লিনাক্স টার্মিনাল মাস্টারি

কমান্ডের ব্যাখ্যা, বাস্তব ওয়ার্কফ্লো এবং DevOps চিটশিট।



জয় চন্দ্র উদয়

সূচনা

শুরুর কথা

ফ্রন্ট ম্যাটার — পড়ার আগে যা জানা জরুরি

কপিরাইট ও সংস্করণ

Copyright © জয় চন্দ্র উদয়, ২০২৬। সর্বস্বত্ব সংরক্ষিত।

প্রথম সংস্করণ, ২০২৬।

এই বইয়ের কোনো অংশ লেখকের অনুমতি ছাড়া অনুলিপি করা যাবে না। টার্মিনাল কমান্ডগুলো Apache License 2.0 এর অধীনে ব্যবহার করা যাবে শিক্ষামূলক কাজে, Credit রেখে।

সংস্করণ ইতিহাস:

* প্রথম সংস্করণ, ২০২৬ — Front Matter + অধ্যায় ১–১৬ + Back Matter (সম্পূর্ণ সংস্করণ)
* পরবর্তী সংস্করণ — নিয়মিত পরিমার্জন ও পাঠকদের ফিডব্যাকের ভিত্তিতে পরিবর্ধন পরিকল্পনা।

এই বই কার জন্য?

- যাঁরা প্রথমবার টার্মিনালে এসেছেন এবং "কীভাবে শুরু করব" বুঝতে পারছেন না
- যাঁরা GUI ব্যবহার করেন কিন্তু সার্ভার, VPS, WSL, Docker, Git-এর মতো কাজের জন্য টার্মিনাল শিখতে চান
- DevOps, সিস্টেম অ্যাডমিন, কম্পিউটার সায়েন্সের শিক্ষার্থী যাঁরা কমান্ড মনে রাখতে চান না, বুঝতে চান
- যাঁরা "চিটশিট কপি করে" আটকে যান — তাঁরা প্রতিটি কমান্ডের ব্যাখ্যা, বাস্তব উদাহরণ আর নিরাপদ ওয়ার্কফ্লো চান

এই বই কী হবে না:

এটি কোনো মুখস্থবিদ্যার বই না। প্রতিটি কমান্ড থেকে আপনি শিখবেন **কী করে, কেন করে, কখন ব্যবহার করবেন, কখন করবেন না**।

কীভাবে এই বই ব্যবহার করবেন?

1. প্রতিটি অধ্যায় ক্রমে পড়ুন। প্রতিটি কমান্ডের `Syntax`, `উদাহরণ` আর `সম্ভাব্য Output` নিজের মেশিনে টাইপ করে দেখুন। না চর্চা করলে মনে থাকবে না।
2. প্রতিটি অধ্যায়ের শেষে Quick recap আর Practice task আছে। সেগুলো না করে পরের অধ্যায়ে যাবেন না।
3. প্রতিটি কমান্ডের ব্যাখ্যায় **মনে রাখো:** আর **সাধারণ ভুল / সতর্কতা** পড়ুন — এগুলো বাস্তব অভিজ্ঞতা থেকে লেখা।
4. **কখন দরকার হবে:** অংশটি বাস্তব ওয়ার্কফ্লো বোঝার জন্য। DevOps কাজে যেন ব্যবহার করতে পারেন, সেজন্য।
5. প্র্যাকটিসের জন্য একটি আলাদা ডিরেক্টরি বানান, যেমন `~/practice`। মূল ডিরেক্টরিতে `rm` প্র্যাকটিস করবেন না।

প্রয়োজনীয়তা / Prerequisites

- অপারেটিং সিস্টেম: Ubuntu 22.04 LTS বা Ubuntu 24.04 LTS (এই সংস্করণের সব উদাহরণ এগুলোতেই পরীক্ষা করা)
- এই বইয়ের ১–৪ অধ্যায় অনেক Linux-এই চলবে, কিন্তু ৫ নম্বর অধ্যায়ের `apt` শুধু Ubuntu/Debian-এই। অন্য ডিস্ট্রোতে Package Manager আলাদা।
- মৌলিক কম্পিউটার জ্ঞান: নিজের ইউজার নাম, পাসওয়ার্ড জানা, ফাইল-ফোল্ডার কী তা বোঝা।
- আগে থেকে Linux অভিজ্ঞতা লাগবে না।

টার্মিনাল কীভাবে খুলবেন

Ubuntu-তে কীবোর্ডে চাপ দিন:

`Ctrl+Alt+T`

আরেকটি ডিরেক্টরি (ওয়ে): Activities মেনুতে `Terminal` লিখে খুলুন।

খোলার পর আপনি দেখবেন কিছুটা এরকম:

```
username@hostname:~$
```

`username` হলো আপনার ইউজার নাম, `hostname` হলো মেশিনের নাম, `~` হলো আপনার Home ডিরেক্টরি, `$` হলো প্রম্পট — এখানে কমান্ড টাইপ করা যাবে।

Table of Contents — বিষয়সূচি

Front Matter — কভার, কপিরাইট, ব্যবহারের নির্দেশনা, Prerequisites ২

অধ্যায় ১ — টার্মিনাল ফান্ডামেন্টালস: Linux, shell, টার্মিনাল, প্রম্পট, কমান্ড, পাথ, `man`, `--help`, ... ৬

অধ্যায় ২ — ফাইলসিস্টেম মাস্টারি: `/`, `/home`, `/etc`, `/var`, `/tmp` সহ মূল পাথ, `ls`, `cd`, ... ১৬

অধ্যায় ৩ — তৈরি, কপি, মুভ, ডিলিট: `touch`, `cp`, `mv`, `rm`, `install`, `file`, `stat` — ২৭

অধ্যায় ৪ — পড়া, এডিট, খোঁজা: `cat`, `less`, `head`, `tail`, `nano`, `grep`, `sort`, ... ৩৪

অধ্যায় ৫ — প্যাকেজ ম্যানেজমেন্ট (Ubuntu/Debian): `sudo`, `apt`, `dpkg` — Safe workflow ও D... ..... ৪৭

অধ্যায় ৬ — ইউজার, গ্রুপ ও পারমিশন: `chmod`, `chown`, `umask`, `ls -l` — least privilege ৫৯

অধ্যায় ৭ — প্রসেস ও মনিটরিং: `ps`, `top`, `htop`, `kill`, `free`, `df`, `du`, `systemctl` ৬৫

অধ্যায় ৮ — নেটওয়ার্ক ও রিমোট: `ip`, `ping`, `curl`, `wget`, `ss`, `ssh`, `scp`, `sftp` —... ..... ৭৭

অধ্যায় ৯ — আর্কাইভ ও ব্যাকআপ: `tar`, `zip`, `gzip`, `xz`, `7z` ৮৭

অধ্যায় ১০ — শেল প্রোডাক্টিভিটি: pipe, redirection, `echo`, `alias`, globbing, quoting ৯৪

অধ্যায় ১১ — সার্ভিস, লগ ও Cron: `systemctl`, `journalctl`, `crontab` ১০৪

অধ্যায় ১২ — ডেভেলপার ওয়ার্কফ্লো: Git, Node.js/npm, PM2, Nginx ১১৩

অধ্যায় ১৩ — Ubuntu VPS: এন্ড-টু-এন্ড ওয়ার্কফ্লো (SSH → Nginx → logs) ১২৬

অধ্যায় ১৪ — ট্রাবলশুটিং: ১১টি কমন সমস্যা ও নিরাপদ সমাধান ১৩০

অধ্যায় ১৫ — প্র্যাকটিস ল্যাব: ১২টি ল্যাব (নেভিগেট থেকে Node.js ট্রেস) ১৩৩

অধ্যায় ১৬ — দ্রুত রেফারেন্স (চিটশিট): টেবিলসমূহ + "Enter চাপার আগে" সেফটি চেকলিস্ট ১৩৭

Back Matter — শব্দকোষ, কমান্ড ইনডেক্স, রেফারেন্স, লেখক পরিচিতি, ব্র্যান্ডিং ও প্লেসমেন্ট (Before You Switc... ..... ১৪২

Learning Roadmap — শেখার পথরেখা

1. **সপ্তাহ ১ — সাহস:** অধ্যায় ১। `pwd`, `whoami`, `whoami`-সহ পরিচিতি, `man` পড়া, `Ctrl+C`, `Tab` আর `Up/Down` অভ্যাস।
2. **সপ্তাহ ২ — ঘুরপথ:** অধ্যায় ২। Home থেকে `/etc`, `/var/log`, `/tmp` পর্যন্ত ঘুরুন, `ls -l`, `cd` দিয়ে ডিরেক্টরি বদলান।
3. **সপ্তাহ ৩ — নিরাপদ ডিলিট:** অধ্যায় ৩। শুধু `~/practice`-এ প্র্যাকটিস করুন। `rm` ব্যবহারের আগে `pwd` আর `ls` দেখার অভ্যাস করুন।
4. **সপ্তাহ ৪ — পড়া-খোঁজা:** অধ্যায় ৪। কনফিগ ফাইল, Log ফাইল পড়ুন। `grep` দিয়ে কনফিগে খুঁজুন।
5. **সপ্তাহ ৫ — ইনস্টল:** অধ্যায় ৫। `git`, `curl`, `wget` ইনস্টল করুন শুধু Ubuntu/Debian-এ। অন্য ডিস্ট্রোতে এই অধ্যায়ের `apt` কমান্ড চলবে না — ধারণাটা নিন।

