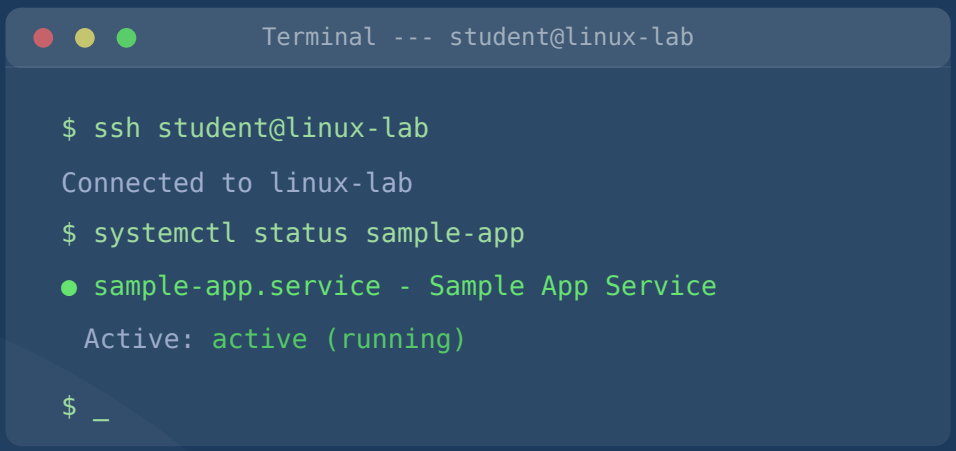
WINDOWS / LINUX / SERVER BASICS

Windows技術者向けの はじめてのLinux

Windows の知識から Linux サーバ運用の入口へ

WSL2 と AWS CloudShell から始める。
設定、権限、サービス、ログを小規模な業務サーバで学ぶ。



haya 株式会社

2026.03

目次

序章本書の読み方

本書の目的

本書は、Windows の運用経験を持つ技術者が Linux サーバの基本操作と運用の入口を無理なく学ぶための入門書です。画面の見え方や設定の持ち方、サービス管理の考え方が Windows とどう違うのかを整理しながら、日常的によく行う確認作業と変更作業を一通り体験できる構成にします。

本書で目指すのは、Linux の専門家になることではありません。まずは「ログインして状態を確認する」「設定ファイルを見つけて内容を把握する」「サービスの状態を調べる」「簡単なトラブルを切り分ける」といった、最初の一歩として必要な作業を一人で進められる状態を目標にします。

対象読者

本書は次のような読者を想定しています。

- Windows サーバや Windows クライアントの運用経験がある
- PowerShell や GUI の管理ツールには慣れている
- Linux はほぼ未経験、またはコマンド操作に不安がある
- 今後 Linux サーバに触れる業務が増える見込みがある

共通題材

本書では、製品固有の事情に引きずられないように、小規模な社内業務サーバを共通題材として扱います。章をまたいで、次の名前を繰り返し使います。

- ホスト名: `linux-lab`
- アプリ配置先: `/opt/sample-app`
- サービス名: `sample-app.service`
- 実行ユーザー: `appsvc`
- 運用グループ: `appops`
- ログ配置先: `/var/log/sample-app`

ポイント

Windows ではアプリごとに GUI の管理画面が用意されていることが多い一方、Linux ではディレクトリ、設定ファイル、サービス、ログという共通要素の組み合わせで管理する場面が増えます。本書はその共通部分に集中します。

本書の全体像

本書は 10 章と付録で構成されています。順番に読むと理解しやすいように並べていますが、復習時は目的に応じて章を引き直せるようにも作っています。

第 1 章 Linux と Windows の考え方の違いをつかみます。まずは「なぜ Linux ではファイルとコマンドが中心になるのか」を腹落ちさせる章です。
第 2 章 学習環境を用意し、最初に何を確認するかを実際に試します。ここで接続先、ユーザー、OS、ネットワークを見る習慣を作ります。
第 3 章 ディレクトリ構造とパスの考え方を学びます。設定、ログ、アプリ本体がどこにあるかを地図として持つ章です。
第 4 章 設定ファイルやログを読み、検索し、必要最小限だけ編集する方法を学びます。
第 5 章 ユーザー、グループ、権限を扱います。読める、書ける、実行できるの違いを見分ける章です。
第 6 章 パッケージ管理の基本を学びます。ソフトウェアを導入、更新、削除するときの流れを理解します。
第 7 章 プロセスとサービスの見方を学びます。サービス障害時の入口になる章です。
第 8 章 ネットワーク確認の基本を学びます。名前解決、疎通、待受、ファイアウォールを切り分けて考えます。
第 9 章 シェルスクリプトで、ここまでの確認作業を繰り返し実行できる形にまとめます。
第 10 章 小さな業務サーバを題材に、接続、配置、権限、サービス、ログ、ネットワークを一連の流れとして扱います。
付録 Windows と Linux の対応表、クイックリファレンス、vi チートシート、用語集をまとめています。作業中の引き直し用です。

読後にできるようになること

- Linux サーバへログインし、基本的な情報を確認できる
- ディレクトリ構造とパスの考え方を理解して移動できる
- 設定ファイル、権限、サービス、ログを基本手順で確認できる
- パッケージ管理と簡単なシェルスクリプトの役割を理解できる
- よくある初歩的な障害を切り分ける観点を持てる

先に知っておきたい優先順位

Linux の学習では、情報量の多さに圧倒されやすくなります。しかし実務では、使う頻度と重要度にはかなり偏りがあります。本書では、次の 3 段階で考えると進めやすくなります。

優先度 1

まず確実に身につけたいのは、`cd`、`ls`、`pwd`、`cat`、`grep`、`tail`、`sudo`、`ls -l`、`systemctl status`、`journalctl -u`、`ssh` です。これらは日常運用と初歩障害の切り分けで何度も使います。

優先度 2

次に重要なのは、パッケージ導入、待受ポート確認、名前解決確認、簡単なスクリプト化です。毎日ではなくても、環境構築や障害時には確実に必要になります。

優先度 3

高度な一括処理、複雑なテキスト加工、細かな最適化は後回しで構いません。最初は「状態を読み、必要最小限の変更を安全に行う」ことを優先してください。

章ごとの最低到達点

本書は順番に読む前提で構成していますが、各章ですべてを均等に覚える必要はありません。まずは次の到達点を目安にしてください。

第 1 章 Linux では GUI よりもファイル、コマンド、ログで状態を読む場面が多いことを理解する。

第 2 章 接続先、ユーザー、OS、IP アドレス、空き容量を最初に確認する習慣を作る。

第 3 章 `pwd`、`cd`、`ls` を迷わず使い、`/etc`、`/var/log`、`/opt`、`/home` の役割を説明できる。

第 4 章 `cat`、`less`、`tail`、`grep` と、最低限の `vi` 操作で設定とログを読める。

第 5 章 `ls -l` の 1 行を読み、`sudo`、`chmod`、`chown` を必要な場面で使い分けられる。

第 6 章 自分の環境で何のパッケージ管理方式を使っているか分かり、導入、更新、削除、導入済み確認を一通り行える。

第 7 章 `systemctl status`、`start`、`stop`、`restart` と `journalctl -u` を使って、サービス状態を追える。

第 8 章 名前解決、疎通、待受、HTTP 応答、ファイアウォールを別々に確認する発想を持てる。

第 9 章 手で行っていた確認作業を短いシェルスクリプトにまとめ、`bash -n` や `bash -x` で見直せる。

第 10 章 接続、配置、権限、サービス、ログ、ネットワーク、バックアップを一連の流れとして再現できる。

本書の進み方

前半では、Linux の基本概念と操作に慣れることを優先します。中盤では、ユーザー管理、パッケージ管理、サービス管理、ネットワーク確認といった運用の基礎を扱います。後半では、複数の知識を組み合わせて、小さな業務サーバを安全に扱うための総合演習へつなげます。

各章は、考え方の説明だけで終わらないようにします。コマンドを実行する目的、確認すべき出力、次にどう読むかを順番に示し、Windows の感覚から Linux の感覚へ橋をかけることを重視します。章末には短い確認課題を置き、読み流しではなく手を動かして定着させる構成にしています。

学習の進め方

- まず読むだけで終えず、短いコマンドでも実際に打つ
- 想定どおりに動かなかったときは、エラーメッセージをそのまま読む
- コマンドを暗記するより、「何を確認するための道具か」を覚える
- 変更前に現在値を読み、変更後に差分と結果を確認する

特に大切なのは、コマンド名よりも順番です。接続先確認、現在値確認、変更、結果確認の順が崩れなければ、初学者でも大きな事故をかなり減らせます。

本書の表記

- ターミナル風の濃い背景ボックスは、端末での入力例と出力例を表します。
- `$` で始まる行は入力するコマンド、続く行はその出力例です。
- 出力例は一例であり、環境差で表示が完全には一致しないことがあります。
- `sample-app`、`linux-lab`、`appsvc`、`appops` は、本書を通して使う共通題材の名前です。

最初に意識しておきたいこと

- Linux では「設定はファイルにある」と考える
- Linux では「状態確認はコマンドとログで行う」と考える
- Linux では「まず読む、次に変える」の順番を徹底する

この 3 点だけでも、最初のつまずきはかなり減ります。第 1 章では、その前提になる Linux の考え方と Windows との違いから始めます。

第 1 章 Linux の基本と Windows との違い

1.1 Linux とは何か

Linux は、サーバ、クラウド、ネットワーク機器、組み込み機器まで幅広く使われている OS の系統です。実務では単に「Linux」と呼ばれることが多いですが、実際にはカーネルと周辺ツールをまとめたディストリビューションとして使います。Ubuntu 系や RHEL 系のように流儀の違いはありますが、本書ではまず共通する基本操作に絞って進めます。

Windows 技術者が最初に戸惑いやすいのは、Linux には 1 つの標準 GUI 管理画面があるわけではない点です。代わりに、テキスト設定、コマンド操作、ログ確認を組み合わせて状態を把握します。最初は遠回りに見えても、このやり方に慣れると、自動化しやすく、差分も追いやすくなります。

1.2 なぜ Windows 技術者が Linux を学ぶのか

Windows 中心の環境でも、周辺システムやクラウド基盤は Linux で動いていることが珍しくありません。監視サーバ、リバースプロキシ、CI 実行環境、コンテナホスト、各種ミドルウェアの導入先として Linux に触れる機会は増えています。

そのため、Linux の専門担当でなくても、最低限の読み方と触り方を知っているだけで対応できる範囲が大きく広がります。本書では、深いチューニングよりも、まず運用の入口で詰まらないことを優先します。

1.3 Windows と Linux の考え方の違い

1.3.1 設定の持ち方

Windows では GUI の管理画面やレジストリを通じて設定を変更する場面が多くあります。一方 Linux では、設定の多くが `/etc` 以下やアプリ固有ディレクトリのテキストファイルに保存されます。

安全な変更の基本

Linux で設定変更を行うときは、いきなり編集するのではなく、まずファイルの場所を確認し、現在値を読み、バックアップを取り、それから変更する順番が基本です。

1.3.2 サービスの扱い方

Windows にはサービス管理コンソールがありますが、Linux では `systemctl` を中心にサービスの起動、停止、再起動、自動起動設定を扱います。アプリの見え方よりも、プロセス、ユニット、ログという単位で理解すると整理しやすくなります。

1.3.3 ファイルシステムの見え方

Windows のドライブレターに対し、Linux は 1 本のディレクトリツリーで構成されます。アプリ本体、設定、ログ、データがどこに置かれるかを知ることが、そのまま運用のしやすさにつながります。

1.4 最初に覚えるべき Linux の見取り図

最初の段階では、すべてを覚える必要はありません。次の 5 つの場所の役割だけ押さえれば、かなりの場面で迷いにくくなります。

- `/home`: ユーザーごとの作業場所
- `/etc`: 設定ファイル
- `/var/log`: ログファイル
- `/opt`: 追加アプリの配置先
- `/tmp`: 一時ファイル

本書の共通題材では、アプリ本体を `/opt/sample-app`、ログを `/var/log/sample-app` に置く前提で進めます。これは特定製品の決まりではなく、役割の分離を理解しやすくするための例です。

1.5 Linux に触るときの最初の手順

新しい Linux サーバに入ったとき、最初に見るべきものはおおむね決まっています。次の順番を体で覚えると、無駄な遠回りが減ります。

最初の確認コマンド

```
$ hostname
$ whoami
$ pwd
$ cat /etc/os-release
$ ip addr show
```

出力例

```
$ hostname
linux-lab
$ whoami
student
$ pwd
/home/student
$ cat /etc/os-release
NAME="Rocky Linux"
VERSION="9.5"
$ ip addr show
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
    inet 192.168.1.20/24 brd 192.168.1.255 scope global eth0
```

この並びには意味があります。どのホストにいるか、どのユーザーで作業しているか、今どこにいるか、OS は何か、ネットワークはどうなっているか、という順で確認しています。Windows でいえば、接続先、実行ユーザー、OS 情報、ネットワーク設定を順番に見ているのと同じです。

出力例で最初に見る点

- `hostname`: 接続先が想定どおりか
- `whoami`: 作業ユーザーが想定どおりか
- `pwd`: 現在位置がホームディレクトリか
- `/etc/os-release`: NAME と VERSION が何か
- `ip addr show`: inet の行にどの IP が出ているか

