

KERNEL PANIC

Bau dir ein Linux: Eine kurze Systemrettung, bei der aus richtigen Diagnoseentscheidungen Schritt für Schritt ein funktionierender Rechner entsteht.

10 × 10

Fragenpool

10 + 1

Erfolgsmomente

Hall of Tux

Gemeinsames Scoreboard

Projekt: André Stiebitz

Abschlussprojekt im Linux-Kurs · Stand 30.07.2026

andrestiebitz.de/kernel-panic/

Vollständige Projektdokumentation · MIT Open Source

Ausgangslage, Ziel und Ergebnis

Als Abschlussprojekt des Linux-Kurses sollte ein frei gewähltes, nachvollziehbares Projekt entstehen. Meine Wahl ist „Kernel Panic — Bau dir dein Linux“: ein Browsergame, in dem der Spieler ein gestörtes Linux-System Schritt für Schritt wieder in Betrieb nimmt. Die fertige Abgabe besteht aus dem öffentlichen Spiel, dieser Dokumentation, einem Download-Paket und dem offenen Quellcode.

ZIEL

Wichtige Inhalte aus LPIC-1 101-500 und 102-500 durch konkrete Diagnoseentscheidungen wiederholen.

ZIELGRUPPE

Linux-Einsteiger im Kurs sowie Mitschüler, die Befehle in einem spielerischen Zusammenhang üben möchten.

PRODUKT

Eine ohne Anmeldung erreichbare, responsive Web-App mit 100 Fragen in zehn Kapiteln und zwei Spielmodi.

ABGABE

Live-Link, Projektdokumentation als PDF, vollständiges ZIP und öffentliches Codeberg-Repository unter MIT.

ABNAHMEKRITERIEN

- Das Spiel öffnet sich per Link ohne Installation oder Login.
- Desktop und Smartphone funktionieren ohne horizontalen Überlauf.
- Zehn Kapitel enthalten jeweils zehn eigenständig formulierte Fragen.
- Kurslauf und Lernmodus vermitteln Inhalte auf unterschiedliche Weise.
- Der Systemplan wächst sichtbar mit den gelösten Kapiteln.
- Es werden keine echten Systembefehle ausgeführt und keine Konten oder Trackingdaten erhoben.
- Die optionale Bestenliste speichert nur Alias und Ergebnisdaten.
- Build, Fragenbank, kompletter Spielweg und Downloads sind automatisiert geprüft.
- Quellcode, Lizenz und reproduzierbarer lokaler Start sind dokumentiert.

Ergebnis: Alle Kriterien sind umgesetzt. Das Spiel ist seit dem 30.07.2026 unter <https://andrestiebitz.de/kernel-panic/> öffentlich erreichbar.

Vom Signal zur sicheren Entscheidung

Das Projekt fragt Befehle nicht isoliert ab. Jede Aufgabe beginnt mit einer beobachtbaren Situation und folgt demselben Denkweg. Damit wird nicht nur „Welcher Befehl?“ gelernt, sondern auch „Warum dieser Befehl und welches Risiko vermeide ich?“.

1 · SIGNAL

Ausgabe, Zustand oder Fehlermeldung lesen.

2 · WERKZEUG

Passenden Befehl oder Mechanismus wählen.

3 · ENTSCHEIDUNG

Den kleinsten sicheren Schritt ableiten.

4 · RISIKO

Unnötige oder destruktive Eingriffe erkennen.

KAPITEL- UND PRÜFUNGSBEZUG

KAPITEL	LPIC-BEZUG	SCHWERPUNKTE IM FRAGENPOOL
Boot	101.1–101.3	Hardware, Kernel, GRUB, Journal und Units
Pakete	102.3–102.5	APT, dpkg, RPM, DNF und Bibliotheken
Text	103.2–103.8	Filter, Streams, Umleitungen und Archive
Prozesse	103.5 / 103.6	Jobs, Signale, nice und Prozesszustände
Storage	104.1–104.3	Blockgeräte, fstab, Inodes, Swap und Mounts
Rechte	104.5–104.7	Modi, Spezialbits, Links, find und stat
Shell	105.1 / 105.2	Variablen, Tests, Schleifen und Skripte
Netzwerk	109.1–109.4	IP, Routing, Sockets, DNS und CIDR
Sicherheit	107 / 110	Konten, Gruppen, SSH-Schlüssel und sudo
Betrieb	107 / 108	Zeitplanung, Locale, Logs und Verify

Die Zuordnung dient der fachlichen Orientierung. Das Spiel bildet nicht jedes einzelne LPIC-Lernziel ab und ersetzt weder Kursunterlagen noch Prüfungsvorbereitung.

