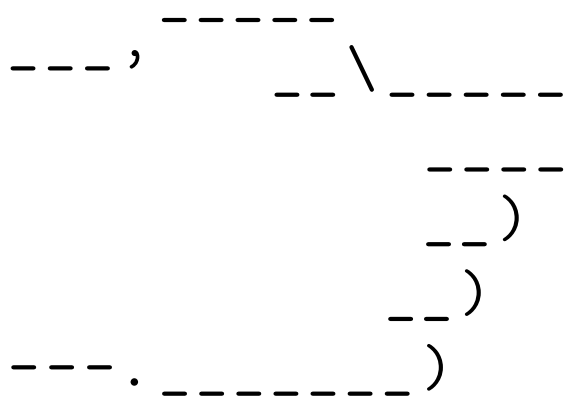


GDB+poke for programmable kernel debugging?

Jose E. Marchesi
jemarch@gnu.org

LPC 2022

Just a hobby, won't be big and professional like drgn
:-P

GNU poke

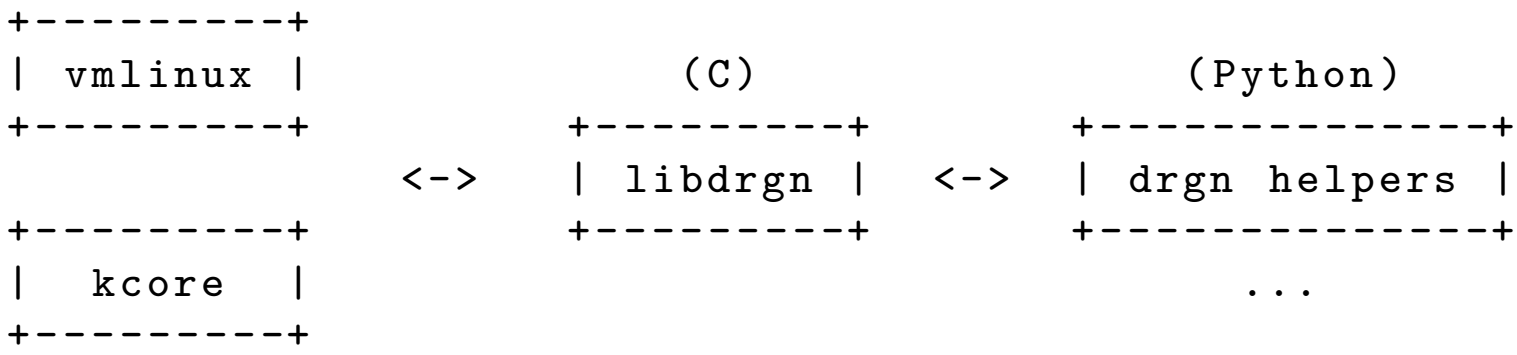
```
(poke) dump
76543210  0011 2233 4455 6677 8899 aabb ccdd eeff  0123456789ABCDEF
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0100 3e00 0100 0000 0000 0000 0000 0000  ..>.............
00000020: 0000 0000 0000 0000 0802 0000 0000 0000  ................
00000030: 0000 0000 4000 0000 0000 4000 0b00 0a00  ....@.....@.....
00000040: 5548 89e5 b800 0000 005d c300 4743 433a  UH.......]..GCC:
00000050: 2028 4465 6269 616e 2036 2e33 2e30 2d31   (Debian 6.3.0-1
00000060: 382b 6465 6239 7531 2920 362e 332e 3020  8+deb9u1) 6.3.0
(poke) load elf
(poke) var ehdr = Elf64_Ehdr @ 0#B
(poke) ehdr.e_ident
struct {
  ei_mag=[0x7fUB,0x45UB,0x4cUB,0x46UB],
  ei_class=0x2UB,
  ei_data=0x1UB,
  ei_version=0x1UB,
  ei_osabi=0x0UB,
  ei_abiversion=0x0UB,
  ei_pad=[0x0UB,0x0UB,0x0UB,0x0UB,0x0UB,...],
  ei_nident=0x0UB#B
}
```

- The extensible editor for structured binary data
- `https://jemarch.net/poke`
- Introductory videos in `https://pokology.org/videos.html`