サービス確認は後続章で扱う

サービスの状態確認も重要ですが、`systemctl` の読み方は第 7 章でまとめて扱います。第 1 章では、まず接続先、ユーザー、OS、ネットワークという土台だけを確実に読めるようになることを優先してください。

1.6 本書で重視する進め方

本書では、難しい理論よりも「現場で最初に必要な判断」を優先します。たとえば、次のような問いにすぐ答えられる状態を目指します。

- 設定ファイルはどこにあるのか
- ログはどこを見ればよいのか
- サービスは起動しているのか
- 変更してよいユーザーなのか
- 失敗したとき、何を先に確認すべきか

これは Linux だけに閉じた話ではありません。Windows の運用経験がある人ほど、確認の順序と変更の手順を揃える価値を理解しやすいはずです。Linux ではそれが、より明示

的にファイルとコマンドの形で表れます。

確認課題

- 自分の業務で Linux が登場しそうな場面を 3 つ書き出し、そこで何を確認する必要がありそうかを考えてください。
- `/home`、`/etc`、`/var/log`、`/opt`、`/tmp` の役割を、Windows の感覚に言い換えて説明してみてください。
- `hostname`、`whoami`、`pwd`、`cat /etc/os-release` の 4 つについて、「何を見るコマンドか」を口頭で説明できるか確認してください。
- 実際に `hostname` と `whoami` を実行し、想定どおりの接続先とユーザーになっているか確認してください。

まとめ

本章では、Linux を学ぶ理由と、Windows との違いの見取り図を整理しました。重要なのは、Linux を特別な世界として捉えすぎないことです。設定、サービス、ログ、権限という観点に分けて見れば、運用の考え方そのものは Windows での経験とつながっています。

次章では、実際に手を動かせる学習環境を整え、本章で見た確認コマンドを安全に試せる状態を作ります。

第 2 章学習環境を構築する

本章では、Linux を安全に試せる学習環境を用意します。いきなり本番サーバで練習するのではなく、失敗してもやり直せる環境を先に持つことで、コマンドの意味と出力の読み方に集中できるようにします。第 1 章で見た確認コマンドも、ここから実際に手を動かして確かめていきます。

2.1 学習環境の選び方

Linux の学習環境は 1 つではありません。大事なのは「すぐ触れること」と「壊しても困らないこと」です。Windows 技術者が最初に選びやすい方法は、次の 3 つです。

2.1.1 WSL2

Windows の中で Linux を動かす方法です。普段使っている PC だけで始められるので、最初の入口として優れています。ファイルパスやネットワークの見え方に Windows 側の事情が残ることはありますが、基本コマンドを覚えるには十分です。

2.1.2 AWS CloudShell

AWS マネジメントコンソールからブラウザで開ける Linux シェル環境です。ソフトウェアのインストールや仮想マシン作成が不要なので、会社 PC で管理者権限を使いにくい場合でも始めやすい方法です。

2.1.3 仮想マシン

VirtualBox や Hyper-V などで Linux 仮想マシンを作る方法です。本番に近い形で学べるため、サービス管理、ネットワーク設定、ユーザー追加などまで試したい場合に向いています。再インストールやスナップショットがしやすい点も利点です。

2.1.4 SSH 接続できる検証用サーバ

SSH で Linux に接続する形です。実際の業務に最も近い反面、操作対象を誤ると他人の環境に影響することがあります。本書の前半を学ぶ段階では、まず手元の検証環境で練習してから使う方が安全です。

本書の前提

本書では、どの環境を使っていても通用するように、Bash 系シェルでの基本操作に寄せて説明します。組織の標準環境が決まっている場合は、それに合わせて構いません。

迷ったときの選び方

最初の 1 台として迷うなら、管理者権限を使えるなら WSL2、ブラウザだけですぐ始めたいなら AWS CloudShell、本番に近い形で練習したいなら仮想マシンを選ぶと進めやすくなります。SSH 接続だけの検証サーバは、基本操作に慣れてからでも遅くありません。

2.2 WSL2 を始める

Microsoft Learn の現行手順では、Windows 11 または Windows 10 Version 2004 以降なら、管理者権限の PowerShell から `wsl --install` を実行して再起動する流れが基本です。本書でも、その最短経路で始めます。

2.2.1 事前確認

- Windows 11、または Windows 10 Version 2004 以降であること
- 管理者権限で PowerShell を開けること
- 会社 PC の制限で仮想化機能や Microsoft Store 利用が禁止されていないこと

2.2.2 最短セットアップ

PowerShell での導入

```
PS> wsl --install
```

既定では Ubuntu が導入されます。別のディストリビューションを選びたい場合は、利用可能な候補を確認してから指定します。

ディストリビューションの確認

```
PS> wsl --list --online
PS> wsl --install -d Ubuntu
```

`wsl --install` の実行後は、Windows の再起動が必要です。再起動後に Ubuntu などの初回起動が始まり、Linux 側のユーザー名とパスワードを設定します。ここで作る名前が、以後 WSL2 上での標準ユーザーになります。

2.2.3 起動確認

導入確認

```
PS> wsl --list --verbose
  NAME      STATE           VERSION
* Ubuntu    Running         2
```

WSL2 側での確認

```
$ whoami
$ pwd
$ cat /etc/os-release
```

導入で止まりやすいところ

`wsl --install` がそのまま通らない場合は、Windows の版が古いか、すでに WSL が部分的に入っていることがあります。古い環境では手動導入手順が必要になるため、その場合は Microsoft Learn の WSL 手順を参照してください。

WSL2 では Linux 側のホームを使う

WSL2 で Linux コマンドを使って練習する場合は、作業用ファイルを `/home/<user>` 配下に置く方が素直です。`/mnt/c/Users/...` でも作業はできますが、速度や権限の見え方で混乱しやすくなります。Windows のエクスプローラーで現在位置を開きたいときは、WSL2 上で `explorer.exe .` を実行します。

2.3 AWS CloudShell を始める

AWS CloudShell は、AWS が提供するブラウザベースのシェル環境です。Microsoft Learn の WSL と違ってローカル PC への導入は不要で、AWS マネジメントコンソールへ入れればすぐに Bash を試せます。

2.3.1 事前確認

- AWS アカウント、または IAM ユーザー、IAM ロールでコンソールにサインインできること
- CloudShell を使う権限があること
- 学習中は同じ AWS リージョンを使い続けること

組織アカウントを使う場合は、少なくとも `AWSCloudShellFullAccess` 相当の権限が必要です。権限が不足している場合は、起動前に管理者へ確認した方が早く進みます。

2.3.2 起動手順

1. Web ブラウザで AWS マネジメントコンソールへサインインする
2. 画面右上のリージョン選択で、学習に使うリージョンを決める
3. 上部の CloudShell アイコン、または検索ボックスから CloudShell を開く
4. 必要に応じて Bash を選ぶ
5. プロンプトが表示されたら準備完了

CloudShell の起動直後

```
[cloudshell-user@ip-10-130-58-72 ~]$
```

2.3.3 CloudShell で最初に知っておくこと

- `$HOME` 配下のファイルは、同じリージョン内ではセッションをまたいで保持される
- 永続ストレージはリージョンごとに 1GB なので、学習ファイルは増やしすぎない
- キーボードやポインタ操作がない状態が 20 から 30 分続くと、セッションは終了する
- 1 つのシェルセッションは、おおよそ 12 時間で終了する

リージョンを途中で変えない

AWS CloudShell のホームディレクトリはリージョンごとに分かれています。同じアカウントでも、リージョンを変えると別のホームディレクトリを見ている状態になります。学習中は、同じリージョンを使い続ける方が混乱しません。

CloudShell での保存場所

学習用ファイルは `/home/cloudshell-user` 配下へ置きます。逆に、`sudo dnf install` のようにシステム領域へ入れたソフトウェアは、セッション再作成後に残らないことがあります。本書の練習ファイルは、必ず `$HOME` 側に置いてください。

2.4 練習用ファイルをどこに置くか

本書では、どの環境でも共通して `/linux-first-lab` を練習場所にします。理由は単純で、消しても困らないことと、管理者権限を使わずに作れることの 2 つです。

練習用ディレクトリ

```
$ mkdir -p ~/linux-first-lab
$ cd ~/linux-first-lab
$ pwd
```

- WSL2 では Linux 側ホームディレクトリ配下に置く
- AWS CloudShell では `$HOME` 配下に置く

- 仮想マシンでも、まずは一般ユーザーのホーム配下で練習する

2.5 最初に確認すること

Linux に接続した直後に、何の情報を見ればよいかを先に覚えておくと迷いにくくなります。最初の確認は次の 6 項目です。

最初の確認

```
$ whoami
$ hostname
$ pwd
$ cat /etc/os-release
$ uname -a
$ ip addr show
```

それぞれの意味は次のとおりです。

- `whoami`: 今どのユーザーで作業しているか
- `hostname`: どのサーバに入っているか
- `pwd`: 今どのディレクトリにいるか
- `cat /etc/os-release`: OS の種類と版
- `uname -a`: カーネルやアーキテクチャの概要
- `ip addr show`: ネットワークインターフェースの状態

Windows でいえば、接続先、実行ユーザー、OS 情報、ネットワーク設定を最初に見ているのと同じです。Linux ではこの確認を GUI ではなくコマンドで行う、という違いだけです。

第 1 章で見た見取り図のうち、この章では接続先、ユーザー、OS、ネットワークの確認に絞ります。サービス管理やログ確認は、後半の章で土台ができてから扱います。

出力例

```
$ ip addr show
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
    inet 192.168.1.20/24 brd 192.168.1.255 scope global eth0
```

`ip addr show` では、まずインターフェース名と `inet` の行だけ見れば十分です。最初は出力全体を読もうとせず、「どの NIC に、どの IP が付いているか」を拾うことから始めてください。

環境によって見え方は違う

WSL2 ではホスト名が Windows 側と少し違って見えたり、`ip addr show` に仮想ネットワークのアドレスが出たりします。AWS CloudShell では `cloudshell-user` や `ip-10-...` のような表示になります。表示そのものが本書と一致することよりも、「今

のユーザー」「今のホスト」「今の OS」が読めることを優先してください。

2.6 プロンプトの読み方

ターミナルの左端に出るプロンプトには、その時点の文脈が詰まっています。次のような表示を見たとき、最低限どこを見ればよいかを押さえましょう。

プロンプト例

```
[student@linux-lab ~]$
```

- `student`: 現在のユーザー名
- `linux-lab`: ホスト名
- `~`: 現在位置がホームディレクトリ
- `$`: 一般ユーザー

root で作業している場合は、末尾が `#` になることが多くあります。作業前に `$` なのか `#` なのかを見る癖をつけるだけでも、事故をかなり減らせます。

2.7 最初のコマンドを実行する

ここでは、Linux に慣れるための最小セットだけを扱います。ファイル一覧を見て、作業ディレクトリを作り、テキストを 1 行書いて、中身を確認するところまで進めます。

2.7.1 ファイル一覧の表示

一覧表示

```
$ ls
$ ls -la
```

`ls` は一覧表示、`ls -la` は隠しファイルも含めた詳細表示です。最初は「何も表示されない」「思ったよりファイルが少ない」と感じるかもしれませんが、それで正常です。

2.7.2 作業ディレクトリの作成

本書では、ホームディレクトリ配下に作業用ディレクトリを用意して進めます。

作業ディレクトリ

```
$ pwd
$ mkdir -p ~/linux-first-lab
$ cd ~/linux-first-lab
$ pwd
```

```
/home/student/linux-first-lab
```

`mkdir -p` は、途中のディレクトリがなくてもまとめて作成する指定です。以後の章では、この `/linux-first-lab` を練習用の安全な場所として使います。

2.7.3 テキストファイルを作る

最初のファイル

```
$ echo "hello linux" > note.txt
$ cat note.txt
hello linux
$ ls -l note.txt
```

`echo` は文字列を出力するコマンドです。`>` を付けると、画面ではなくファイルに書き込みます。`cat` はその内容を確認するコマンドです。

2.8 システム情報を読む

Linux サーバに入ったら、CPU、メモリ、ディスクの確認もよく行います。最初は値を細かく解釈できなくても構いません。まずは「どのコマンドで見るか」を覚えましょう。

基本的なシステム情報

```
$ nproc
$ free -h
$ df -h
```

- `nproc`: 利用可能な CPU 数
- `free -h`: メモリ使用量
- `df -h`: ディスク使用量

この 3 つは、アプリが重い、ディスクが埋まりそう、環境要件を満たしているか確認したい、といった場面で繰り返し使います。

出力例

```
$ free -h
               total        used        free      shared  buff/cache
    available
Mem:           1.8Gi       420Mi       1.1Gi        20Mi       310Mi       1.2
    Gi
$ df -h
Filesystem      Size  Used Avail Use\% Mounted on
/dev/sda1        20G  3.2G   17G   16\% /
```

最初はどこを見れば十分か

- `free -h`: Mem 行の `available` を見て、今すぐ不足していないかを確認する
- `df -h`: 作業対象のマウントポイントで `Avail` と `Use%` を見る

2.9 SSH 接続の考え方

仮想マシンやリモートサーバを使う場合は、SSH で接続することになります。Windows ではリモートデスクトップに近い感覚で捉えがちですが、Linux では「テキストの操作窓を開く」と考えた方が実態に合っています。

SSH 接続例

```
$ ssh student@linux-lab
```

接続先がまだない段階では、この節は「こういう形で入る」と把握できれば十分です。接続後は、いきなり設定変更を始めるのではなく、まず `whoami`、`hostname`、`pwd` で状況を確認してください。接続先の取り違えは、初学者だけでなく経験者でも起きます。

