IT資格道場

LPIC-1 101試験 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

LPI公式 LPIC-1 Exam 101 Objectives Version 5.0 準拠
2026年7月版

本書のご利用にあたって

- 本書は「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルであっても著作物です。本書の転載・改変しての再配布・二次販売を禁じます。
- 本書を AI モデルの学習データとして利用すること(学習目的の入力・データセット化を含む)を禁じます。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

不正な配布を見つけた場合は info@shikaku-doujou.com までご連絡ください。

目次

この教材群の使い方(3点セット)

種類	役割
① 学習用テキスト(本書)	これだけで問題が解けるようになる本編
② 頻出度別ノート	テキストを読んだら次はこれ。頻出順に絞った問題と解説で「解ける」に橋渡し。試験直前の最終チェックにも
③ 暗記用ノート	覚えるべき要点だけを一問一答に凝縮。空き時間の反復と用語の再確認用

使い方: ① 学習用を読む → ② 頻出度別で解き方を掴む → ③ 問題演習で腕試し → ④ 頻出度別に戻って再確認(試験直前もここ)。暗記用は空き時間や用語の再確認に。

このテキストの位置づけ: 本書は①の学習用テキストです。まずはここを通読し、これだけで問題が解けるようになることを目標にしてください。

はじめに —— この教材の使い方

このテキストは、Linux Professional Institute（LPI）が実施する **LPIC-1 101 試験（試験コード 101-500）** に合格することを目的に作られています。

101試験が問うのは、Linux を「使う側」の土台です。電源を入れてからログインプロンプトが出るまでに何が起きているか（Topic 101）、ディスクを切ってパッケージを入れる（Topic 102）、コマンドラインでファイルとプロセスとテキストを自在に扱う（Topic 103）、ファイルシステムを作って守ってマウントし、パーミッションとリンクと配置場所を正しく扱う（Topic 104）—— この4つの Topic に、公式の到達目標 23 本が並びます。

覚え方は3点に絞ります。

1. **似ているコマンドの役割の違い**（例: `df` と `du`、`fsck` と `tune2fs`、`grep -E` と `grep -F`）
2. **書式とオプションを「打てる」まで**（本試験は択一に加えて、コマンド名やパスを直接入力する記述入力があります。本書は主要コマンドに書式行と代表オプション表を必ず置きます）
3. **設定が「今だけ」効くのか「次回起動後も」効くのか**（例: `mount` と `/etc/fstab`、`export` とプロファイル）

試験の基本情報

項目	内容
試験名	LPIC-1 101試験（**101-500**）
運営	Linux Professional Institute（LPI）
出題範囲のバージョン	**Version 5.0**
認定要件	101試験と102試験の両方に合格すると LPIC-1 に認定される。前提資格は不要
問題数	**60問**
出題形式	**択一（multiple-choice）と記述入力（fill-in-the-blank）**
試験時間	**90分**
合格ライン	**200〜800 のスケールで 500点**。必要な正答数は出題の組み合わせによって変わる

試験の設計図 —— 4つの Topic と weight

公式は到達目標ごとに **weight**（重み）を公表しており、「weight n の到達目標は本番で n 問出る」と明記しています。合計は 60 です。本書の章立てはこの Topic の順番どおりで、章の分量も weight に比例させています。

Topic	内容	weight	本書の章
101	システムアーキテクチャ	8	第1章
102	Linux のインストールとパッケージ管理	12	第2章
103	GNU と Unix コマンド	26	第3章
104	デバイス、Linux ファイルシステム、ファイルシステム階層標準	14	第4章

4 ステップ学習法

1. 第1章から順に読む（Topic 103 が最大の山です。第3章は2回に分けて読んでも構いません）
2. 各節の「書式行」と「取り違えやすい対」の表を、手を動かして確かめる
3. 問題演習で腕試しをし、間違えた問題の該当節に戻る
4. 頻出度別ノートで直前の総仕上げ

第1章 システムアーキテクチャ(Topic 101・weight 8)

この章では、Linux が動いている土台 —— ハードウェアをどう認識し、電源投入から何を経て使える状態になり、その状態をどう切り替えて落とすのか —— を扱います。扱う objective は 101.1 ハードウェア設定の決定と設定(weight 2)、101.2 システムの起動(weight 3)、101.3 ランレベル / ブートターゲットの変更とシャットダウンまたは再起動(weight 3)の 3 つです。

この章を貫く考え方はひとつです。**起動とは、担当者が次の担当者へバトンを渡していく連鎖**であり、どこで止まったかが分かれば何を直すかが逆算できます。用語には初出でひとこと説明を付けますので、前提知識は要りません。

101 試験は選択式に加えて**記述入力(fill-in-the-blank)**が出ます。コマンド名・オプション・ファイルパスは、読んだあとに**その綴りのまま手で打てる**ところまで持っていってください。本文では主要コマンドに書式行とオプション表を必ず添えています。

1-1 ハードウェア設定の決定と設定 [101.1]

この objective の狙いは「**目の前の機械に何がつながっていて、Linux はそれをどう見ているのか**を、コマンドで確かめられること」です。weight は 2 と小さいものの、`lspci` / `lsusb` / `lsmod` / `modprobe` の 4 語と、`/proc/` `/sys/` `/dev/` の 3 つのディレクトリは記述入力でそのまま問われます。

1-1-1 Linux から見たハードウェア —— デバイスファイルと `/dev/` [101.1]

Linux は、ハードウェアを**ファイルとして**扱います。ディスクもキーボードもプリンタも、`/dev/` の下に置かれた**デバイスファイル**(スペシャルファイル)を読み書きすることで操作できます。「すべてはファイルである」という Unix の設計思想が、いちばん目に見える形で現れている場所です。

デバイスファイルは 2 種類に分かれます。この対比は選択肢で入れ替えられます。

種別	記号	読み書きの単位	代表例
キャラクタデバイス	`c`	**1 文字ずつ**順に流す	端末(`/dev/tty`)・シリアルポート・ `/dev/random`
ブロックデバイス	`b`	**一定の大きさの塊(ブロック)単位**で、任意の位置へアクセスできる	ディスク(`/dev/sda`)・光学ドライブ

`ls -l /dev/` の先頭 1 文字が `b` ならブロック、`c` ならキャラクタです。**ディスクはブロック、端末はキャラクタ**という対応だけは必ず押さえてください。

代表的なデバイスファイルの名前も出題されます。

デバイスファイル	指すもの
`/dev/sda` ・ `/dev/sdb`	1 台目・2 台目の SATA / SCSI / USB ディスク**全体**
`/dev/sda1` ・ `/dev/sda2`	`/dev/sda` の**1 番目・2 番目のパーティション**

デバイスファイル	指すもの
`/dev/nvme0n1`・`/dev/nvme0n1p1`	NVMe SSD の 1 台目**全体**と、その 1 番目のパーティション
`/dev/sr0`	光学ドライブ
`/dev/null`	書き込んだものを捨てる。読むと即座に終端になる
`/dev/zero`	読むと `0` が無限に出てくる

NVMe だけ命名規則が違う点が引っかけになります。SATA ディスクは末尾に数字を付けるだけ(`sda` → `sda1`)ですが、NVMe は**パーティションを表す `p` が入り**ます(`nvme0n1` → `nvme0n1p1`)。「 `/dev/nvme0n11` 」のような選択肢は誤りです。

1-1-2 `/proc/` —— カーネルとプロセスの情報を見る窓 [101.1]

`/proc/` は、カーネルが**メモリ上に作り出している仮想ファイルシステム**(procfs)です。ディスク上に実体はありません。`cat` で読むと、その瞬間のカーネルの内部状態が文字列で返ってきます。

パス	内容
`/proc/cpuinfo`	CPU の型番・コア数・対応機能(`flags`)
`/proc/meminfo`	物理メモリとスワップの容量・使用量
`/proc/interrupts`	**IRQ(割り込み要求)番号**と、それを使っているデバイス
`/proc/ioports`	**I/O ポートアドレス**の割り当て
`/proc/dma`	**DMA チャネル**の割り当て