INHALTLICHE ARBEITSWEISE

Die bereitgestellten Linux-Kursunterlagen wurden ausschließlich genutzt, um die Themenabdeckung zu kontrollieren. Sämtliche Spieltexte und Szenarien wurden für dieses Projekt neu formuliert. Kursoriginale, Lösungssammlungen und Prüfungs-Screenshots sind nicht enthalten.

Zwei Modi, drei Aufgabentypen

KURSLAUF • 30 FRAGEN

Pro Kapitel werden drei Fragen zufällig ausgewählt. Der Lauf dauert ungefähr 30 Minuten und eignet sich für Vorführung und Wiederholung.

LERNMODUS • 100 FRAGEN

Alle zehn Fragen jedes Kapitels werden gespielt. Der gespeicherte Fortschritt ermöglicht mehrere Sitzungen.

AUFGABENTYPEN

AUSWAHL

Eine Diagnose oder sichere nächste Aktion wird aus mehreren Antworten gewählt. Genau eine Antwort ist richtig.

SIMULIERTES TERMINAL

Der Spieler tippt einen Befehl selbst. Leerzeichen und Pipes werden normalisiert, Groß-/Kleinschreibung bleibt absichtlich relevant.

SEQUENZ

Gemischte Befehls- oder Skriptbausteine werden in eine ausführbare Reihenfolge gesetzt. Ablenker müssen bewusst weggelassen werden.

FEEDBACK

Fehler kosten XP und Systemintegrität, Hinweise kosten XP. Nach einer Antwort werden Signal, Werkzeug, Entscheidung und Risiko erklärt.

FORTSCHRITT UND ABSCHLUSS

- Fragen und Antwortbausteine werden gemischt; Sequenzen verraten die Lösung nicht.
- Die zufällige Auswahl bleibt nach Neuladen stabil.
- Kapitel werden nacheinander freigeschaltet.
- Jedes abgeschlossene Kapitel schaltet ein Modul im Systemplan ein.
- Nach jedem Kapitel feiert eine kurze Animation den Erfolg und fasst drei gelernte Fähigkeiten zusammen.
- Der Abschlussbericht zeigt je Kapitel Lösungen ohne Hilfe, Fehler und Hinweise.
- Der finale Systemstart bietet einen anonymen Eintrag in die gemeinsame „Hall of Tux“ an.
- „Neu starten“ verwirft den lokalen Spielstand kontrolliert.

Ein Fehler kostet, aber vernichtet nichts

Lernen entsteht an der Stelle, an der eine Antwort nicht stimmt. Die Fehlerbehandlung ist deshalb kein Abbruch, sondern ein eigener Lernschritt: sie benennt den Preis, zeigt die Lösung und führt weiter.

SICHTBARER PREIS

Eine falsche Antwort kostet zehn XP und zwei Prozentpunkte Integrität. Der Abzug steht in der Fehlermeldung, statt unbemerkt in der Statistik zu verschwinden.

LÖSUNG SOFORT

Die Meldung nennt den richtigen Befehl beziehungsweise die richtige Antwort. Das Schema Signal → Werkzeug → Entscheidung → Risiko klappt automatisch auf.

AUTOMATISCH WEITER

Nach dreieinhalb Sekunden erscheint die nächste Frage. Ein „Weiter“-Knopf ist immer vorhanden und überspringt die Wartezeit sofort.

DREI FEHLER

Drei Fehler in einem Kapitel setzen auf den Kapitelanfang zurück. Gelöste Fragen und gesammelte XP bleiben erhalten; nur der Einstiegspunkt wandert zurück.

WARUM EINE VERPASSTE FRAGE ZURÜCKKOMMT

Eine falsch beantwortete Frage gilt nicht als gelöst und wird am Ende des Kapitels erneut vorgelegt. Ein Kapitel ist erst abgeschlossen, wenn alle seine Fragen sitzen. Das hält den Lauf vollständig abschließbar und bewahrt gleichzeitig den Vertrag mit der Score-API, die für jede Frage des Laufs ein Einzelergebnis erwartet.

