000    0x0 0x00000000    0x0 ---- NULL
1   0x00000350   0x20 0x00000350   0x20 -r-- NOTE        .note.gnu.property
2   0x00000370   0x24 0x00000370   0x24 -r-- NOTE        .note.gnu.build-id
3   0x00000394   0x1c 0x00000394   0x1c -r-- PROGBITS    .interp
4   0x000003b0   0x24 0x000003b0   0x24 -r-- GNU_HASH    .gnu.hash
5   0x000003d8   0xa8 0x000003d8   0xa8 -r-- DYNSYM      .dynsym
6   0x00000480   0x8d 0x00000480   0x8d -r-- STRTAB      .dynstr
7   0x0000050e    0xe 0x0000050e    0xe -r-- GNU_VERSYM  .gnu.version
8   0x00000520   0x30 0x00000520   0x30 -r-- GNU_VERNEED .gnu.version_r
9   0x00000550   0xc0 0x00000550   0xc0 -r-- RELA        .rela.dyn
10  0x00000610   0x18 0x00000610   0x18 -r-- RELA        .rela.plt
11  0x00001000   0x17 0x00001000   0x17 -r-x PROGBITS    .init
12  0x00001020   0x20 0x00001020   0x20 -r-x PROGBITS    .plt
13  0x00001040    0x8 0x00001040    0x8 -r-x PROGBITS    .plt.got
14  0x00001050  0x103 0x00001050  0x103 -r-x PROGBITS    .text
15  0x00001154    0x9 0x00001154    0x9 -r-x PROGBITS    .fini
16  0x00002000   0x13 0x00002000   0x13 -r-- PROGBITS    .rodata
17  0x00002014   0x2c 0x00002014   0x2c -r-- PROGBITS    .eh_frame_hdr
18  0x00002040   0xac 0x00002040   0xac -r-- PROGBITS    .eh_frame
19  0x000020ec   0x20 0x000020ec   0x20 -r-- NOTE        .note.ABI-tag
20  0x00002dd0    0x8 0x00003dd0    0x8 -rw- INIT_ARRAY  .init_array
21  0x00002dd8    0x8 0x00003dd8    0x8 -rw- FINI_ARRAY  .fini_array
22  0x00002de0  0x1e0 0x00003de0  0x1e0 -rw- DYNAMIC     .dynamic
23  0x00002fc0   0x28 0x00003fc0   0x28 -rw- PROGBITS    .got
```

[0x...]> iM ; memory maps



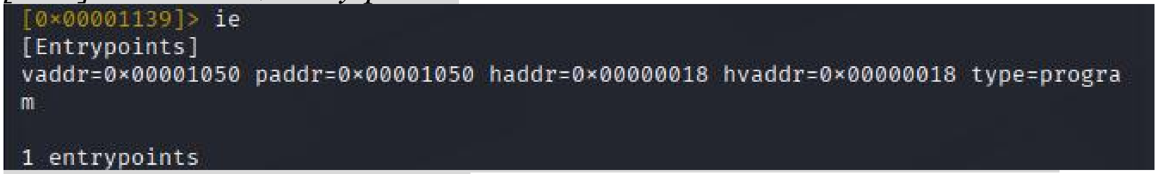
```
[0x00001139]> iM
[Main]
vaddr=0x00001139 paddr=0x00001139
```

[0x...]> iH ; file headers (ELF/PE fields)



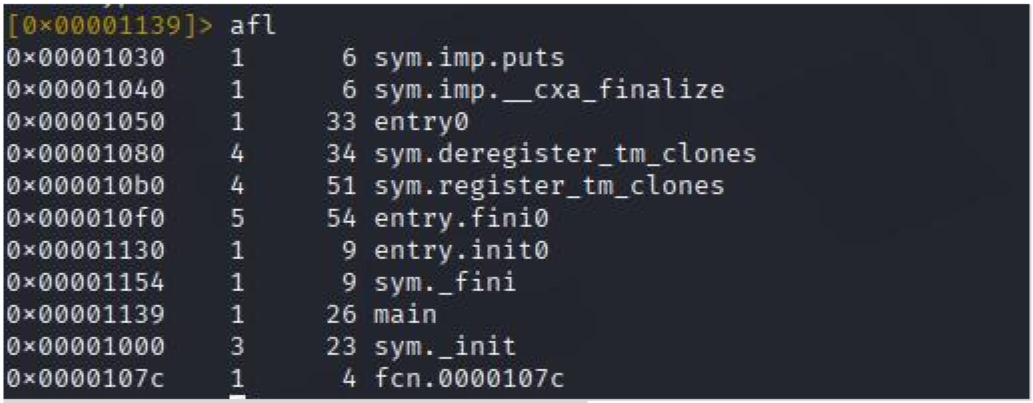
```
[0x00001139]> iH
0x00000000  ELF64       0x464c457f
0x00000010  Type        0x0003
0x00000012  Machine     0x003e
0x00000014  Version     0x00000001
0x00000018  Entrypoint  0x00001050
0x00000020  PhOff       0x00000040
0x00000028  ShOff       0x00003690
0x00000030  Flags       0x00000000
0x00000034  EhSize      64
0x00000036  PhentSize   56
0x00000038  PhNum       14
0x0000003a  ShentSize   64
0x0000003c  ShNum       31
0x0000003e  ShrStrndx   30
```

[0x...]> ie ; entry points



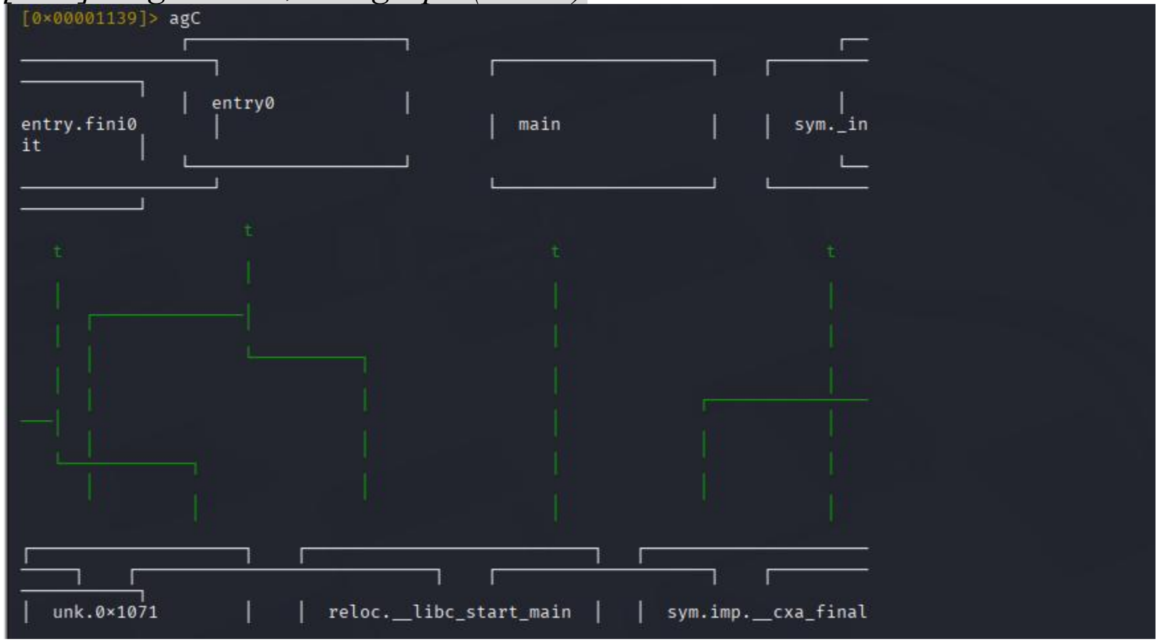
```
[0x00001139]> ie
[Entrypoints]
vaddr=0x00001050 paddr=0x00001050 haddr=0x00000018 hvaddr=0x00000018 type=progra
m