パス	内容
`/proc/devices`	登録済みのデバイスドライバとメジャー番号
`/proc/partitions`	カーネルが認識しているパーティション
`/proc/modules`	ロード済みカーネルモジュール(`lsmod` の元データ)
`/proc/cmdline`	**起動時にカーネルへ渡されたパラメータ**
`/proc/mounts`	現在マウントされているファイルシステム
`/proc/<数字>/`	**その PID のプロセス**に関する情報

`/proc/` の数字のディレクトリはプロセス ID ごとの情報で、名前が付いたものはシステム全体の情報、という切り分けを覚えてください。

「デバイスが使っているハードウェア資源を調べたい」という設問では、**IRQ なら `/proc/interrupts`、I/O ポートなら `/proc/ioports`、DMA なら `/proc/dma`** の三択が問われます。名前がそのまま内容なので、綴りを覚えれば取れる 1 問です。

1-1-3 `/sys/` —— sysfs [101.1]

`/sys/` (sysfs) も同じく仮想ファイルシステムですが、見せているものが違います。`/proc/` が「カーネルとプロセスの情報」を雑多に集めた場所であるのに対し、**`/sys/` はデバイス・ドライバ・バスの関係を階層構造で整理して見せる**ために後から作られました。

観点	/proc/	/sys/
主な対象	**プロセス**とカーネル全般の情報	**デバイス・ドライバ・バス**の情報

観点	/proc/	/sys/
構造	歴史的経緯で雑多	**デバイスの親子関係を反映した階層**
導入	古くからある	2.6 系カーネルで導入
使う側	人間・各種コマンド	**udev** が特に強く依存する

代表的なサブディレクトリは `/sys/block/`(ブロックデバイス)、`/sys/class/`(種類別)、`/sys/devices/`(物理的な接続関係)、`/sys/bus/`(バス別)です。**「デバイスの階層なら `/sys/`、プロセス情報なら `/proc/`」**という一言で選択肢を切れます。

`/sys/` の下のファイルには**書き込めるもの**もあり、電源管理やドライバのパラメータを実行中に変更できます。ただし再起動すると失われる点は `/proc/sys/` と同じです。

1-1-4 udev —— `/dev/` を動的に作る仕組み [101.1]

昔の Linux では、`/dev/` の下にありうるデバイスファイルを**あらかじめ全部作り置き**していました。現在は **udev** が、**カーネルからの通知を受けて、実際につながっている機器の分だけデバイスファイルを作る**方式になっています。USB メモリを挿すと `/dev/sdb` が現れ、抜くと消えるのは udev の働きです。

流れは次のとおりです。

1. 機器が接続され、カーネルがそれを検出する
2. カーネルが **uevent** という通知を出す
3. udev のデーモン(`systemd-udevd`)が受け取る
4. **`/sys/` の情報**と**ルールファイル**を突き合わせ、`/dev/` にデバイスファイルを作る

ルールファイルの置き場所は 2 か所で、**管理者が書くのは** `/etc/udev/rules.d/` です。

ディレクトリ	誰のもの
`/lib/udev/rules.d/`（`/usr/lib/udev/rules.d/`）	**パッケージが提供する既定のルール**。直接編集しない
`/etc/udev/rules.d/`	**管理者が置くルール**。同名なら管理者側が優先される

ルールを書き換えたあとは `udevadm control --reload-rules` で読み直させ、`udevadm monitor` で uevent の流れを観察できます。**「特定の USB メモリを常に同じ名前で見えるようにしたい」という要求の答えは udev ルール**であり、`/etc/fstab` だけでは実現できません。

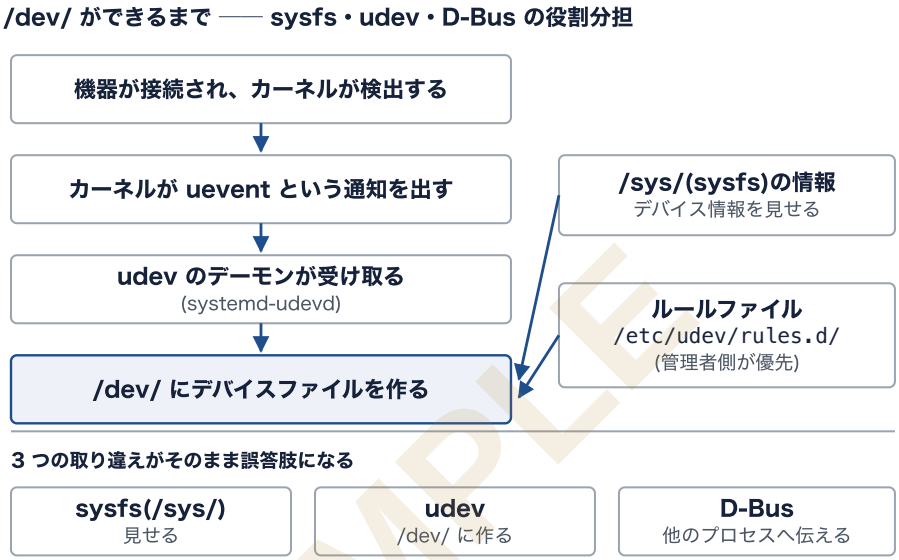
1-1-5 D-Bus ―― プロセス同士の連絡係 [101.1]

D-Bus は、**プロセス間通信(IPC)** の仕組みです。デバイスファイルを作るのが udev、その「機器がつながりました」という出来事をデスクトップ環境などの**他のプログラムへ知らせる**のが D-Bus、という役割分担になります。

名称	受け持つもの
sysfs(`/sys/`)	カーネルが持つデバイス情報を**見せる**場所
udev	`/dev/` にデバイスファイルを**作る**仕組み
D-Bus	出来事を**他のプロセスへ伝える**通信路

この 3 つは公式の到達目標に「conceptual understanding of sysfs, udev and dbus」と並べて挙げられており、**役割の取り違えがそのまま誤答肢**にな

ります。D-Bus はデバイスファイルを作りませんし、udev はプロセス間の汎用メッセージ配送を行いません。



D-Bus はデバイスファイルを作らず、udev はプロセス間の汎用メッセージ配送を行わない。

図 1-1　/dev/ ができるまで —— sysfs・udev・D-Bus の役割分担

D-Bus にはシステム全体で 1 本走る**システムバス**と、ログインセッションごとの**セッションバス**があります。各システムには `/etc/machine-id` に**マシン ID** という固有の識別子が置かれ、D-Bus もこれを使います(1-2 章末および Topic 102 の仮想化ゲストで再登場します)。

1-1-6 カーネルモジュール —— `lsmod` と `modprobe` [101.1]

カーネルモジュールとは、必要なときだけカーネルに読み込んで機能を足す部品(多くはデバイスドライバ)です。ドライバを全部カーネル本体に入れると巨大になるため、外付けにして必要な分だけ載せます。

`lsmod` —— ロード済みモジュールの一覧

書式　`lsmod`

`lsmod` は引数を取らず、`/proc/modules` の内容を読みやすく整形して表示するだけのコマンドです。出力の 3 列は次の意味を持ちます。

列	内容
`Module`	モジュール名
`Size`	メモリ上の大きさ(バイト)
`Used by`	**使用回数**と、**そのモジュールに依存しているモジュール名**

`lsmod` に出てくるのはロード済みのものだけです。「システムに存在するがロードされていないモジュールの一覧」も「モジュールファイルのパス」も表示しません —— ここが誤答肢の作り所です。ファイルのパスやライセンス・作者を知りたいときは `modinfo <名前>` を使います。

`modprobe` —— 依存関係を解決してロードする

書式　`modprobe [オプション] <モジュール名> [パラメータ=値 ...]`

オプション	意味
`-r`	モジュールを**アンロード**する(依存も考慮する)
`-a`	複数のモジュールをまとめてロードする

オプション	意味
`-n`	実際には実行せず、何をするかだけ表示する
`-v`	詳しく表示する
`-c`	現在の設定内容をすべて表示する
`-f`	バージョンの不一致を無視して強制する

例　`modprobe e1000e`　（ネットワークドライバをロードする）
例　`modprobe -r e1000e`　（アンロードする）
例　`modprobe -n -v vfat`　（何がロードされるかだけ確認する）