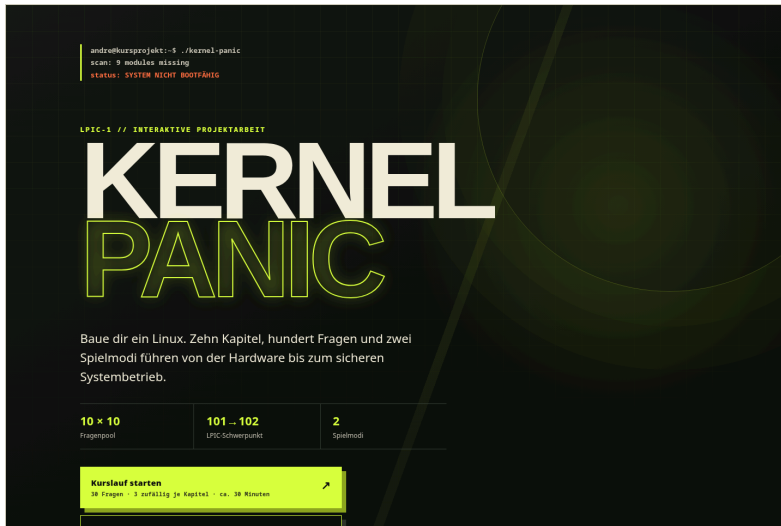
LERNÜBERSICHT ZUM MITNEHMEN

Am Ende eines Laufs gibt die Abschlusseite einen persönlichen Lern-Spickzettel heraus, als Markdown-Datei und als Druckansicht. Er beginnt mit den Fragen, die einen zweiten Versuch oder einen Hinweis gebraucht haben — jeweils mit Lösung und vollständigem Lernschema —, und endet mit der Bilanz pro Kapitel. Er entsteht vollständig im Browser, ohne Serveranfrage und ohne persönliche Daten.

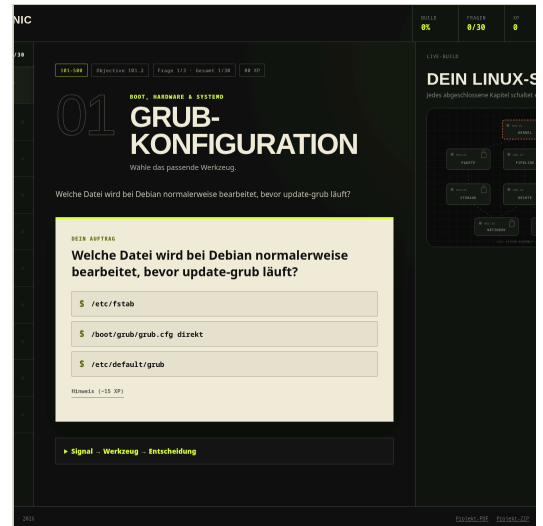
- Fehlversuche sind pro Frage bei zwanzig gedeckelt; die Integrität wird aus ihnen abgeleitet und bleibt so mit der serverseitigen Berechnung deckungsgleich.
- Ein ausstehender Kapitel-Reset lässt sich nicht durch Neuladen der Seite umgehen.
- Bei aktivierter Systemeinstellung für reduzierte Bewegung ersetzt ein ruhiger Hinweistext die Animation; der automatische Wechsel bleibt angekündigt.

Benutzerführung und sichtbarer Systemaufbau

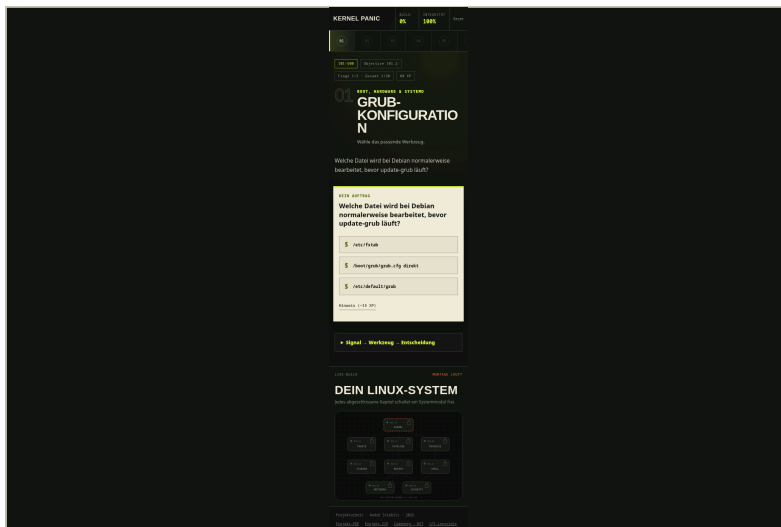
Die Oberfläche verbindet eine reduzierte Terminal-Ästhetik mit klaren hellen Aufgabenflächen. Kapitelstatus, XP, Integrität und Fortschritt bleiben jederzeit sichtbar. Die vier Ansichten zeigen den vollständigen Weg von der Moduswahl bis zum Kapitelbericht.