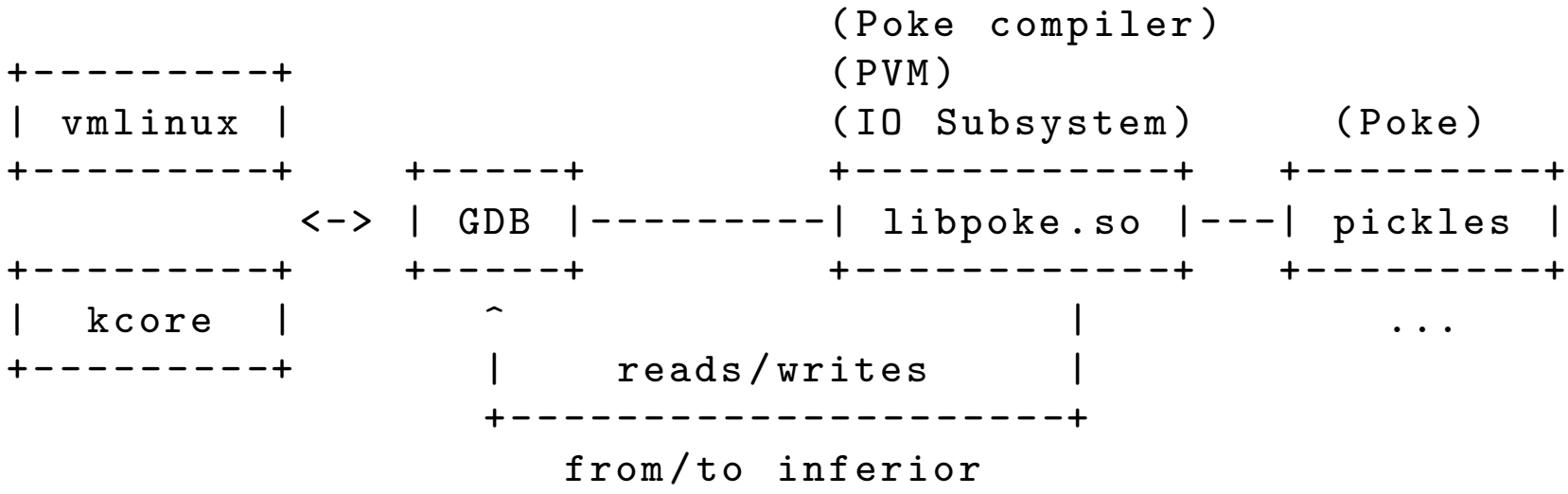
poke and GDB

- poke integrates in GDB via `libpoke.so`
- Allows executing Poke code in GDB to poke at the inferior's memory.
- Provides access to GDB symbols from Poke programs.
- Can translate GDB types (from DWARF, CTF, etc) to Poke types.
- Not merged yet, functional WIP available in branch `users/jemarch/poke-gdb` in `binutils-gdb.git`

drgn



GDB+poke



GDB vs. libdrgn
- Full-fledged debugger
- No reinventing the wheel
  (disasm, stacktraces, ...)

pickles vs. drgn helpers
- Carefully designed DSL
- kernel-specific pickles
- General purpose pickles
  (For payloads or otherwise.)

Need a debugger? Use one!

- `GDB` knows about programs, objects, core dump files, memory segments and regions, stack frames, disassembling, etc etc.
- **No** need to reimplement any of that. Really.
- However, GDB can be made to know the kernel better like libdrgn does:
  - pgtable
  - kallsyms
  - `VMCOREINFO`
  - ...
- ... **this project is a good opportunity to do so!**

Poke is a DSL designed with love

- The Poke language is specificaly designed to poke at structured binary data.
- Also to be used interactively.
- Statically typed.
- Arbitrarily complex and well-defined data structures can be mapped in bit-addressable IO spaces.
- **All** aspects of storage are well defined.

