Automating Meshtastic: Practical Scripts

The Meshtastic Python library enables powerful automation workflows. This page provides four complete, ready-to-use scripts: a position logger, a message forwarder to Telegram, a battery monitor with alerts, and an automated network health reporter. Each script is self-contained and includes setup instructions.

Script 1: Position Logger to CSV

Logs GPS position updates from all mesh nodes to a CSV file in real time. Useful for tracking mobile assets, recording deployment surveys, or building a trace of network coverage over time. **Privacy note:** this logger writes every observed node's GPS coordinates to a plaintext file. Treat that file as sensitive location data and protect it accordingly.

```
# position_logger.py
# Logs all position packets from the Meshtastic mesh to a CSV file.
#
# Usage:
# pip install meshtastic
# python position_logger.py [--port /dev/ttyUSB0] [--output positions.csv]
import argparse
import csv
import datetime
import os
import sys
import time

import meshtastic
import meshtastic.serial_interface
import meshtastic.tcp_interface
from pubsub import pub

OUTPUT_FILE = "positions.csv"
```

```
CSV_FIELDS = ["timestamp", "node_id", "long_name", "short_name",
 "latitude", "longitude", "altitude_m", "speed_kmh",
 "heading_deg", "snr_db", "hop_count"]


def get_interface(args):
 if args.host:
 return meshtastic.tcp_interface.TCPInterface(hostname=args.host)
 port = args.port or None
 return meshtastic.serial_interface.SerialInterface(devPath=port)


def main():
 parser = argparse.ArgumentParser(description="Log Meshtastic positions to CSV")
 parser.add_argument("--port", help="Serial port (e.g. /dev/ttyUSB0 or COM4)")
 parser.add_argument("--host", help="TCP hostname or IP address")
 parser.add_argument("--output", default=OUTPUT_FILE, help="Output CSV file path")
 args = parser.parse_args()

 file_exists = os.path.isfile(args.output)
 outfile = open(args.output, "a", newline="", encoding="utf-8")
 writer = csv.DictWriter(outfile, fieldnames=CSV_FIELDS)
 if not file_exists:
 writer.writeheader()

 iface_ref = [None] # mutable container for the interface

 def on_position(packet, interface):
 decoded = packet.get("decoded", {})
 pos = decoded.get("position", {})
 if not pos.get("latitudeI"):
 return # no valid fix

 node_id = packet.get("fromId", "unknown")
 nodes = interface.nodes or {}
 node_info = nodes.get(node_id, {})
 user = node_info.get("user", {})

 lat = pos["latitudeI"] / 1e7
```

```
lon = pos["longitudeI"] / 1e7
alt = pos.get("altitude", 0)
speed = pos.get("groundSpeed", 0)
# NOTE: verify the groundTrack/heading scaling in the Position protobuf for your
# firmware version before dividing by 100; the field's units have varied.
heading = pos.get("groundTrack", 0)
snr = node_info.get("snr", "")
# Guard against missing hopStart/hopLimit; record None rather than a bogus 0.
if "hopStart" in packet and "hopLimit" in packet:
hops = packet["hopStart"] - packet["hopLimit"]
else:
hops = None

row = {
"timestamp": datetime.datetime.now(datetime.timezone.utc).isoformat(),
"node_id": node_id,
"long_name": user.get("longName", ""),
"short_name": user.get("shortName", ""),
"latitude": f"{lat:.7f}",
"longitude": f"{lon:.7f}",
"altitude_m": alt,
"speed_kmh": speed * 3.6 if speed else 0,
"heading_deg": heading / 100 if heading else 0,
"snr_db": snr,
"hop_count": hops,
}
writer.writerow(row)
outfile.flush()
print(f"[{row['timestamp']}] {node_id} ({row['long_name']}) "
f"@ {lat:.5f},{lon:.5f} alt={alt}m")

pub.subscribe(on_position, "meshtastic.receive.position")

iface = get_interface(args)
iface_ref[0] = iface
print(f"Logging positions to {args.output}. Press Ctrl-C to stop.")

try:
while True:
```

```
 time.sleep(1)
 except KeyboardInterrupt:
 pass
 finally:
 iface.close()
 outfile.close()
 print("Logger stopped.")

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

Script 2: Message Forwarder to Telegram

Forwards all text messages received on the mesh to a Telegram chat. Requires a Telegram bot token (create one via `@BotFather`) and a chat ID. This is a popular pattern for monitoring a community mesh from a smartphone without needing Bluetooth or Wi-Fi proximity to a node.