`modprobe` と `insmod` の違いが定番の出題です。

コマンド	指定するもの	依存関係
`modprobe <名前>`	**モジュール名**だけでよい	**自動で解決してロードする**
`insmod <ファイルパス>`	**`.ko` ファイルのフルパス**	**解決しない**。足りないと失敗する
`modprobe -r <名前>`	モジュール名	依存を考慮してアンロード
`rmmod <名前>`	モジュール名	依存を考慮しない

依存関係の情報は `depmod` が作る `modules.dep` に入っており、`modprobe` はこれを見て順番にロードします。**「名前だけ渡せば依存も一緒に入るのが `modprobe`、パス指定で依存を見ないのが `insmod`」**—— 実務でも試験でも通常使うのは `modprobe` です。

モジュールへのパラメータや別名を恒久的に設定する場所は `/etc/modprobe.d/` 配下のファイルです。ここに `blacklist <名前>` と書けば、そのモジュールの自動ロードを抑止できます。これはカーネル本体へ渡す起動パラメータ(1-2-5)とは別系統である点に注意してください。

1-1-7 接続機器を一覧する —— `lspci` と `lsusb` [101.1]

`lspci` —— PCI バスの機器を一覧する

書式　`lspci [オプション]`

オプション	意味
`-v` / `-vv` / `-vvv`	詳しく表示する(重ねるほど詳細)
`-k`	**その機器を使っているカーネルドライバ(モジュール)名**を表示する
`-n`	名前ではなく**数値の ID**(ベンダ ID:デバイス ID)で表示する
`-t`	バスの接続関係を**ツリー**で表示する
`-s <指定>`	特定のバス・スロットだけに絞る

例　`lspci`
例　`lspci -k`　　（どのドライバが使われているかを確認する）
例　`lspci -nn`　　（名前と数値 ID の両方を出す）

PCI は**マザーボードに内蔵・増設された機器**をつなぐバスです。ネットワークコントローラ、ビデオコントローラ、SATA コントローラ、サウンドなどが並びます。

`lsusb` —— USB 機器を一覧する

書式　`lsusb [オプション]`

オプション	意味
`-v`	詳しく表示する
`-t`	**USB の接続関係をツリー(階層)状に**表示する
`-s <bus>:<dev>`	特定の機器に絞る
`-d <ID>`	ベンダ ID:プロダクト ID で絞る

例　`lsusb`

例　`lsusb -t`　　（ハブとその配下の機器の階層を見る）

「PCI か USB か」で選択肢を切る素直な設問が多い領域です。内蔵のネットワークカードやビデオカードは `lspci`、あとから挿した USB メモリやキーボードは `lsusb`。両方に共通して `-v` が詳細、`-t` がツリーである点も覚えておくと迷いません。

同系統の情報源として、`lsblk`(ブロックデバイスの一覧)、`lscpu`(CPU の情報)、`lsdev`(資源の一覧)も出てくることがあります。ただし公式の到達目標が名指ししているのは `lspci` と `lsusb` の 2 つです。

1-1-8 USB デバイスを操作する [101.1]

USB は、**同じコネクタに何でもつなげる**ことを目指した規格です。試験では次の性質が問われます。

- **ホットプラグ**に対応する。電源を入れたまま抜き挿しできる
- **ハブ**で分岐でき、ツリー状に最大 127 台まで接続できる
- **USB クラス**(大容量記憶・HID・プリンタなど)ごとに汎用ドライバが用意されており、機器固有のドライバがなくても動くものが多い

Linux 側では、USB 大容量記憶装置(USB メモリや外付け HDD)は `usb-storage` モジュールを介して `/dev/sdb` のような**ブロックデバイス**として見えます。つまり内蔵ディスクとまったく同じ扱いで `mount` できます。キーボードやマウスは **HID(Human Interface Device)クラス**として扱われます。

USB 機器を扱う道具は次のとおりです。

道具	用途
`lsusb`	接続されている USB 機器の一覧
`usb-devices`	接続機器の詳細をテキストで一覧する
`/proc/bus/usb/`(古い環境)・`/sys/bus/usb/`	USB の情報を持つ仮想ファイル
`modprobe usb-storage`	USB 大容量記憶のドライバをロードする
udev ルール	挿したときの名前や権限を決める

「USB メモリを挿したのに `/dev/` に何も出ない」という場面の調査手順は、**`lsusb` で機器そのものが見えているかを確認 → `dmesg` で認識時のメッセージを確認 → `lsmod` でドライバがロードされているかを確認**、という順になります。この 3 段の順番はそのまま設問になります。

1-1-9 内蔵周辺機器の有効化と無効化 [101.1]

マザーボードに載っているシリアルポート・パラレルポート・オンボード LAN・オンボードサウンドなどを**内蔵周辺機器(integrated peripherals)**と呼びます。これらを使うかどうかは、次の 2 か所で決められます。

場所	できること
BIOS / UEFI の設定画面	機器そのものを**有効化・無効化**する。無効にすると OS からは存在しないものとして扱われる
Linux 側(モジュールの blacklist・udev ルール)	機器はあるが**ドライバを読み込ませない**、あるいは使い方を変える

BIOS / UEFI で無効化した機器は、Linux 側でどんな設定をしても使えません。逆に、機器は生きているのにドライバが載っていないだけなら `modprobe` で解決します。「オンボードのシリアルポートが `lspci` にも出てこない」なら、疑うのはまずファームウェアの設定です。

1-1-10 大容量記憶装置(mass storage)の種類 [101.1]

公式の到達目標に「Differentiate between the various types of mass storage devices」とあり、**種類ごとの性質の違い**が問われます。

種類	仕組み	性質
HDD	磁気ディスクを**回転**させ、ヘッドで読み書きする	容量あたりの単価が安い。**駆動部があるので衝撃に弱く、ランダムアクセスが遅い**
SSD	フラッシュメモリ	**駆動部が無く高速・静か・衝撃に強い。**書き換え回数に上限がある

種類	仕組み	性質
光学メディア	CD・DVD・Blu-ray	読み出し中心。配布や長期保管に使う
磁気テープ	テープに順次記録	**順次アクセスのみ**。大容量・長期保管のバックアップ向け

接続方式(インタフェース)も区別が必要です。

方式	特徴	Linux でのデバイス名
SATA	内蔵ディスクの標準的な接続方式	`/dev/sda` 系
SCSI / SAS	サーバー向け。SATA と同じドライバ層で扱われる	`/dev/sda` 系
USB	外付け。内部的には SCSI として扱われる	`/dev/sdb` 系
NVMe	**PCI Express に直結**する SSD 向けの規格。SATA より速い	**`/dev/nvme0n1` 系**

SATA・SCSI・USB がまとめて `sd` 系の名前になるのに対し、NVMe だけ `nvme` 系になります。SSD だから必ず `nvme` になるわけではなく、**SATA 接続の SSD は `/dev/sda`** です —— ここが引っかけです。

SSD には **TRIM** という仕組みがあり、削除された領域を SSD 側に知らせて性能低下を防ぎます。Linux では `fstrim` コマンドやマウントオプション `discard` で扱います。

1-1-11 ハードウェア資源を確かめる手順 [101.1]

ここまでの道具を、調べたいものから引ける形にまとめます。

知りたいこと	見る場所
PCI 機器の一覧	`lspci`
USB 機器の一覧	`lsusb`
ロード済みモジュール	`lsmod` (元データは `/proc/modules`)
モジュールのロード	`modprobe`
IRQ の割り当て	`/proc/interrupts`
I/O ポートの割り当て	`/proc/ioports`
DMA チャネル	`/proc/dma`
CPU の機能	`/proc/cpuinfo`
デバイスの階層・ドライバの結び付き	`/sys/`
デバイスファイルの実体	`/dev/`
認識時のカーネルメッセージ	`dmesg` (1-2-7)

「`/proc/` `/sys/` `/dev/` の 3 つを取り違えない」ことが、この objective で最も得点に直結します。**`/dev/` は操作するための入り口、`/sys/` は構造を見る窓、`/proc/` は状態を見る窓**、と役割で覚えてください。

1-2 システムの起動 [101.2]

この objective は、**電源投入から起動完了までの順番**と、**その各段階でどこに介入できるか**、そして**起動時のログをどう読むか**を問います。weight は 3 で、