2.10 この章でつまずきやすい点

- PowerShell の `PS>` と Linux の `$` を見分けないままコマンドを打ってしまう
- WSL2 で `/mnt/c/...` を作業場所にして、権限や改行コードの違いに戸惑う
- AWS CloudShell でリージョンを変えてしまい、前回のファイルが消えたように見える
- Linux に入った直後に `whoami` や `hostname` を見ず、接続先を取り違える
- いきなり `sudo` を多用し、一般ユーザーで十分な練習まで管理者権限で進めてしまう

最初は一致より再現

本書の画面例と、自分の環境のプロンプトや IP アドレスが完全に一致しなくても問題ありません。大事なのは、同じ手順をたどって同じ種類の情報が読めることです。

2.11 この段階では何を覚えればよいか

本章の時点では、すべてのコマンドを暗記する必要はありません。次のことができれば十分です。

- WSL2 または AWS CloudShell を起動して Linux のシェルを開ける
- Linux に接続できる
- プロンプトを見て、ユーザーとホストを判別できる
- 作業ディレクトリを作って移動できる
- 1 行のテキストファイルを作って確認できる
- OS、CPU、メモリ、ディスクの基本情報を確認できる

確認課題

- 学習環境を 1 つ選び、起動手順を自分の言葉で 3 から 5 行にまとめてください。
- なぜその方式を選ぶのかを、「始めやすさ」「壊しても困らないか」の 2 点で説明してください。
- `whoami`、`hostname`、`pwd`、`cat /etc/os-release`、`uname -a`、`ip addr show` を実行し、それぞれの出力から 1 つずつ読み取れた情報を書き出してください。
- `/linux-first-lab` を作り、`note.txt` に 1 行書いて、`cat` と `ls -l` で確認してください。

まとめ

本章では、学習環境の選び方と、Linux に入った直後に確認すべき基本項目を整理しました。重要なのは、学習環境の優劣よりも、安心して何度でも試せる状態を先に作ることです。

次章では、Linux のディレクトリ構造とパスの考え方を学び、どこに何が置かれているのかを読めるようにします。

第 3 章ファイルシステムとディレクトリ構造

本章では、Linux のディレクトリ構造を地図として理解します。Windows のドライブレターに慣れている読者が迷いやすい点を避けながら、設定、ログ、アプリ本体、作業ファイルがどこに置かれるかを整理します。第 2 章で作成した `/linux-first-lab` を土台に、ここからは「どこにあるか」を読める状態を目指します。

3.1 Linux は 1 本の木で考える

Windows では `C:` や `D:` といったドライブレターで場所を区別します。一方 Linux では、ルートディレクトリ `/` を起点に、すべてが 1 本のディレクトリツリーとしてつながっています。

ルートの確認

```
$ cd /
$ pwd
/
$ ls
bin  boot  dev  etc  home  opt  proc  run  srv  tmp  usr  var
```

最初は名前を全部覚える必要はありません。まずは「どの種類の情報がどこに置かれやすいか」を知ることが重要です。

3.2 最初に覚えるべきディレクトリ

3.2.1 `/home`

ユーザーごとの作業場所です。第 2 章で作成した `/linux-first-lab` もここに置かれます。最初の練習は、この配下で行うのが安全です。

3.2.2 /etc

設定ファイルが置かれる場所です。ネットワーク、サービス、ユーザー、アプリ設定など、運用で頻繁に確認する情報が集まっています。

3.2.3 /var

変化しやすいデータの置き場です。特に /var/log はログ確認の入口として重要です。

3.2.4 /opt

追加アプリをまとめて配置する場所としてよく使われます。本書の共通題材では、アプリ本体を /opt/sample-app に置く前提で進めます。

3.2.5 /tmp

一時ファイルの置き場です。再起動やクリーンアップで消える前提のデータを置きます。恒久的に残したいものは置きません。

最初に持つ地図

Linux の運用では、「設定は /etc」「ログは /var/log」「追加アプリは /opt」といった見取り図を持っているだけで、調査の初動がかなり速くなります。

3.3 パスの考え方

3.3.1 絶対パス

ルート / から始まる書き方です。どこから実行しても同じ場所を指せるため、運用手順やドキュメントでは基本的に絶対パスを使います。

絶対パス

```
$ cd /etc
$ pwd
/etc
$ ls /var/log
```

3.3.2 相対パス

現在位置を基準にした書き方です。短く書ける反面、今どこにいるかを取り違えると別の場所を操作してしまいます。

相対パス

```
$ cd ~/linux-first-lab
$ mkdir -p sample-app/config
$ cd sample-app
$ pwd
/home/student/linux-first-lab/sample-app
$ ls config
```

3.3.3 よく使う記号

- `.`: 現在のディレクトリ
- `..`: 1 つ上のディレクトリ
- `~`: 自分のホームディレクトリ

記号の例

```
$ cd ~/linux-first-lab/sample-app/config
$ cd ..
$ pwd
/home/student/linux-first-lab/sample-app
$ cd ~
$ pwd
/home/student
```

3.4 よく使うファイル操作

3.4.1 一覧表示と移動

一覧と移動

```
$ pwd
$ ls
$ ls -la
$ cd ~/linux-first-lab
```

`pwd` で現在位置を確認し、`ls` で中身を見る。この順番は基本です。迷ったらまず現在位置を確認します。

3.4.2 作成とコピー

作成とコピー

```
$ mkdir -p ~/linux-first-lab/sample-app/config
$ mkdir -p ~/linux-first-lab/sample-app/logs
$ mkdir -p ~/linux-first-lab/sample-app/data
$ cp /etc/hosts ~/linux-first-lab/sample-app/config/hosts.sample
$ ls -R ~/linux-first-lab/sample-app
```

`mkdir -p` は階層をまとめて作るときに使います。`cp` はコピー、`-R` を付けるとディレクトリを再帰的に扱えます。ここでは 1 行でまとめる書き方もありますが、最初は 1 ディレクトリずつ作る形のほうが追いやすいので、この形で進めます。

`ls -R` で最初に見る範囲

`ls -R` は出力が長くなりやすいので、最初は一番上のディレクトリ名と、その直下に `config`、`logs`、`data` が見えているかだけ確認すれば十分です。全部を細かく読む必要はありません。

3.4.3 移動と名前変更

移動と名前変更

```
$ mv ~/linux-first-lab/sample-app/config/hosts.sample \
    ~/linux-first-lab/sample-app/config/hosts.backup
$ ls ~/linux-first-lab/sample-app/config
```

`mv` は移動だけでなく、名前変更にも使います。Linux ではこの 2 つは同じ操作として扱われます。

3.4.4 削除

削除

```
$ rm ~/linux-first-lab/sample-app/config/hosts.backup
$ rmdir ~/linux-first-lab/sample-app/data
```

`rm` はファイル削除、`rmdir` は空のディレクトリ削除です。再帰削除の `rm -r` は強力なので、最初は対象を `ls` で確認してから使う癖をつけてください。

3.5 業務サーバらしい配置を見てみる

学習用ディレクトリで操作に慣れたら、実際のサーバで見かける配置も知っておきます。

業務サーバの例

```
$ sudo mkdir -p /opt/sample-app/bin
$ sudo mkdir -p /opt/sample-app/config
$ sudo mkdir -p /opt/sample-app/data
$ sudo mkdir -p /var/log/sample-app
$ sudo ls -R /opt/sample-app
$ sudo ls -ld /var/log/sample-app
```

この配置には意味があります。

- `/opt/sample-app`: アプリ本体や配布物
- `/opt/sample-app/config`: アプリ固有設定
- `/opt/sample-app/data`: アプリが扱うデータ
- `/var/log/sample-app`: ログ出力先

ここでは形だけ見る

この節の `/opt` や `/var/log` の例は、業務サーバでよく見る配置を知るためのものです。権限設計や所有者の意味は、第 5 章で詳しく扱います。

3.6 ファイル検索

Linux では、場所を推測して探すより、コマンドで探す方が早い場面が多くあります。

3.6.1 find

find の基本

```
$ find ~/linux-first-lab -name "*.txt"
$ find ~/linux-first-lab -type d
$ find ~/linux-first-lab -name "hosts.*"
```

`find` は名前、種類、更新時刻などで対象を探せます。最初は `-name` と `-type` を押さえれば十分です。

3.6.2 which

which

```
$ which bash
$ which systemctl
$ which ssh
```

`which` は、コマンド本体がどこにあるかを確認するために使います。コマンドが見つからないときに、「そもそも入っているか」を確かめる入口として便利です。

3.7 Windows との見え方の違い

ファイルシステムで特に戸惑いやすい違いは次の 3 つです。

- ドライブレターがない
- パス区切りが \ ではなく /
- 拡張子よりも配置場所と権限が重要

たとえば Windows では「このファイルはどのドライブにあるか」と考えがちですが、Linux では「このファイルはツリーのどこに置かれているか」と考えます。この考え方に切り替わると、設定ファイルやログの所在が理解しやすくなります。

確認課題

- `/linux-first-lab/sample-app` 配下に `config`、`logs`、`data` を作成し、`ls -R` で確認してください。
- `/etc/hosts` を練習用ディレクトリへコピーし、`mv` で名前変更し、不要になったら削除してください。
- `/opt/sample-app/config` のような絶対パスと、`sample-app/config` のような相対パスの違いを、自分の言葉で説明してください。

まとめ

本章では、Linux のディレクトリ構造とパスの考え方を整理しました。重要なのは、`/etc`、`/var/log`、`/opt`、`/home` の役割を地図として持つことです。

次章では、その地図の上にある設定ファイルやログを読み、検索し、必要最小限の編集を行う方法を学びます。

第 4 章テキスト処理とエディタ

本章では、Linux 運用で避けて通れないテキストファイルの読み書きを扱います。設定確認、ログ確認、簡単な編集作業に必要なコマンドを、小さな手順に分けて学びます。第 3 章で整理したディレクトリの地図を使いながら、今度はその中身を読む段階へ進みます。

Windows との対応

Windows では GUI の設定画面、イベントビューア、メモ帳やエディタを行き来しながら情報を見る場面が多くあります。Linux では、その役割を `cat`、`less`、`grep`、`sed`、`vi` といったテキスト中心の道具が担います。

4.1 Linux では設定がテキストで見える

Windows では GUI の設定画面やレジストリを介して情報を扱うことが多くあります。一方 Linux では、多くの設定がテキストファイルとして置かれています。これは最初は地味に見えても、運用のしやすさに直結します。

- 内容をそのまま読める
- 差分を取りやすい
- スクリプトで加工しやすい
- バックアップしやすい

4.2 練習用の設定ファイルを作る

以後の例では、練習用に単純なアプリ設定ファイルを作って扱います。次の例は、そのまま端末へ貼り付けて作って構いません。最後の `EOF` まで入力すると、まとめてファイルが作られます。

設定ファイル作成

```
$ mkdir -p ~/linux-first-lab/sample-app/config
$ cd ~/linux-first-lab/sample-app/config
$ cat > app.conf << 'EOF'
# sample-app configuration
host=linux-lab
```

```
port=8080
log_level=INFO
data_dir=/opt/sample-app/data
log_dir=/var/log/sample-app
run_user=appsvc
EOF
$
$ cat > app.log << 'EOF'
2026-03-21 09:00:00 INFO service started
2026-03-21 09:05:12 WARN response delayed
2026-03-21 09:07:31 ERROR database timeout
2026-03-21 09:08:02 INFO retry succeeded
EOF
```

4.3 ファイル内容を読む

4.3.1 cat

cat

```
$ cat app.conf
$ cat -n app.conf
```

cat は短いファイルの全体像を見るのに向いています。-n を付けると行番号が出るので、「何行目を修正するか」を把握しやすくなります。

4.3.2 less

less

```
$ less app.log
```

less は長いファイルをページ送りしながら読むコマンドです。ログ確認では特に重要です。最低限、次の操作だけ覚えておけば十分です。

- Space: 次のページ
- b: 前のページ
- /文字列: 検索
- n: 次の検索結果
- q: 終了

4.3.3 head と tail

head と tail

```
$ head -n 3 app.conf
$ tail -n 2 app.log
$ tail -f app.log
```

head は先頭、tail は末尾を表示します。特に tail -f は、ログが追記される様子をリアルタイムで追うときに使います。終わるときは Ctrl+C を押します。

4.4 検索と抽出

4.4.1 grep

grep

```
$ grep "port" app.conf
$ grep "log" app.conf
$ grep -n "ERROR" app.log
$ grep -i "info" app.log
```

grep は「この文字列を含む行を探す」ための基本コマンドです。設定値を見つけるときも、ログの異常行を探すときも、入口になることが多くあります。

4.4.2 awk

awk

```
$ awk -F= '/port/ {print $2}' app.conf
$ awk '{print $1, $2, $3}' app.log
```

awk は列の抽出や整形に使います。最初は「区切り文字を決めて、欲しい列だけ出す」程度で十分です。

抽出結果の例

```
$ grep -n "ERROR" app.log
3:2026-03-21 09:07:31 ERROR database timeout
$ awk -F= '/port/ {print $2}' app.conf
8080
```

