

# Command Line and Scripting Essentials for Heavy Vehicle Security

Phil Lapczynski

Version 2025-05-15





# About Me

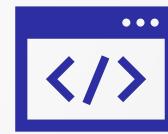
Phil Lapczynski (lap chin ski)

- Principal engineer at Renesas Electronics America
- ~20 years in transportation security
- Teaching Experience:
  - Taught graduate embedded security course at UD Mercy
  - Auto-ISAC ACT Program instructor for OTA security
- 8x CyberTruck Challenge Mentor
- SAE, Auto-ISAC, Global Platform Contributor
- LinkedIn - <https://www.linkedin.com/in/philip-lapczynski/>

# Command Line Interfaces



**Who has  
experience..**



...using the  
command  
line in  
Windows?



...using the  
command  
line in Linux?



...with CAN  
and J1939?

Processes: 880 total, 4 running, 11 stuck, 865 sleeping,  
Load Avg: 3.31, 2.75, 2.40 CPU usage: 7.0% user, 6.34%  
SharedLibs: 1067M resident, 155M data, 56M linkedit.  
MemRegions: 534128 total, 9757M resident, 476M private, 61M  
PhysMem: 31G used (2991M wired, 7207M compressor), 201M unused.  
VM: 371T vsize, 4915M framework vsize, 36(0) swapins, 156(0) swapouts  
Networks: packets: 1513801004/1764G in, 934992249/564G out.  
Disks: 297082080/4063G read, 294118055/2685G written.

PID	COMMAND	%CPU	TIME	#TH	#WQ	#PORT	MEM	PURG	CMPRS	PGRP
60820	Logic Pro	19.6	08:11:17	33	2	1601	1292M	0B	949M	60820
0	kernel_task	16.3	65:35:00	610/10	0	0	18M+	0B	0B	0
400	WindowServer	12.0	38:59:49	21	5	5364	3742M-	100M+	1728M	400
61207	top	6.7	00:03.10	1/1	0	42-	16M+	0B	0B	61207 9027
361	mds	5.8	15:25:25	11	6	5123-	210M-	0B	165M	361
61345	mdworker_sha	4.4	00:00.10	5	2	47+	2785K-	0B	0B	61345
463	coreaudiod	4.1	18:18:41	11	3	4571	55M	0B	28M	463
59132	diskimagergio	2.2	02:20.90	15	14	83	36M-	0B	400K	59132
2240	MSTeams	2.0	11:32:55	39	5	2388	406M	96K	348M	2240
1	launchd	1.7	03:50:21	5	3	4593-	27M	0B	5248K	1 0
61344	mdworker_sha	1.5	00:00.09	5	2	48	3202K	0B	0B	61344
394	notifyd	1.1	40:36.49	2	1	1185-	4481K	0B	976K	394
57666	Signal Help	1.0	05:31.53	39	1	874+	255M+	0B	45M	57657 57657
64236	OneDrive	1.0	08:58.69	16	2	287	214M	16K	173M	64236
42201	Microsoft Ou	1.0	35:59.09	24	4	1029	573M	512K	356M	42201
372	opendirector	1.0	95:03.52	8/1	7/1	2377-	15M	80K	4368K	372
599	mds_stores	0.9	04:36:51	6	4	209	69M	48K	21M	599
57663	Signal Help	0.8	00:49.01	19	3	194+	85M+	96K-	13M	57657 57657
332	logd	0.7	05:35:18	5	4	2867-	17M	0B	18M	332
59533	mdsync	0.6	00:04.05	3	1	58	4993K+	0B	912K	59533
39573	Private Inte	0.6	03:12:03	8	1	279	105M	16K	43M	39573
9025	Terminal	0.5	00:15.35	10	3	509	120M+	31M	20M	9025
403	cfprefsd	0.4	24:15.24	4/1	3	1299-	5233K	96K	784K	403
2392	Google Chrom	0.3	69:42.08	18	1	167	64M	0B	25M	2392
2384	Google Chrom	0.3	05:30:34	49	2	5527	389M	32K	112M	2384
364	diskarbitrat	0.3	07:29.90	2	1	1282-	4353K	0B	912K	364
530	FocusriteCon	0.2	53:16.78	10	2	108	4993K	0B	4160K	530
341	systemstats	0.2	05:04.56	3	2	104	6721K	0B	2112K	341
452	lsd	0.2	16:56.99	4	3	202-	8258K	0B	1248K	452
346	powerd	0.2	35:22.27	3	2	151	6545K	0B	1648K	346
57466	com.apple.We	0.2	00:12.92	9	1	306	26M	28M	4256K	57466
880	com.apple.Am	0.1	22:18.44	3	1	79	4946K	0B	2832K	880
57243	Google Chrom	0.1	00:30.57	22	1	215	141M	0B	16M	2384
86069	NIHostIntegr	0.1	04:25.17	16	1	314	39M	0B	37M	86069
536	CodeMeterMac	0.1	64:20.59	11	1	640	37M	0B	6256K	536
396	corebrightne	0.1	19:29.99	4	3	137	4529K	0B	22M	396
58360	com.apple.We	0.1	00:07.01	5	2	85	70M	0B	40M	58360
61001	com.apple.We	0.1	00:02.11	4	2	81	98M	16K	16M	61001
57784	com.apple.We	0.1	00:07.14	4	1	85	71M	0B	37M	57784
671	fileprovider	0.0	07:54:20	7	6	241	90M	0B	22M	671
550	audioclocksy	0.0	22:12.46	3	2	48	521K	0B	16M	550
61290	screenCaptur	0.0	00:00.35	2	1	64	8002K	0B	16M	61290
57657	Signal	0.0	01:04.73	48	1	479	201M	0B	16M	57657