章の中でも重い部類です。

1-2-1 起動の全体像 —— 5 つのバトン [101.2]

典型的な x86 系サーバーでは、次の順に進みます。

1. **電源投入 → ファームウェア(BIOS または UEFI)が起動**し、ハードウェアの自己診断を行う
2. ファームウェアが**ブートローダー(bootloader)**を読み込んで実行する
3. ブートローダーが**カーネル(kernel)**と **initramfs** をメモリへ読み込み、カーネルへ制御を渡す
4. カーネルが起動し、**initramfs を足場にして本来のルートファイルシステムをマウント**する
5. カーネルが**最初のプロセス(init)**を起動する。現在は多くの環境で **systemd** が PID 1 になる

この 5 段の順番はそのまま出題されます。とくに迷いやすいのが initramfs の位置です。**initramfs はカーネルが読み込んで使うもの**なので、カーネルより前ではなく「カーネルと一緒にメモリへ載り、カーネルが使う」と覚えてください。

この 5 段の順番はそのまま出題される。initramfs はカーネルが読み込んで使うもの ―― カーネルより前ではなく「カーネルと一緒にメモリへ載る」と覚える。

図 1-2　起動の 5 つのバトンと、止まった段階の見分け

各段階で「どこまで進んだか」を見分ける目印も対で覚えます。

症状	止まっている段階
画面に何も出ない・メーカーのロゴで止まる	ファームウェア(BIOS / UEFI)
ブートメニューが出ない・`grub rescue>` が出る	ブートローダー
カーネルのメッセージが流れた後に kernel panic	**カーネルまたは initramfs**(ルートをマウントできていない)
サービスの起動メッセージの途中で止まる	**init(systemd)以降**

1-2-2 BIOS と UEFI [101.2]

ファームウェアは、マザーボード上の ROM に書かれている起動用のプログラムです。2 世代あり、その差が問われます。

観点	BIOS(レガシー)	UEFI
ブートローダーの置き場所	**ディスク先頭の MBR**(512 バイト)	**ESP(EFI System Partition)上のファイル**
対応するパーティション方式	主に MBR	**GPT に対応**(MBR も可)
ディスク容量の上限	**約 2 TB**	2 TB を大きく超えて扱える
起動の仕組み	MBR のコードを読んで実行	**ファイルシステムを理解し、`.efi` ファイルを実行できる**
セキュリティ	無し	**Secure Boot**(署名されていないブートローダーの実行を拒否する)
設定の保存先	CMOS	**NVRAM**(`efibootmgr` で操作する)

Secure Boot は UEFI の機能であって BIOS の機能ではありません —— 定番のひっかけです。もうひとつ、**UEFI は自分でファイルシステム(FAT)を読めるので、512 バイトに収まる小さなコードを置く必要がありません。**この「ファイルを直接読める」ことが、MBR の 2 TB 制限や 4 区画制限から解放された理由になっています。

ESP については Topic 102(102.1・102.2)で詳しく扱いますが、ここでは**ESP は FAT 系(vfat)でフォーマットされ、多くの環境で `/boot/efi` にマウントされる**という 1 点を押さえてください。

UEFI 環境でファームウェアの起動順を操作するコマンドが `efibootmgr` です。`efibootmgr -v` で登録済みのブートエントリを一覧できます。

1-2-3 ブートローダー(bootloader) [101.2]

ブートローダーは、「どのカーネルを、どんなオプションで起動するか」を決めて実行する小さなプログラムです。Linux では **GRUB(GRand Unified Bootloader)** が事実上の標準で、GRUB Legacy と GRUB 2 の 2 世代があります。設定ファイルとインストール方法は Topic 102(102.2)で扱います。

ここで押さえるのは**ブートローダーに対して起動時に何ができるか**です。

操作	何が起きるか
メニューでエントリを選ぶ	起動するカーネル(バージョン)を選べる
`e` **キーを押す**	**そのエントリの内容を一時的に編集**できる。カーネルパラメータをその場で追記できる
`c` **キーを押す**	**GRUB のコマンドラインに入る**。`ls` や `set root=` などを手で打てる
編集後に `Ctrl+X`（または `F10`）	編集した内容で起動する

`e` で加えた変更はその 1 回の起動にだけ効き、次回には引き継がれません。恒久的に変えたい場合は設定ファイル側を直します(102.2)。「一時的に試すならメニュー編集、恒久化するなら設定ファイル+再生成」という使い分けが、そのまま設問になります。

1-2-4 カーネル(kernel)と initramfs [101.2]

カーネルは OS の中核で、CPU・メモリ・デバイスを配分する管理者です。ブートローダーがメモリへ載せると、カーネルは自分自身を展開し、検出したハー

ドウェアを初期化していきます。

initramfs(initial RAM filesystem。古い形式は **initrd**)は、**一時的な小さなルートファイルシステム**です。なぜ必要かというと、本来のルートファイルシステムが LVM の上や RAID の上、暗号化された領域、あるいは特殊なストレージドライバを要する装置の上にあると、**カーネル単体ではそれをマウントできない**からです。

そこで、必要なモジュールと最小限のコマンドを詰めた「道具箱」を先にメモリへ置き、そこから本来のルートをマウントし、成功したら役目を終えて退場します。

用語	内容
initrd	古い形式。**ブロックデバイスのイメージ**として扱われる
initramfs	現在の形式。**cpio アーカイブを gzip 等で圧縮したもの**をメモリ上に展開する
置き場所	`/boot/initramfs-<バージョン>.img` (Red Hat 系)・`/boot/initrd.img-<バージョン>` (Debian 系)
作り直すコマンド	`dracut` (Red Hat 系)・`update-initramfs` (Debian 系)・`mkinitramfs`

「initramfs はルートをマウントするための踏み台であり、マウントできたら消える」という役割が理解できていれば、`kernel panic - not syncing: VFS: Unable to mount root fs` というメッセージを見たときに initramfs とルートの指定を疑えます。

1-2-5 起動時にカーネルへ渡すパラメータ [101.2]

ブートローダーは、カーネルへ**カーネルパラメータ**(起動オプション)を渡せます。これが「boot loader に共通のコマンドを与え、カーネルにオプションを渡す」という到達目標の実体です。

パラメータ	効果
`root=/dev/sda1` ・ `root=UUID=...`	**ルートファイルシステムの場所**を指定する
`ro` / `rw`	ルートを読み取り専用 / 読み書き可能でマウントする
`single` または `1`	**シングルユーザーモード**で起動する
`systemd.unit=rescue.target`	systemd 環境で**起動先のターゲットを指定**する
`init=/bin/bash`	PID 1 として systemd ではなく**シェルを起動する**(パスワードを忘れた際の復旧など)
`quiet`	起動メッセージを抑制する
`nomodeset`	グラフィックのモード設定を無効にする(画面が映らないときの回避)
`mem=2G`	カーネルが使うメモリ量を制限する
`selinux=0`	SELinux を無効にする

現在動いているシステムに渡されたパラメータは `/proc/cmdline` で確認できます。「今の起動でどんなオプションが使われたか」を問われたらこのファイルです。

シングルユーザーモードへ入る方法は、**SysVinit 系なら `single` または `1`、systemd 系なら `systemd.unit=rescue.target`** という 2 系統がある点に注意

してください。両方を選択肢に混ぜてくる出題があります。

1-2-6 init の三世代 —— SysVinit・Upstart・systemd [101.2]

カーネルが最後に起動する**最初のプロセス**を **init** と呼びます。PID は必ず **1** です。この init には歴史的に 3 つの実装があり、公式の到達目標も「Understanding of SysVinit and systemd」「Awareness of Upstart」と、**systemd と SysVinit は理解、Upstart は存在を知っていること**という差を付けています。

	SysVinit	Upstart	systemd
起動のしかた	起動スクリプトを**番号順に逐次実行**	**イベント駆動**(あることが起きたら次を起動)	**依存関係を解決して並列起動**
管理単位	**ランレベル**	ジョブ	**ユニット**と**ターゲット**
設定ファイル	`/etc/inittab`、`/etc/init.d/` の各スクリプト	`/etc/init/` 配下の `.conf`	**ユニットファイル**(`.service` など)
主なコマンド	`init`・`telinit`・`service`・`chkconfig`	`initctl`	`systemctl`
位置づけ	伝統的な方式	**過渡期の方式。現在はほぼ使われない**	現在の主流