Start: Projektidee, Kurslauf, Lernmodus und direkte Abgabe-Downloads.



Desktop: Kapitelrail, Aufgabe, Hilfesystem und fachliche Einordnung.



Mobil: dieselbe Mission responsiv auf 390 Pixel Breite.



Abschluss: Systemstart, Ergebnisbericht und gemeinsame Hall of Tux.

Statisches Spiel mit optionaler Score-API

„Kernel Panic“ ist mit Next.js 16 und React 19 umgesetzt und wird als statische Website exportiert. Das gesamte Lernen funktioniert ohne Anwendungsserver. Nur die optionale Bestenliste spricht mit einer getrennten Python-API und einer privaten SQLite-Datei. CSS erzeugt die Oberfläche; der Systemplan ist ein eigenes SVG.

FRAGENBANK 100 strukturierte JavaScript-Objekte	SPIELLOGIK Auswahl, Prüfung, Punkte und Kapitel	REACT-UI Rätsel, Navigation, Systemplan, Bericht	BROWSER HTML/CSS/JS + localStorage
---	---	--	--

DATENFLUSS

START	Der Spieler wählt Kurslauf oder Lernmodus.
AUSWAHL	<code>createRunQuestionIds()</code> mischt je Kapitel 3 oder 10 eindeutige IDs.
SPEICHERUNG	Modus, IDs, Fortschritt, XP, Integrität, Fehler und Hinweise landen unter <code>kernel-panic-progress-v2</code> im Browser.
VALIDIERUNG	Beim Laden entfernt <code>sanitizeProgress()</code> unbekannte oder widersprüchliche Werte.
DARSTELLUNG	React rendert die aktuelle Frage und den aus dem Zustand abgeleiteten Kapitelstatus.
SCOREBOARD	Erst am Abschluss lädt oder speichert die optionale „Hall of Tux“ einen anonymen Alias.
AUSFALLSICHERHEIT	Ist die API nicht erreichbar, bleiben Spiel, Fortschritt und Abschluss vollständig nutzbar.

WICHTIGE PROJEKTDATEN

```

app/                Einstieg und globales Design
components/         Spiel, Rätsel, Kapitelrail, Systemplan, Bericht
lib/question-bank.js 10 Kapitel × 10 Fragen
lib/game.js         Auswahl, Mischen, Speichern, Auswertung
server/             gehärtete Score-API und SQLite-Schema
treasure/           Seed der getrennten Fake-Datenbank
deploy/             Nginx- und Container-Konfiguration
lib/*.test.js       Fragenbank- und Logiktests
tests/              Browser- und visuelle Tests
docs/               Dokumentation und PDF-Quelle
public/             PDF, ZIP und Favicon
scripts/serve-static.mjs lokaler Server für den Export

```

Lokal starten, prüfen und bauen

VORAUSSETZUNGEN

Für das Spiel werden Node.js 20 oder neuer, npm und ein aktueller Browser benötigt. Die lokale Score-API braucht zusätzlich Python 3.13; ihr Produktionsbetrieb ist per Docker Compose dokumentiert. Das Repository kann von Codeberg geklont oder als ZIP geladen werden.

LOKAL AUSFÜHREN

```
git clone https://codeberg.org/andremadstop/lpic-kernel-panic.git
cd lpic-kernel-panic
npm install
npm run dev

# Browser:
http://localhost:3000/kernel-panic/
```

QUALITÄT UND EXPORT

```
npm audit
npm run check
npx playwright test

# npm run check erzeugt die Schatz-DB, testet die Score-API
# und führt Linter, Logiktests sowie statischen Build aus.
# Das fertige Webprojekt liegt danach in out/.
```

EINE FRAGE ERGÄNZEN

Neue Aufgaben werden in `lib/question-bank.js` einem Kapitel zugeordnet. Jede Frage enthält eine eindeutige ID, Typ, LPIC-Ziel, Aufgabenstellung, Lösung, Signal, Werkzeug, Entscheidung, Risiko, Hinweis und XP. Abhängig vom Typ kommen Auswahlmöglichkeiten, akzeptierte Terminalantworten oder Sequenzbausteine hinzu.