Pickles: kernel-specific

- Roughly equivalent to the drgn helpers.
- Example: `linux-list.pk`

```
type Linux_List =
  struct
  {
    offset<uint<64>,B> next;
    offset<uint<64>,B> prev;

    method is_empty = int<32>:
    {
      ...
    }
    ...
  };
```

Pickles: general purpose

- Many of them distributed with poke
  - Instruction sets: BPF, RISC-V
  - Object file related: ELF, DWARF, CTF, BTF
  - File systems: MBR, JFFS2, USTAR, ZFS
  - ...
- Third party distribution: `https://pokology.org`, `https://github.com`, etc.
- Written and maintained by the community for other purposes.
- Available for the kernel hackers to debug **payloads**

For discussion...

- **helpers/pickles**: What kernel data structures are most important to abstract for debugging purposes?
- **Payloads**: what do you store/process/transmit in your data buffers that you may want to inspect and poke at for debugging purposes?

Can we make a wishlist? Is there interest? What is the right place where to discuss such things?

(I have a few poke stickers to give away)

Bonus slides

Poking at Bytes

- Data stored in modern computers is fundamentally a sequence of entities called “bytes”:

```
+--------+--------+--------+ ... +--------+
| byte 0 | byte 1 | byte 2 |     | byte N |
+--------+--------+--------+ ... +--------+
```

- Each byte stores a little unsigned integer in the range `0..255`
- In Poke parlance, that is an `uint<8>`, aka `byte`
- `byte @ 4#B`
- `uint<8> @ 4#B = 66`

Mapping

The central concept in poke

- Poke variables are in memory.
- The IO space is the data being edited (file, memory, ...)
- Both can be manipulated **in the same way**.
- ... or that's the idea.

Mapping

TYPE @ OFFSET `-> MAPPED_VALUE`

- Simple types

```
(poke) var a = 10
(poke) var b = int @ 0#B
```

- Arrays

```
(poke) var a = [1,2,3]
(poke) var b = int[3] @ 0#B
```

- Structs

```
(poke) var a = Packet { i = 10, j = 20 }
(poke) var b = Packet @ 0#B
```

From Bytes to Integers

- `uint<16> @ 1#B`

```
poke values                    |         uint<16> @ 1#B          |
-----------                    |                                 |
IO space      |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----------   |                |                 |               |
IO device     |      byte0     |      byte1      |      byte2    |
```

- From 1 to 64 bits, signed and unsigned.
- Bit ordering is always big-endian.
- Byte ordering may be either big-endian or little-endian.

Weird Integers - Incomplete Bytes

- `uint<12> @ 0#B`

```
poke values  |        uint<12>       |
-----------  |                       |
IO space     |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----------  |               |               |               |
IO device    |     byte0     |     byte1     |     byte2     |
```

- But what is “the first half of the second byte”?

```
|                  uint<12>                   |
+-------------------------+-------------+-------------+
| a7 a6 a5 a4 a3 a2 a1 a0   b7 b6 b5 b4 :               |
+-------------------------+-------------+-------------+
|           byte 0          |             byte 1            |
```

- In big-endian: `a7 a6 a5 a4 a3 a2 a1 a0 b7 b6 b5 b4`
- In little-endian: `b7 b6 b5 b4 a7 a6 a5 a4 a3 a2 a1 a0`

Weird Integers - Quantum Bytenics

- `uint<5> @ 0#B`

```
poke values  | uint<5> |
-----------  |         |
IO space     |b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|b|
-----------  |               |               |               |
IO device    |     byte0     |     byte1     |     byte2     |
```

- Let's view the byte as a sequence of bits:

```
|     uint<5>      |
+------------------+-----------+
|  b7  b6  b5  b4  b3  |            |
+------------------+-----------+
|              byte0              |
```

- The value is `b7 b6 b5 b4 b3` regardless of byte endianness.