「逐次実行の SysVinit、イベント駆動の Upstart、並列起動の systemd」という 3 語の対応で覚えます。「systemd は起動スクリプトを順番に実行する」という選択肢は誤りで、**並列起動によって起動時間が短くなる**のが systemd の売りです。

Upstart は Ubuntu が採用していたことで広まりましたが、現在の主要ディストリビューションはいずれも systemd へ移行しています。試験では**名前と「イベント駆動である」ことが分かれば十分**です。

1-2-7 `dmesg` —— カーネルのリングバッファを読む [101.2]

カーネルは起動時から、検出したハードウェアやエラーのメッセージを**カーネルリングバッファ**という固定長のメモリ領域に書き続けます。これを表示するのが `dmesg` です。

書式　`dmesg [オプション]`

オプション	意味
`-H`	ページャーを使って読みやすく表示する
`-T`	**時刻を人間が読める形式で表示する**(既定は起動からの経過秒数)
`-w`	**新しいメッセージを待ち受けて表示し続ける**(追跡)
`-l <レベル>`	`err`・`warn` などの**深刻度で絞り込む**
`-f <facility>`	facility で絞り込む
`-c`	表示したあと**バッファを消去する**
`-C`	表示せずにバッファを消去する
`-k`	カーネルのメッセージだけ
`-n <レベル>`	コンソールへ出力するレベルを設定する

例　`dmesg | less`

例　`dmesg -T | grep -i usb`　　（USB 機器の認識状況を時刻付きで見る）

例　`dmesg -l err,warn`　　（エラーと警告だけを見る）

例　`dmesg -w`　　（挿抜を試しながら流れを見る）

`dmesg` はリングバッファなので、古いメッセージは新しいものに押し出されて消えます。起動から時間が経ったシステムでは、起動時のメッセージがもう残っていないことがあります。この性質は「なぜ起動時のログが `dmesg` に出てこないのか」という設問の答えになります。

同じ内容は多くの環境で `/var/log/dmesg` にも保存されます。恒久的に残っているのはこちらのファイルの側です。

1-2-8 `journalctl` —— systemd のジャーナルを読む [101.2]

systemd を採用した環境には、従来の syslog とは別に **journal(ジャーナル)** というログの仕組みがあります。テキストではなく**バイナリ形式**で記録されるため、`cat` では読めず、専用コマンド `journalctl` を使います。

書式　`journalctl [オプション] [一致条件]`

オプション	意味
`-b`	**今回の起動以降**のログ。`-b -1` で**1 つ前の起動時**のログ
`-k`	**カーネルのメッセージだけ**（`dmesg` に相当）
`-u <ユニット>`	サービス(ユニット)を指定して絞り込む

オプション	意味
`-f`	`tail -f` のように**リアルタイムで追跡**する
`-p <レベル>`	**深刻度で絞る**(`-p err` なら err 以上)
`-n <件数>`	直近の指定件数だけ表示する
`--since` / `--until`	期間で絞る(`--since "1 hour ago"` など)
`--no-pager`	ページャーを使わずそのまま出力する
`--list-boots`	記録されている起動の一覧を表示する
`--disk-usage`	ジャーナルが使っている容量を表示する

```
例  journalctl -b                 (今回の起動のログを全部見る)
例  journalctl -b -1 -p err       (前回の起動でのエラーだけ見る)
例  journalctl -k -b              (今回の起動のカーネルメッセージ)
例  journalctl -u sshd.service    (sshd のログだけ見る)
```

起動時のトラブルを追うときの中心は `journalctl -b` と `journalctl -b -1` です。`dmesg` と違ってジャーナルは**前回以前の起動のログも残せる**のが強みで、「再起動してしまったので `dmesg` にはもう残っていない」という場面を救えます。

ただし既定では**メモリ上(`/run/log/journal/`)にしか保存されず、再起動で消える**設定の環境があります。**`/var/log/journal/` ディレクトリを作る**か、`/etc/systemd/journald.conf` で `Storage=persistent` を指定すると永続化されます。「前回の起動のログを見ようとしたら出てこない」の答えはこれです。

1-2-9 ログファイルで起動イベントを確認する [101.2]

到達目標の「Check boot events in the log files」に対応する部分です。見る場所は 3 つあり、**それぞれ残り方が違う**点が問われます。

見る場所	内容	残り方
`dmesg`	カーネルリングバッファ	**メモリ上。古いものから消える。再起動で失われる**
`/var/log/dmesg`	起動時のカーネルメッセージを保存したファイル	ファイルとして残る
`journalctl`	systemd のジャーナル	既定はメモリ、`/var/log/journal/` を作れば**再起動をまたいで残る**
`/var/log/messages`（Red Hat 系）・`/var/log/syslog`（Debian 系）	syslog が受け取った一般的なログ	ファイルとして残る
`/var/log/boot.log`	起動時のサービス開始メッセージ	ファイルとして残る

Red Hat 系は `/var/log/messages`、Debian 系は `/var/log/syslog` という対応は、系統を問う設問でそのまま使われます。

1-3 ランレベル / ブートターゲットの変更とシャットダウン・再起動 [101.3]

この objective は、**起動した後のシステムの「状態」をどう切り替え、どう安全に落とすか**を問います。SysVinit のランレベルと systemd のターゲットを**両**

方扱い、さらに「切り替える前に利用者へ知らせる」「プロセスを正しく終わらせる」という運用面まで含みます。weight は 3 です。

1-3-1 ランレベル —— SysVinit の状態 [101.3]

ランレベルは、SysVinit におけるシステムの動作状態を 0 から 6 の数字で表したものです。

ランレベル	意味
0	**システム停止(halt)**
1(または `s`・`S`)	**シングルユーザーモード**。管理者だけが使える保守用の状態
2	マルチユーザー(ディストリビューションにより差がある。Debian 系では GUI も含む)
3	**マルチユーザー + ネットワーク(CUI)**
4	未使用(管理者が自由に定義してよい)
5	**マルチユーザー + GUI**
6	**再起動(reboot)**

必ず覚えるのは 0 = 停止、1 = シングルユーザー、3 = CUI、5 = GUI、6 = 再起動の 5 つです。とくに**0 と 6 を既定のランレベルにしてはいけない**理由(起動した瞬間に落ちる・無限に再起動する)は理屈で問われます。

現在のランレベルは `runlevel` コマンドで確認できます。出力は「前のランレベル 現在のランレベル」の 2 語で、前がなければ `N`(none)と表示されます。systemd 環境でも互換のために `runlevel` が動く場合があります。

1-3-2 `/etc/inittab` —— 既定のランレベルを決めるファイル [101.3]

SysVinit では、`/etc/inittab` が init の設定ファイルです。行の書式は 4 つのフィールドをコロンで区切ったものです。

書式　`<ID>:<ランレベル>:<アクション>:<実行するコマンド>`

フィールド	内容
ID	その行を識別する 1 〜 4 文字の名前
ランレベル	この行を適用するランレベル(複数可。`235` のように並べる)
アクション	いつ実行するか(`initdefault`・`sysinit`・`wait`・`respawn`・`ctrlaltdel` など)
コマンド	実行する内容

最重要は `initdefault` の行です。

例　`id:3:initdefault:`　　(既定のランレベルを 3 にする)

例　`id:5:initdefault:`　　(既定のランレベルを 5 にする)

`initdefault` の行には実行するコマンドを書きません(4 つ目のフィールドは空)。この行のランレベル欄を書き換えることが、SysVinit における「**既定のランレベルの設定**」そのものです。

主なアクションの意味も出ます。

アクション	意味
initdefault	**既定のランレベルを指定する**
`sysinit`	起動時に、ランレベルに関わらず最初に一度実行する
`wait`	指定ランレベルへ移行したときに一度実行し、**終了を待つ**
`once`	一度実行するが終了を待たない
respawn	**プロセスが終了したら自動的に再起動する**(ログインプロンプトなど)
`ctrlaltdel`	`Ctrl+Alt+Del` が押されたときの動作

respawn は「落ちたら起こし直す」、これがログイン用の `getty` を常に生かしておくために使われます。

なお、**systemd を採用した環境では `/etc/inittab` は使われません**。ファイルが残っていても「このファイルは使われていない」という趣旨のコメントだけが書かれていることがあります。systemd 環境で既定の状態を変えるのは `systemctl set-default` です(1-3-5)。