4.5 置換と一括変更

4.5.1 sed

sed

```
$ sed 's/INFO/DEBUG/' app.conf
$ cp app.conf app.conf.bak
$ sed -i 's/port=8080/port=8081/' app.conf
$ grep "port" app.conf
```

sed は置換処理に便利ですが、いきなり上書きすると危険です。最初は次の順番を守ってください。

1. 置換結果を画面表示で確認する
2. バックアップを作る
3. 上書き置換する
4. grep や diff で確認する

4.6 パイプとリダイレクト

Linux のコマンドは、単体でも使えますが、つなげると力を発揮します。

4.6.1 パイプ

パイプ

```
$ cat app.log | grep "ERROR"
$ cat app.conf | grep "dir" | awk -F= '{print $2}'
```

| は、前のコマンドの出力を次のコマンドへ渡します。最初は「一覧を出す」「必要な行だけ絞る」「必要な列だけ抜く」という 3 段階で考えると理解しやすくなります。

4.6.2 リダイレクト

リダイレクト

```
$ grep "dir" app.conf > config-paths.txt
$ cat config-paths.txt
$ grep "timeout" app.log >> config-paths.txt
```

> は上書き、>> は追記です。調査結果を一時ファイルに残すときによく使います。

4.7 エディタを使う

4.7.1 vi の最低限

vi の基本

```
$ vi app.conf
```

vi では、まず次の 3 つだけ覚えれば実務の入口には立てます。

- i: 挿入モードへ入る
- Esc: コマンドモードへ戻る
- :wq: 保存して終了
- :q!: 保存せず終了

4.7.2 nano

nano

```
$ nano app.conf
```

nano は画面下に操作ヒントが出るため、初学者にはとっつきやすいエディタです。環境によっては入っていないこともありますが、その場合は vi か vim を使います。ただし実務では vi 系が入っている前提で話が進むことが多いため、最終的には両方に触れておくのが無難です。

4.8 変更前後を確認する

編集作業では、変えたつもりで変わっていない、別の値まで変わっていた、という事故が起きます。確認には diff が便利です。

diff

```
$ cp app.conf app.conf.before
$ sed -i 's/log_level=INFO/log_level=DEBUG/' app.conf
$ diff app.conf.before app.conf
```

小さな変更でも、変更前後を比較する習慣を持っておくと、トラブル時の切り戻しが楽になります。

確認課題

- app.conf と app.log を作成し、cat -n、less、tail -f を試してください。
- grep -n "ERROR" app.log を実行し、どの行のエラーを拾っているか確認してください。

- 続けて `awk -F= '/port/ {print $2}' app.conf` を実行し、どの値だけを抜き出しているか説明してください。
- `port=8080` を別の値へ変更し、`diff` で変更前後の差分を確認してください。

まとめ

本章では、設定ファイルやログを読む、検索する、必要最小限だけ編集するための基本操作を学びました。Linux のテキスト処理は、難しい加工を覚える前に「まず読む」「必要な行だけ絞る」「変更後を確認する」という順番を守ることが重要です。

次章では、ユーザー、グループ、権限の考え方を学び、なぜ読めるファイルと読めないファイルがあるのかを理解します。

第 5 章ユーザー・グループ・パーミッション

本章では、Linux の権限管理を扱います。Windows の管理者権限の感覚と似ている部分、違う部分を整理しながら、ユーザー、グループ、ファイル権限を読み取る力を身につけます。第 4 章で触れた設定ファイルやログが「なぜ読めるのか、なぜ書けないのか」を理解する章でもあります。

Windows との対応

Windows で管理者権限や共有アクセス権を意識するのに近い章です。ただし Linux では、「所有者」「グループ」「その他」の 3 つに分けて、より明示的に権限を読みます。

実行前の前提

本章の一部の演習では sudo が必要です。共有サーバではなく、自分で管理できる検証環境で試すことを前提にしてください。sudo が使えない環境では、sudo 付きの例は読み物として追い、自分のホーム配下でできる ls -l や chmod の確認を優先してください。

5.1 root と一般ユーザー

Linux では、管理者権限を持つ特別なユーザーが root です。Windows のローカル管理者やドメイン管理者に近い感覚で考えられますが、Linux では root を普段使いしないことが強く意識されます。

ユーザー確認

```
$ whoami
$ id
$ ls -l /etc/shadow
```

/etc/shadow のような重要ファイルは、一般ユーザーでは読めません。これは不便さではなく、事故や侵害の範囲を小さくするための基本設計です。

5.2 sudo の考え方

sudo は、一時的に別ユーザーの権限でコマンドを実行する仕組みです。多くの環境では root 権限で実行するために使います。

sudo の基本

```
$ sudo whoami
root
$ sudo -l
```

sudo -l では、自分がどのコマンドを昇格実行できるかを確認できます。環境によってはパスワードを求められたり、許可がないと表示されたりしますが、それも権限設計の結果です。

大切なのは、「シェル全体を root に切り替える」よりも、「必要な 1 コマンドだけ sudo を付ける」方が安全だという点です。たとえばサービス再起動、設定ファイルの配置、ユーザー追加のような操作だけに権限昇格を限定します。

5.3 権限の読み方

5.3.1 ls -l の見方

権限表示

```
$ cd ~/linux-first-lab
$ touch permission-demo.txt
$ ls -l permission-demo.txt
-rw-r--r--  1 student  student  0 Mar 21 10:00 permission-demo.txt
```

先頭の -rw-r--r-- は、次の 4 つに分けて読みます。

- 1 文字目: ファイル種別
- 2 から 4 文字目: 所有者の権限
- 5 から 7 文字目: グループの権限
- 8 から 10 文字目: その他の権限

r は読み取り、w は書き込み、x は実行です。

5.3.2 数値表現

- r = 4
- w = 2
- x = 1

したがって、rw- は 6、r-- は 4、rwx は 7 になります。

数値指定

```
$ chmod 600 permission-demo.txt
$ ls -l permission-demo.txt
-rw-------  1 student  student  0 Mar 21 10:00 permission-demo.txt
```

5.4 所有者とグループ

Linux のアクセス制御では、「誰のファイルか」と「どのグループのファイルか」が重要です。

所有者確認

```
$ ls -l permission-demo.txt
$ id
$ grep "^$(whoami):" /etc/passwd
```

一般ユーザーで作成したファイルは、通常は自分自身が所有者になります。そこへ別ユーザーや運用グループをどう関与させるかが、サーバ運用の設計になります。

5.5 アカウント情報をどう読むか

ユーザーとグループは、単に名前だけで管理されているわけではありません。Linux では、ユーザー情報、グループ情報、パスワード関連情報が別の場所に分かれています。

アカウント情報の確認

```
$ getent passwd appsvc
$ getent group appops
$ grep "^$(whoami):" /etc/passwd
$ sudo grep "^root:" /etc/shadow
```

- `/etc/passwd`: ユーザー名、UID、ホームディレクトリ、ログインシェル
- `/etc/group`: グループ名と所属メンバー
- `/etc/shadow`: パスワード関連情報。通常は root 権限でのみ読める

`getent` を使うと、ローカルファイルだけでなく、環境によっては外部の認証基盤も含めた見え方で確認できます。最初は `/etc/passwd` を直接読むより、「まず `getent` を使う」と覚える方が実務で迷いにくくなります。

5.6 chmod, chown, chgrp

5.6.1 chmod

chmod

```
$ chmod 644 permission-demo.txt
$ chmod 600 permission-demo.txt
$ ls -l permission-demo.txt
```

chmod は「誰が何をできるか」を変えるコマンドです。

5.6.2 chown

chown

```
$ sudo chown root permission-demo.txt
$ ls -l permission-demo.txt
```

chown は所有者を変えるコマンドです。後半の例では、appsvc のようなサービス実行ユーザーへ寄せる場面でも使います。

5.6.3 chgrp

chgrp

```
$ sudo chgrp root permission-demo.txt
$ ls -l permission-demo.txt
```

chgrp はグループを変えるコマンドです。運用では、後半のように appops のような運用グループへ寄せる設計でも使います。

5.7 業務アプリのユーザー設計

本書の共通題材では、次の役割を置きます。

- appsvc: アプリ実行用のサービスアカウント
- appops: 運用担当者のグループ

ユーザーとグループ作成例

```
$ sudo groupadd appops
$ sudo useradd -r -m -g appops -s /usr/sbin/nologin appsvc
$ id appsvc
$ getent group appops
```

環境によっては /usr/sbin/nologin の代わりに別のログイン禁止シェルを使うことがあります。重要なのは、サービス用アカウントを人が普段ログインする用途と分けることで

す。

5.8 アプリ配置先の権限を考える

/opt/sample-app を例に、役割ごとに権限を考えてみます。

権限設定例

```
$ sudo chown -R appsvc:appops /opt/sample-app
$ sudo chmod 755 /opt/sample-app
$ sudo chmod 750 /opt/sample-app/config
$ sudo chmod 770 /opt/sample-app/data
$ sudo mkdir -p /var/log/sample-app
$ sudo chown -R appsvc:appops /var/log/sample-app
```

ここでの意図は次のとおりです。

- アプリ本体は読み取り中心なので 755
- 設定ファイルは広く読ませたくないので 750
- データやログは実行ユーザーと運用グループで管理する

5.9 Windows ACL との違い

Windows では、ユーザーやグループごとに細かく許可と拒否を積み上げる ACL に触れることが多くあります。一方 Linux の入口では、まず次の 3 つで考える方が整理しやすくなります。

- 所有者には何を許すか
- グループには何を許すか
- その他には何を許すか

Linux にも拡張 ACL はありますが、本書の段階では基本パーミッションを確実に読めることの方が重要です。最初から細かな例外規則に進むより、ls -l の 1 行を正しく読めることを優先してください。

まずは基本権限を優先

Windows の ACL に慣れていると、「もっと細かく制御したい」と考えがちです。しかし Linux 初学段階では、所有者、グループ、その他の 3 区分で十分に整理できる場面が大半です。例外的な制御は、基本権限で足りないと分かってから検討する方が安全です。

5.10 よくある失敗

- root のままアプリを起動してしまう
- 設定ファイルを 777 にしてしまう
- ログディレクトリの所有者がずれて書き込めない

- sudo を付け忘れて失敗し、原因を権限以外だと思い込む
  困ったときは、まず次の 3 つを見ます。

切り分けの基本

```
$ whoami
$ ls -ld /opt/sample-app /opt/sample-app/config /var/log/sample-app
$ ls -l /opt/sample-app/config
```

確認課題

- `ls -l` の 1 行を読み、所有者、グループ、権限を分けて説明してください。
- 練習用ファイルに対して `chmod 600` と `chmod 644` を試し、表示の違いを確認してください。
- `/opt/sample-app` と `/var/log/sample-app` を例に、`appsvc` と `appops` にどの権限を持たせたいか整理してください。

まとめ

本章では、Linux の権限管理の基本を整理しました。重要なのは、root を常用しないこと、`ls -l` を読めること、そして「所有者」「グループ」「権限」を分けて考えることです。

次章では、Linux でソフトウェアを導入し、更新し、削除するためのパッケージ管理を学びます。

第 6 章パッケージ管理とソフトウェアインストール

本章では、Linux でソフトウェアを導入し、更新し、削除する基本手順を整理します。Windows のインストーラー文化との違いを押さえつつ、パッケージ管理の考え方を理解します。第 5 章で整理した実行ユーザーや配置先の考え方を前提に、今度は「どう入れるか」を見ていきます。

Windows との対応

Windows では、セットアッププログラムを個別に入手して実行する流れが中心です。Linux では、OS が用意するパッケージ管理の仕組みを通して、検索、導入、更新、削除をまとめて扱うのが基本です。

6.1 パッケージ管理とは何か

Windows では、Web サイトからインストーラーをダウンロードして実行する流れが一般的です。Linux でも配布物を直接展開する方法はありますが、日常的にはパッケージマネージャを使って導入する方が基本です。

パッケージマネージャを使う利点は次のとおりです。

- 依存関係をまとめて解決できる
- インストール済みソフトを一覧管理できる
- 更新と削除の手順が揃う
- リポジトリと署名により真正性を確認しやすい

6.2 まず自分の環境を知る

Linux では、ディストリビューションによって標準のパッケージマネージャが異なります。最初に、自分の環境がどちらの系統かを確認します。

環境確認

```
$ cat /etc/os-release
$ which dnf
$ which apt
$ which rpm
$ which dpkg
```

大まかな目安は次のとおりです。

- RHEL 系、Rocky、AlmaLinux、Amazon Linux: `dnf` または `yum`
- Ubuntu、Debian 系: `apt`

`dnf` と `apt` は日常的な導入、更新、削除に使う高水準のコマンドです。一方 `rpm` と `dpkg` は、パッケージ情報を直接確認するときに使う低水準のコマンドだと考えると整理しやすくなります。

自分の系統だけ追えばよい

RHEL 系ならこの後の `dnf` 節、Ubuntu / Debian 系なら `apt` 節を主に読めば十分です。もう片方は比較用として流し読みし、まずは自分の環境で使う経路を 1 本持つことを優先してください。

6.3 RHEL 系での基本操作

6.3.1 情報更新

dnf makecache

```
$ sudo dnf makecache
```

`dnf` では、まずリポジトリのメタデータを更新します。Windows Update のカタログ同期に近い感覚で捉えると分かりやすいでしょう。

6.3.2 検索と詳細確認

dnf search

```
$ dnf search tree
$ dnf info tree
```

検索結果で見る点

検索結果では、左側にパッケージ名、右側に短い説明が出ます。最初は「目的の名前があるか」「説明が意図に合っているか」だけ見れば十分です。

検索結果の例

```
$ dnf search tree
================ Name Exactly Matched: tree ================
tree.x86_64 : File system tree viewer
```