# What Is the Command Line Interface (CLI)?

- The CLI is a **text-based interface** used to interact with a computer's operating system.
- Unlike graphical user interfaces (GUIs), the CLI uses **typed commands** to perform tasks.
- It's lightweight, powerful, and often the **preferred tool in professional diagnostics and scripting**.
- Common CLIs:
  - Windows Command Prompt / PowerShell**
  - Linux Terminal / Bash / ZSH / etc**
- Widely used in **vehicle diagnostics, firmware interaction, and cybersecurity tools**.

```
user@hostname:~$ pwd  
/home/user
```

```
user@hostname:~$ ls
```

# Basic Command-Line Desktop Operations

Documents

Downloads

```
user@hostname:~$ cd
```

# Today's Objectives

<b>Command-line Basics</b>	<b>Scripting Introduction</b>	<b>Python and CAN Bus</b>	<b>AI Tools for Scripting</b>	<b>Advanced CLI Tools</b>
Overview of essential command-line commands and operations for both Windows and Linux systems.	Introduction to scripting for diagnostics and automation tasks, enhancing efficiency and troubleshooting capabilities.	Using Python scripts to interact with CAN bus systems for data communication and vehicle diagnostics.	Leveraging AI tools like ChatGPT to assist in writing and debugging scripts, optimizing development processes.	Overview of advanced command-line tools such as cantools, python-can, TruckDevil, CaringCaribu, and Scapy for enhanced diagnostics and automation.

# **Let's Boot Into Linux!**

- For a majority of this course we will be in Linux
- We will touch on both Linux and Windows, however it will be fine to stay booted in Linux the whole time

# Navigating the File System

## Key Commands:

- **cd** – Change directory (no argument will show current)
- **ls** (Linux) / **dir** (Windows) – List contents
- **pwd** (Linux) – Show current directory

## Quick Examples:

- **cd** /home/user/logs (Linux)
- **cd** C:\Logs\2024 (Windows)

# Basic File Operations

- **Common Commands:**

- **mkdir** – Make new folder
- **rm / del** – Remove files
- **copy** (Windows) / **cp** (Linux) – Copy files
- **move** (Windows) / **mv** (Linux) – Move or rename files

- **Examples:**

- **mkdir** diagnostics
- **cp can\_dump.log backup/** (Linux)
- **copy can\_dump.log backup\** (Windows)

# Windows CMD Resources

- <https://github.com/security-cheatsheet/cmd-command-cheat-sheet>

> help

```
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Madhushala>help
For more information on a specific command, type HELP command-name
ASSOC           Displays or modifies file extension associations.
ATTRIB          Displays or changes file attributes.
BREAK           Sets or clears extended CTRL+C checking.
BCDEDIT         Sets properties in boot database to control boot loading.
CACLS           Displays or modifies access control lists (ACLs) of files.
CALL            Calls one batch program from another.
CD               Displays the name of or changes the current directory.
CHCP             Displays or sets the active code page number.
CHDIR           Displays the name of or changes the current directory.
CHKDSK          Checks a disk and displays a status report.
CHKNTFS         Displays or modifies the checking of disk at boot time.
CLS              Clears the screen.
CMD              Starts a new instance of the Windows command interpreter.
```



# Linux Command Line Resources

- Lots of resources out there for help
  - **man <command>** - Show the users manual

```
MAN(1)                                General Commands Manual                               MAN(1)

NAME
    man, apropos, whatis – display online manual documentation pages

SYNOPSIS
    man [-adho] [-t | -w] [-M manpath] [-P pager] [-S mansect] [-m arch[:machine]] [-p [eprtv]]
        [mansect] page ...
    man -f [-d] [-M manpath] [-P pager] [-S mansect] keyword ...
    whatis [-d] [-s mansect] keyword ...
    man -k [-d] [-M manpath] [-P pager] [-S mansect] keyword ...
    apropos [-d] [-s mansect] keyword ...

DESCRIPTION
    The man utility finds and displays online manual documentation pages. If mansect is
    provided, man restricts the search to the specific section of the manual.

    The sections of the manual are:
    1. General Commands Manual
    2. System Calls Manual
    3. Library Functions Manual
    4. Kernel Interfaces Manual
    5. File Formats Manual
    6. Games Manual
    7. Miscellaneous Information Manual
    8. System Manager's Manual
    9. Kernel Developer's Manual
```