1-3-3 `/etc/init.d/` と起動スクリプト [101.3]

SysVinit では、サービスごとの起動スクリプトを `/etc/init.d/` に置きます。スクリプトは `start` / `stop` / `restart` / `status` といった引数を受け取る形で書かれています。

例　`/etc/init.d/sshd start`

例　`/etc/init.d/sshd status`

例　`service sshd restart`　　（同じことを `service` コマンド経由で行う）

どのランレベルでどのサービスを起動するかは、`/etc/rc<数字>.d/` ディレクトリ(例: `/etc/rc3.d/`)に置かれた**シンボリックリンク**で決まります。リンク名の先頭 1 文字と番号に意味があります。

リンク名の例	意味
`S20sshd`	そのランレベルへ入るときに **start(開始)** する。**番号の小さい順**に実行
`K80sshd`	そのランレベルから出るときに **kill(停止)** する。番号の小さい順に実行

`S` **が start、**`K` **が kill、数字は実行順**。これが SysVinit の「逐次実行」の正体です。リンクを手で作らずに管理するコマンドが `chkconfig`(Red Hat 系)と `update-rc.d`(Debian 系)です。

`/etc/init.d/` と `/etc/rc<数字>.d/` の役割の違いは頻出です。**実体のスクリプトは** `/etc/init.d/`**、ランレベルごとの起動順の指定は** `/etc/rc<数字>.d/` **のリンク**、と覚えてください。

1-3-4 `init` と `telinit` —— 実行中にランレベルを変える [101.3]

書式　`init <ランレベル>`

書式　`telinit <ランレベル>`

`init` と `telinit` は、実行中のランレベルを切り替えるという点で同じ働きをします。`telinit` は init に対して指示を送るための名前で、多くの環境では `init` へのシンボリックリンクです。

コマンド	動作
`init 3` / `telinit 3`	ランレベル 3(CUI)へ切り替える
init 1 / **telinit 1**	**シングルユーザーモードへ切り替える**
init 0	**システムを停止する**
init 6	**システムを再起動する**
telinit q（`Q`）	**`/etc/inittab` を読み直させる**。ランレベルは変えない

`init 0` が停止、`init 6` が再起動は、記述入力でもそのまま問われます。`0` と `6` を取り違えないでください。

`telinit q` は「設定ファイルを読み直すだけ」で、ランレベルの移行は起きません。`/etc/inittab` を編集した後に反映させる手段として問われます。

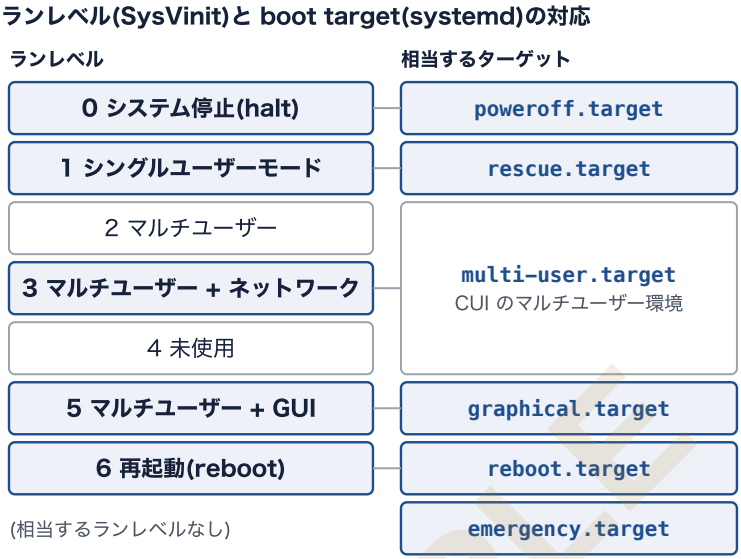
`init` は PID 1 のプロセス名でもあり、ランレベル変更コマンドの名前でもあります。**同じ名前で 2 つの顔がある**点は、設問文を読むときに意識してください。

1-3-5 systemd のターゲットと `systemctl` [101.3]

systemd では、ランレベルにあたる概念を**ターゲット(boot target)**と呼びます。ターゲットは「システムがどの状態まで立ち上がるか」を表すユニットの集まりです。

ターゲット	相当するランレベル	内容
`poweroff.target`	0	停止
`rescue.target`	**1**	**ローカルファイルシステムをマウントしたうえで単一のシェルを起動**(シングルユーザー相当)
`multi-user.target`	**2・3・4**	**CUI のマルチユーザー環境**
`graphical.target`	**5**	**GUI 付きのマルチユーザー環境**
`reboot.target`	6	再起動
`emergency.target`	——	**マウント処理をほぼ行わない最小構成のシェル**

「3 = `multi-user.target` (CUI)、5 = `graphical.target` (GUI)」は必ず覚えます。もうひとつ頻出なのが **`rescue.target` と `emergency.target` の違い**で、`rescue` はローカルファイルシステムをマウント**してから**シェルを出し、`emergency` はそれすら行いません —— **「より深刻なほうが emergency」**です。



rescue はローカルファイルシステムをマウントしてからシェルを出し、emergency はそれすら行わない——「より深刻なほうが emergency」。3 = CUI、5 = GUI は必ず覚える。

図 1-3　ランレベル(SysVinit)と boot target(systemd)の対応

互換のため、`runlevel3.target` のような名前が `multi-user.target` へのシンボリックリンクとして用意されている環境もあります。

`systemctl` によるターゲットの操作は次の 3 つです。

書式　`systemctl get-default`

書式　`systemctl set-default <ターゲット>`

書式　`systemctl isolate <ターゲット>`

コマンド	効く時点	内容
systemctl get-default	——	**現在の既定ターゲットを表示する**

コマンド	効く時点	内容
`systemctl set-default <target>`	**次回起動から**	**既定ターゲットを恒久的に変更する**
`systemctl isolate <target>`	**今すぐ**	**再起動せずに指定ターゲットへ切り替える**

```
例  systemctl get-default
例  systemctl set-default multi-user.target     （次回から CUI で起動する）
例  systemctl isolate rescue.target             （今すぐレスキューへ移る）
```

`set-default` は次回から、`isolate` は今すぐ。「システムを再起動せずにシングルユーザー相当へ切り替えたい」という設問の答えは `systemctl isolate rescue.target` です。逆に「次回の起動から GUI をやめたい」なら `systemctl set-default multi-user.target` です。

`systemctl set-default` は、内部的には `/etc/systemd/system/default.target` というシンボリックリンクを張り替えています。この仕組みを知っていると、「既定のターゲットはどこで決まるか」という設問に答えられます。

サービスそのものの操作も、ターゲットの理解と対で問われます。

コマンド	効く時点	内容
`systemctl start <unit>`	**今すぐ**	起動する。次回起動時の設定は変えない
`systemctl stop <unit>`	今すぐ	停止する。自動起動の設定は変えない

コマンド	効く時点	内容
`systemctl enable <unit>`	**次回起動から**	自動起動を有効にする。**今は起動しない**
`systemctl disable <unit>`	次回起動から	自動起動を無効にする。**今は止まらない**
`systemctl enable --now <unit>`	今も次回も	有効にし、同時に今すぐ起動する
`systemctl status <unit>`	——	稼働状態・自動起動設定・直近のログ

`start` と `enable` の違いは最頻出です。「今すぐ起動し、かつ次回起動時にも自動的に起動させたい」なら両方を実行するか `enable --now` を使います。片方だけの選択肢は誤りです。

状態を一語で確かめる操作と、起動そのものを禁じる操作も対で押さえます。

コマンド	効く時点	内容
`systemctl restart <unit>`	**今すぐ**	**いったん停止してから起動し直す**。設定ファイルを変更したあとの反映に使う
`systemctl is-active <unit>`	——	**いま稼働しているかどうかを `active` / `inactive` の 1 語で返す**
`systemctl is-enabled <unit>`	——	**自動起動の設定を `enabled` / `disabled` の 1 語で返す**