1 entrypoints
```

[0x...]> afl ; all functions



```
[0x00001139]> afl
0x00001030    1      6 sym.imp.puts
0x00001040    1      6 sym.imp.__cxa_finalize
0x00001050    1     33 entry0
0x00001080    4     34 sym.deregister_tm_clones
0x000010b0    4     51 sym.register_tm_clones
0x000010f0    5     54 entry.fini0
0x00001130    1      9 entry.init0
0x00001154    1      9 sym._fini
0x00001139    1     26 main
0x00001000    3     23 sym._init
0x0000107c    1      4 fcn.0000107c
```

[0x...]> agC   ; call graph (ASCII)

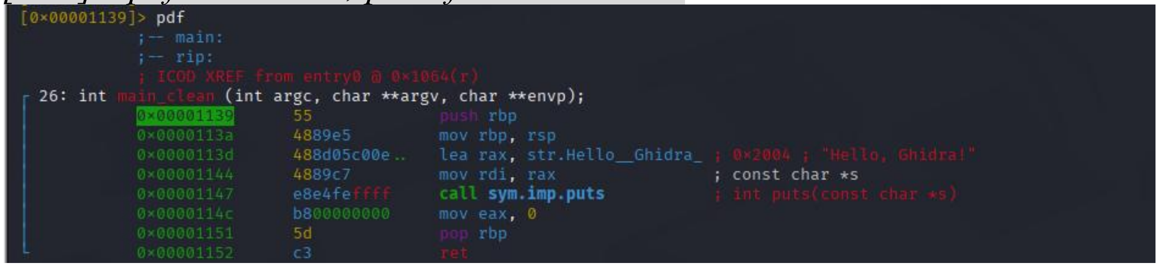


```
[0x00001139]> agC
```

3. *Work with functions*

[0x...]> s sym.main   ; jump to main
[0x...]> afn main_clean @ $$; rename current function ("$$" is current addr)
[0x...]> af @ 0x401000   ; create function at address if missing
[0x...]> af- @ 0x401000   ; delete function definition
[0x...]> afl~main_clean   ; verify rename