# Linux Command Line Resources (cont.)

- Lots of resources out there for help
  - **tldr** – an attempt to make reading the manuals even easier with examples.

```
➜ ~ tldr man

man
Format and display manual pages.
More information: <https://manned.org/man>.

- Display the man page for a command:
  man command

- Open the man page for a command in a browser (`BROWSER` environment variable can replace `=browser_name`):
  man [-browser_name] [-html=browser_name] command

- Display the man page for a command from section 7:
  man 7 command

- List all available sections for a command:
  man [-fl--whatis] command

- Display the path searched for manpages:
  man [-wl--path]

- Display the location of a manpage rather than the manpage itself:
  man [-wl--where] command

- Display the man page using a specific locale:
  man [-ll--locale] locale command

- Search for manpages containing a search string:
  man [-kl--apropos] "search_string"
```

```
➜ ~ tldr pip

pip
Python package manager.
Some subcommands such as `install` have their own usage documentation.
More information: <https://pip.pypa.io>.

- Install a package (see `pip install` for more install examples):
  pip install package

- Install a package to the user's directory instead of the system-wide default location:
  pip install --user package

- Upgrade a package:
  pip install [-Ul--upgrade] package

- Uninstall a package:
  pip uninstall package

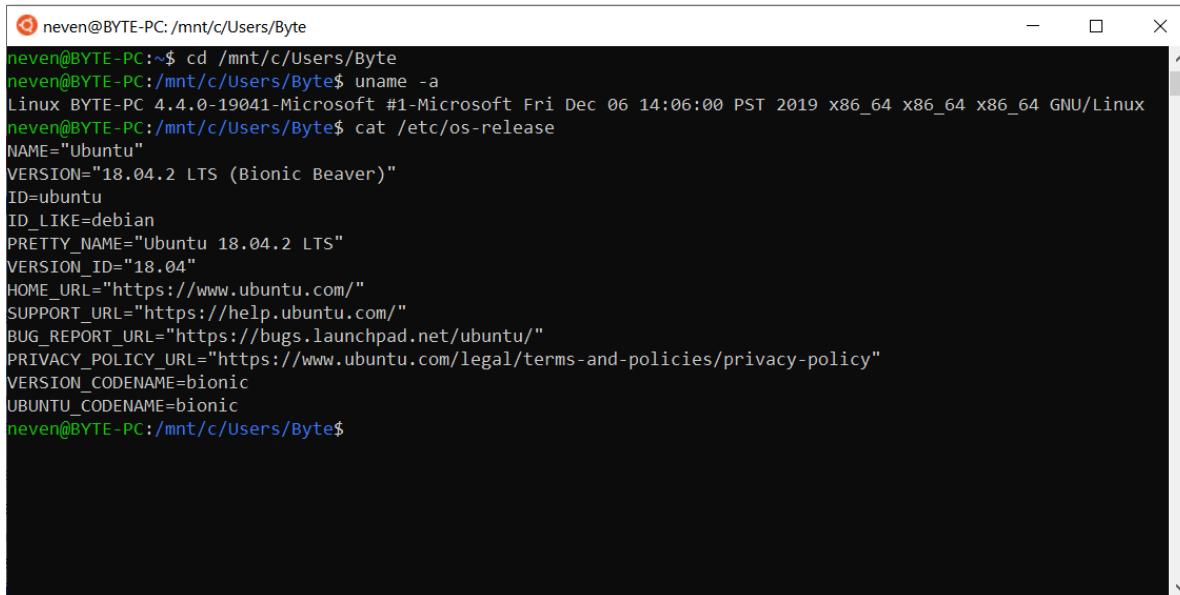
- Save installed packages to file:
  pip freeze > requirements.txt

- Show installed package info:
  pip show package

- Install packages from a file:
  pip install [-rl--requirement] requirements.txt
```

# Windows Subsystem for Linux

- Windows has a subsystem that allows for use of the GNU/Linux environment within Windows without a VM
- **NOTE** – YMMV when trying to use things like socketCAN in WSL
- Better to work directly in Linux if you want to use Linux cantools

