Using the Meshtastic Python CLI for Diagnostics

The Meshtastic Python package ships both a library and a command-line interface (CLI). The CLI is the fastest way to interrogate a connected node, export its configuration, watch live packet traffic, and run one-off diagnostic commands from a laptop. This page covers installation, every useful diagnostic command, how to read the output, and the difference between serial and TCP connections.

Installation

```
# Requires Python 3.9 or later
pip install meshtastic

# Verify installation
meshtastic --version
# Output: meshtastic v2.x.x
```

On Linux you may need to add your user to the `dialout` group to access serial ports without sudo:

```
sudo usermod -aG dialout $USER
# Log out and back in for the change to take effect
```

Connecting to a Node

Serial (USB)

The most common connection method. Plug the device in via USB and run any command without extra flags - the CLI auto-detects the first available serial port:

```
meshtastic --info
```

To specify a port explicitly:

```
meshtastic --port /dev/ttyUSB0 --info # Linux
meshtastic --port /dev/tty.usbserial-* --info # macOS
meshtastic --port COM4 --info # Windows
```

TCP (Wi-Fi)

Nodes running ESP32 with Wi-Fi enabled expose a TCP port (default 4403). Use `--host` instead of `--port`:

```
meshtastic --host 192.168.1.42 --info
meshtastic --host meshtastic.local --info # mDNS name on the local LAN
```

BLE

BLE support requires the `bleak` extras package on some platforms:

```
pip install "meshtastic[ble]"
meshtastic --ble-scan # lists visible Meshtastic BLE devices
meshtastic --ble <MAC-or-name> --info
```

--info: Device Summary

```
meshtastic --info
```

This is the single most useful diagnostic command. It prints a comprehensive JSON-structured summary of the connected node, including:

```
Connected to radio
Owner: WB5ABC (4 char id: !aabbccdd)
My info:
{
 "myNodeNum": 2864434397,
 "hasGps": true,
 ...
}
Metadata: {
 "firmwareVersion": "2.5.3.abcdef",
 "deviceStateVersion": 23,
```

```
 "hasWifi": true,
 "hasBluetooth": true,
 "hasEthernet": false,
 "role": "CLIENT",
 "positionFlags": 811,
 "hwModel": "TLORA_V2_1_1P6",
 "hasRemoteHardware": false,
 "canShutdown": false
}
Nodes in mesh: {
 "!aabbccdd": {
 "num": 2864434397,
 "user": { "id": "!aabbccdd", "longName": "WB5ABC", "shortName": "W5", "hwModel":
"TLORA_V2_1_1P6" },
 "position": { "latitudeI": 323456789, "longitudeI": -970123456, "altitude": 312, "time":
1714000000 },
 "snr": 6.5,
 "lastHeard": 1714010000,
 "deviceMetrics": { "batteryLevel": 87, "voltage": 3.98, "channelUtilization": 4.25,
"airUtilTx": 0.81 }
 },
 ...
}
Preferences: { ... full radio config ... }
Channels: [ { "index": 0, "role": "PRIMARY", "settings": { ... } }, ... ]
```

Key fields to check during diagnostics:

- `channelUtilization`: percentage of channel used in the last 5 minutes.
- `airUtilTx`: TX duty cycle percentage.
- `snr`: last-heard SNR from each remote node (dB).
- `lastHeard`: Unix timestamp. Subtract from current time to see how stale a node entry is.
- `firmwareVersion`: compare against the latest release to check if updates are needed.

--nodes: Node Table

```
meshtastic --nodes
```

Prints a compact tabular summary of all nodes in the mesh database:

```
 N Num User AKA Hardware Latitude Longitude Altitude Battery SNR LastHeard
 1 2864434397 WB5ABC W5 TLORA_V2 32.3456 -97.0123 312m 87% 6.5 2024-04-25 10:31:22
 2 3109276812 K5ZZZ K5 RAK4631 32.3512 -97.0099 289m 72% 2.1 2024-04-25 10:29:55
 3 4012345678 N5XYZ N5 HELTEC_V3 32.3390 -97.0210 278m -- -4.7 2024-04-25 09:15:10
```

The `LastHeard` column quickly shows stale nodes (entries in the node DB that haven't been heard in hours or days). These are candidates for manual removal if you are troubleshooting ghost-node problems. The SNR column shows the signal quality of the last packet heard *from* that node (not necessarily a direct link - the packet may have been relayed).

--export-config: Full Configuration Export

```
meshtastic --export-config > node_backup_$(date +%Y%m%d).yaml
```

Exports the complete device configuration as YAML. This includes radio settings, channel definitions, module configs (telemetry intervals, MQTT settings, serial module, etc.), and device metadata. Use this to:

- Back up a node before a firmware update.
- Clone configuration from one device to another with `meshtastic --configure config.yaml`.
- Audit configuration differences between nodes in a community mesh.

--listen: Live Packet Watch

```
meshtastic --listen
```

Subscribes to all decoded packets from the connected node and prints them as they arrive. This is the equivalent of a packet sniffer for the mesh. Press `Ctrl-C` to stop. Example output:

```
TEXT_MESSAGE_APP: from=!aabbccdd, to=^all, id=0x12345678, hop_limit=3, want_ack=False
 payload: b'Hello from WB5ABC'

POSITION_APP: from=!aabbccdd, to=^all, id=0xdeadbeef, hop_limit=3, want_ack=False
 payload: Position(latitudeI=323456789, longitudeI=-970123456, altitude=312, time=1714010500)

TELEMETRY_APP: from=!ccccdddd, to=^all, id=0x87654321, hop_limit=1, want_ack=False
 payload: Telemetry(deviceMetrics=DeviceMetrics(batteryLevel=72, voltage=3.91,
```

```
  channelUtilization=11.3, airUtilTx=2.1))
```

During a `--listen` session, watch for:

- Packets arriving with `hop_limit=0` (the original TTL minus hops taken). A `hop_limit` of 0 on arrival means the packet used all of its allowed hops - the hop budget may be too low.
- Repeated identical packet IDs arriving within seconds (duplicate storm).
- High-frequency position packets from a single node ID indicating that node's smart position interval is too aggressive.

Reading Telemetry from --listen

Every 15 minutes by default each node broadcasts device telemetry. While listening you can redirect output and grep for telemetry to build a quick health log:

```
meshtastic --listen 2>&1 | grep TELEMETRY_APP | tee mesh_telemetry.log
```

Practical Diagnostic Workflow

1. **Start with `--info`** to get baseline node count and channelUtilization. If utilization > 25%, immediately check individual node position intervals.
2. **Run `--nodes`** and flag any node where `LastHeard` is more than 2 hours old during an active session. These are likely ghost nodes.
3. **Run `--listen` for 5 - 10 minutes** during peak mesh activity to count duplicate packet IDs and identify high-frequency transmitters.
4. **Export config with `--export-config`** and compare `hop_limit` across nodes. A community mesh should generally use a maximum hop limit of 3 (the default).

Serial vs TCP: When to Use Each

Method	Best For	Limitations
Serial (USB)	Local bench testing, initial setup, firmware examination	Requires physical access; occupies USB port preventing simultaneous app connection
TCP (Wi-Fi)	Remote diagnostics on deployed nodes, scripted automation, Raspberry Pi gateways	Requires node to have Wi-Fi enabled and be on local network or accessible via port forward
BLE	Temporary field diagnostics without Wi-Fi	Short range, platform-specific BLE stack issues, slower data rate

Revision #2
Created 2026-05-03 05:39:37 UTC by Mesh America Admin
Updated 2026-05-03 12:59:11 UTC by Mesh America Admin