> মনে রাখুন: প্রতিদিন ২০ মিনিট টার্মিনালে থাকলেই যথেষ্ট। অযথা কমান্ড মুখস্থ করবেন না।

০১

অধ্যায় ১

টার্মিনাল ফান্ডামেন্টালস: ভয় থেকে ঘনিষ্ঠতা

এই অধ্যায়ে আমরা ভাষা শিখছি। আপনি যদি এই অধ্যায়ের ধারণাগুলো না বোঝেন, পরের সব কমান্ডই মুখস্থ হয়ে যাবে।

লিনাক্স কী?

লিনাক্স একটি অপারেটিং সিস্টেম। Windows-এর মতোই, কিন্তু বেশি নমনীয়, সার্ভারে বহুল ব্যবহৃত। Ubuntu হলো লিনাক্সের একটি জনপ্রিয় ডিস্ট্রিবিউশন (সংস্করণ)।

শেল (Shell) কী?

Shell হলো সেই প্রোগ্রাম যা আপনার কমান্ড বোঝে আর কাজ করতে পাঠায়। Ubuntu-তে ডিফল্ট Shell হলো `bash`।

টার্মিনাল কী?

টার্মিনাল হলো সেই উইন্ডো যেখানে আপনি Shell-এর সাথে কথা বলেন। আপনি টাইপ করেন, Shell উত্তর দেয়।

কমান্ড, প্রম্পট

`$` চিহ্নের পর আপনি যা লেখেন তা হলো কমান্ড। কখনো কখনো প্রম্পটে `#` দেখলে বুঝবেন আপনি `root` ইউজার।

সাধারণ Syntax: `command [options] [arguments]`

প্রায়ণ্ন কমান্ড এই কাঠামোতে চলে:

- `command` — কী করতে চান, যেমন `ls`, `pwd`
- `[options]` — কীভাবে করবেন, যেমন `-l`, `-a`, `-h`। অপশন সাধারণত `-` দিয়ে শুরু।
- `[arguments]` — কীসের ওপর করবেন, যেমন ফাইলের নাম, পথ।

উদাহরণ: `ls -l /home` — এখানে `ls` হলো কমান্ড, `-l` হলো অপশন, `/home` হলো আর্গুমেন্ট।

Case Sensitivity — বড়-ছোট হাতের অক্ষর গুরুত্বপূর্ণ

লিনাক্সে `LS`, `Ls`, `ls` আলাদা। সাধারণত সব কমান্ড ছোট হাতের অক্ষরে। ফাইলের নামেও `Report.txt` আর `report.txt` আলাদা ফাইল।

Absolute vs Relative Paths — সম্পূর্ণ vs আংশিক পথ

- Absolute Path: একদম মূল থেকে পুরো পথ। যেমন `/home/username/file.txt`
- Relative Path: এখন যে ডিরেক্টরিতে আছেন, সেখান থেকে আংশিক পথ। যেমন `file.txt` বা `Documents/file.txt`

প্র্যাকটিস: `pwd` দিয়ে দেখুন আপনি কোথায় আছেন, তারপর Relative Path ব্যবহার করুন।

মূল কী: `man`, `--help`, `Ctrl+C`, `Tab`, Up/Down, `clear`

- `man` দিয়ে কমান্ডের বিস্তারিত সাহায্য পাওয়া যায়।
- `command --help` দিয়ে সংক্ষিপ্ত সাহায্য পাওয়া যায়।
- `Ctrl+C` দিয়ে চলমান প্রসেস আটকানো যায়। এটিকে Interupt বলে।
- `Tab` দিয়ে স্বয়ংক্রিয়ভাবে নাম পূরণ হয় (Auto-completion)। বারবার ট্যাব চাপলে সম্ভাব্য বিকল্প দেখাবে।
- Up/Down বাটন দিয়ে আগের কমান্ড আবার আনা যায়।
- `clear` বা `Ctrl+L` দিয়ে স্ক্রিন পরিষ্কার হয়।