6.3.3 インストールと削除

dnf install

```
$ sudo dnf install -y tree
$ tree --version
$ sudo dnf remove -y tree
```

6.3.4 更新

dnf upgrade

```
$ dnf check-update
$ sudo dnf upgrade
```

6.4 Ubuntu / Debian 系での基本操作

6.4.1 情報更新

apt update

```
$ sudo apt update
```

6.4.2 検索と詳細確認

apt search

```
$ apt search tree
$ apt-cache show tree
```

APT 系でも、まずはパッケージ名と短い説明だけ読めれば十分です。最初の 1 回は、名前が `tree` で、説明が「ディレクトリ構造を表示する道具」になっていることを確認してください。

6.4.3 インストールと削除

apt install

```
$ sudo apt install -y tree
$ tree --version
$ sudo apt remove -y tree
```

6.4.4 更新

apt upgrade

```
$ apt list --upgradable
$ sudo apt upgrade
```

6.5 依存関係を見る

Linux のパッケージ管理で重要なのが依存関係です。単体のソフトに見えても、背後では複数のライブラリや補助パッケージが必要になります。

依存関係の例

```
$ apt-cache depends bash
$ rpm -q --requires bash
```

Ubuntu / Debian 系では `apt-cache depends`、RHEL 系では既に導入済みのパッケージに対して `rpm -q --requires` のような形で要件を確認できます。環境によって見える情報は違いますが、少なくとも「1 つのソフトを入れると他のパッケージも一緒に入る」という感覚は持っておくべきです。これを手作業で管理しなくてよいことが、パッケージマネージャの大きな価値です。

6.6 配布物を直接置く場合との違い

業務システムでは、ベンダー配布物や社内ビルド成果物を `/opt/sample-app` に直接配置する場面もあります。この場合でも、次の観点はパッケージ管理と共通です。

- OS が対応しているか
- 必要ライブラリがあるか
- 実行ユーザーをどうするか
- サービス登録をどうするか
- ログと設定ファイルの置き場をどうするか

6.7 導入前に確認したいこと

sample-app のような業務アプリを導入する前には、少なくとも次の情報を見ておくと安全です。

導入前の確認

```
$ cat /etc/os-release
$ uname -m
$ free -h
$ df -h
$ id appsvc
$ ls -ld /opt /var/log
```

ここで見ているのは、OS 種類、CPU アーキテクチャ、メモリ、ディスク、実行ユーザーの有無、配置先ディレクトリの前提です。製品ごとの細かい要件に入る前に、この基本情報を押さえるだけで導入トラブルを減らせます。

6.8 リポジトリをむやみに増やさない

学習を進めていると、検索結果から外部リポジトリを追加したくなることがあります。しかし本番環境では、出所の曖昧なリポジトリを増やすと、更新経路や依存関係が読みにくくなります。

標準リポジトリ優先

最初の原則は単純です。標準リポジトリで足りるかを確認し、足りない場合も「なぜ追加するのか」「誰が保守するのか」を明確にしてから増やします。

6.9 この章の到達点

本章の時点で、次のことができれば十分です。

- 自分の環境で dnf と apt のどちらを使うか判断できる
- パッケージを検索し、情報を確認できる
- パッケージを導入、更新、削除できる
- 配布物を直接配置する場合でも、事前に見るべき観点を持てる

確認課題

- cat /etc/os-release と which dnf、which apt を実行し、自分の環境で使うパッケージマネージャを判断してください。
- tree などの小さなパッケージを対象に、検索、詳細確認、導入、削除の流れを手元で試してください。
- sample-app を入れる前提で、OS、CPU、メモリ、ディスク、実行ユーザー、配置

先の 6 観点を確認できるかチェックしてください。

まとめ

本章では、Linux のパッケージ管理の基本を整理しました。重要なのは、コマンドの細部を暗記することよりも、「OS を確認する」「検索する」「情報を見る」「導入する」「更新する」「不要なら削除する」という流れを持つことです。

次章では、Linux 上で動いているプロセスとサービスをどう見て、どう扱うかを学びます。

第 7 章プロセス管理とサービス制御

本章では、Linux サーバ上で動いているプログラムをどう見て、どう扱うかを学びます。単発のコマンド実行と、常駐するサービスの違いを理解することが主眼です。第 6 章でソフトウェアを入れる流れを見たので、ここからは「入れたものが実際にどう動くか」を確認します。

Windows との対応

Windows でいえば、タスクマネージャーでプロセスを見て、サービス管理ツールで常駐サービスを扱う感覚に近い章です。Linux では、その役割を `ps`、`top`、`systemctl`、`journalctl` が担います。

7.1 プロセスとは何か

プロセスとは、実行中のプログラムです。Windows のタスクマネージャーで見ているものと同じ概念ですが、Linux ではコマンドで状態を確認することが多くあります。

まずは、現在動いているプロセスの一覧を見てみます。

ps

```
$ ps aux
```

出力には主に次の情報が含まれます。

- `USER`: 実行ユーザー
- `PID`: プロセス ID
- `%CPU`: CPU 使用率
- `%MEM`: メモリ使用率
- `COMMAND`: 実行コマンド

7.2 目的のプロセスを見つける

運用では、一覧全部を見るより、目的のプロセスだけを探すことの方が多くあります。

検索

```
$ ps -ef | grep sample-app
$ pgrep -af sample-app
$ pgrep -af ssh
```

pgrep -af は、PID とコマンドラインをまとめて見られるので便利です。最初は ps と grep の組み合わせでも十分ですが、目的が明確なら pgrep の方が速い場面もあります。

grep で自分自身も見えることがある

ps -ef | grep sample-app では、grep 自身の行が出ることがあります。最初は不思議に見えても問題ありません。目的のプロセスだけを見たいときは、pgrep -af の方が読みやすい場面があります。

7.3 リアルタイムで状態を見る

7.3.1 top

top

```
$ top
```

top は、CPU やメモリの使用状況をリアルタイムで確認するコマンドです。最低限、次の見方を覚えてください。

- 上部: 全体の負荷、メモリ状況
- 中央以降: 各プロセスの一覧
- P: CPU 使用率順に並び替え
- M: メモリ使用率順に並び替え
- q: 終了

7.3.2 htop

htop

```
$ htop
```

htop は色分けやスクロールがあり、top より見やすいことが多いツールです。入っていない場合は無理に追加しなくても構いませんが、検証環境では使ってみる価値があります。

7.4 プロセスを止める

異常なプロセスや、テスト用に起動したプロセスを終了したい場合は kill を使います。

テストプロセス

```
$ sleep 300 &
$ jobs
$ pgrep -af sleep
$ kill <PID>
```

通常は kill だけで十分です。これは強制終了ではなく、終了を依頼するシグナルを送っています。どうしても止まらない場合だけ、強いシグナルを使います。

強制終了

```
$ kill -9 <PID>
```

強制終了は最後

kill -9 は便利ですが、後始末の機会を与えずに止めます。まず通常の kill を試し、それでも止まらないときだけ使う方が安全です。

7.5 優先度とバックグラウンド実行

7.5.1 nice

CPU を使う処理の優先度を少し下げたいときは nice を使います。

nice

```
$ nice -n 10 long-running-command
```

数値が大きいほど優先度は低くなります。長時間かかる集計や一時的な検証処理を、本番サービスより控えめに動かしたいときに使います。

7.5.2 バックグラウンドと nohup

バックグラウンド

```
$ ./long-task.sh &
$ jobs
$ nohup ./long-task.sh > /tmp/long-task.log 2>&1 &
```

& はバックグラウンド実行、nohup はログアウトしても処理を続けたいときに使います。ただし、常駐サービスの管理には後述の systemd を使う方が基本です。

7.6 systemd とサービス管理

Linux では、常駐プログラムを「サービス」として管理することが一般的です。多くのディストリビューションでは、その管理に systemd が使われます。ここでの sample-app は本書の共通題材です。まだそのサービスが存在しない環境では、実際に入っているサービス名へ読み替えて構いません。

sample-app がまだないとき

手元に sample-app がない場合は、まず systemctl list-units --type=service --state=running で実在するサービスを 1 つ見つけ、その名前で status や journalctl -u を試してください。共有環境では start や stop は読み物として把握するだけでも十分です。

systemctl の基本

```
$ systemctl status sample-app
$ sudo systemctl start sample-app
$ sudo systemctl stop sample-app
$ sudo systemctl restart sample-app
```

最初に覚えるべき操作は、この 4 つで十分です。特に status は、障害時の入口として最もよく使います。

status の出力例

```
$ systemctl status sample-app
● sample-app.service - Sample App Service
     Loaded: loaded (/etc/systemd/system/sample-app.service; enabled)
     Active: active (running) since Sat 2026-03-21 10:20:11 JST; 8s ago
   Main PID: 2145 (python3)
```

status でまず見る点

- Loaded で定義ファイルが読めているか
- Active が active (running) か
- Main PID が出ていて、実プロセスにつながっているか

7.7 自動起動

サーバ再起動後も自動的にサービスを起動したい場合は、enable を使います。

自動起動

```
$ sudo systemctl enable sample-app
$ systemctl is-enabled sample-app
$ sudo systemctl disable sample-app
```

enable は今すぐ起動する操作ではなく、次回ブート時に起動するよう登録する操作です。今すぐ起動もしたい場合は enable --now を使います。

7.8 ユニットファイルの見方

サービス定義は、ユニットファイルに書かれています。最初から全部を覚える必要はありません。まずは主要項目の役割だけ押さえます。

ユニットファイル例

```
[Unit]
Description=Sample App Service
After=network.target

[Service]
Type=simple
User=appsvc
WorkingDirectory=/opt/sample-app
ExecStart=/opt/sample-app/bin/start.sh
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- User: 実行ユーザー
- WorkingDirectory: 作業ディレクトリ
- ExecStart: 起動コマンド
- Restart: 失敗時の再起動方針

7.9 ログを見る

サービスの状態確認では、status と並んで journalctl が重要です。

journalctl

```
$ journalctl -u sample-app -n 50
$ journalctl -u sample-app -f
$ journalctl -u sample-app --since "1 hour ago"
```

`-n 50` は直近 50 行、`-f` は追尾、`--since` は時間範囲指定です。`-f` で追尾中に抜けるときは `Ctrl+C` を押します。テキストログが `/var/log/sample-app` に出る構成なら、`tail -f` と併用して確認します。

journal の出力例

```
$ journalctl -u sample-app -n 3
Mar 21 10:20:11 linux-lab systemd[1]: Started Sample App Service.
```

7.10 障害時の最初の切り分け

`sample-app` が動かないときは、次の順番で見ると整理しやすくなります。

切り分けの順番

```
$ systemctl status sample-app
$ journalctl -u sample-app -n 50
$ pgrep -af sample-app
$ ls -ld /opt/sample-app /var/log/sample-app
```

見ている観点は、状態、ログ、実プロセス、権限の 4 つです。いきなり設定ファイルを触るのではなく、まず現状を読む方が安全です。

確認課題

- `ps aux` と `pgrep -af` を使って、目的のプロセスを探してください。
- `sleep 300 &` でテスト用プロセスを起動し、`kill` で終了させてください。
- `systemctl status`、`journalctl -u`、`pgrep -af`、権限確認の 4 つを、障害時にどの順で使うか説明してください。

まとめ

本章では、Linux のプロセスとサービス管理の基本を学びました。重要なのは、単発のプロセスと常駐サービスを分けて考えること、そして障害時に `status`、`journalctl`、`pgrep` を使って順に確認することです。

次章では、ネットワーク設定と通信確認の基本を学びます。

第 8 章ネットワーク設定と確認

本章では、Linux サーバのネットワーク状態を確認し、通信トラブルの入口を切り分けるための基本を扱います。IP アドレス、名前解決、ポート、ファイアウォールの関係を整理します。第 7 章でサービスが動いているかを見たので、ここからは「動いているのに届かない」を切り分けます。

8.1 最初に見るべきネットワーク情報

Linux サーバに入って最初に見るべき情報は、インターフェース、経路、待受ポートです。

基本確認

```
$ ip addr show
$ ip route show
$ ss -tlnp
```

`ip addr show` はインターフェースと IP アドレス、`ip route show` はルーティング、`ss -tlnp` は TCP 待受ポートを表示します。

8.2 Windows との対応で考える

Windows 技術者にとっては、次の対応で考えると整理しやすくなります。

- `ipconfig` に近いもの: `ip addr show`
- `route print` に近いもの: `ip route show`
- 待受ポート確認: `ss -tlnp`

Linux では、設定確認の粒度が少し細かく分かれているだけです。何を見るかの発想自体は大きく変わりません。

8.3 名前解決

8.3.1 /etc/hosts

hosts

```
$ cat /etc/hosts
127.0.0.1   localhost
192.168.1.20 linux-lab
```

/etc/hosts は、DNS を使わずにローカルでホスト名を解決するためのファイルです。検証環境や一時的な切り分けで使うことがあります。

8.3.2 DNS の確認

DNS 確認

```
$ cat /etc/resolv.conf
$ getent hosts linux-lab
$ nslookup linux-lab
```

getent hosts は、システムの名前解決設定に従って結果を返すので、実際の挙動確認に向いています。nslookup は DNS サーバに問い合わせる確認です。環境によっては nslookup が入っていないこともあるため、最初は getent hosts を主役に考えると迷いにくくなります。

8.4 疎通確認

8.4.1 ping

ping