A screenshot of a Windows terminal window titled "neven@BYTE-PC". The window contains a black terminal session with white text. The user runs several commands: "cd /mnt/c/Users/Byte", "uname -a", "cat /etc/os-release", and "ls" (which is partially visible at the bottom). The output shows the system is a Microsoft #1-Microsoft build of Linux, running on an x86\_64 architecture, and is identified as Ubuntu 18.04.2 LTS (Bionic Beaver).

```
neven@BYTE-PC:~$ cd /mnt/c/Users/Byte
neven@BYTE-PC:/mnt/c/Users/Byte$ uname -a
Linux BYTE-PC 4.4.0-19041-Microsoft #1-Microsoft Fri Dec 06 14:06:00 PST 2019 x86_64 x86_64 x86_64 GNU/Linux
neven@BYTE-PC:/mnt/c/Users/Byte$ cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.2 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.2 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
neven@BYTE-PC:/mnt/c/Users/Byte$ ls
```

# Working with CAN in Linux

# What is SocketCAN?

- **SocketCAN** is the Linux-native framework for Controller Area Network (CAN) support
- Treats CAN interfaces like network interfaces (e.g., Ethernet) — makes CAN sockets accessible via standard Linux networking APIs
- Developed by Volkswagen Research and integrated into the Linux kernel
- Supports a variety of CAN hardware adapters: USB, PCIe, and virtual interfaces
- Powers tools like **can-utils**, **python-can**, and custom scripts for diagnostics and security

# Exercise 1: Enabling a Virtual CAN Interface

```
sudo modprobe can  
sudo modprobe vcan  
sudo ip link add dev vcan0 type vcan  
sudo ip link set up vcan0  
~
```

Since we will be using this many times throughout CyberTruck, is it best to add these commands to a script.

**Example:**

```
$ ./bringup_vcan.sh
```

# Exercise 2: Enabling a HW CAN Interface

**STEP 1:** Connect your interface and make sure it is connected

- You can use the **lsusb** command to verify that it is connected

**STEP 2:** Configure interface and bring it up

```
sudo ip link set can0 down
sudo ip link set can0 up type can bitrate 50000
sudo ip link set can1 down
sudo ip link set can1 up type can bitrate 50000
```

# Using can-utils – the CANbus hello world

To verify your interface and CAN is set up properly, it is wise to verify functionality using the Linux can-tools. Using two separate terminals...

## TERMINAL A

```
$ candump can0
```

## TERMINAL B

```
$ cangen can0
```

# You should see output!

can0	43F	[8]	D6	BA	05	3B	50	C1	47	7C
can0	209	[8]	72	24	42	58	2A	4A	78	71
can0	605	[8]	4E	8B	FD	53	4C	6D	7E	1E
can0	16C	[8]	80	D4	CD	19	D3	46	6C	7D
can0	0D7	[3]	24	E0	E7					
can0	682	[8]	E6	CF	08	4F	8E	AB	49	2D
can0	409	[4]	E2	B2	0E	77				
can0	7A7	[8]	9B	90	8D	76	E7	5B	EB	14
can0	245	[1]	37							
can0	5EC	[8]	78	66	03	0F	F1	8B	09	54
can0	555	[7]	3E	F9	87	72	C1	F6	53	
can0	531	[8]	94	3D	C0	5B	08	B6	70	5B
can0	091	[8]	F8	8E	43	5F	13	CF	EB	64
can0	277	[8]	A2	7A	35	12	81	56	12	1D
can0	752	[4]	3E	DD	F5	33				
can0	4EF	[8]	E0	82	22	17	34	27	65	4C
can0	14A	[7]	82	43	F0	69	36	17	F6	
can0	1C0	[8]	28	A3	FF	17	16	57	F4	69