```
[0x...]> pdf                    ; print function disasm
```



```
[0x00001139]> pdf
            ;-- main:
            ;-- rip:
            ; ICOD XREF from entry0 @ 0x1064(r)
┌ 26: int main_clean (int argc, char **argv, char **envp);
│           0x00001139      55             push rbp
│           0x0000113a      4889e5         mov rbp, rsp
│           0x0000113d      488d05c00e..   lea rax, str.Hello__Ghidra_ ; 0x2004 ; "Hello, Ghidra!"
│           0x00001144      4889c7         mov rdi, rax                ; const char *s
│           0x00001147      e8e4feffff     call sym.imp.puts           ; int puts(const char *s)
│           0x0000114c      b800000000     mov eax, 0
│           0x00001151      5d             pop rbp
└           0x00001152      c3             ret
```

4. Strings & references

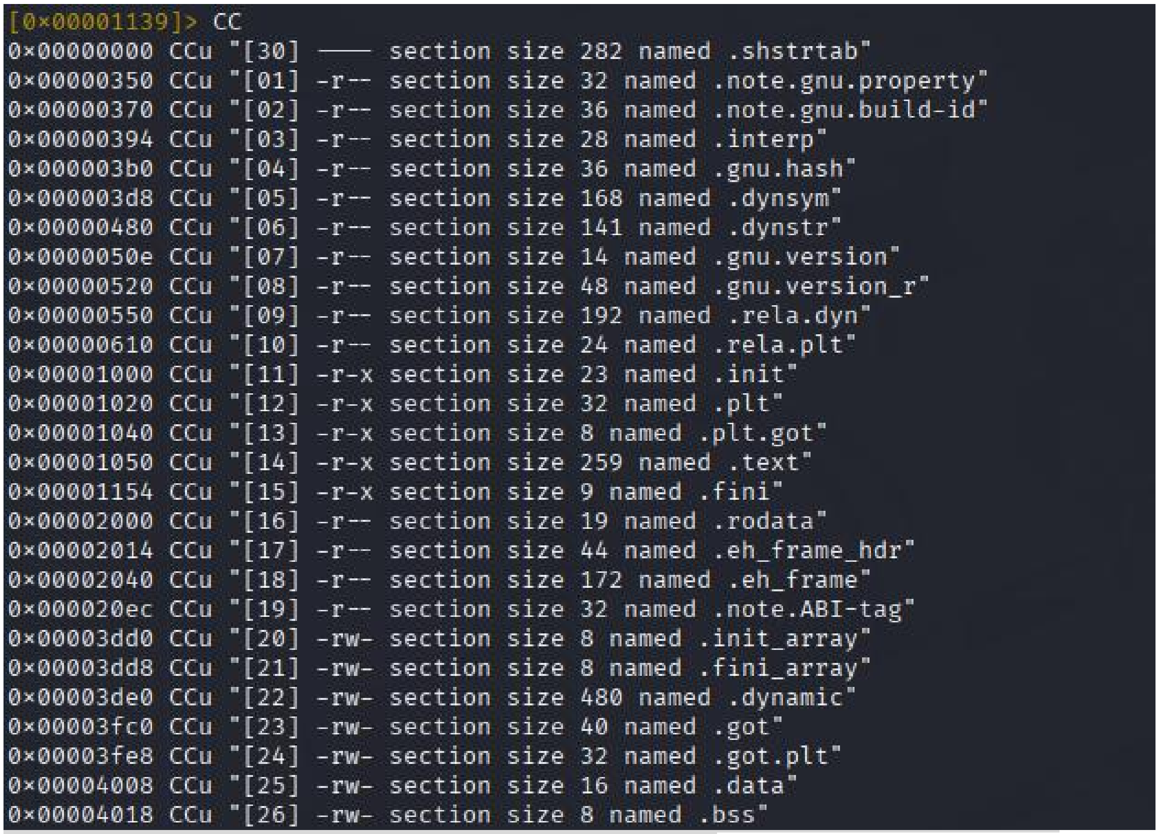
```
[0x...]> iz                   ; list strings
[0x...]> iz~hello             ; filter strings containing "hello"
[0x...]> axt @ str.hello      ; show who references a specific string
[0x...]> s `axt~[1]`          ; seek to the first xref (example of using back ticks)
```

5. Search

```
[0x...]> /c hello             ; search ASCII string "hello"
[0x...]> /x 9090              ; search hex pattern 90 90
[0x...]> /i call              ; search for instruction mnemonic
[0x...]> ?/                   ; help for search family
```

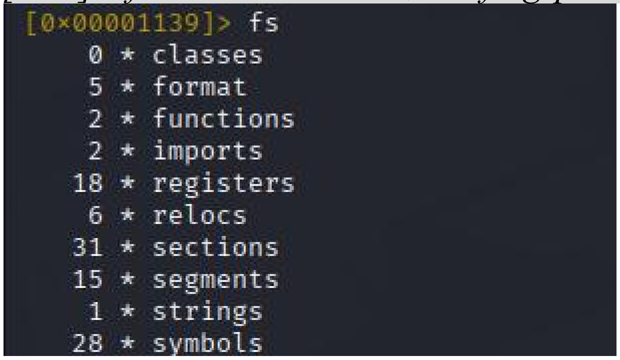
6. Comments, flags, bookmarks