コマンド	効く時点	内容
`systemctl mask <unit>`	今も次回も	**そのユニットを起動できない状態にする。**`start` も、他のユニットからの依存関係による起動も行われなくなる。解除は `unmask`

`disable` **は「次回から自動的には起動しない」だけで、他のユニットが必要としていれば依存関係によって起動されます。**依存関係によっても起動させたくないなら `mask` を使う —— この差がそのまま設問になります。また `status` が複数行の詳細を返すのに対し、`is-active` **と** `is-enabled` **は 1 語だけを返す**ので、スクリプトの中での判定に向きます。

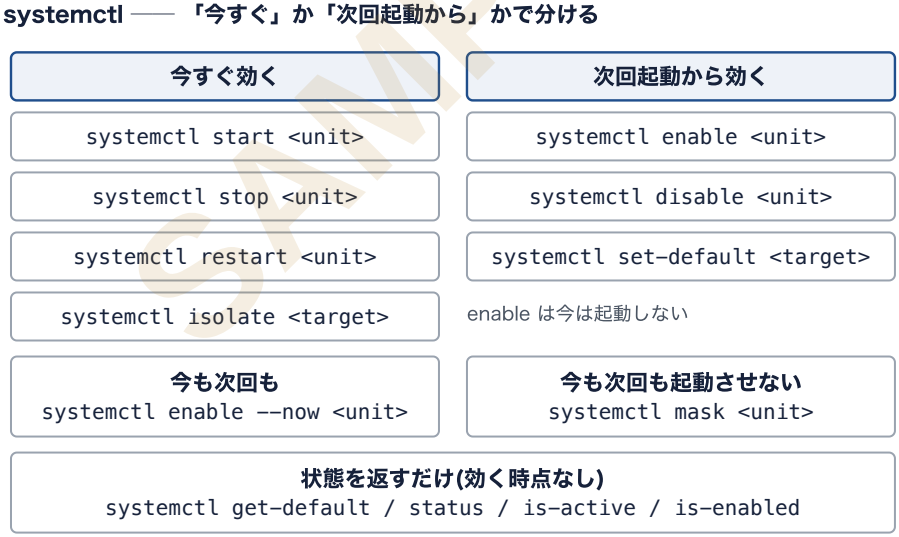


図 1-4　systemctl ——「今すぐ」か「次回起動から」かで分ける

1-3-6 `/etc/systemd/` と `/usr/lib/systemd/` [101.3]

systemd のユニットファイルと設定ファイルは、**複数のディレクトリに分かれて置かれ、優先順位が決まっています。**

ディレクトリ	誰のもの	優先度
`/usr/lib/systemd/system/`	**パッケージが提供する既定のユニットファイル。**直接編集しない	低い
`/run/systemd/system/`	実行中に動的に生成されるもの	中
`/etc/systemd/system/`	**管理者が置く / 上書きするユニットファイル**	**高い(最優先)**

同名のユニットファイルがあれば、`/etc/systemd/system/` に置いたものが勝ちます。パッケージ提供の定義を直接書き換えると更新時に上書きされてしまうため、**優先度の高い場所に同名ファイルを置くか、`.d` ディレクトリにドロップイン設定を追加する**のが作法です。

`/etc/systemd/` の直下には、systemd 本体の設定ファイルも置かれます。

ファイル	内容
`/etc/systemd/system.conf`	systemd 本体の全体設定
`/etc/systemd/journald.conf`	ジャーナル(ログ)の設定
`/etc/systemd/logind.conf`	ログインセッションの管理設定
`/etc/systemd/system/default.target`	**既定ターゲットへのシンボリックリンク**

ユニットファイルを編集したあとは `systemctl daemon-reload` を実行して読み直させないと、古い定義のまま動きます。「設定を変更したのに反映されない」の答えはたいていこれです。

ディストリビューションによっては `/lib/systemd/system/` が `/usr/lib/systemd/system/` への symlink になっています。**「パッケージ側は `/usr/lib/systemd/`、管理者側は `/etc/systemd/`」**という対で覚えるのが実戦的です。

1-3-7 シングルユーザーモードへ入る [101.3]

シングルユーザーモードは、ネットワークサービスも他の利用者のログインも止めて、**管理者ひとりだけが作業できる保守用の状態**です。ファイルシステムの修復やパスワードの復旧に使います。入り方は 3 通りあり、状況によって使い分けます。

状況	方法
稼働中の SysVinit 環境から移る	`init 1` または `telinit 1`
稼働中の systemd 環境から移る	`systemctl isolate rescue.target`
起動時に最初からその状態で立ち上げる	ブートローダーで `e` を押し、カーネル行に `single`(SysVinit)または `systemd.unit=rescue.target`(systemd)を追記する

それ以上に深刻な状態が `emergency.target` で、ルートを読み取り専用でマウントしただけの最小構成になります。`rescue` でも起動しないときの最後の手段です。

シングルユーザーモードへ移行すると**動いていたサービスは停止され、他の利用者はログアウトさせられます。**だからこそ、次に述べる「事前の通知」が必要になります。

1-3-8 `shutdown` —— 予約と通知ができる停止コマンド [101.3]

書式　`shutdown [オプション] <時刻> [メッセージ]`

オプション	意味
`-h`	**停止(halt / poweroff)する**
`-r`	**再起動する**
`-c`	**予約済みの停止処理を取り消す**
`-k`	**実際には停止せず、警告メッセージだけを送る**
`-H`	halt する
`-P`	poweroff する(電源を切る)
`--no-wall`	警告メッセージを送らない

時刻の書き方は 3 通りです。

書き方	意味
`now`	ただちに
`+10`	**10 分後**
`23:00`	その日の 23 時 00 分(24 時間表記の `hh:mm`)

例　`shutdown -h now`　（ただちに停止する）
例　`shutdown -r now`　（ただちに再起動する）
例　`shutdown -r +10 "10分後に再起動します"`　（10 分後に再起動し、全端末へ通知する）
例　`shutdown -h 23:00`　（23 時に停止する）
例　`shutdown -c`　（予約を取り消す）
例　`shutdown -k +5 "テストです"`　（実際には止めず、警告だけ送る）

`-h` **が halt(停止)、**`-r` **が reboot(再起動)、**`-c` **が cancel(取り消し)。**そして `+10` **は「10 分後」であって「10 秒後」ではありません**——ここが定番の引っかけです。

`shutdown -r +10 "..."` を実行すると、**その場ですぐ再起動するのではなく、**10 分後に実行されます。その間、ログイン中の利用者の端末にメッセージが表示され、さらに停止の直前には**新規ログインが禁止**されます(`/etc/nologin` が作られる仕組み)。この「すぐには落ちない」点が設問の狙いどころです。

`-k` **は「脅すだけ」**で実際には停止しません。訓練や告知の練習に使います。

停止・再起動の他のコマンドとの違いも押さえます。

コマンド	動作
`halt`	システムを停止する。**電源が切れるかどうかは環境による**
`poweroff`	システムを停止し、**電源を切る**ことを明示する
`reboot`	システムを再起動する

コマンド	動作
`systemctl poweroff` / `systemctl reboot` / `systemctl halt`	systemd 経由で同じことを行う
`shutdown`	**時刻の予約と利用者への通知ができる。**運用ではこれが基本

`shutdown` **だけが「予約」と「通知」と「取り消し」を持っている**、というのが他コマンドとの決定的な違いです。「利用者に知らせてから 15 分後に停止したい」という設問で `halt` や `poweroff` を選ぶのは誤りになります。

これらのコマンドの実行には、一般に管理者権限が必要です。

1-3-9 `wall` —— ログイン中の全員へ知らせる [101.3]

```
書式  wall [ファイル]
書式  echo "メッセージ" | wall
```

`wall` **(write all)** は、**ログイン中のすべての利用者の端末へメッセージを流す**コマンドです。ランレベルやターゲットを切り替える前、保守で落とす前の告知に使います。

```
例  wall "20時からメンテナンスに入ります"
例  echo "5分後に再起動します" | wall
例  wall /etc/motd.notice                  （ファイルの内容を送る）
```