Der Banktest kontrolliert Anzahl, eindeutige IDs, Pflichtfelder, genau eine richtige Auswahlantwort sowie vollständige Terminal- und Sequenzlösungen. Eine Änderung ist erst fertig, wenn alle Prüfungen grün sind.

Automatisierte Abnahme

PRÜFERGEBNISSE DES ABGABESTANDS

100 / 100 FRAGEN VALIDE

6 / 6 BROWSERTESTS

BUILD GRÜN

0 VULNERABILITIES

PRÜFUNG	WAS KONTROLLIERT WIRD	ERGEBNIS
Fragenbank	Exakt 10 × 10, eindeutige IDs, Pflichtfelder und typspezifische Lösungen	grün
Spiel-Logik	Normalisierung, Mischen, 30/100-Auswahl, Punkte, Speicher-Sanitisierung und Kapitelbericht	grün
Komplettweg	Ein zufälliger 30-Fragen-Kurslauf wird automatisch bis zum Systembericht gelöst	grün
Lernmodus	Start mit allen 100 Fragen und 10 Fragen pro Kapitel	grün
Persistenz	Fragenauswahl bleibt nach Neuladen identisch	grün
Score-API	Validierung, serverseitige Neuberechnung, SQL-Constraints, Ranking und Idempotenz	grün
Mobil	390 × 844 Pixel ohne horizontalen Überlauf	grün
Auslieferung	PDF, ZIP, API-Health, Schatzdatei und Security-Header live vorhanden	grün
Abhängigkeiten	npm audit gegen die npm-Sicherheitsdatenbank	0 Befunde

DATENSCHUTZ- UND SICHERHEITSMODELL

- Keine Anmeldung, Cookies, Werbung, Analyse oder Telemetrie.
- Fortschritt liegt ausschließlich im `localStorage` des Geräts.
- Das Terminal ist eine Simulation; Eingaben werden nie als Betriebssystembefehl ausgeführt.
- Die private Score-DB speichert Alias und Ergebniswerte, aber keine Konten, E-Mails oder IP-Adressen.
- Die API läuft ohne Root, ohne Capabilities, schreibgeschützt, ressourcenlimitiert und ohne Internetroute.
- Nginx und API begrenzen Herkunft, Methode, Größe, Rate und Laufzeit; Scores werden serverseitig berechnet.
- Repository und ZIP enthalten keine Secrets, personenbezogenen Kursdaten oder internen Hosting-Metadaten.

BEKANNTE GRENZEN

Fortschritt synchronisiert nicht zwischen Geräten. Freitext wird nur gegen definierte Befehlsvarianten geprüft. Weil Fragen und Clientcode offen sind, ist die Bestenliste ein spielerischer Vergleich und kein manipulationssicherer Prüfungsnachweis.

Lernen soll sich nach Fortschritt anfühlen

Ein reines Quiz quitiert meist nur „richtig“ oder „falsch“. Dieses Projekt setzt zusätzlich auf erkennbare Etappen: Nach jedem Kapitel fährt ein neues Systemmodul hoch. Eine kurze Animation nennt das erreichte Kapitelziel, fasst den Lerngewinn zusammen und zeigt drei konkrete Fähigkeiten. Nach Kapitel zehn startet das Gesamtsystem mit Abschlussanimation, Statistik und Ergebnisbericht.



Kapitel geschafft: sichtbarer Erfolg, Lernzusammenfassung und nächster Schritt.

HALL OF TUX

Wer möchte, veröffentlicht am Ende einen frei gewählten Alias und seinen auf 1.000 Punkte normierten Score. Die API berechnet das Ergebnis aus den Einzelergebnissen neu. Klarnamen sind unnötig.

TREASURE HUNT

Fortgeschrittene können im Seitenquelltext eine Ops-Notiz und eine absichtlich öffentliche SQLite-Datei finden. SQL-Abfragen führen durch erfundene Wartungslogs bis zu einem kleinen Internet-Klassiker.

WICHTIGE TRENNUNG

Die Schatzdatei ist ein statisches Fake-Artefakt ohne echte Nutzer-, Score- oder Serverdaten. Die produktive SQLite-Datei liegt außerhalb des Webroots und ist ausschließlich über die eingeschränkte Score-API erreichbar. So bleibt der Witz lehrreich, ohne die Infrastruktur zu öffnen.