```
[0x...]> CC This prints the greeting   ; add a comment at current address
```



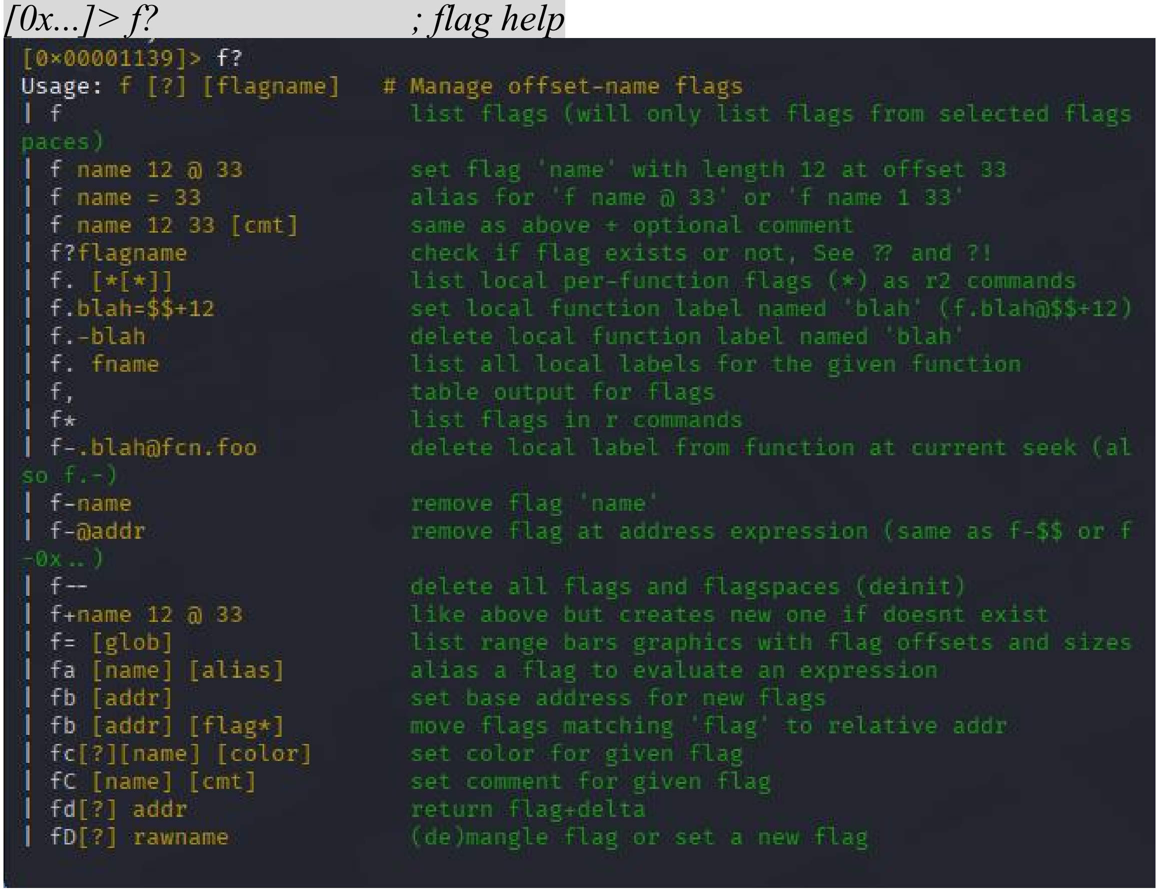
```
[0×00001139]> CC
0×00000000 CCu "[30] ---- section size 282 named .shstrtab"
0×00000350 CCu "[01] -r-- section size 32 named .note.gnu.property"
0×00000370 CCu "[02] -r-- section size 36 named .note.gnu.build-id"
0×00000394 CCu "[03] -r-- section size 28 named .interp"
0×000003b0 CCu "[04] -r-- section size 36 named .gnu.hash"
0×000003d8 CCu "[05] -r-- section size 168 named .dynsym"
0×00000480 CCu "[06] -r-- section size 141 named .dynstr"
0×0000050e CCu "[07] -r-- section size 14 named .gnu.version"
0×00000520 CCu "[08] -r-- section size 48 named .gnu.version_r"
0×00000550 CCu "[09] -r-- section size 192 named .rela.dyn"
0×00000610 CCu "[10] -r-- section size 24 named .rela.plt"
0×00001000 CCu "[11] -r-x section size 23 named .init"
0×00001020 CCu "[12] -r-x section size 32 named .plt"
0×00001040 CCu "[13] -r-x section size 8 named .plt.got"
0×00001050 CCu "[14] -r-x section size 259 named .text"
0×00001154 CCu "[15] -r-x section size 9 named .fini"
0×00002000 CCu "[16] -r-- section size 19 named .rodata"
0×00002014 CCu "[17] -r-- section size 44 named .eh_frame_hdr"
0×00002040 CCu "[18] -r-- section size 172 named .eh_frame"
0×000020ec CCu "[19] -r-- section size 32 named .note.ABI-tag"
0×00003dd0 CCu "[20] -rw- section size 8 named .init_array"
0×00003dd8 CCu "[21] -rw- section size 8 named .fini_array"
0×00003de0 CCu "[22] -rw- section size 480 named .dynamic"
0×00003fc0 CCu "[23] -rw- section size 40 named .got"
0×00003fe8 CCu "[24] -rw- section size 32 named .got.plt"
0×00004008 CCu "[25] -rw- section size 16 named .data"
0×00004018 CCu "[26] -rw- section size 8 named .bss"
```