```
$ ping -c 4 linux-lab
```

ping はホストへの到達性を確認します。ただし、相手が応答しない設定でもサービス自体は動いている場合があるため、これだけで判断しないことが重要です。

8.4.2 nc

netcat

```
$ nc -zv linux-lab 8080
```

nc は、特定ポートへの接続可否を確認できます。本書の共通題材では、sample-app が 8080 で待ち受ける例を使います。

8.4.3 curl

curl

```
$ curl -I http://linux-lab:8080
```

HTTP 系のサービスであれば、curl でヘッダだけ確認するのが手早い方法です。

成功時の見え方

```
$ ping -c 2 linux-lab
64 bytes from linux-lab: icmp_seq=1 ttl=64 time=0.4 ms
$ nc -zv linux-lab 8080
Connection to linux-lab port 8080 [tcp/http-alt] succeeded!
$ curl -I http://linux-lab:8080
HTTP/1.0 200 OK
```

8.5 待受ポートを確認する

「サービスは起動したはずなのに接続できない」ときは、まず待受ポートを見ます。

待受確認

```
$ ss -tlnp | grep 8080
$ sudo lsof -iTCP:8080 -sTCP:LISTEN
```

ここで確認したいのは、次の 3 点です。

- 本当にポートが開いているか
- どのアドレスで待ち受けているか
- どのプロセスが使っているか

見え方が違うとき

ss -tlnp のプロセス情報は、権限がないと一部見えないことがあります。lsof も環境によっては入っていないため、最初は ss を基本にし、必要なときだけ補助的に使うと考えるのが安全です。

8.6 ファイアウォールの見方

待受ポートがあっても接続できない場合、ファイアウォールで止められている可能性があります。

8.6.1 RHEL 系

firewalld

```
$ sudo firewall-cmd --state
$ sudo firewall-cmd --list-ports
$ sudo firewall-cmd --permanent --add-port=8080/tcp
$ sudo firewall-cmd --reload
```

8.6.2 Ubuntu 系

ufw

```
$ sudo ufw status
$ sudo ufw allow 8080/tcp
```

ファイアウォール節の扱い

ファイアウォール設定は環境差と影響範囲が大きいため、共有環境や権限がない環境では `status` や一覧表示だけ確認し、ポート追加は読み物として追えば十分です。最初は「どこで遮断されるかを切り分ける発想」を持つことを優先してください。

SSH を閉じない

SSH で接続しているサーバでは、ファイアウォール設定変更前に SSH ポートが許可されているか確認してください。そこを閉じると、自分自身が入れなくなります。

8.7 SSH とファイル転送

Linux サーバ運用では、SSH が標準的な接続手段です。状態確認、設定変更、ログ取得の多くを SSH 経由で行います。

SSH

```
$ ssh student@linux-lab
$ ssh student@linux-lab "hostname"
```

ログや設定ファイルを手元に持ってきたい場合は、`scp` が使えます。

scp

```
$ scp student@linux-lab:/var/log/sample-app/app.log .
$ scp ./app.conf student@linux-lab:/tmp/
```

8.8 トラブル時の順番

通信トラブルでは、いきなり設定を触るよりも、順番に確認した方が速く解決します。

1. 相手ホスト名が正しいか
2. 名前解決できるか
3. IP 到達性があるか
4. サービスが起動しているか
5. ポートが待ち受けているか
6. ファイアウォールで止まっていないか

切り分け例

```
$ getent hosts linux-lab
$ ping -c 2 linux-lab
$ systemctl status sample-app
$ ss -tlnp | grep 8080
$ nc -zv linux-lab 8080
```

確認課題

- `ip addr show`、`ip route show`、`ss -tlnp` を実行し、IP、経路、待受ポートを 1 つずつ読み取ってください。
- `getent hosts linux-lab`、`ping`、`nc -zv`、`curl -I` を順に試し、どの層を確認しているか説明してください。
- 接続できないと仮定したとき、名前解決、ルーティング、起動状態、待受、ファイアウォールのどこを疑うかを順番に書き出してください。

まとめ

本章では、ネットワーク確認の基本を学びました。重要なのは、IP、名前解決、待受ポート、ファイアウォールを別々の層として切り分けることです。

次章では、こうした確認作業をまとめて自動化するためのシェルスクリプトを学びます。

第 9 章シェルスクリプト入門

本章では、繰り返し行う確認作業を小さく自動化するためのシェルスクリプトを学びます。難しい言語機能ではなく、変数、条件分岐、ループ、実行権限といった基本に集中します。第 8 章までで手で行ってきた確認を、ここからは再実行しやすい形へまとめます。

Windows との対応

Windows のバッチファイルや PowerShell スクリプトで繰り返し作業をまとめるのと同じ発想です。Linux では Bash を入口にし、必要になれば `cron` で定期実行へつなげます。

9.1 シェルスクリプトとは何か

シェルスクリプトは、ターミナルで毎回手で打っているコマンドを、1 つのファイルにまとめたものです。Windows のバッチファイルや PowerShell スクリプトに近い役割と考えれば十分です。

最初のスクリプト

```
$ cd ~/linux-first-lab
$ cat > hello.sh << 'EOF'
#!/bin/bash
echo "Hello from Bash"
echo "hostname: $(hostname)"
EOF
$ chmod +x hello.sh
$ ./hello.sh
```

実行結果例

```
$ ./hello.sh
Hello from Bash
hostname: linux-lab
```

ここでのポイントは 3 つです。

- 1 行目の `#!/bin/bash` は実行シェルの指定
- `chmod +x` で実行権限を付ける
- `./hello.sh` で実行する

9.2 変数

スクリプトでは、繰り返し使う値を変数に入れると読みやすくなります。

変数

```
$ cat > variables.sh << 'EOF'
#!/bin/bash
APP_HOME="/opt/sample-app"
APP_LOG="/var/log/sample-app/app.log"
PORT=8080

echo "APP_HOME=$APP_HOME"
echo "APP_LOG=$APP_LOG"
echo "PORT=$PORT"
EOF
$ chmod +x variables.sh
$ ./variables.sh
```

`=` の前後に空白を入れない点に注意してください。参照するときは `$ 変数名` または `${変数名}` を使います。

9.3 条件分岐

存在確認や状態確認には `if` がよく使われます。

if

```
$ cat > check-paths.sh << 'EOF'
#!/bin/bash
APP_HOME="/opt/sample-app"

if [ -d "$APP_HOME" ]; then
  echo "[OK] app directory exists"
else
  echo "[NG] app directory not found"
fi
EOF
$ chmod +x check-paths.sh
$ ./check-paths.sh
```

最初は次の判定だけ覚えておけば十分です。

- -f: ファイルか
- -d: ディレクトリか
- -r: 読み取り可能か
- -x: 実行可能か

9.4 複数条件の分岐

サービスのように、開始、停止、再起動、状態確認といった複数分岐には case が向いています。

case

```
$ cat > service-control.sh << 'EOF'
#!/bin/bash
ACTION="${1:-status}"

case "$ACTION" in
  start)
    echo "sudo systemctl start sample-app"
    ;;
  stop)
    echo "sudo systemctl stop sample-app"
    ;;
  restart)
    echo "sudo systemctl restart sample-app"
    ;;
  status)
    echo "systemctl status sample-app"
    ;;
  *)
    echo "usage: $0 {start|stop|restart|status}"
    exit 1
    ;;
esac
EOF
$ chmod +x service-control.sh
$ ./service-control.sh status
```

この例では、安全のため実際に systemctl を実行せず、何を打つべきかを表示するだけにしています。最初はこの形で分岐の書き方に慣れる方が安全です。

9.5 ループ

同じ確認を複数対象に対して行うときは、for が役に立ちます。

for

```
$ cat > check-ports.sh << 'EOF'
#!/bin/bash
for port in 22 80 8080; do
  echo "checking port $port"
done
EOF
$ chmod +x check-ports.sh
$ ./check-ports.sh
```

ログファイル一覧、ポート一覧、複数ホストの確認など、運用ではループが頻繁に出てきます。

9.6 実行権限と実行方法

スクリプトは、次の 2 通りの実行を押さえておけば十分です。

実行方法

```
$ ./hello.sh
$ bash hello.sh
```

`./hello.sh` は実行権限が必要、`bash hello.sh` は実行権限がなくても動きます。実務では、実行ファイルとして扱いたいスクリプトには `chmod +x` を付ける方が分かりやすくなります。

9.7 実用例 1: 状態確認スクリプト

ここまでの要素を組み合わせると、小さな確認スクリプトが作れます。

status-check.sh

```
$ cat > status-check.sh << 'EOF'
#!/bin/bash
APP_HOME="/opt/sample-app"
LOG_DIR="/var/log/sample-app"

echo "=== sample-app check ==="
echo "[1] host: $(hostname)"
echo "[2] user: $(whoami)"

if [ -d "$APP_HOME" ]; then
  echo "[3] app dir: OK"
else
```

```
  echo "[3] app dir: NG"
fi

if [ -d "$LOG_DIR" ]; then
  echo "[4] log dir: OK"
else
  echo "[4] log dir: NG"
fi

echo "[5] service:"
if systemctl list-unit-files --type=service 2>/dev/null | grep -q '^sample-app
    \.service'; then
  systemctl status sample-app --no-pager
else
  echo "sample-app.service is not installed yet"
fi
EOF
$ chmod +x status-check.sh
```

このスクリプトは、接続先、実行ユーザー、ディレクトリ、サービス状態をまとめて見ています。毎回同じ確認を手で打つよりも、順番が固定されるのが利点です。

実行結果例

```
$ ./status-check.sh
=== sample-app check ===
[1] host: linux-lab
[2] user: student
[3] app dir: NG
[4] log dir: NG
[5] service:
sample-app.service is not installed yet
```

第 10 章の前にこのスクリプトを試すと、まだ `sample-app` が存在しないため上のような結果になることがあります。それでも「何を確認するスクリプトか」を掴めれば十分です。

9.8 実用例 2: ログの抜粋

ログ調査では、「直近のエラーだけ見る」スクリプトもよく使います。

log-snapshot.sh

```
$ cat > log-snapshot.sh << 'EOF'
#!/bin/bash
LOG_FILE="/var/log/sample-app/app.log"
```

```
if [ -f "$LOG_FILE" ]; then
  echo "=== tail ==="
  tail -n 20 "$LOG_FILE"
  echo ""
  echo "=== errors ==="
  grep -i "error" "$LOG_FILE" | tail -n 10
else
  echo "log file not found: $LOG_FILE"
fi
EOF
$ chmod +x log-snapshot.sh
```

実行結果例

```
$ ./log-snapshot.sh
log file not found: /var/log/sample-app/app.log
```

こちらも第 10 章の前ではログファイルが存在しないことがあります。その場合は失敗ではなく、「まだ対象がない」と分かるだけでも十分です。

9.9 定期実行の入口

同じスクリプトを毎日動かしたい場合は、cron が入口になります。

cron

```
$ crontab -e
```

crontab -e を実行すると、環境で既定になっているエディタが開きます。第 4 章で触れた vi や nano の基本操作を思い出せば十分です。

たとえば毎朝 8 時に状態確認スクリプトを動かすなら、次のような 1 行になります。

crontab 例

```
0 8 * * * /home/student/linux-first-lab/status-check.sh > /tmp/status-check.
    log 2>&1
```

最初の段階では、時刻書式を暗記するよりも「繰り返し作業を定期実行へ移せる」という発想を持つことが重要です。

cron が使えないとき

学習環境によっては cron が動いていなかったり、crontab を使えなかったりします。その場合は、この節を概念理解として読み、まずはスクリプトを手動で実行して期待どおりの出力になることを優先してください。

9.10 よくある失敗

- シェバンを書いていない
- 実行権限がない
- 変数の = の前後に空白を入れている
- パスを相対指定して、実行場所によって壊れる
- 手動実行では動くが、cron では環境変数差で動かない
  迷ったときは、まず次を確認します。

確認ポイント

```
$ ls -l script.sh
$ bash -n script.sh
$ bash -x script.sh
```

bash -n は構文確認、bash -x は実行トレースです。スクリプトのデバッグでは特に有効です。

確認課題

- hello.sh と variables.sh を作成し、chmod +x と bash 実行の両方を試してください。
- ディレクトリ存在確認を行う if スクリプトか、開始と停止を切り替える case スクリプトを 1 本書いてください。
- 自作スクリプトに対して bash -n と bash -x を実行し、何を確認するコマンドか説明してください。

まとめ

本章では、シェルスクリプトの基本を学びました。重要なのは、高度な構文を覚えることではなく、普段手で打っている確認作業を「再実行できる形」にすることです。

次章では、ここまでの内容をまとめて使い、小さな業務サーバを運用する総合演習に進みます。

第 10 章小さな業務サーバを運用する総合演習

本章では、第 1 章から第 9 章までの内容をまとめて使います。新しい Linux サーバに入り、配置先を確認し、設定を読み、サービスを起動し、ログを確認し、必要なら簡単な切り分けを行うまでを一連の流れとして扱います。

Windows でたとえると

Windows でいえば、サービス登録、構成ファイル配置、イベントログやアプリログ確認、疎通確認をまとめて追う演習に近い章です。前半章で分けて学んだ確認を、ここで 1 本の運用手順としてつなげます。

10.1 シナリオ

次のような状況を想定します。

- サーバ名は `linux-lab`
- アプリは `/opt/sample-app` に配置する
- 実行ユーザーは `appsvc`
- ログは `/var/log/sample-app/app.log` に出す
- サービス名は `sample-app.service`
- HTTP 待受ポートは `8080`

目的は、高度な本番設計ではありません。Linux サーバに入ってから「何をどの順で確認し、どこを触るか」を一連の流れとして持つことです。

10.2 フェーズ 1 事前確認

最初に、接続先と作業ユーザー、OS、基本リソースを確認します。

事前確認