Deployment, Download und Open Source

PRODUKTIONSWEG

1 · CHECK Linter, Logik-, API- und Browsertests	2 · BUILD Webexport, PDF, ZIP und Schatz-DB	3 · DEPLOY Webpfad + isolierter API- Container	4 · VERIFY Health, Limits und kompletter Live-Weg
--	--	---	--

```
npm run check
rsync -a --delete out/ "$HOST:$WEBROOT/kernel-panic/"
ssh "$HOST" "cd \"$COMPOSE_DIR\" && docker compose up -d --build"
BASE_URL=https://andrestiebitz.de npx playwright test
```

Host, Webroot und Compose-Verzeichnis stehen als Variablen, weil die Zielumgebung nicht Teil der Veröffentlichung ist.

ÖFFENTLICHE ARTEFAKTE

SPIEL	https://andrestiebitz.de/kernel-panic/
PDF	.../kernel-panic/kernel-panic-projektarbeit.pdf
ZIP	.../kernel-panic/kernel-panic-lpic-projekt.zip
GIT	https://codeberg.org/andremadstop/lpic-kernel-panic
API-HEALTH	.../kernel-panic/api/health

OPEN-SOURCE-LIZENZ

Quellcode und eigene Projektttexte stehen unter der MIT-Lizenz, Copyright © 2026 André Stiebitz. Nutzung, Kopieren, Änderung und Weitergabe sind erlaubt, sofern Copyright- und Lizenzhinweis erhalten bleiben. Die Software wird ohne Gewährleistung bereitgestellt. Der vollständige rechtsverbindliche Text liegt in der Datei LICENSE im Repository und im ZIP.

BEWUSSTE AUSSCHLÜSSE

Die MIT-Freigabe umfasst ausschließlich die selbst entwickelte Anwendung und die selbst formulierten Projektttexte. Bereitgestellte Kursunterlagen, fremde Lösungsblätter und Prüfungsbilder wurden nicht veröffentlicht und sind nicht Teil der Lizenz.

Präsentation, Fazit und Quellen

VORSCHLAG FÜR EINE KURZE LIVE-PRÄSENTATION

1. **Problem und Idee:** LPIC-Befehle in eine zusammenhängende Systemrettung einbetten.
2. **Zwei Modi:** 30-Fragen-Kurslauf und vollständigen 100-Fragen-Lernmodus zeigen.
3. **Drei Aufgaben:** je eine Auswahl-, Terminal- und Sequenzfrage vorführen.
4. **Lernmodell:** Signal → Werkzeug → Entscheidung → Risiko erklären.
5. **Systemplan:** sichtbaren Aufbau und Kapitel-Erfolgsmoment zeigen.
6. **Technik:** statische React-App, localStorage und getrennte Score-API nennen.
7. **Community:** Hall of Tux zeigen und die SQLite-Schatzsuche nur andeuten.
8. **Qualität:** Komplettweg, Fragenbank- und API-Tests zeigen.
9. **Abgabe:** Live-Link, PDF, ZIP, Codeberg und MIT-Lizenz teilen.

EIN-SATZ-ERKLÄRUNG

„Ich wollte die LPIC-Befehle nicht nur als Quiz abfragen, sondern als zusammenhängende Systemrettung, bei der man aus einem Signal das passende Werkzeug und eine sichere Entscheidung ableitet.“

FAZIT UND MÖGLICHE WEITERENTWICKLUNG

Das Projekt verbindet Linux-Fachinhalte, Webentwicklung und Qualitätssicherung in einer überschaubaren, direkt nutzbaren Abgabe. Sinnvolle spätere Erweiterungen wären ein frei wählbares Kapitel, barriereärmere Tastatursteuerung, weitere Erklärvarianten oder ein Export des persönlichen Ergebnisberichts. Die jetzige Fassung ist mit Lernpfad, Erfolgsmomenten, Bestenliste und Schatzsuche abgeschlossen.

QUELLENBASIS

- Linux Professional Institute: LPIC-1 Exam 101 and 102 Objectives, Version 5.0 — <https://www.lpi.org/our-certifications/exam-101-102-objectives/>
- Bereitgestellte Unterlagen des Linux-Kurses zur Kontrolle der Themenabdeckung; keine Originaltexte oder Prüfungsbilder übernommen.
- Next.js-Dokumentation — <https://nextjs.org/docs>
- React-Dokumentation — <https://react.dev/>
- MIT-Lizenztext im Projekt — Datei LICENSE.

Live: <https://andrestiebitz.de/kernel-panic/>

Download: <https://andrestiebitz.de/kernel-panic/kernel-panic-lpic-projekt.zip>

Quellcode: <https://codeberg.org/andremadstop/lpic-kernel-panic>

Lizenz: MIT · Copyright © 2026 André Stiebitz