```
[0x...]> CCu                    ; remove comment
[0x...]> f my.flag @ $$         ; create a named flag here
[0x...]> fs                     ; list flagspaces
```



```
[0×00001139]> fs
    0 * classes
    5 * format
    2 * functions
    2 * imports
   18 * registers
    6 * relocs
   31 * sections
   15 * segments
    1 * strings
   28 * symbols
```

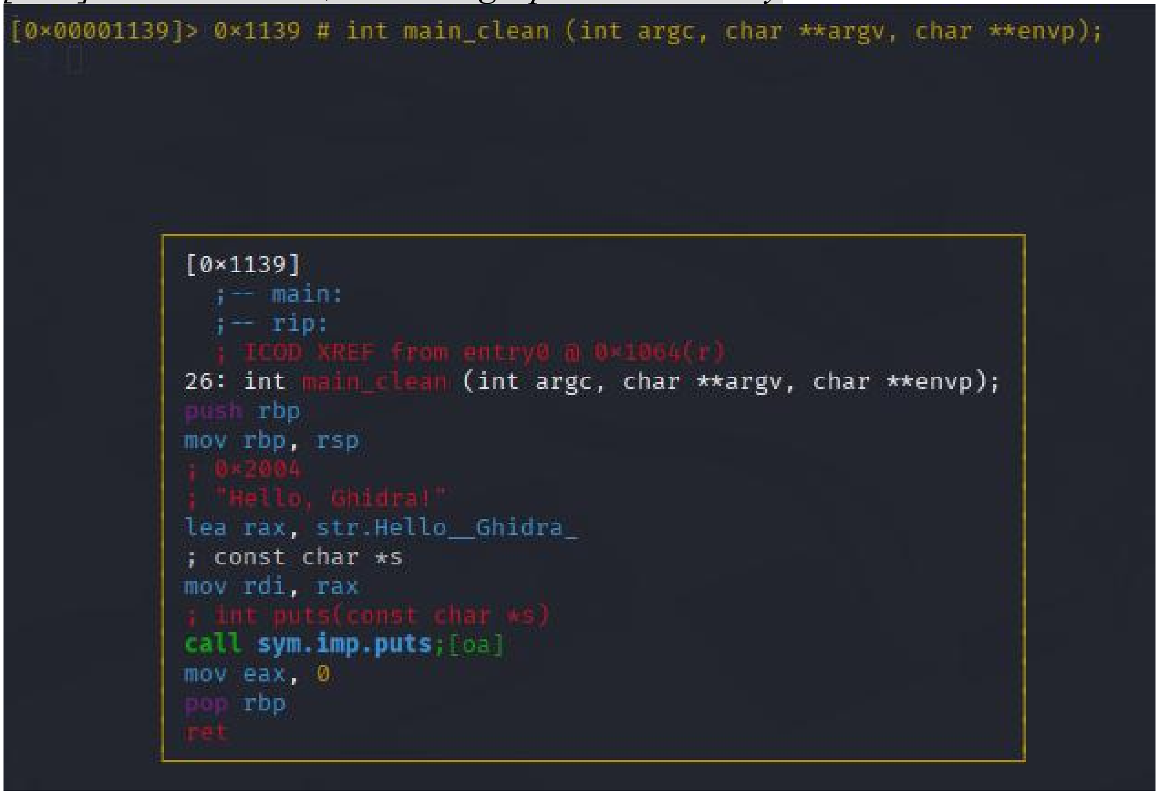
[0x...]> f?   ; flag help



