LESSON WORKBOOK

Linux Terminal 03 - Troubleshoot with Evidence

Linux Terminal

Study the lesson, work through the lab, and check your understanding with the quiz. Keep the reference sheet nearby as you practice.

Includes the complete lesson, guided lab, knowledge check, answer explanations, and quick reference.

Lesson

Linux Terminal 03 - Troubleshoot with Evidence

Track: IT Foundations -> Linux operations **Prerequisites:** Linux Terminal 01 and 02: paths, permissions, pipes, grep, processes, and SSH **Format:** 35-45 minutes teaching + 20-minute local lab **Outcome:** distinguish a running process from a working service, collect bounded evidence, and explain a next action without guessing.

1. The checkout button stopped working

Imagine you help operate a small online shop. Someone says, "The website is down." The home page opens, but the orders page shows an error. One teammate suggests restarting the server. Another suggests changing the firewall. You have enough terminal knowledge to do either. This lesson is about deciding whether either action makes sense.

First, turn the report into an observable statement: "At 14:05 UTC, requesting `/orders` from the application returns HTTP 503. `/health` returns HTTP 200." That sentence gives you a time, a test, and a boundary. It leaves room for the site to be partially working.

The exercise uses a deliberately broken teaching service. Its inventory timeout is simulated; no real shop, customer, or external service is involved. Your task is to explain the evidence. A restart is not part of this lab because the process already answers requests.

Use a simple loop: **observe -> form one hypothesis -> run a discriminating test -> record what changed**. A useful test distinguishes plausible explanations. Repeating the same command ten times without a question rarely does.

2. Know which machine you are examining

After connecting through SSH, orient yourself before collecting output:

```
hostname
date -u
whoami
pwd
```

On a real incident, record the application, environment, machine, and time zone. An accurate command on a staging server does not explain a production failure. Compare timestamps in a common time zone; clock differences can make events appear in the wrong order.

Keep observations separate from conclusions. "One request returned 503" is an observation. "The database is broken" is a hypothesis. "The log names inventory as the failing dependency" is another observation. That distinction helps the next person reproduce your thinking rather than inherit a guess.

Do not copy whole environment dumps, authorization headers, or complete customer logs into a public discussion. Collect the smallest useful excerpt. A request identifier, timestamp, route, status, and redacted error often explain more than a thousand unrelated lines.

3. A process is only one layer

Processes have IDs, owners, parent processes, and states. For a PID you already identified, take a focused snapshot:

```
ps -p 12345 -o pid,ppid,stat,etime,comm
```

Replace `12345` with your actual target; the number is illustrative. `pid` identifies the process, `ppid` its parent, `stat` its state, and `etime` its elapsed lifetime. `comm` keeps the view focused on the command name rather than full arguments, which sometimes contain secrets. Linux and macOS both support these fields, although details vary. Linux ps documentation

An `S` state usually means interruptible sleep: a normal service may be waiting for work. An `R` means running or runnable. A `Z` means the process has exited but its parent has not yet collected its status; it is not a healthy worker simply because a row remains visible.

One snapshot does not establish sustained resource pressure. Watch over an interval and correlate with request timing. A busy process can be doing useful work; a quiet process can be stalled on a dependency. Avoid diagnosing by one large number.

Signals are actions, not observations. If you own a disposable process, `kill -TERM PID` requests termination. The lab remembers the PID of the child it created and cleans up that child only. On shared infrastructure, use the service's operating procedure rather than broad name-based termination.

4. Resources provide context

These read-only checks are common on a Linux host:

```
uptime
free -h
df -h .
```

`uptime` includes load averages over one, five, and fifteen minutes. On Linux, load includes runnable tasks and tasks in uninterruptible sleep, often waiting on I/O. It is not a CPU percentage. Compare load with available CPUs and other evidence. Linux uptime documentation

`free -h` describes memory. On modern Linux, the available estimate is generally more useful than treating every cached byte as unavailable. `df -h .` reports filesystem capacity for the current directory's filesystem, not the size of the current directory. A filesystem can also exhaust inodes; `df -i .` is a separate useful check when many tiny files are involved.

These are context checks, not requirements for the portable lab. macOS has `uptime` and `df`, but not Linux's `free`; its Activity Monitor and `vm_stat` use different presentations. Containers may show host-related values or imposed limits differently. Name the environment in your notes.

5. Read a window of logs

Start small. For a known application log:

```
tail -n 40 app.log
grep -n 'status=503' app.log
grep -n -C 2 'dependency=inventory' app.log
```

The exact strings depend on the application. A severity label alone is not proof of user impact. Some applications log handled failures at ERROR; others return failed requests without that label. Correlate route, time, request ID, and status.

On a Linux machine using systemd, the journal provides filters. For an existing service named `shop.service`:

```
journalctl -u shop.service --since '15 minutes ago' -n 100 --no-pager
```

This restricts the unit, time range, and output count. Service names and access rights vary. An empty result might mean the wrong unit, a different log destination, or limited access. It does not automatically mean no errors. macOS does not use systemd, and some Linux containers do not either. systemd journalctl reference

The lab emits an explicit synthetic clue: `dependency=inventory reason=timeout`. In a real application that would justify investigating the inventory path next. It would not yet prove why inventory timed out. The dependency might be overloaded, unreachable, or responding too slowly for this caller's deadline.