Unaligned Integers

- `uint<16> @ 2#b`

```
poke values       |         uint<16> @ 2#b          |
-----------       |                                 |
IO space      |0|1|1|1|1|1|1|1|0|1|0|0|0|1|0|1|0|1|0|0|1|1|0|0|
-----------   |               |               |               |
IO device     |     0x7f      |      0x45     |      0x4c     |
```

- This helps to visualize it:

```
poke values        |         uint<16> @  2#b         |
-----------        |                                 |
Virtual bytes      | virt. byte1   |  virt. byte2  |
-----------        |               |               |
IO space       |0|1|1|1|1|1|1|1|0|1|0|0|0|1|0|1|0|1|0|0|1|1|0|0|
-----------    |               |               |               |
IO device      |     0x7f      |      0x45     |      0x4c     |
```

Simple Values

- Integers:

```
10, 0xff, 8UB, 0b1100, 0o777
```

- Strings:

```
"foo\nbar"
""
```

- Arrays:

```
[1,2,3]
[[1,2],[3,4]]
[[1,2,3],[4]]
```

- Structs:

```
Person { name = "Donald␣Knuth", age = 84 }
Person {}
```

Bytes or bits? Neither!

- The offset problem.
- bytes? bits? both?
- Solution: **united values**.

Offset values

- Named units:

```
8#b
23#B
2#Kb
```

- Numeric units:

```
8#8
2#3
```

- Even better:

```
type Packet = struct { int i; long j; }
23#Packet
```

- Operations:

```
OFF +- OFF -> OFF
OFF *  INT -> OFF
OFF /  OFF -> INT
OFF /  INT -> OFF
OFF %  OFF -> OFF
```

Offset values

Offsets avoid explicit unit conversions

```
type Elf64_Shdr =
  struct
  {
    ...
    offset<Elf64_Xword,B> sh_size;
    ...
  };

...
shdr.sh_size = 10#Elf64_Rela;
```

Simple Types

- Integral types:

  ```
  int<N>
  uint<N>
  ```

- Offset types:

  ```
  offset<INT_TYPE,UNIT>
  ```

- String type:

  ```
  string
  ```

Array Types

- Unbounded:

```
int[]
int[][]
```

- Bounded by number of elements:

```
int[2]
int[foo+bar]
```

- Bounded by size:

```
int[8#B]
```

Array Constructors

- `TYPE[BOUND](INITVAL)`
- Examples:

```
int<32>[]() -> []
string[3]() -> ["","",""]
string[3]("foo'') -> ["foo","foo","foo"]
```

Structs

- Definition of a struct type

```
type Packet =
  struct
  {
    byte magic;
    uint<32> data_length;
    byte[data_length] data;
  };
```

- Constructing a struct value

```
(poke) Packet {data_length = 2}
Packet {
  magic=0x0UB,
  data_length=0x2U,
  data=[0x0UB,0x0UB]
}
```

- Mapping a struct value

```
(poke) Packet @ 0#B
Packet {
  magic=0x0UB,
  data_length=0x2U,
  data=[0x0UB,0x0UB]
}
```

Pickles

- Useful definitions are stored in **pickles**
- Many of them are included in poke's distribution
- You can write and share your own pickles
- Loading pickles
  - `.load`
  - `load`
- Inspecting pickles
  - `.info types|variables|functions`
  - `.info type TYPENAME`
- `load_path`

Field Labels

```
type Packet =
  struct
  {
    byte magic;
    uint<32> data_length;
    offset<int,B> data_offset;

    byte[data_length] data @ data_offset;
  };
```

Pinned Structs

All fields have an offset of zero bytes from the beginning of the struct.

```
type Elf_Sym_Info =
  pinned struct
  {
    uint32 st_info;
    struct
    {
      uint<28> st_bind;
      uint<4> st_type;
    } st_info_sct;
  };
```

Integral Structs

- Structs that are handled like integers in input/output

```
type Elf_Sym_Info =
  struct uint<8>
  {
    uint<4> st_bind;
    uint<4> st_type;
  };
```

- Can be used as integers explicitly or in the right contexts

```
(poke) Elf_Sym_Info { st_bind = 1 } as int<32>
0x10
(poke) Elf_Sym_Info { st_bind = 1 } + 2
0x12U
```

- Integers can be cast to integral structs

```
(poke) 0x12 as Elf_Sym_Info
Elf_Sym_Info {
  st_bind=0x1UN,
  st_type=0x2UN
}
```

Constraints in Structs

- Explicit:

```
struct
{
  byte[4] ei_mag : ei_mag == [0x7fUB, 'E', 'L', 'F'];
  byte ei_class;
  byte ei_data;
  ...
} e_ident;
```