# Useful Commands – Send and Receive

## Reading from the can bus

```
$ candump any
```

## Transmitting CAN frames

```
$ cansend can0 123#DEADBEEF
```

- Explanation:
  - Sends frame with **ID 0x123** and data **DE AD BE EF**

## Transmitting 29-bit CAN frames (as used by J1939)

```
$ cansend can0 1823ff40#0123
```

# Useful Commands – Logging and Replay

**Logging traffic**

```
$ candump -l can0
```

**Replay a log file**

```
$ canplayer -I candump_*.log
```

# Getting Saucy With It

**Split log files and do a  
binary search for desired  
replay behavior**

Keep splitting and replaying  
until you have found the  
exact message

```
phil@ubuntu:~/Desktop/Projects/canplayground$ ./bifurcator.sh candump-2025-05-24_023038.log
Current segment: bifurcate_current.log (53 lines)
Replaying first half (A)...
Did the behavior occur in A? (y/n): □
```

```
phil@ubuntu:~/Desktop/Projects/canplayground
can0 0CD [5] 62 A1 D7 08 65
can1 0CD [5] 62 A1 D7 08 65
can0 452 [6] D4 80 F4 31 B6 01
can1 452 [6] D4 80 F4 31 B6 01
can0 3F0 [4] 82 0B 9D 3F
can1 3F0 [4] 82 0B 9D 3F
can0 3C8 [8] A6 7C A2 3F B6 F2 B6 7C
can1 3C8 [8] A6 7C A2 3F B6 F2 B6 7C
can0 360 [7] F9 76 2A 78 09 E0 1D
can1 360 [7] F9 76 2A 78 09 E0 1D
can0 5BC [8] 03 B7 8F 44 AE 30 61 79
can1 5BC [8] 03 B7 8F 44 AE 30 61 79
can0 1E1 [8] BD B9 C2 56 34 94 7F 60
can1 1E1 [8] BD B9 C2 56 34 94 7F 60
can0 056 [7] 70 9F 8F 30 23 C1 CE
can1 056 [7] 70 9F 8F 30 23 C1 CE
can0 4FC [2] 88 E8
can1 4FC [2] 88 E8
can0 198 [8] 05 8B 0F 75 88 1D F4 55
can1 198 [8] 05 8B 0F 75 88 1D F4 55
can0 730 [7] 44 73 A2 77 F8 0A 72
can1 730 [7] 44 73 A2 77 F8 0A 72
can0 7A0 [8] AE FD 28 42 00 43 F4 57
can1 7A0 [8] AE FD 28 42 00 43 F4 57
```

# Diving Deeper into can-utils

Linux can-utils are a suite of utilities and tools for SocketCAN which include...

## Basic tools to display, record, generate and replay CAN traffic

- candump : display, filter and log CAN data to files
- canplayer : replay CAN logfiles
- cansend : send a single frame
- cangen : generate (random) CAN traffic
- cansequence : send and check sequence of CAN frames with incrementing payload
- cansniffer : display CAN data content differences

## CAN access via IP sockets

- canlogserver : log CAN frames and serves them
- bcmserver : interactive BCM configuration (remote/local)
- [socketcan](#) : use RAW/BCM/ISO-TP sockets via TCP/IP sockets
- [cannelloni](#) : UDP/SCTP based SocketCAN tunnel

## ISO-TP tools [ISO15765-2:2016 for Linux](#)

- isotpsend : send a single ISO-TP PDU
- isotprecv : receive ISO-TP PDU(s)
- isotpsniffer : 'writetap' ISO-TP PDU(s)
- isotpdump : 'writetap' and interpret CAN messages (CAN\_RAW)
- isotpserver : IP server for simple TCP/IP <-> ISO 15765-2 bridging (ASCII HEX)
- isotpperf : ISO15765-2 protocol performance visualisation
- isotptun : create a bi-directional IP tunnel on CAN via ISO-TP

## J1939/ISOBus tools

- j1939acd : address claim daemon
- j1939cat : take a file and send and receive it over CAN
- j1939spy : spy on J1939 messages using SOC\_J1939
- j1939sr : send/recv from stdin or to stdout
- testj1939 : send/receive test packet

# Can-j1939 Kickstarter

<https://github.com/linux-can/can-utils/blob/master/can-j1939-kickstart.md>

Can-utils have a dedicated  
J1939 module

**First steps with j1939**

Use [testj1939](#)

When *can-j1939* is compiled as module, opening a socket will load it, or you can load it manually

```
modprobe can-j1939
```

Most of the subsequent examples will use 2 sockets programs (in 2 terminals). One will use CAN\_J1939 sockets using *testj1939*, and the other will use CAN\_RAW sockets using *cansend+candump*.

*testj1939* can be told to print the used API calls by adding *-v* program argument.

**receive without source address**

Do in terminal 1

```
testj1939 -B -r can0
```

Send raw CAN in terminal 2

```
cansend can0 1823ff40#0123
```

You should have this output in terminal 1

```
40 02300: 01 23
```

This means, from NAME 0, SA 40, PGN 02300 was received, with 2 databytes, 01 & 23.

now emit this CAN message:

```
cansend can0 18234140#0123
```



# Working with python-can

# Working with python-can

**Install using:**

```
$ pip install python-can
```

**Documentation:**

<https://python-can.readthedocs.io/en/stable/index.html>

**Why choose this tool?**

- Automation and scripting power
- Integration with larger python workflows and tools (cantools, pandas, scapy)
- Programmatic logic and conditional handling
  - Python gives you full control over logic
  - Wait for specific responses
  - Branch based on message contents
  - Retry, delay, or log conditionally

# Receiving CAN with python-can

- Define a bus, message, and use the send() API

```
import can

bus = can.interface.Bus(channel='vcan0', bustype='socketcan')
print("Listening for messages...")
for msg in bus:
    print(msg)
```

# Sending CAN with python-can

- Define a bus, message, and use the send() API

```
import can

bus = can.interface.Bus(channel='vcan0', bustype='socketcan')
msg = can.Message(arbitration_id=0x123, data=[0x11, 0x22, 0x33], is_extended_id=False)

try:
    bus.send(msg)
    print("Message sent")
except can.CanError:
    print("Message NOT sent")
```