`shutdown` **は内部で** `wall` **と同じ仕組みを使って警告を送っています。**到達目標の「Alert users before switching runlevels / boot targets or other

major system events」に対応するのが、この `wall` と `shutdown` の警告機能です。

関連するコマンドとの区別も出ます。

コマンド	宛先
`wall`	**ログイン中の全員**
`write <ユーザー名>`	**特定の 1 人**の端末
`mesg n` / `mesg y`	**自分の端末が**他人からの書き込みを受け取るかどうかを切り替える
`/etc/motd`	**ログインに成功した後**に表示されるメッセージ(すでにログイン中の人には届かない)
`/etc/issue`	**ログインプロンプトの前**に表示されるメッセージ

`/etc/motd` **はログイン後、**`/etc/issue` **はログイン前**、そして**すでに作業中の人へ今すぐ届けたいなら** `wall` —— この三択が問われます。

1-3-10 プロセスを正しく終了させる [101.3]

到達目標には「Properly terminate processes」が含まれます。システムを落とす前に、動いているプロセスへ**正しい順番で終了を促す**という考え方です。

シグナル	番号	動作
`SIGTERM`	**15**	**終了を「お願い」する。既定のシグナル。**プロセスは後始末をしてから終われる
`SIGKILL`	**9**	**強制終了。**プロセスは受け取れず、後始末もできない

シグナル	番号	動作
`SIGHUP`	1	端末の切断。多くのデーモンでは**設定の再読み込み**として使われる
`SIGINT`	2	`Ctrl+C` による割り込み
`SIGSTOP` / `SIGCONT`	19 / 18	一時停止 / 再開

正しい手順は「まず `SIGTERM` (15)、それでも終わらなければ `SIGKILL` (9)」です。いきなり `kill -9` を使うと、書きかけのファイルが壊れたり、一時ファイルが残ったりします。**`SIGKILL` は最後の手段**、という順序が設問の核心です。

```
例  kill 1234          (PID 1234 へ SIGTERM を送る)
例  kill -15 1234      (同じ。番号を明示した形)
例  kill -9 1234       (強制終了させる)
例  killall sshd       (名前を指定して終了させる)
例  pkill -u alice     (alice のプロセスをまとめて終了させる)
```

`kill` は PID、`killall` と `pkill` はプロセス名で指定します。`kill` にプロセス名を渡しても動きません —— これが取り違えの定番です。

なお、systemd や SysVinit はシャットダウン処理の中で、**全プロセスに `SIGTERM` を送り、一定時間待ってから `SIGKILL` を送る**という手順を自動で行っています。管理者が `shutdown` を使うべき理由のひとつがこれです。

1-3-11 `acpid` —— 電源ボタンを押したときの動作 [101.3]

ACPI(Advanced Configuration and Power Interface) は、電源管理のための規格です。`acpid` は、ACPI が発生させるイベント —— 電源ボタンが押

された、ノート PC のふたが閉じられた、バッテリーが少なくなった —— を受け取り、**設定に従ってスクリプトを実行するデーモン**です。

項目	内容
デーモン名	`acpid`
設定の置き場所	`/etc/acpi/events/`（イベント定義）・`/etc/acpi/`（実行するスクリプト）
典型的な用途	**電源ボタンを押したら `shutdown -h now` を実行する**

到達目標は「Awareness of acpid」、つまり**存在と役割を知っていれば十分**です。押さえるのは 1 点、**「サーバーの電源ボタンを押すと、いきなり電源が切れるのではなく、`acpid` が受け取って正常なシャットダウン処理を始める」**という動きです。

systemd 環境では `systemd-logind` が同種の処理を担い、`/etc/systemd/logind.conf` の `HandlePowerKey` などで挙動を指定できます。**「電源ボタンの挙動を変えたい」という設問では、`acpid` の設定か `logind.conf` が答えになります。**

1-4 第1章の試験ポイント [101.1/101.2/101.3]

ハードウェア設定 [101.1]

- **`/dev/` は操作の入り口、`/sys/` はデバイスの階層を見る窓、`/proc/` は状態を見る窓。**`/proc/` と `/sys/` はディスク上に実体を持たない仮想ファイルシステム

- **IRQ は `/proc/interrupts`、I/O ポートは `/proc/ioports`、DMA は `/proc/dma`**
- **`lsmod` はロード済みのみ**を表示(元データは `/proc/modules`)。ファイルパスやロードされていないモジュールは出ない
- **`modprobe` は名前指定で依存も解決、`insmod` はパス指定で依存を解決しない**。アンロードは `modprobe -r` と `rmmod`
- **`lspci` は PCI、`lsusb` は USB**。どちらも `-v` が詳細、`-t` がツリー、`lspci -k` はドライバ名
- **udev が `/dev/` を動的に作り、D-Bus はプロセス間の連絡係、sysfs は情報源**。管理者のルールは `/etc/udev/rules.d/`
- **SATA・SCSI・USB は `/dev/sd` 系、NVMe だけ `/dev/nvme0n1p1` の形**。SATA 接続の SSD は `/dev/sda`

システムの起動 [101.2]

- 起動順は **ファームウェア(BIOS / UEFI) → bootloader → kernel + initramfs → init(systemd)**
- **Secure Boot は UEFI の機能**。UEFI は GPT に対応し、ESP 上の `.efi` ファイルを直接読める
- **initramfs はルートをマウントするための一時的な踏み台**。マウント後は退場する
- カーネルパラメータは `/proc/cmdline` で確認。`single` / `1` は SysVinit、`systemd.unit=rescue.target` は systemd
- init の三世代は **SysVinit(逐次)・Upstart(イベント駆動)・systemd(並列)**。systemd は **PID 1**

- `dmesg` **はリングバッファで消える。**`-T` で時刻表示、`-w` で追跡、`-c` で消去
- `journalctl -b` **が今回、**`-b -1` **が前回の起動。**`-k` はカーネル、`-u` はユニット。永続化は `/var/log/journal/`
- ログの場所は **Red Hat 系が** `/var/log/messages`**、Debian 系が** `/var/log/syslog`

ランレベル / ブートターゲット [101.3]

- ランレベルは **0 = 停止、1 = シングルユーザー、3 = CUI、5 = GUI、6 = 再起動**
- 既定のランレベルは `/etc/inittab` の `initdefault` 行。`telinit q` **は inittab の再読み込みだけ**
- 起動スクリプトの実体は `/etc/init.d/`、ランレベルごとの起動順は `/etc/rc<数字>.d/` の `S`**(start)/** `K`**(kill)+ 番号**のリンク
- `init 0` **は停止、**`init 6` **は再起動。**`init` と `telinit` は同じ働き
- ターゲットは **3 =** `multi-user.target`**、5 =** `graphical.target`**。**`rescue` **はマウントする、**`emergency` **はほぼしない**
- `systemctl set-default` **は次回から、**`systemctl isolate` **は今すぐ。**`start` **は今すぐ、**`enable` **は次回から**
- `restart` **は止めてから起動し直す、**`is-active` **は稼働状態、**`is-enabled` **は自動起動設定を 1 語で返す。**`disable` **は依存関係による起動までは止められず、それを止めるのは** `mask`
- ユニットファイルは **パッケージ側が** `/usr/lib/systemd/`**、管理者側が** `/etc/systemd/` **で管理者側が優先。**編集後は `systemctl daemon-reload`

- **`shutdown` の `-h` は停止、`-r` は再起動、`-c` は取り消し、`-k` は警告のみ。**`+10` は 10 分後
- **`wall` はログイン中の全員へ、**`write` は 1 人へ、`/etc/motd` はログイン後、`/etc/issue` はログイン前
- 終了は **`SIGTERM` (15)が先、`SIGKILL` (9)は最後の手段**
- **`acpid` は電源ボタンやふたの開閉などの ACPI イベントを受けてスクリプトを実行する**デーモン

サンプルはここまでです

続き(第2章〜巻末)は、LPIC-1 101 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

LPIC-1 101試験 学習用テキスト(無料サンプル)

版
2026年7月版(LPI公式 LPIC-1 Exam 101 Objectives Version 5.0 準拠)

発行
IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ
info@shikaku-doujou.com