- Implicit along with an initialization value:

```
struct
{
  byte[4] ei_mag == [0x7fUB, 'E', 'L', 'F'];
  byte ei_class;
  byte ei_data;
  ...
} e_ident;
```

- No constraint, only initialization value:

```
struct
{
  byte[4] ei_mag = [0x7fUB, 'E', 'L', 'F'];
  byte ei_class;
  byte ei_data;
  ...
} e_ident;
```

Optional and absent Fields

- Fields may be optional depending on the result of an expression

```
type Elf64_File =
  struct
  {
    Elf64_Ehdr ehdr;

    Elf64_Shdr[ehdr.e_shnum] shdr @ ehdr.e_shoff
      if ehdr.e_shnum > 0;

    Elf64_Phdr[ehdr.e_phnum] phdr @ ehdr.e_phoff
      if ehdr.e_phnum > 0;
  };
```

- Accessing an absent field results in an `E_elem` exception.

Struct Methods

```
type Elf64_File =
  struct
  {
    ...
    method get_group_signatures = string[]:
      {
        var signatures = string[]();
        for (section in shdr where section.sh_type == SHT_GROUP)
          signatures += [get_group_signature (section)];
        return signatures;
      }

  };
```

Pretty Printers

Implemented as methods with a special name: `_print`.

```
type BER_Data_Value =
  struct
  {
    ...
    method _print = void:
    {
      print "#<";
      if (length.is_indefinite)
        print "length=indefinite";
      else
        printf "length=%v", length.get;
      print ">";
    }
  };
```

Declarations inside Struct Types

```
type Packet =
  struct
  {
    byte magic = 0xab;
    byte size;

    var real_size = (size == 0xff ? 0 : size);

    byte[real_size] payload;
    byte[real_size] control;

    fun corrected_crc = int:
    {
      try return calculate_crc (payload, control);
      catch if E_div_by_zero { return 0; }
    }

    int crc = corrected_crc;
  };
```

Functions are **NOT** methods!

Unions

```
type Id3v2_Frame =
  struct
  {
    char id[4] : id[0] != 0;
    uint32 size;
    ...
    union
     {
       /* Frame contains text related data.  */
       union
       {
         struct
         {
           char id_asciiz_str = 0;
           char[size - 1] frame_data;
         } : size > 1;

         char[size] frame_data;
       } : id[0] == 'T';

       /* Frame contains other data.  */
       char[size] frame_data;
     };
  };
```

Polymorphism in Poke

- Static typing is important for Poke.
- But we support a poor man's type polymorphism:
  - **any**, **any[]**
  - everything coerces to any.
  - any coerces to nothing.
- The plan is to eventually transition into **gradual typing**, in a backwards-compatible way.

Functions

- Function definition

```
fun ctf_section = (Elf64_Ehdr ehdr) Elf64_Shdr:
{
 for (s in Elf64_Shdr[ehdr.e_shnum] @ ehdr.e_shoff)
   if (elf_string (ehdr, s.sh_name) == ".ctf")
     return s;

 raise E_generic;
}
```

- Lambdas

```
setter = lambda (any val) int:
  {
    var s = val as string;

    if (s != "info" && s != "less")
      return 0;

    pk_doc_viewer = s;
    return 1;
  }
```

Optional Function Arguments

```
fun elf_string = (Elf64_Ehdr ehdr, offset<Elf_Word,B> offset,
                  Elf_Half strtab = ehdr.e_shstrndx) string:
{
 var shdr = Elf64_Shdr[ehdr.e_shnum] @ ehdr.e_shoff;
 return string @ (shdr[strtab].sh_offset + offset);
}
```

Variable Number of Arguments

Last argument is an array of **any**s.

```
fun format = (string fmt, args...) string:
  {
    ...
    if (fmt[fi + 1] == 'x')
       res = res + tohex (args[narg] as uint<64>);
    ...
  }
```

Parameterless Functions

- Algol68ism: parameterless functions are homoiconic to variables

```
(poke) fun beast = int: { return 666; }
(poke) beast() + 1
667
(poke) beast + 1
667
```

- Use parentheses to refer to the function value itself

```
(poke) (beast)
#<closure>
```

Terminal Hyperlinks

- Terminals support **implicit** links by recognizing URLs and DTRT.
- Egmont Koblinger proposed using **explicit** links instead , printed by applications using sequences of escape character.