```
[0x00001139]> f?
Usage: f [?] [flagname]   # Manage offset-name flags
| f                         list flags (will only list flags from selected flags
paces)
| f name 12 @ 33            set flag 'name' with length 12 at offset 33
| f name = 33               alias for 'f name @ 33' or 'f name 1 33'
| f name 12 33 [cmt]        same as above + optional comment
| f?flagname                check if flag exists or not, See ?? and ?!
| f. [*[*]]                 list local per-function flags (*) as r2 commands
| f.blah=$$+12              set local function label named 'blah' (f.blah@$$+12)
| f.-blah                   delete local function label named 'blah'
| f. fname                  list all local labels for the given function
| f,                        table output for flags
| f*                        list flags in r commands
| f-.blah@fcn.foo           delete local label from function at current seek (al
so f.-)
| f-name                    remove flag 'name'
| f-@addr                   remove flag at address expression (same as f-$$ or f
-0x..)
| f--                       delete all flags and flagspaces (deinit)
| f+name 12 @ 33            like above but creates new one if doesnt exist
| f= [glob]                 list range bars graphics with flag offsets and sizes
| fa [name] [alias]         alias a flag to evaluate an expression
| fb [addr]                 set base address for new flags
| fb [addr] [flag*]         move flags matching 'flag' to relative addr
| fc[?][name] [color]       set color for given flag
| fC [name] [cmt]           set comment for given flag
| fd[?] addr                return flag+delta
| fD[?] rawname             (de)mangle flag or set a new flag
```

7. ***Visual graph mode (recommended for class demos)***

```
[0x...]> s sym.main
[0x...]> VV                ; visual + graph mode directly
```



```
[0×00001139]> 0×1139 # int main_clean (int argc, char **argv, char **envp);

[0×1139]
  ;-- main:
  ;-- rip:
  ; ICOD XREF from entry0 @ 0×1064(r)
26: int main_clean (int argc, char **argv, char **envp);
push rbp
mov rbp, rsp
; 0×2004
; "Hello, Ghidra!"
lea rax, str.Hello__Ghidra_
; const char *s
mov rdi, rax
; int puts(const char *s)
call sym.imp.puts;[oa]
mov eax, 0
pop rbp
ret
```

Keys inside VV: hjkl or arrow keys to move, ENTER follow edge, x xrefs, ? help, q quit.

6) (Optional) Decompiler plugins

radare2 itself focuses on disassembly. If your lab image includes a decompiler plugin (e.g., **r2ghidra-dec**), you can try:

```
[0x...]> pdg        ; Ghidra-based pseudocode (if plugin available)
```

If not installed, keep using pdf and graphs; students still learn core RE skills.

7) (Optional) Debugging with r2

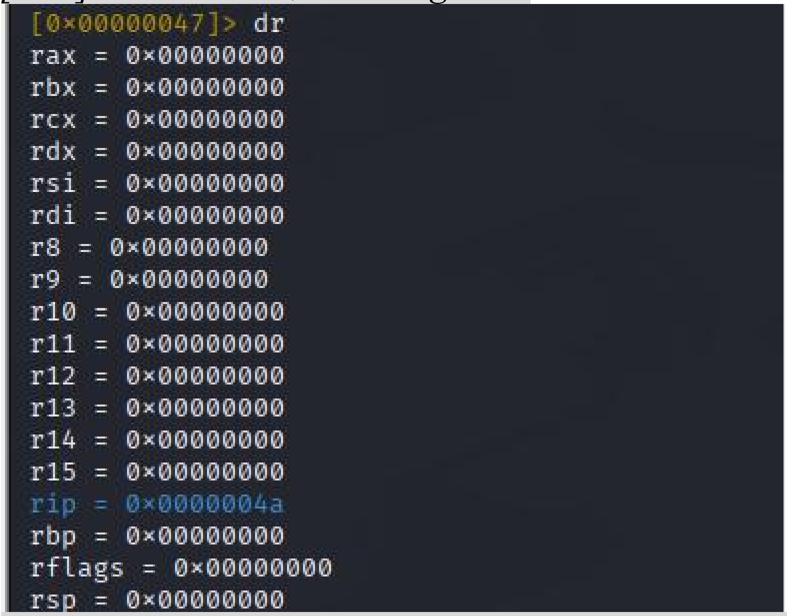
Debug only your own binaries or those you are authorized to analyze.

1. **Start under debugger**

   ```
   r2 -d ./hello arg1 arg2
   ```

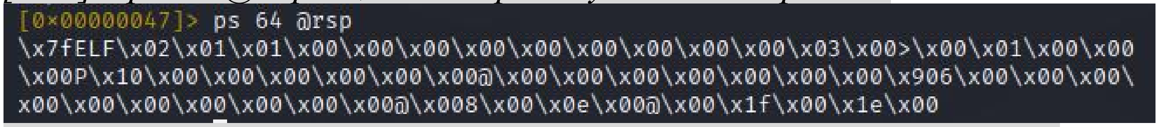
2. **Common debug commands** (inside r2):