```
$ hostname
$ whoami
```

```
$ pwd
$ cat /etc/os-release
$ free -h
$ df -h
$ id appsvc
$ getent group appops
```

ここで見ているのは、接続先の取り違え防止、OS 種類、メモリ、ディスク、実行ユーザーの有無です。アプリに手を触れる前に、土台があるかを見ています。

この段階で見たいこと

- `hostname` が想定どおり `linux-lab` になっているか
- `whoami` が一般運用ユーザーであり、最初から root ではないか
- `df -h` で `/opt` や `/var` を置ける空き容量があるか
- `id appsvc` や `getent group appops` が失敗した場合は、第 5 章の手順でサービスユーザーと運用グループを先に作る必要があること

10.3 フェーズ 2 ディレクトリと設定

次に、アプリ配置先とログ配置先を作り、最小限の起動スクリプトと設定ファイルを置きます。

配置作成

```
$ sudo mkdir -p /opt/sample-app/bin
$ sudo mkdir -p /opt/sample-app/config
$ sudo mkdir -p /opt/sample-app/data
$ sudo mkdir -p /var/log/sample-app
$ sudo chown -R appsvc:appops /opt/sample-app /var/log/sample-app
```

起動スクリプト例

```
$ sudo tee /opt/sample-app/bin/start.sh > /dev/null << 'EOF'
#!/bin/bash
exec python3 -m http.server 8080 --directory /opt/sample-app/data \
  >> /var/log/sample-app/app.log 2>&1
EOF
$ sudo chmod 755 /opt/sample-app/bin/start.sh
```

設定ファイル例

```
$ sudo tee /opt/sample-app/config/app.conf > /dev/null << 'EOF'
host=linux-lab
port=8080
log_dir=/var/log/sample-app
run_user=appsvc
EOF
```

動作確認用ページ

```
$ sudo tee /opt/sample-app/data/index.html > /dev/null << 'EOF'
<html>
  <body>
    <h1>sample-app works</h1>
  </body>
</html>
EOF
```

ここでは説明のために単純な HTTP サーバを例にしています。本質は、配置先、設定、実行ユーザー、ログの置き場を分けて管理することです。サービスの起動状態は `journalctl`、HTTP アクセスのようなアプリ側のログは `/var/log/sample-app/app.log` で見る構成にします。

配置確認例

```
$ sudo ls -R /opt/sample-app
/opt/sample-app:
bin  config  data

/opt/sample-app/bin:
start.sh

/opt/sample-app/config:
app.conf

/opt/sample-app/data:
index.html
$ sudo cat /opt/sample-app/config/app.conf
host=linux-lab
port=8080
log_dir=/var/log/sample-app
run_user=appsvc
```

ここで空の `data` ディレクトリが見えていても問題ありません。まずは配置の形を作り、起動に必要な最小ファイルがそろっていることを確認します。

10.4 フェーズ 3 サービス登録

起動スクリプトがあるだけでは、再起動や状態確認が面倒です。そこで systemd へ登録します。

ユニットファイル

```
$ sudo tee /etc/systemd/system/sample-app.service > /dev/null << 'EOF'
[Unit]
Description=Sample App Service
After=network.target

[Service]
Type=simple
User=appsvc
Group=appops
WorkingDirectory=/opt/sample-app
ExecStart=/opt/sample-app/bin/start.sh
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOF
```

反映と起動

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now sample-app
$ systemctl status sample-app
```

この段階で重要なのは、起動したかどうかだけではありません。どのユーザーで動くか、どこを作業ディレクトリにするか、再起動方針をどうするか、といった運用上の基本も一緒に決まります。

status の出力例

```
$ systemctl status sample-app
● sample-app.service - Sample App Service
     Loaded: loaded (/etc/systemd/system/sample-app.service; enabled)
     Active: active (running) since Sat 2026-03-21 10:20:11 JST; 8s ago
   Main PID: 2145 (python3)
      Tasks: 1
     Memory: 9.8M
        CPU: 85ms
```

status の見方

- Loaded でユニットファイルが正しく読み込まれているかを見る
- Active: active (running) なら少なくとも現時点では起動している
- Main PID が出ていれば、実プロセスが結び付いていると分かる
- failed や activating のまま止まる場合は、次に journalctl -u を見る

10.5 フェーズ 4 ログとポートの確認

サービスが起動したら、次は「実際に待ち受けているか」を見ます。

状態確認

```
$ journalctl -u sample-app -n 50
$ ss -tlnp | grep 8080
$ curl -I http://localhost:8080
$ nc -zv localhost 8080
$ tail -n 5 /var/log/sample-app/app.log
```

status で起動済みに見えても、ポートが開いていない、直後に落ちている、といったケースはあります。だからこそ、ログとポート確認を続けて行います。

正常時の出力例

```
$ journalctl -u sample-app -n 3
Mar 21 10:20:11 linux-lab systemd[1]: Started Sample App Service.
$ ss -tlnp | grep 8080
LISTEN 0      5            0.0.0.0:8080       0.0.0.0:*    users:(("python3",
    pid=2145,fd=3))
$ curl -I http://localhost:8080
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.12.2
$ nc -zv localhost 8080
Connection to localhost port 8080 [tcp/http-alt] succeeded!
$ tail -n 5 /var/log/sample-app/app.log
127.0.0.1 - - [21/Mar/2026 10:21:08] "HEAD / HTTP/1.1" 200 -
```

ここでの判断

- journalctl に起動成功が出ているか
- ss で 0.0.0.0:8080 または想定アドレスの待受になっているか
- curl が 200 OK を返すか
- nc が接続成功になり、TCP レベルでも到達できるか
- app.log にアクセス記録が出ていて、アプリ側のログも取得できているか

journal とアプリログの役割

`journalctl -u sample-app` では、サービスが起動したか、失敗したかといった systemd 側の情報を見ます。実際の HTTP アクセスのようなアプリ側の記録は、ここでは `/var/log/sample-app/app.log` で確認します。

10.6 フェーズ 5 リモートからの確認

ローカルでは動いていても、別ホストから見えないことがあります。その場合は、名前解決と疎通を分けて確認します。

リモート確認

```
$ getent hosts linux-lab
$ ping -c 2 linux-lab
$ nc -zv linux-lab 8080
$ curl -I http://linux-lab:8080
```

ここで失敗した場合は、次のどこで止まっているかを切り分けます。

- 名前解決
- ルーティング
- サービス起動
- ポート待受
- ファイアウォール

リモート確認の見え方

```
$ getent hosts linux-lab
192.168.1.20    linux-lab
$ nc -zv linux-lab 8080
Connection to linux-lab port 8080 [tcp/http-alt] succeeded!
$ curl -I http://linux-lab:8080
HTTP/1.0 200 OK
```

localhost と外部到達は別

`localhost` では成功するのに `linux-lab` 宛てで失敗する場合は、アプリ自体よりも待受アドレス、経路、ファイアウォールを疑う方が筋がよい切り分けになります。

10.7 フェーズ 6 よくある初歩障害

10.7.1 権限で起動できない

権限確認

```
$ ls -ld /opt/sample-app /opt/sample-app/bin /var/log/sample-app
$ ls -l /opt/sample-app/bin/start.sh
$ sudo journalctl -u sample-app -n 20
```

実行ユーザーに読み取りや実行権限がないと、サービスは起動しません。

権限不備の例

```
$ sudo journalctl -u sample-app -n 5
Mar 21 10:28:04 linux-lab systemd[1]: sample-app.service: Failed at step EXEC
    spawning /opt/sample-app/bin/start.sh: Permission denied
Mar 21 10:28:04 linux-lab systemd[1]: sample-app.service: Main process exited,
     code=exited, status=203/EXEC
Mar 21 10:28:04 linux-lab systemd[1]: sample-app.service: Failed with result '
    exit-code'.
```

`Permission denied` や `203/EXEC` が見えたら、まず実行権限と所有者を疑います。設定内容の読み直しより、権限確認の方が先です。

10.7.2 ポート競合

ポート競合

```
$ ss -tlnp | grep 8080
$ sudo lsof -iTCP:8080 -sTCP:LISTEN
```

別プロセスが同じポートを使っている場合は、既存プロセスを止めるか、設定側のポートを変更する必要があります。

ポート競合の例

```
$ ss -tlnp | grep 8080
LISTEN 0      128          0.0.0.0:8080       0.0.0.0:*    users:(("nginx",pid
    =980,fd=6))
$ sudo journalctl -u sample-app -n 5
Mar 21 10:31:12 linux-lab python3[2250]: OSError: [Errno 98] Address already
    in use
```

`Address already in use` が出たら、設定ファイルより先に「誰がそのポートを使っているか」を調べます。これはネットワーク障害ではなく、起動条件の衝突です。

10.7.3 設定変更が反映されない

設定反映

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart sample-app
$ systemctl status sample-app
```

ユニットファイルを変えたのに反映されない場合は、daemon-reload を忘れていることがあります。

10.8 フェーズ 7 バックアップの最小単位

本格的なバックアップ設計は環境によりますが、最低限「設定」と「ログ」を別々に確保する発想は持っておくべきです。

最小バックアップ

```
$ mkdir -p ~/linux-first-lab/backup
$ sudo tar -czf ~/linux-first-lab/backup/sample-app-config.tar.gz \
    /opt/sample-app/config
$ sudo tar -czf ~/linux-first-lab/backup/sample-app-logs.tar.gz \
    /var/log/sample-app
$ ls -lh ~/linux-first-lab/backup
```

バックアップ確認例

```
$ ls -lh ~/linux-first-lab/backup
total 16K
-rw-r--r-- 1 student student 1.2K Mar 21 10:40 sample-app-config.tar.gz
-rw-r--r-- 1 student student 4.8K Mar 21 10:40 sample-app-logs.tar.gz
```

設定とログを分けておくと、切り戻ししたいのか、調査資料を残したいのかを後から選びやすくなります。最初から全部を 1 つに固めるより、用途ごとに分ける方が運用では扱いやすくなります。

10.9 フェーズ 8 最小リストアの考え方

バックアップを取っただけでは不十分です。必要になったときに、どこへ戻すのか、いきなり上書きしてよいのかを考えておく必要があります。

リストア前の確認

```
$ mkdir -p ~/linux-first-lab/restore-check
$ tar -tzf ~/linux-first-lab/backup/sample-app-config.tar.gz
$ tar -xzf ~/linux-first-lab/backup/sample-app-config.tar.gz \
    -C ~/linux-first-lab/restore-check
$ ls -R ~/linux-first-lab/restore-check
```

まずは中身を確認し、どのファイルが入っているかを把握します。バックアップファイルを持っていても、中身を理解しないまま上書きすると、直したい問題とは別の差分まで戻してしまうことがあります。

設定を戻す前の比較

```
$ diff -r ~/linux-first-lab/restore-check/opt/sample-app/config \
    /opt/sample-app/config
$ sudo cp /opt/sample-app/config/app.conf \
    /opt/sample-app/config/app.conf.before-restore
$ sudo cp ~/linux-first-lab/restore-check/opt/sample-app/config/app.conf \
    /opt/sample-app/config/app.conf
```

いきなり全体を上書きしない

最初はアーカイブ全体を戻すのではなく、どのファイルを戻したいのかを確認してから、必要な設定ファイルだけを差し戻す方が安全です。ログのバックアップは通常、調査資料として参照する用途が中心で、運用中のログディレクトリへそのまま戻す場面は多くありません。

10.10 フェーズ 9 日次運用の定例確認

総合演習が終わったあとに実務で繰り返すのは、派手な操作より定例確認です。毎朝あるいは変更作業後に、同じ順番で見るだけでも障害の早期発見につながります。

朝の定例確認

```
$ hostname
$ whoami
$ systemctl status sample-app
$ journalctl -u sample-app -n 20
$ df -h
$ free -h
$ ss -tlnp | grep 8080
$ tail -n 20 /var/log/sample-app/app.log
```

- 接続先とユーザーを取り違えていないか

- サービスが落ちていないか
- 直近ログに失敗や再起動が出ていないか
- ディスクやメモリに極端な逼迫がないか
- 期待ポートで待ち受けているか
- アプリ側ログに不自然なエラーが連続していないか

10.11 フェーズ 10 変更作業の前後で見る点

設定を変更するときは、変更内容そのものより、前後で何を確認したかが重要です。安全な変更は、変更前確認と変更後確認のセットで成り立ちます。

変更前

```
$ systemctl status sample-app
$ sudo cat /opt/sample-app/config/app.conf
$ sudo cp /opt/sample-app/config/app.conf \
    /opt/sample-app/config/app.conf.before-change
$ sudo tar -czf ~/linux-first-lab/backup/sample-app-config-quick.tar.gz \
    /opt/sample-app/config
```

変更後