এই অধ্যায়ের কমান্ডগুলো এই প্রসঙ্গেই ব্যবহার করবে।

pwd — বর্তমান ডিরেক্টরির পুরো পথ দেখায়

সহজ কথায়: আপনি এখন কোন ডিরেক্টরিতে আছেন তা বলবে।

Syntax

```
pwd
```

উদাহরণ

```
pwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username
```

কখন দরকার হবে: হারিয়ে গেলে, `cd` করার পর, `rm` করার আগে নিজের অবস্থান যাচাই করতে।

মনে রাখো: `rm` বা `mv` করার আগে সবসময় `pwd` চালান, যাতে ভুল ডিরেক্টরিতে কাজ না হয়।

whoami — বর্তমান ইউজারের নাম দেখায়

সহজ কথায়: আপনি এখন কার হিসেবে লগইন করেছেন।

Syntax

```
whoami
```

উদাহরণ

```
whoami
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
joy
```

কখন দরকার হবে: সার্ভারে SSH-এর পর নিশ্চিত হতে, স্ক্রিপ্ট ডিবাগে।

মনে রাখো: `id` এর চেয়ে সংক্ষিপ্ত। শুধু নাম চাইলে এটাই ব্যবহার করুন, বিস্তারিত চাইলে `id`।

`id` — ইউজার ID, গ্রুপ ID আর গ্রুপসহ বিস্তারিত পরিচিতি

সহজ কথায়: ইউজারের সংখ্যাসূচক পরিচিতি দেখাবে।

Syntax

```
id
id <username>
```

উদাহরণ

```
id
id root
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
uid=1000(joy) gid=1000(joy) groups=1000(joy),27(sudo),1000(docker)
uid=0(root) gid=0(root) groups=0(root)
```

কখন দরকার হবে: পারমিশন সমস্যা সমাধানে, ইউজার সার্ভারে ঢুকতে পারছে কিনা যাচাইয়ে।

মনে রাখো: `uid=0` মানেই `root`। নিজের পাসওয়ার্ড `<password>` দিয়ে এটা মেলাবেন না।

`uname -a` — সিস্টেমের নাম, ভার্সন, আর্কিটেকচার দেখায়

সহজ কথায়: অপারেটিং সিস্টেম কী চলছে বলবে।

Syntax

```
uname -a
```

উদাহরণ

```
uname -a
uname -r
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Linux joy-laptop 6.8.0-31-generic #31-Ubuntu SMP x86_64 GNU/Linux
6.8.0-31-generic
```

কখন দরকার হবে: সাপোর্টে সমস্যা বোঝাতে, সার্ভারে কার্নেল ভার্সন দেখতে।

↳ `uname -a` — চলমান

মনে রাখো: `-a` মানে সব তথ্য। শুধু নাম চাইলে `uname`।

`hostname` — মেশিনের নাম দেখায়

সহজ কথায়: আপনার এই মেশিনের নাম কী।

Syntax

```
hostname
hostnamectl
```

উদাহরণ

```
hostname
hostnamectl
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
joy-laptop
```

কখন দরকার হবে: SSH বা সার্ভারে পরিচিতি বোঝাতে, প্রম্পট দেখে নিজেকে চিনতে।

মনে রাখো: নাম বদলাতে `sudo hostnamectl set-hostname <newname>` লাগবে, যেটা পারমিশন চাইবে।

`date` — বর্তমান তারিখ আর সময় দেখায়

সহজ কথায়: আজ কয়টা বাজে, আজকের তারিখ কত।

Syntax

```
date
date +"%Y-%m-%d %H:%M:%S"
```

উদাহরণ

```
date
date +"%Y-%m-%d"
```

↳ date — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Fri Nov  6 12:00:00 UTC 2026
2026-11-06
```

কখন দরকার হবে: Log ফাইলে সময় মেলাতে, স্ক্রিপ্টে ফাইলের নামে তারিখ বসাতে।

মনে রাখো: `date` এর ফরম্যাট `%Y-%m-%d` মনে রাখুন। `%` তারিখের কোড।

cal — মাসের ক্যালেন্ডার দেখায়

সহজ কথায়: এই মাসের ক্যালেন্ডার প্রিন্ট করে।

Syntax

```
cal
cal <month> <year>
cal -y
```

উদাহরণ

```
cal
cal 11 2026
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
   November 2026
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
```

কখন দরকার হবে: তারিখ-দিন বের করতে, Log বিশ্লেষণেতারিখের দিন জানতে।

মনে রাখো: কিছু Ubuntu মিনিমালে `cal` নাও থাকতে পারে। না থাকলে সতর্কতা: `sudo apt install util-linux -y` দিয়ে ইনস্টল করুন (Ubuntu/Debian-এ)।

`man` — ম্যানুয়াল পেজ পড়া

সহজ কথায়: যেকোনো কমান্ডের বিস্তারিত লাইব্রেরি।

Syntax