6. Separate listening, reaching, and succeeding

Think of three questions. Is a socket listening? Can this client reach it? Does the requested operation succeed? Each answer covers a different layer.

On Linux, `ss -ltn` lists listening TCP sockets with numeric addresses. `ss -ltnp` also attempts to show process information; visibility depends on permissions. On macOS, an available `lsof` can inspect listeners with `lsof -nP -iTCP -sTCP:LISTEN`. These tools have different output; read local help when a flag is unfamiliar. Linux ss reference

Binding to `127.0.0.1` means loopback: clients on that same machine can reach the listener. It is intentional for our local lab. A service listening locally does not prove that traffic from the internet, a load balancer, or another subnet can reach it. That path introduces routing, access controls, name resolution, and sometimes TLS.

For HTTP, make the actual request and preserve its status:

```
curl -sS --connect-timeout 2 --max-time 5 \
  -o response.txt -w 'HTTP %{http_code}\n' \
  http://127.0.0.1:8080/orders
```

Use the actual lab port, which is printed when the script starts. Connection timeout bounds the connection phase; maximum time bounds the whole transfer. A timeout and an HTTP 503

describe different evidence. A 503 means you received an HTTP response from something on that request path. curl manual

7. HTTP status and command exit status are different

By default, curl can exit successfully after receiving an HTTP error response: the transfer itself completed. For a health check that should fail on HTTP errors, add `--fail`. For the lab's 503, curl then returns exit status 22.

That behavior matters in scripts. A monitor that only checks "did curl run?" may report a broken endpoint as healthy. Conversely, adding `--fail` without handling its result can make an intentional negative test look like a broken lab. Capture the status immediately, because the next command replaces `$?`.

HTTP 200 on `/health` can mean only that the process is alive. A business route might need a database, an inventory service, or configuration that the liveness endpoint never checks. Readiness checks often examine whether traffic should be sent to an instance, but the actual checks depend on the application. Verify what your endpoint measures.

8. Run the complete local exercise

The supplied files require Bash 3.2 or later, Python 3, and curl. No AWS account, administrator access, installation, internet request, or firewall change is needed. The server binds only to loopback and asks the operating system for an available port. Python's HTTP server is used as a teaching fixture, not a production deployment. Python HTTP server documentation

From this lesson's directory:

```
bash lab/run-lab.sh
```

Read the script first. It checks prerequisites, creates a private temporary directory, launches its own child, requests two routes, inspects the generated log, and tests curl's error behavior. The port and PID vary, but these lines should appear:

```
health HTTP: 200
orders HTTP: 503
Count of logged 503 responses: 1
curl --fail exit: 22 (expected HTTP failure)
PASS: process alive, HTTP reachable, orders unavailable; synthetic dependency evidence found.
```

The log includes `dependency=inventory reason=timeout synthetic=true`. The count is one at the moment it is printed; the later negative curl test makes a second orders request. That sequencing is intentional. Explain the count before assuming a discrepancy.

The exit handler stops only the owned server and removes only the named files the run created, then removes the empty temporary directory. Ctrl+C also triggers cleanup. If the process listing is restricted by an execution environment, the script prints that limitation while continuing the endpoint tests. A loopback restriction may prevent this lab from running; do not change machine security settings merely to bypass such a restriction.

9. Write an incident note someone can use

Complete this five-line handoff after the run:

1. **Symptom:** `/orders` returns 503 in the local teaching service.
2. **Scope:** `/health` still returns 200; requests from this local client reach the service.
3. **Evidence:** the owned process exists; the log associates orders failures with the synthetic inventory timeout.
4. **Next test:** in a real system, inspect the inventory caller's timeout and the dependency's matching time window.
5. **Limit:** this exercise does not test real DNS, TLS, external routing, or a real inventory service.

That last line strengthens the note. It tells the next engineer where your evidence stops. Avoid "network fine" when you tested only loopback, or "fixed" when you never reran the affected operation.

For a portfolio exercise, save your explanation and a redacted excerpt of actual output. Describe what you tested, what you observed, and the next discriminating check. The learning artifact is the reasoning, not a fictional production incident.

10. Apply the method to the next problem

If the process is missing, investigate startup and supervision. If it is listening only on loopback but needs remote clients, examine the intended binding and network design. If the request connects but returns 503, move up to application evidence. If a change is made, repeat the original user-facing test and compare the same time window.

Next, Linux Terminal 04 turns the same evidence habit into automation. A backup command that exits successfully is only one layer. You will create an archive, restore it into a separate directory, compare the files, and make the script tell the truth about failure.

Sources checked: September 18, 2026. Command examples use portable Bash where stated; systemd, `ss`, and `free` sections target Linux. Platform command availability and permissions still depend on the learner's installation. Additional memory reference: procps free manual.

Knowledge check and answers

Linux Terminal 03 - Knowledge Check

Eight questions. Answer before opening the explanation. Suggested pass: 6/8, then retry missed concepts using the lab evidence.