  `https://gist.github.com/egmontkob/eb114294efbcd5adb1944c9f3cb5feda`
- Emitting hyperlinks:

```
# $1 = href, $2 = text
emit_hyperlink ()
{
  printf '\e]8;;'"$1"'\e\\'"$2"'\e]8;;\e\\'
}
```

The app:// protocol

- Originally designed by the Rabbit Herd
  `http://www.jemarch.net/rhhw.html`
- Evolved by Bruno Haible into two proposals: **appstart** and **appsocket**.
  `https://lists.freedesktop.org/archives/xdg/2020-January/014214.html`
  `https://lists.freedesktop.org/archives/xdg/2020-January/014215.html`
- `app-client.c`

The app:// protocol

GNU poke hyperlinks

- We use terminal hyperlinks to improve the CLI experience, but they are also available to Poke programs and scripts.
- GNU poke supports three kinds of hyperlinks:
  - **insert hyperlinks**
    Some given text is inserted at the current pointer in the prompt.
  - **execute hyperlinks**
    Some given text is executed as a command, like if it was typed at the prompt.
  - **closure hyperlinks**
    Like **execute**, but execute some given Poke function (closure) instead of a command.
- The kind and its arguments are **not** explicitly included in the payload, which consists on an opaque token:

  `app://malditobastardo:39957/12345`

Write your Own Commands

- What to do: DSL vs. command language.
- We avoided the need for a command language using a syntax trick:

```
fun foo = (int a, int b = 30, int c) void: { ... }
...
foo (10, 20, 40);
...
foo :c 10 :a 20
...
```

- All poke commands are implemented like this, and you can hook in the commands infrastructure (help system, settings, etc.)

Write your Own Binary Utilities

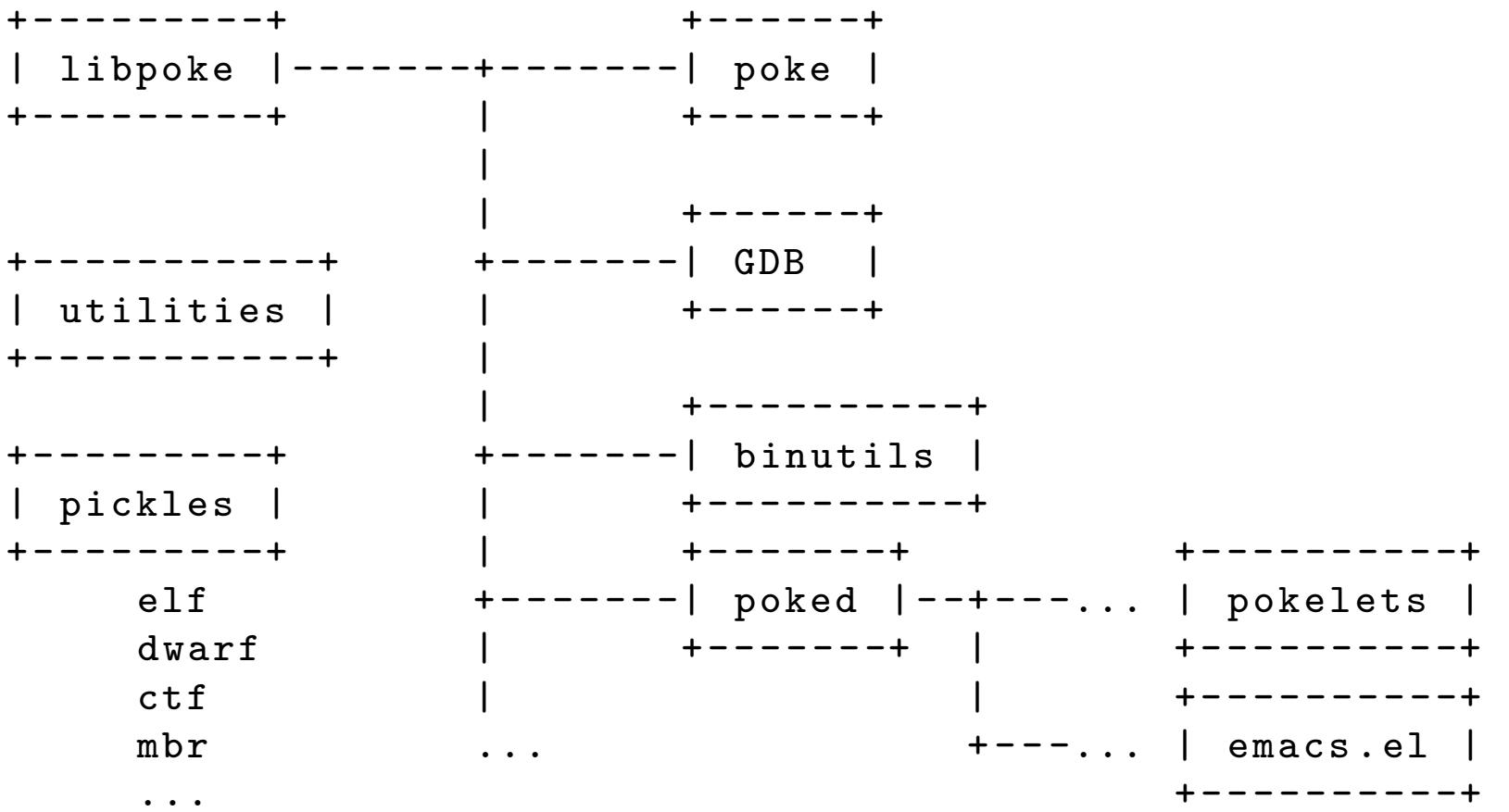
- Use the shebang:

```
#!/usr/bin/poke -L
!#
```