# Logging CAN with python-can

- python-can provides a built-in Logger to save CAN messages
- Works with real and virtual interfaces (e.g., can0, vcan0)
- Saves data in a format compatible with can-utils (candump style)
- Ideal for capturing traffic, analyzing diagnostics, and reproducibility

```
import can

bus = can.interface.Bus(channel='vcan0', bustype='socketcan')
logger = can.Logger("logfile.log")

for i in range(10):
    msg = bus.recv(timeout=1.0)
    if msg:
        logger(msg)
        print(f"Logged: {msg}")
```



Tools That Use Python-Can

# SAE J1939 for Python

**Install using:**

```
$ pip install can-j1939
```

**Examples:**

<https://github.com/juergenH87/python-can-j1939/tree/master/examples>

**Why choose this tool?**

- Support for J1939-22 (J1939 on CANFD)
- ECU (CA) Naming according SAE J1939/81
- Full featured address claiming procedure according SAE J1939/81
- Full support of transport protocol (up to 1785 bytes) according SAE J1939/21
- Full support of fd-transport protocol according SAE J1939/22 (J1939-FD)

# TruckDevil

**Install using:**

```
$ git clone https://github.com/LittleBlondeDevil/TruckDevil.git
```

**Examples:**

<https://github.com/LittleBlondeDevil/TruckDevil/tree/master/truckdevil/modules>

**Why choose this tool?**

- Easy to learn module based
- Support for M2 device or any python-can supported device
- Message parsing and display of standard J1939 messages
- Module for UDS, ECU discovery, fuzzing, dumping, and sending J1939 messages
- Easy to add your own modules

# Pretty-J1939

**Install using:**

```
$ pip install pretty_j1939
```

**Examples:**

[https://github.com/nmfta-repo/pretty\\_j1939](https://github.com/nmfta-repo/pretty_j1939)

**Why choose this tool?**

- Pretty-print J1939 traffic captured in candump logs
- Convert a J1939 Digital Annex (Excel) file into a JSON structure

# Pretty-J1939

```
usage: pretty_j1939.py [-h] [--da-json [DA_JSON]] [--candata] [--no-candata] [--pgn] [--no-pgn] [--link] [--no-link] [--include-na] [--no-include-na] [--real-time] [--no-real-time] [--format] [--no-format] candump

pretty-printing J1939 candump logs

positional arguments:
  candump            candump log, use - for stdin

optional arguments:
  -h, --help          show this help message and exit
  --da-json [DA_JSON] absolute path to the input JSON DA (default=".J1939db.json")
  --candata          print input can data
  --no-candata       (default)
  --pgn              (default) print source/destination/type description
  --no-pgn           (default)
  --spn              (default) print signals description
  --no-spn           (default)
  --transport        print details of transport-layer streams found (default)
  --no-transport     (default)
  --link              print details of link-layer frames found
  --no-link           (default)
  --include-na        include not-available (0xff) SPN values
  --no-include-na     (default)
  --real-time         emit SPNs as they are seen in transport sessions
  --no-real-time      (default)
  --format            format each structure (otherwise single-line)
  --no-format          (default)
```

## Some examples of pretty printing

Formatted content (one per line) next to candump data:

```
$ pretty_j1939.py --candata --format example.candump.txt | head
(1543509533.000838) can0 10FDA300#FFFF07FFFFFF ; {
;   "DA": "All(255)",
;   "PGN": "EEC6(64931)",
;   "SA": "Engine #1( 0)",
;   "Name": "Engine Variable Geometry Turbocharge"
}
(1543509533.000915) can0 18FEE000#FFFFFFFFFFB05C6800 ; {
;   "DA": "All(255)",
;   "PGN": "VD(65248)",
;   "SA": "Engine #1( 0)",
```

Single-line contents next to candump data:

```
$ pretty_j1939.py --candata example.candump.txt | head
(1543509533.000838) can0 10FDA300#FFFF07FFFFFF ; {"SA":"Engine #1( 0)","DA":"All(255)","P
(1543509533.000915) can0 18FEE000#FFFFFFFFFFB05C6800 ; {"SA":"Engine #1( 0)","DA":"All(255)","P
(1543509533.000991) can0 08FE6E0B#0000000000000000 ; {"SA":"Brakes - System Controller( 11)","D
(1543509533.001070) can0 18FD8255#FFFFFF0100FFFF ; {"SA":"Diesel Particulate Filter Control
(1543509533.001145) can0 0CF00400#207D87481400F087 ; {"SA":"Engine #1( 0)","DA":"All(255)","P
(1543509533.001220) can0 18FF4500#6D00FA00FF00006A ; {"SA":"Engine #1( 0)","DA":"All(255)","P
(1543509533.001297) can0 18FEDF00#82FFFFFF7DE70300 ; {"SA":"Engine #1( 0)","DA":"All(255)","P
(1543509533.001372) can0 1CFE9200#FFFFFF00000000 ; {"SA":"Engine #1( 0)","DA":"All(255)","P
(1543509533.001447) can0 18F00131#FFFFFF3F00FFFFFF ; {"SA":"Cab Controller - Primary( 49)","DA
(1543509533.001528) can0 18FEF131#F7FFFF07CCFFFFFF ; {"SA":"Cab Controller - Primary( 49)","DA
```