```
# mesh_to_telegram.py
# Forwards Meshtastic text messages to a Telegram chat.
#
# Setup:
# pip install meshtastic requests
# Set environment variables:
# TELEGRAM_TOKEN=<your bot token>
# TELEGRAM_CHAT_ID=<chat id, e.g. -1001234567890>
# python mesh_to_telegram.py [--host 192.168.1.42]
import os
import sys
import time
import datetime
import requests
import meshtastic
import meshtastic.serial_interface
import meshtastic.tcp_interface
from pubsub import pub
```

```
TELEGRAM_TOKEN = os.environ.get("TELEGRAM_TOKEN", "")
TELEGRAM_CHAT_ID = os.environ.get("TELEGRAM_CHAT_ID", "")

if not TELEGRAM_TOKEN or not TELEGRAM_CHAT_ID:
 print("ERROR: Set TELEGRAM_TOKEN and TELEGRAM_CHAT_ID environment variables.")
 sys.exit(1)


def send_telegram(text: str):
 # Send a message to the configured Telegram chat.
 url = f"https://api.telegram.org/bot{TELEGRAM_TOKEN}/sendMessage"
 data = {"chat_id": TELEGRAM_CHAT_ID, "text": text, "parse_mode": "HTML"}
 try:
 resp = requests.post(url, json=data, timeout=10)
 resp.raise_for_status()
 except requests.RequestException as exc:
 print(f"Telegram send failed: {exc}")


def on_receive(packet, interface):
 decoded = packet.get("decoded", {})
 if decoded.get("portnum") != "TEXT_MESSAGE_APP":
 return

 text = decoded.get("text", "")
 from_id = packet.get("fromId", "unknown")
 to_id = packet.get("toId", "^all")
 hops = (packet["hopStart"] - packet["hopLimit"]
 if "hopStart" in packet and "hopLimit" in packet else None)
 timestamp = datetime.datetime.now(datetime.timezone.utc).strftime("%H:%M:%S UTC")

 nodes = interface.nodes or {}
 node_info = nodes.get(from_id, {})
 long_name = node_info.get("user", {}).get("longName", from_id)

 dest_str = "broadcast" if to_id in ("^all", "4294967295") else to_id
 tg_msg = (
 f"Mesh Message [{timestamp}]
```

```
"
 f"From:
{long_name}
 ({from_id})
"
 f"To: {dest_str} | {hops} hop(s)
"
 f"
{text}"
 )

 print(f"Forwarding to Telegram: {text!r} from {long_name}")
 send_telegram(tg_msg)


import argparse
parser = argparse.ArgumentParser()
parser.add_argument("--host", help="TCP hostname or IP")
parser.add_argument("--port", help="Serial port")
args = parser.parse_args()

pub.subscribe(on_receive, "meshtastic.receive.text")

if args.host:
 iface = meshtastic.tcp_interface.TCPInterface(hostname=args.host)
else:
 iface = meshtastic.serial_interface.SerialInterface(devPath=args.port)

print(f"Forwarding mesh messages to Telegram chat {TELEGRAM_CHAT_ID}. Press Ctrl-C to stop.")
try:
 while True:
 time.sleep(1)
except KeyboardInterrupt:
 pass
finally:
 iface.close()
```

Script 3: Battery Monitor with Alerts

Monitors battery levels of all nodes in the mesh and sends a text-message alert over the mesh when any node's battery falls below a configurable threshold. Useful for solar-powered relay nodes where low battery means imminent network degradation.

```
# battery_monitor.py
# Sends a mesh alert when any node battery drops below a threshold.
#
# Usage:
# python battery_monitor.py [--threshold 20] [--interval 300]
import argparse
import time
import meshtastic
import meshtastic.serial_interface
from pubsub import pub

ALERT_COOLDOWN = {} # node_id -> last_alert_time to avoid spamming

def check_battery(iface, threshold, interval):
 # Periodically check all nodes and alert on low battery.
 while True:
 time.sleep(interval)
 nodes = iface.nodes or {}
 now = time.time()

 for node_id, node_data in nodes.items():
 metrics = node_data.get("deviceMetrics", {})
 battery = metrics.get("batteryLevel")

 if battery is None:
 continue # no telemetry available

 # Skip nodes that are charging (level > 100 indicates USB power on some firmware)
 if battery > 100:
 continue
```

```
 if battery < threshold:
 last_alert = ALERT_COOLDOWN.get(node_id, 0)
 # Only alert once per hour per node
 if now - last_alert > 3600:
 long_name = node_data.get("user", {}).get("longName", node_id)
 alert_msg = (
 f"LOW BATTERY ALERT: {long_name} ({node_id}) "
 f"is at {battery}%"
 )
 print(alert_msg)
 # Broadcast alert on the primary channel
 iface.sendText(alert_msg)
 ALERT_COOLDOWN[node_id] = now

def main():
 parser = argparse.ArgumentParser(description="Meshtastic battery monitor")
 parser.add_argument("--threshold", type=int, default=20,
 help="Battery percentage threshold for alerts (default: 20)")
 parser.add_argument("--interval", type=int, default=300,
 help="Check interval in seconds (default: 300)")
 parser.add_argument("--port", help="Serial port")
 parser.add_argument("--host", help="TCP hostname")
 args = parser.parse_args()

 if args.host:
 import meshtastic.tcp_interface
 iface = meshtastic.tcp_interface.TCPInterface(hostname=args.host)
 else:
 iface = meshtastic.serial_interface.SerialInterface(devPath=args.port)

 print(f"Battery monitor started. Alert threshold: {args.threshold}%. "
 f"Check interval: {args.interval}s.")