```
$ diff /opt/sample-app/config/app.conf.before-change \
    /opt/sample-app/config/app.conf
$ sudo systemctl restart sample-app
$ systemctl status sample-app
$ journalctl -u sample-app -n 20
$ curl -I http://localhost:8080
$ tail -n 20 /var/log/sample-app/app.log
```

ユニットファイルを変えたとき

- ユニットファイルを変更したら、まず `sudo systemctl daemon-reload` を実行する
- その後でサービスを再起動する
- 代表例は `/etc/systemd/system/sample-app.service`

設定ファイル変更とユニットファイル変更では、反映手順が少し違う点に注意してください。

10.12 フェーズ 11 作業記録を残す

実務では、変更そのものよりも「何を見て、何を変えて、結果がどうだったか」を残しておくことが重要です。あとで切り戻すとき、別の担当者へ引き継ぐとき、障害原因を振り返ると

きに効いてきます。

作業記録のひな型

```
$ mkdir -p ~/linux-first-lab/worklog
$ tee ~/linux-first-lab/worklog/20260321-sample-app.txt > /dev/null << 'EOF'
作業対象: linux-lab / sample-app
目的: app.conf の設定確認とサービス再起動
変更前確認:
- systemctl status sample-app
- cat /opt/sample-app/config/app.conf
- cp /opt/sample-app/config/app.conf app.conf.before-change
変更内容:
- /opt/sample-app/config/app.conf を編集
変更後確認:
- systemctl restart sample-app
- systemctl status sample-app
- curl -I http://localhost:8080
- tail -n 20 /var/log/sample-app/app.log
切り戻し:
- app.conf.before-change を戻して再起動
EOF
```

記録に残したい最小項目

- どのホストで何を変えるのか
- 変更前に何を確認したか
- どのファイルやサービスを変更したか
- 変更後に何を確認し、結果がどうだったか
- 切り戻すなら何を戻すのか

10.13 この章で身につけたい順番

総合演習で重要なのは、細かいコマンドの数ではなく順番です。

1. 接続先とユーザーを確認する
2. 配置先と権限を整える
3. サービスとして登録する
4. ログとポートを確認する
5. リモートから疎通確認する
6. 問題が出たら、状態、ログ、権限、ポートの順で切り分ける
7. 設定とログを分けてバックアップし、戻し方も確認しておく
8. 日次確認と変更後確認を同じ順番で繰り返す

確認課題

- `linux-lab` を新規サーバと見立てて、事前確認からサービス起動までの手順を自分だけで再現してください。
- サービスが起動しない場合に、状態、ログ、権限、ポートのどの順で見るかを書き出してください。
- 設定バックアップとログバックアップを別々に取得する理由を、自分の言葉で説明してください。

まとめ

本章では、小さな業務サーバを題材に、Linux 運用の基本的な流れを通して確認しました。ここまでの章を個別の知識として終わらせず、接続、配置、権限、サービス、ログ、ネットワークという順番でつなげて考えることが、運用の入口では特に重要です。

付録

付録 A Windows コマンドとの対応

ここでは、Windows 技術者が最初に戸惑いやすい操作を中心に、Linux 側でまず試す代表的なコマンドをまとめます。完全な 1 対 1 対応ではありませんが、「何をしたいときに何を打つか」を引くための早見表として使ってください。

ファイルとディレクトリ

dir → **ls -la** ファイル一覧を詳細表示する。
cd → **cd** ディレクトリを移動する。
cd のみ → **pwd** 現在位置を表示する。
mkdir dirname → **mkdir dirname** ディレクトリを作成する。
rmdir dirname → **rmdir dirname** 空ディレクトリを削除する。
copy src dst → **cp src dst** ファイルをコピーする。
move src dst → **mv src dst** ファイル移動や名前変更を行う。
del filename → **rm filename** ファイルを削除する。
dir /s /b pattern → **find . -name 'pattern'** ファイルを検索する。

テキストとログ

type file → **cat file** ファイル全体を表示する。
more file → **less file** 長いファイルをページ送りで確認する。
findstr word file → **grep word file** 文字列を検索する。
PowerShell Select-String → **grep -n**, **grep -i** 行番号付き検索や大文字小文字を無視した検索を行う。
Get-Content -Tail → **tail -n** 末尾の数行だけを確認する。
Get-Content -Wait → **tail -f** ログの追記をリアルタイムで追う。

システム、サービス、プロセス

systeminfo → **cat /etc/os-release**, **uname -a** OS 情報を確認する。
tasklist → **ps aux** 実行中のプロセスを一覧表示する。
taskkill /pid PID → **kill PID** プロセスを終了する。
services.msc → **systemctl status** サービス状態を確認する。

net start service → **systemctl start service** サービスを起動する。
net stop service → **systemctl stop service** サービスを停止する。
Event Viewer → **journalctl -u service** サービスログを確認する。

ネットワーク

ipconfig → **ip addr show** IP アドレスとインターフェースを確認する。
route print → **ip route show** ルーティングを確認する。
netstat -ano → **ss -tlnp** 待受ポートを確認する。
ping host → **ping -c 4 host** 到達性を確認する。
nslookup host → **getent hosts host**, **nslookup host** 名前解決を確認する。
curl.exe -I URL → **curl -I URL** HTTP ヘッダを確認する。
scp 相当 PowerShell の scp や WinSCP に対して、Linux 側でも scp を使う。

ユーザー、権限、導入

whoami → **whoami** 実行ユーザーを確認する。
icacls に近い確認 → **ls -l** 基本的な所有者、グループ、権限を確認する。
runas に近い操作 → **sudo** 一時的に権限を上げてコマンドを実行する。
winget install に近い操作 → **dnf install**, **apt install** パッケージを導入する。
Programs and Features に近い確認 → **rpm -q**, **dpkg -l** 導入済みパッケージを確認する。

付録 B よく使うコマンドクイックリファレンス

最初の 30 秒で見るもの

基本確認

```
$ whoami
$ hostname
$ pwd
$ cat /etc/os-release
$ ip addr show
$ df -h
```

- 接続先を取り違えていないか
- 想定外のユーザーで作業していないか
- OS と IP アドレスは何か
- ディスク空きは十分か

ファイルとディレクトリ

ファイル操作

```
$ pwd
$ ls -la
$ mkdir -p ~/linux-first-lab
$ cp src.txt dst.txt
$ mv old.txt new.txt
$ rm file.txt
$ find ~/linux-first-lab -name "*.conf"
```

テキストとログ

テキスト処理

```
$ cat -n app.conf
$ less app.log
$ head -n 5 app.conf
$ tail -n 20 app.log
$ tail -f app.log
$ grep -n "ERROR" app.log
$ awk -F= '/port/ {print $2}' app.conf
$ sed -i 's/8080/8081/' app.conf
$ diff app.conf.before app.conf
```

ユーザーと権限

ユーザーと権限

```
$ id
$ ls -l file.txt
$ chmod 600 file.txt
$ sudo chown appsvc:appops /opt/sample-app
$ getent passwd appsvc
$ getent group appops
```

パッケージ管理

パッケージ管理

```
$ which dnf
$ which apt
$ dnf search tree
$ sudo dnf install -y tree
$ apt search tree
$ sudo apt install -y tree
$ rpm -q bash
$ dpkg -l | grep bash
```

プロセスとサービス

プロセスとサービス

```
$ ps aux
$ pgrep -af sample-app
$ top
$ systemctl status sample-app
$ sudo systemctl restart sample-app
$ journalctl -u sample-app -n 50
$ journalctl -u sample-app -f
```

ネットワーク

ネットワーク確認

```
$ ip addr show
$ ip route show
$ getent hosts linux-lab
$ ping -c 2 linux-lab
$ ss -tlnp | grep 8080
$ nc -zv linux-lab 8080
$ curl -I http://linux-lab:8080
```

バックアップと調査

バックアップと調査

```
$ tar -czf backup.tar.gz /opt/sample-app/config
$ du -sh /opt/sample-app /var/log/sample-app
$ free -h
$ df -h
$ bash -n script.sh
$ bash -x script.sh
```

付録 C vi チートシート

最初に覚える操作

- `vi file`: ファイルを開く
- `i`: 挿入モードに入る
- `Esc`: ノーマルモードへ戻る
- `:w`: 保存
- `:q`: 終了
- `:wq`: 保存して終了
- `:q!`: 保存せず終了

移動

- `h`, `j`, `k`, `l`: 左、下、上、右
- `0`: 行頭へ移動
- `$`: 行末へ移動
- `gg`: ファイル先頭へ移動
- `G`: ファイル末尾へ移動
- `w`: 次の単語先頭へ移動
- `b`: 前の単語先頭へ移動

編集

- `x`: 1 文字削除
- `dd`: 1 行削除
- `yy`: 1 行コピー
- `p`: 貼り付け
- `u`: 取り消し
- `Ctrl+r`: やり直し

検索と置換

- `/word`: 文字列検索
- `n`: 次の検索結果
- `:s/old/new/`: 現在行の最初の一致を置換
- `:%s/old/new/g`: ファイル全体を一括置換

固まったときの最小復帰手順

- 何を押しても意図どおりに動かないときは、まず `Esc` を数回押す
- 画面下に `:` を出したいときは、ノーマルモードに戻ってから入力する
- 保存せず抜けたいときは `:q!`
- 保存して抜けたいときは `:wq`

付録 D 用語集

ディストリビューション Linux カーネルと各種ツール、パッケージ管理の仕組みをまとめた配布単位。
カーネル OS の中核部分。ハードウェア制御やプロセス管理を担う。
シェル コマンドを受け付ける対話環境。Bash が代表例。
シェルスクリプト コマンド列をまとめて再実行できるようにしたファイル。
プロセス 実行中のプログラム。
PID プロセスを識別する番号。
サービス 常駐して動作するプログラム、または systemd で管理する単位。
デーモン バックグラウンドで動き続けるプロセスを指す古い呼び方。
systemd サービス起動や依存関係、自動起動を管理する仕組み。
ユニットファイル systemd がサービス定義を読む設定ファイル。
リポジトリ パッケージを配布する保管場所。
パッケージ 導入、更新、削除を管理しやすくしたソフトウェアのまとまり。
依存関係 あるソフトが動作に必要とする別のライブラリやパッケージ。
FHS `/etc`、`/var`、`/opt` などの役割を定める考え方。
絶対パス ルート `/` から始まる完全なパス。
相対パス 現在位置を基準にしたパス。
所有者 ファイルやディレクトリを主に管理するユーザー。
グループ 複数ユーザーへまとめて権限を与える単位。
権限 読み取り、書き込み、実行の可否。
root Linux の管理者ユーザー。
sudo 一時的に別ユーザー権限でコマンドを実行する仕組み。
標準出力 コマンドの通常結果が出る出力先。
標準エラー出力 エラー内容が出る出力先。
リダイレクト 出力先を画面以外のファイルへ切り替える操作。

パイプ あるコマンドの出力を次のコマンドへ渡す操作。
待受ポート サービスが接続を受け付けるために開いているポート。
名前解決 ホスト名から IP アドレスを求めること。
SSH リモートサーバへ安全に接続する標準的な手段。
ワーキングディレクトリ コマンドやサービスが基準にする現在位置。
Shebang スクリプト 1 行目の `#!/bin/bash` のような実行シェル指定。

付録 E 障害時の確認順メモ

何かがうまく動かないときは、設定変更より先に次の順で見ます。

1. 接続先とユーザーは合っているか
2. 対象ファイルやディレクトリは存在するか
3. サービスは起動しているか
4. ログに失敗理由が出ているか
5. 権限や所有者は合っているか
6. ポートは開いているか
7. 名前解決と疎通はできているか

順番を崩していきなり編集すると、原因が分からないまま状態だけ変えてしまいます。まず読む、次に変える、最後に結果を確認するという原則を、この付録も含めて繰り返し参照してください。

付録 F よく見るパス一覧

Windows では管理ツールやイベントビューアから辿る情報も、Linux では「どのディレクトリを見るか」を知っていると探しやすくなります。最初によく使う代表的な場所をまとめます。

`/etc` 設定ファイルを置く基本の場所。サービス設定や OS 設定を探す入口になります。
`/etc/systemd/system` 自分で追加、変更したサービスのユニットファイルを置く場所。`sample-app.service` のような定義を確認します。
`/var/log` ログの基本配置先。OS や各種サービスのログを探すときに最初に見ます。
`/var/lib` アプリやサービスが保持する内部データの置き場。設定ではなく状態データを持つ場面が多くあります。
`/opt` 業務アプリをまとめて置くときに使いやすい場所。本書では `/opt/sample-app` を共通題材にします。
`/home` 一般ユーザーのホームディレクトリ。作業メモや一時的な検証ファイルを置く場所として使います。
`/tmp` 一時ファイルの置き場。再起動や定期清掃で消える前提のため、永続化したい設定は置きません。
`/var/tmp` 一時ファイルでも、`/tmp` より長めに保持されることを想定した置き場です。
`/usr/bin` 多くの基本コマンドが入る場所。`which` や `type` で実体を調べるときの候補にな

ります。

/usr/local/bin 手動配置したコマンドや自作スクリプトを置く候補です。パッケージ管理下のファイルと分けやすくなります。

/proc カーネルやプロセスの状態を仮想ファイルとして見せる場所。メモリ、PID、マウント情報などの確認に使います。

/dev デバイスファイルの置き場。ディスクや疑似端末などをファイルとして扱う Linux の考え方が表れます。

迷ったときの入口

設定を探すなら `/etc`、ログを探すなら `/var/log`、業務アプリの配置先を見るなら `/opt`、自分の作業ファイルを見るなら `/home` から始めると、初学段階では大きく外しにくくなります。