```
[0x...]> db sym.main     ; set breakpoint at main
[0x...]> dbi             ; list breakpoints
[0x...]> dc              ; continue execution
[0x...]> ds              ; single step
[0x...]> dso             ; step over
[0x...]> dr              ; show registers
```



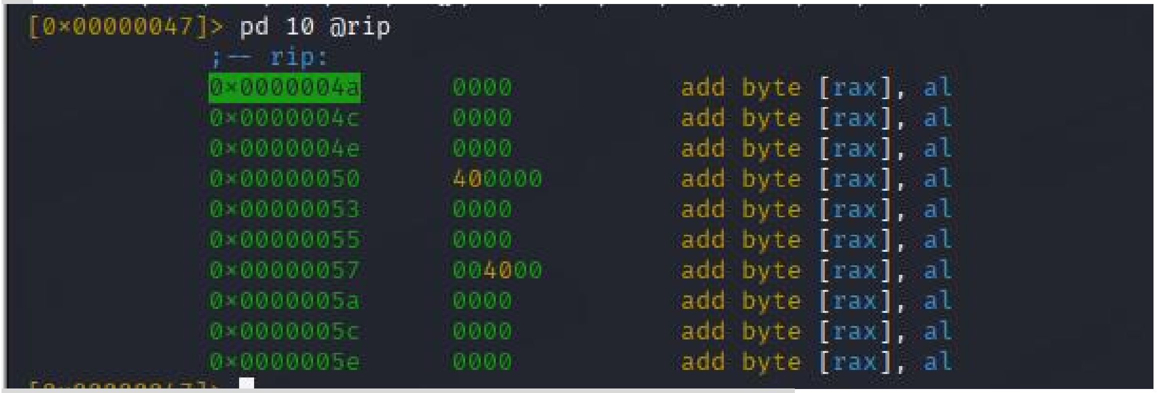
```
[0x00000047]> dr
rax = 0x00000000
rbx = 0x00000000
rcx = 0x00000000
rdx = 0x00000000
rsi = 0x00000000
rdi = 0x00000000
r8  = 0x00000000
r9  = 0x00000000
r10 = 0x00000000
r11 = 0x00000000
r12 = 0x00000000
r13 = 0x00000000
r14 = 0x00000000
r15 = 0x00000000
rip = 0x0000004a
rbp = 0x00000000
rflags = 0x00000000
rsp = 0x00000000
```

```
[0x...]> px 64 @ rsp     ; hexdump 64 bytes at stack pointer
```



```
[0x00000047]> ps 64 @rsp
\x7fELF\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03\x00>\x00\x01\x00\x00
\x00P\x10\x00\x00\x00\x00\x00\x00@\x00\x00\x00\x00\x00\x00\x00\x906\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00@\x008\x00\x0e\x00@\x00\x1f\x00\x1e\x00
```

```
[0x...]> pd 10 @ rip     ; show next 10 instructions at instruction pointer
```



```
[0x00000047]> pd 10 @rip
            ;-- rip:
            0x0000004a      0000           add byte [rax], al
            0x0000004c      0000           add byte [rax], al
            0x0000004e      0000           add byte [rax], al
            0x00000050      400000         add byte [rax], al
            0x00000053      0000           add byte [rax], al
            0x00000055      0000           add byte [rax], al
            0x00000057      004000         add byte [rax], al
            0x0000005a      0000           add byte [rax], al
            0x0000005c      0000           add byte [rax], al
            0x0000005e      0000           add byte [rax], al
```

```
[0x...]> dcu sym.main+0x20 ; continue until address
[0x...]> dpt             ; show backtrace (threads)
[0x...]> doo             ; restart the program
[0x...]> q               ; quit debugger
```

8) (Optional) Safe patching basics

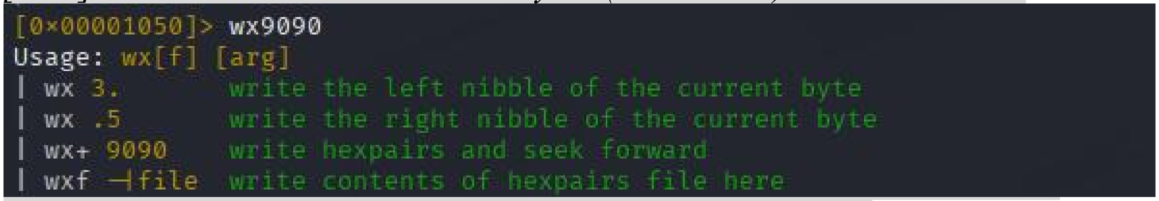
Only patch your own **toy** binaries for learning. Do not use patching to bypass protections.

1. **Reopen file in write mode** (or open initially with -w):

```
[0x...]> oo+             ; reopen with write permissions
```

2. **Write bytes / assembly**

```
[0x...]> wx 9090         ; write hex bytes (NOP,NOP) at current addr
```



```
[0x00001050]> wx9090
Usage: wx[f] [arg]
| wx 3.       write the left nibble of the current byte
| wx .5       write the right nibble of the current byte
| wx+ 9090    write hexpairs and seek forward
| wxf -|file  write contents of hexpairs file here
```