 import threading
 t = threading.Thread(target=check_battery,
 args=(iface, args.threshold, args.interval),
 daemon=True)
 t.start()
```

```
 try:
 while True:
 time.sleep(1)
 except KeyboardInterrupt:
 pass
 finally:
 iface.close()

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

Script 4: Automated Network Health Reporter

Generates a periodic network health report and either prints it to the console or sends it as a mesh broadcast. Summarizes node count, online/offline status, average SNR, channel utilization, and identifies any nodes not heard in the last configured window.

```
# health_reporter.py
# Generates periodic Meshtastic network health reports.
#
# Usage:
# python health_reporter.py [--interval 3600] [--broadcast]
import argparse
import time
import datetime
import meshtastic
import meshtastic.serial_interface
import meshtastic.tcp_interface


def generate_report(iface, offline_threshold_minutes=60) -> str:
 # Build a health report string from current node data.
 nodes = iface.nodes or {}
 now = time.time()
 total = len(nodes)
 online = []
```

```
offline = []
snr_values = []
util_values = []

for node_id, node_data in nodes.items():
last_heard = node_data.get("lastHeard", 0)
age_min = (now - last_heard) / 60 if last_heard else None
long_name = node_data.get("user", {}).get("longName", node_id)
metrics = node_data.get("deviceMetrics", {})
snr = node_data.get("snr")
util = metrics.get("channelUtilization")

if snr is not None:
snr_values.append(snr)
if util is not None:
util_values.append(util)

if age_min is not None and age_min < offline_threshold_minutes:
online.append((long_name, age_min, snr))
else:
offline.append((long_name, age_min))

avg_snr = sum(snr_values) / len(snr_values) if snr_values else None
avg_util = sum(util_values) / len(util_values) if util_values else None

lines = [
f"=== Mesh Health Report {datetime.datetime.now(datetime.timezone.utc).strftime('%Y-%m-%d
%H:%M UTC')} ===",
f"Total nodes: {total} | Online (<{offline_threshold_minutes}m): {len(online)}"
f" | Offline: {len(offline)}",
]

if avg_util is not None:
# These bands are custom to this script. The app's conventional channel-utilization
# bands are roughly green < 25% / orange 25-50% / red > 50%.
health = "OK" if avg_util < 15 else "WARN" if avg_util < 25 else "HIGH"
lines.append(f"Avg channel utilization: {avg_util:.1f}% [{health}]")

if avg_snr is not None:
lines.append(f"Avg SNR (last heard): {avg_snr:.1f} dB")
```

```
 if offline:
 lines.append(f"Offline nodes ({len(offline)}):")
 for name, age in offline:
 age_str = f"{age:.0f}m ago" if age is not None else "never"
 lines.append(f" - {name}: last heard {age_str}")

 return "
".join(lines)


def main():
 parser = argparse.ArgumentParser(description="Meshtastic network health reporter")
 parser.add_argument("--interval", type=int, default=3600,
 help="Report interval in seconds (default: 3600)")
 parser.add_argument("--broadcast", action="store_true",
 help="Broadcast report as mesh text message")
 parser.add_argument("--offline", type=int, default=60,
 help="Minutes without a packet to consider a node offline (default: 60)")
 parser.add_argument("--port", help="Serial port")
 parser.add_argument("--host", help="TCP hostname")
 args = parser.parse_args()

 if args.host:
 iface = meshtastic.tcp_interface.TCPInterface(hostname=args.host)
 else:
 iface = meshtastic.serial_interface.SerialInterface(devPath=args.port)

 print("Health reporter running. First report in", args.interval, "seconds.")

 try:
 while True:
 time.sleep(args.interval)
 report = generate_report(iface, offline_threshold_minutes=args.offline)
 print(report)
 print()

 if args.broadcast:
 # Mesh text messages cap at ~200 bytes of application payload; send a summary
 nodes = iface.nodes or {}
 now = time.time()
 online = sum(
```

```
 1 for n in nodes.values()
 if (now - n.get("lastHeard", 0)) / 60 < args.offline
 )
 metrics = [n.get("deviceMetrics", {}).get("channelUtilization")
 for n in nodes.values()
 if n.get("deviceMetrics", {}).get("channelUtilization") is not None]
 avg_util = sum(metrics) / len(metrics) if metrics else 0
 short_report = (
 f"Mesh status: {online}/{len(nodes)} online, "
 f"chan util {avg_util:.1f}%"
 )
 iface.sendText(short_report)
 print(f"Broadcast: {short_report!r}")
 except KeyboardInterrupt:
 pass
 finally:
 iface.close()

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

Revision #3
Created 2026-05-03 05:39:40 UTC by Mesh America Admin
Updated 2026-06-09 12:15:19 UTC by Mesh America Admin