# Caring Caribou

**Install using:**

```
$ git clone https://github.com/CaringCaribou/caringcaribou.git
$ cd caringcaribou && python setup.py install
```

**Examples:**

<https://github.com/CaringCaribou/caringcaribou/tree/master/caringcaribou/modules>

**Why choose this tool?**

- Dedicated modules for CAN message reverse engineering
- Powerful module for UDS RE such as attacking security seeds, DID enumeration, ECU discovery
- Support for DoIP and XCP reverse engineering

# Caring Caribou

```
(venv) phil@ubuntu:~/Desktop/Projects/canplayground/caringcaribou$ caringcaribou --help
usage: caringcaribou [-h] [-i INTERFACE] module ...

-----
CARING CARIBOU v0.7 - python 3.10.12 (main, Feb 4 2025, 14:57:36) [GCC 11.4.0]
\_\_/_/_
 \_/\_/
 (oo)\_____)\/
  (_)\-----|||
   ||----|||
   ||----|||
-----

A friendly car security exploration tool

positional arguments:
  module      Name of the module to run
  ...
  Arguments to module

options:
  -h, --help    show this help message and exit
  -i INTERFACE  force interface, e.g. 'can1' or 'vcan0'

available modules:
  dcm, doip, dump, fuzzer, listener, module_template, send, test, uds, uds_fuzz, xcp
```

```
(venv) phil@ubuntu:~/Desktop/Projects/canplayground/caringcaribou$ caringcaribou -i vcan0 uds -h
-----
CARING CARIBOU v0.7 - python 3.10.12 (main, Feb 4 2025, 14:57:36) [GCC 11.4.0]
-----

Loading module 'uds'

usage: caringcaribou uds [-h]
                  {discovery,services,subservices,ecu_reset,testerpresent,security_seed,dump_dids,read_mem,auto}
                  ...
                  ...

Universal Diagnostic Services module for CaringCaribou

positional arguments:
  {discovery,services,subservices,ecu_reset,testerpresent,security_seed,dump_dids,read_mem,auto}

options:
  -h, --help            show this help message and exit

Example usage:
  caringcaribou uds discovery
  caringcaribou uds discovery -blacklist 0x123 0x456
  caringcaribou uds discovery -autoblacklist 10
  caringcaribou uds services 0x733 0x633
  caringcaribou uds ecu_reset 1 0x733 0x633
  caringcaribou uds testerpresent 0x733
  caringcaribou uds security_seed 0x3 0x1 0x733 0x633 -r 1 -d 0.5
  caringcaribou uds dump_dids 0x733 0x633
  caringcaribou uds dump_dids 0x733 0x633 --min_did 0x6300 --max_did 0xffff -t 0.1
  caringcaribou uds read_mem 0x733 0x633 --start_addr 0x0200 --mem_length 0x10000

(venv) phil@ubuntu:~/Desktop/Projects/canplayground/caringcaribou$ █
```

<https://github.com/CaringCaribou/caringcaribou/tree/master> for help

# Scapy

Install using:

```
$ git clone https://github.com/CaringCaribou/caringcaribou.git  
$ cd caringcaribou && python setup.py install
```

Example:

<https://github.com/CaringCaribou/caringcaribou/tree/master/caringcaribou/modules>

Why choose this tool?

- Dedicated modules for CAN message reverse engineering
- Powerful module for UDS RE such as attacking security seeds, DID enumeration, ECU discovery
- Support for DoIP and XCP reverse engineering

Fixme

# Scapy – CAN send message

```
from scapy.all import *
load_layer("can")

# Configure Socket
conf.contribs['CANSocket'] = {'use-python-can': False} # default
load_contrib('cansocket')
socket = CANSocket(channel='vcan0')

#Send a Packet
packet = CAN(identifier=0x123, data=b'01020304')
socket.send(packet)
~
```

# Scapy – Low Level CAN attacks

- <https://munich.dissec.to/kb/chapters/can/can-lowlevelattacks.html>
- Enumerates many low-level attacks on CAN devices including
  - Bus Flood Attacks
  - Simple Frame Spoofing
  - Adaptive Frame Spoofing
  - Frame Tampering
  - Error Passive Spoofing Attacks
  - Bus-off Attacks



# Other Tools

Wireshark

Fixme

# Commercial Scan Tools

Fixme

A close-up photograph of a man with a beard and mustache, wearing a blue denim shirt. He is looking down at a black smartphone held in his hands. The screen of the phone displays several lines of white text, which appear to be code or technical text. The background is dark and out of focus.