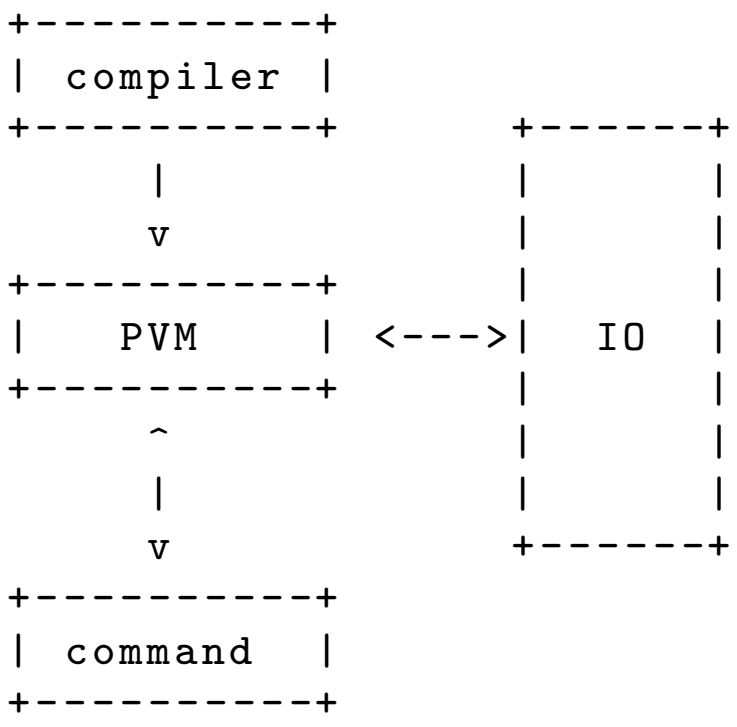
- Support for parsing command-line arguments
- Support for writing filters

The program

The pokesphere



Architecture



The project

poke is a GNU program

- Developed by the GNU Project
- Part of the GNU system
- Licensed under GPLv3+, both applications and libraries

Project Resources

- Homepage: **https://www.jemarch.net/poke**
- Savannah: **https://savannah.gnu.org/p/poke**
- Mailing list: **poke-devel@gnu.org**
- IRC channel: **#poke** in **irc.libera.chat**
- Pokology: **https://pokology.org**

Current Work

- Current released poke version is 2.1
- poke 2.x is in maintenance mode
- poke 3.x is under development in the `master` branch.
- Other poke-related projects and applications are appearing.