```
[0x...]> wa nop          ; assemble & write instruction here
[0x...]> wa mov eax,0    ; example assemble write (x86)
[0x...]> wv?             ; write values help
```

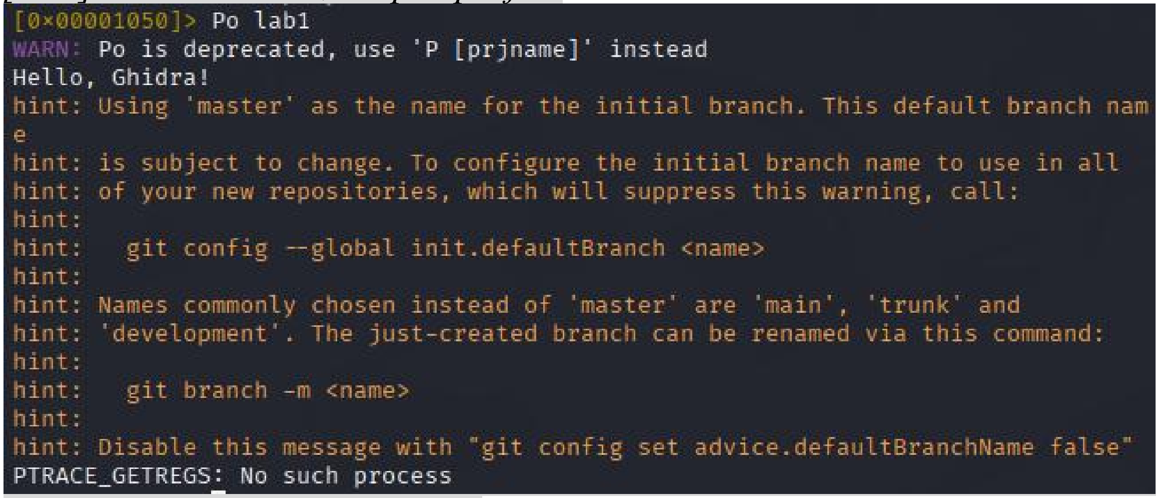
3. **Save changes**

```
[0x...]> wq              ; write and quit
```

9) Projects & scripting (automation)

Projects (keep your analysis database):

```
[0x...]> Ps lab1       ; save project as "lab1"
[0x...]> Po lab1       ; reopen project
```



```
[0x00001050]> Po lab1
WARN: Po is deprecated, use 'P [prjname]' instead
Hello, Ghidra!
hint: Using 'master' as the name for the initial branch. This default branch nam
e
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
hint:
hint: Disable this message with "git config set advice.defaultBranchName false"
PTRACE_GETREGS: No such process
```

```
[0x...]> Pl            ; list projects
[0x...]> Pd lab1       ; delete project
```

One-liner reports (great for grading):

```
r2 -Aqc "afl; pdf @ sym.main" ./hello > report.txt
```

- -q = quiet (exit after commands), -c runs commands in quotes.

r2pipe (Python/Node bindings) is available for advanced automation, but the above one-liners are enough for most beginner labs.

10) Instructor lab recipe

Objective: Identify a function that prints a message and the exact string used.

Setup: Provide a tiny C program (ELF on Linux) that prints a greeting.

Student steps: 1. r2 -A ./hello — open with auto-analysis. 2. iz — list strings; filter with iz~hello. 3. axt @ str.hello — find who references the string. 4. s sym.main (or the xref target) — jump to the function. 5. pdf — read disassembly and confirm the call path to puts/printf. 6. Add a comment CC Prints greeting. 7. Ps lab1_<rollno> — save project.

Deliverable: A short report with: - The function name & address - The exact string content and address - One screenshot of pdf or VV graph with a comment visible

11) Troubleshooting & tips

- **No functions found:** run aaa (deep analysis) and then afl.
- **Can’t find main:** try afl~main, or check entry with ie, then follow init code to the main caller.
- **Decompiler command fails:** plugin not installed; stick to pdf/agf/VV.
- **Write failed:** reopen with -w or use oo+; ensure filesystem permissions allow writing.
- **Debugger won’t attach:** check ptrace_scope (see setup step), or run as root only in a dedicated lab VM.
- **Help on any topic:** type the command followed by ? (e.g., pd?, af?, ax?).

12) Quick cheat-sheet (most-used commands)

Open & analyze - r2 -A file — open and auto-analyze - aa | aaa — analyze (fast | deep)

Info - i, iS, ii, iE, iz, is, iH, ie — file/sections/imports/exports/strings/symbols/headers/entry

Navigation & print - s sym.main — seek to symbol - pd N @ addr — disassemble N instructions - pdf — disassemble current function - agf — ASCII graph of function - VV — visual graph mode

X-refs & search - axt @ addr|sym — refs *to* - axf @ addr|sym — refs *from* - /c <str>, /x <hex>, /i <mnem> — search

Comments & flags - CC <text> — add comment - f name @ addr — create flag

Debug (optional) - -d to start debugging, db/dc/ds/dr/px basics

Patch (optional) - oo+ → wa <asm> or wx <hex> → wq

Appendix: Extra exercises

1. **Control-flow reading:** Use VV on main, follow call edges, and write a 3-sentence summary of the branch conditions.
2. **String hunt:** Use /c and iz to find a hidden flag string; submit its address and the x-ref function.
3. **Mini debug:** Set a breakpoint before a puts call, run dc, step ds, and capture register state with dr.
4. **Pattern search:** Find all NOP sleds with /x 9090 and mark top 3 results with flags.

You're ready to analyze. Start with small, legal samples, annotate thoroughly, and save projects so your findings are reproducible.