Coding with AI  
(or vibe-coding your way  
through CyberTruck)

# Do's and Don'ts

## DO NOT!

- Transmit any information or data that is in scope of your NDA
- This includes
  - Private information
  - CAN logs
  - Files and documents
  - Pictures
  - **If you are not 100% sure it's okay, you should not send it**



# Do's and Don'ts

## What you can do...

- Ask AI about public knowledge
- Ask AI about how to use a public tool (such as the tools in can-gen)
- Ask AI to write you code or code samples, provided you aren't breaking any NDA restrictions

## Do's and Don'ts

## Examples of acceptable use...

- “Write a bash script to bifurcate a candump log file and replay it using canplayer. Ask me after each round if I found what I was looking for. If yes, continue splitting the confirmed file and replaying until the single CAN message is identified.”
- “Write a python-can script to send out periodic tester present messages at a rate of 2 seconds.”

# AI-Generated Script to Read CAN Messages

```
import can

bus = can.interface.Bus(channel='can0', bustype='socketcan')

print("Listening on can0...")
for msg in bus:
    print(msg)
```

## Prompt to AI:

“Generate a Python script that listens on can0 using python-can and prints each message.”

# Adding complexity

## Prompt to AI:

“Write a python-can script that will iterate through sending a read request for all diagnostic DIDs from 0x0000 to 0xffff. The script should use 29bit IDs.

```
import can
import time

# Constants
TX_ID = 0x18DB33F1 # 29-bit CAN ID: Functional request (example)
SERVICE_ID = 0x22    # UDS ReadDataByIdentifier

def send_did_requests(channel='can0', bustype='socketcan'):
    bus = can.interface.Bus(channel=channel, bustype=bustype, fd=False)

    print("Starting DID scan on extended CAN ID...")

    try:
        for did in range(0x0000, 0x10000): # 0x0000 to 0xFFFF
            # Format DID into 2 bytes
            did_bytes = did.to_bytes(2, byteorder='big')

            # Construct UDS request: [SID, DID_High, DID_Low]
            data = bytes([SERVICE_ID]) + did_bytes

            # Pad to 8 bytes if necessary
            data += bytes(8 - len(data))

            # Create CAN frame
            msg = can.Message(arbitration_id=TX_ID,
                               is_extended_id=True,
                               data=data)

            # Send frame
            bus.send(msg)
            print(f"Sent: SID=0x{SERVICE_ID:02X}, DID=0x{did:04X}")

            time.sleep(0.01) # Slight delay to avoid bus overload

    except KeyboardInterrupt:
        print("\nScan interrupted by user.")
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    send_did_requests()
```



# Additional Tools



# Interfacing with J1708

- **Use RP1210 Adapter with J1708 support**
  - Examples: DG Tech DPA4, DPA5, Nexiq USBLINK
- **Use py-hv-networks tool to interface over python**
  - <https://github.com/TruckHacking/py-hv-networks>
  - Windows – use RP1210 DLL (NEEDS TO USE 32-BIT!!!!!!)
  - Linux – You can use the TruckDuck hardware
- **Mentors who are knowledgeable with J1708 and PLC**
  - Ben Gardiner



**RESOURCES**

**Additional Resources**

# Additional Resources

- **My Cybertruck Repository**
  - <https://github.com/hexsecs/cybertruck>
- **Awesome CANbus**
  - <https://github.com/iDoka/awesome-canbus>
- **Sharika Kumar – 2023 – Diagnostic Systems (Includes RP1210 Shimming)**
  - <https://www.cybertruckchallenge.org/wp-content/uploads/2023/06/Diagnostic-Systems-by-Sharika-Kumar.pdf>
- **Dr. Daily - Intro to J1939 (Includes RP1210 Shimming)**
  - <https://www.cybertruckchallenge.org/wp-content/uploads/2023/06/Introduction-to-SAE-J1939-CyberTruck-2023.pdf>
- **Hannah Silva – 2022 CyberTruck Training (Includes TruckDevil)**
  - [https://www.cybertruckchallenge.org/wp-content/uploads/2022/08/Silva\\_Heavy\\_Vehicle\\_Networks.pdf](https://www.cybertruckchallenge.org/wp-content/uploads/2022/08/Silva_Heavy_Vehicle_Networks.pdf)
  - [https://www.cybertruckchallenge.org/wp-content/uploads/2022/08/Silva\\_Vehicle\\_Network\\_Security.pdf](https://www.cybertruckchallenge.org/wp-content/uploads/2022/08/Silva_Vehicle_Network_Security.pdf)
  - <https://www.youtube.com/@NMFTA-Inc> <-- Truck Devil Training Videos
- **NMFTA – Blind Wireless Seed Key Unlock**
  - <https://nmfta.org/wp-content/media/2025/01/Blind-Wireless-Seed-Key-Unlock-Whitepaper-final.pdf>

# Additional Resources

- **Heavy Vehicle CAN data**
  - <https://www.engr.colostate.edu/~jdaily/J1939/candata.html>