1. The health route returns 200; orders returns 503. What is established?

A. Every application dependency is healthy. B. The process answers requests, but the tested business operation fails. C. DNS is broken. D. The machine must be restarted.

Answer: B. The health endpoint and orders endpoint test different work. A running process and an HTTP response do not establish dependency health. The observation supports investigating the affected route.

2. Which observation best supports an inventory-specific next test?

A. The server has been running for a week. B. The terminal uses a dark theme. C. A matching failed-request log includes `dependency=inventory reason=timeout`. D. One process is sleeping.

Answer: C. This correlates the failed operation with a named dependency. It motivates investigating that path, but does not by itself distinguish dependency overload, routing trouble, or an overly short caller deadline.

3. What does `df -h .` report?

A. The size of every file inside the current directory. B. The filesystem capacity and use for the filesystem containing the current directory. C. The application's memory use. D. The current directory's inode count only.

Answer: B. `df` reports filesystem usage. Directory content size is a different question. When many small files are involved, inspect inode availability separately with an appropriate `df -i` check.

4. A Linux load average of 4.0 proves the CPU is at 400%. True or false?

Answer: False. Linux load counts runnable tasks and tasks in uninterruptible sleep, often related to I/O. It is not a percentage. Interpret it with CPU count, workload, time interval, and other evidence.

5. Curl prints an HTTP 503, but its exit status is zero. Is this necessarily a curl bug?

Answer: No. Without an option such as `--fail`, receiving an HTTP error can still be a successful transfer. The lab checks the HTTP status explicitly and also demonstrates `--fail`, which returns 22 for its 503 response.

6. Which conclusion follows from successfully reaching `127.0.0.1`?

A. Every remote user can reach the service. B. The public DNS record is correct. C. This local request path reaches the loopback listener. D. The external firewall allows the application.

Answer: C. Loopback stays on the same host. External name resolution, routing, access controls, and load balancers are outside that test's scope.

7. You want recent logs for a known systemd unit without an unbounded dump. Which is best?

A. `journalctl -u shop.service --since '15 minutes ago' -n 100 --no-pager` B. `killall shop` C. `chmod -R 777 /var/log` D. Reboot and hope.

Answer: A. Unit, time, and count filters limit the evidence window. The unit must exist and you need permission to read the relevant journal. macOS and Linux installations without systemd require different log tooling.

8. Write a two-sentence handoff for the lab.

Example answer: "From the local client, `/health` returned 200 and `/orders` returned 503 while the owned fixture process was running. The application log names a synthetic inventory timeout; a real follow-up would compare the caller and dependency time window, and this test establishes nothing about public DNS, TLS, or remote connectivity."

Explanation: A useful handoff contains observations, a justified next test, and the limits of the test. It does not claim a real incident, a root cause, or a repair that did not occur.

Quick reference

Linux Terminal 03 - Evidence Before Action

Method: observe -> one hypothesis -> discriminating test -> record result.

Question	Focused check	What it establishes
Which machine and time?	`hostname`; `date -u`; `whoami`; `pwd`	Context for subsequent evidence
Does the known process exist?	`ps -p PID -o pid,ppid,stat,etime,comm`	Identity and a process-state snapshot
Is resource pressure plausible?	Linux: `uptime`, `free -h`, `df -h .`	Load, memory, filesystem context
What did the application report?	`grep -n -C 2 'status=503' app.log`	Matching log lines with context
What happened recently in a unit?	`journalctl -u shop.service --since '15 minutes ago' -n 100 --no-pager`	Bounded systemd journal window
Is TCP listening?	Linux: `ss -ltn`	Listening TCP addresses and ports
Does the operation work?	Request the exact route with bounded curl	Result from this client's request path

HTTP test template: replace the example port with the actual port.

```
curl -sS --connect-timeout 2 --max-time 5 \
  -o response.txt -w 'HTTP %{http_code}\n' \
  http://127.0.0.1:8080/orders
```

Two statuses: HTTP status describes the response; shell exit status describes curl's result. Curl may exit zero for an HTTP error unless you use `--fail` or explicitly inspect the HTTP status. The lab's 503 produces exit 22 with `--fail`.

Three boundaries: a process exists ≠ it is listening; a listener exists ≠ every client reaches it; a health route succeeds ≠ every business operation succeeds.

Portable lab: `bash lab/run-lab.sh` - Bash 3.2+, Python 3, curl. Local loopback only. Expected health 200, orders 503, synthetic inventory timeout, then PASS. Automatic cleanup stops its own child and removes exact temporary files.

Linux versus macOS: `ss`, `free`, and systemd's `journalctl` are Linux-oriented. macOS can use `lsof -nP -iTCP -sTCP:LISTEN` for a listener view when available; memory and system logs use

different tools. The supplied local lab supports both platforms.

Handoff: symptom · scope · evidence · next test · limit. After a change, repeat the original failing operation. Use small, redacted excerpts.

References checked September 18, 2026: ps, ss, journalctl, curl. Full teaching context is in `lesson.md`.