```
man <command>
man <section> <command>
```

উদাহরণ

```
man ls
man mkdir
man 2 mkdir
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
LS(1)                   User Commands                   LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...
```

কখন দরকার হবে: কমান্ডের সব অপশন বুঝতে, `man` ছাড়া অনুমান করা বন্ধ করতে।

মনে রাখো: `man` থেকে বের হতে `q` চাপুন। `/` দিয়ে খুঁজুন, `n` দিয়ে পরের ম্যাচে যান।

`command --help` — সংক্ষিপ্ত সাহায্য

সহজ কথায়: কমান্ডটা কী কী অপশনে চলে তার সংক্ষিপ্ত তালিকা।

Syntax

```
<command> --help
<command> -h
```

উদাহরণ

```
ls --help
mkdir --help
```

↳ command --help — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
  -a, --all                  do not ignore entries starting with .
  -l                         use a long listing format
```

কখন দরকার হবে: দ্রুত অপশন দেখতে, `man`-এ যাওয়ার আগে সংক্ষেপ জানতে।

মনে রাখো: সব কমান্ডে `--help` থাকে না। না থাকলে `man` দেখুন।

clear — স্ক্রিন পরিষ্কার

সহজ কথায়: স্ক্রল ওপরে পাঠিয়ে মনোযোগ পুনরুদ্ধার।

Syntax

```
clear
```

উদাহরণ

```
clear
ls -la
clear
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# স্ক্রিন খালি হয়ে যাবে, কোনো আউটপুট আসবে না
```

কখন দরকার হবে: লম্বা আউটপুটের পর, স্ক্রিন ভরে গেলে পরিষ্কার করতে।

মনে রাখো: `Ctrl+L` ও একই কাজ করে। ইতিহাস মুছতে `rm ~/.bash_history` নয় — বা `history -c`।

history — আগের কমান্ডের তালিকা

সহজ কথায়: আগে চালানো কমান্ডগুলো সংখ্যাসহ দেখাবে।

Syntax

```
history
history | tail -20
```

↳ history — চলমান

উদাহরণ

```
history
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
1  ls -la
2  cd Documents
3  pwd
```

কখন দরকার হবে: ভুলে যাওয়া কমান্ড খুঁজতে, আগের কমান্ড ব্যাং `!2` দিয়ে চালাতে।

মনে রাখো: `!!` দিলে শেষ কমান্ড আবার চলবে। যেকোনো কমান্ডের আগে `history` দিয়ে নিশ্চিত করুন — ভুল কমান্ড আবার চলতে পারে।

exit — টার্মিনাল বা Shell বন্ধ

সহজ কথায়: এই টার্মিনাল সেশান শেষ করবে।

Syntax

```
exit
```

উদাহরণ

```
exit
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# SSH সেশান হলে কানেকশন ছিঁড়বে, জানালা বন্ধ হবে
```

কখন দরকার হবে: কাজ শেষে, SSH থেকে বের হতে, স্ক্রিপ্টে।

মনে রাখো: না ব্যবহার করলেও টার্মিনাল বন্ধ করলে একই হয়। `logout` ও প্রায় একই কাজ করে।

দ্রুত পুনরালোচনা (Quick recap) — অধ্যায় ১

- Shell (`bash`) কমান্ড বোঝে, টার্মিনাল হলো কথা বলার উইন্ডো।
- সিনট্যাক্স: `command [options] [arguments]`
- Absolute পাথ শুরু হয় `/` দিয়ে, Relative পাথ এখনকার ডিরেক্টরি থেকে।
- `man <command>` = সম্পূর্ণ সাহায্য, `<command> --help` = সংক্ষিপ্ত সাহায্য।
- `Ctrl+C` আটকায়, `Tab` পূরণ করে, Up/Down ইতিহাস আনে, `clear` / `Ctrl+L` পরিষ্কার করে।

অনুশীলনী (Practice Task)

1. `Ctrl+Alt+T` দিয়ে টার্মিনাল খুলুন।
2. `pwd`, `whoami`, `id`, `uname -a`, `hostname`, `date`, `cal` পরপর চালান। আউটপুট পড়ুন।
3. `man pwd` খুলে `q` দিয়ে বের হন।
4. `ls --help` দেখে `history` চালিয়ে নিজের আগের লেখা দেখুন।
5. `clear` দিয়ে স্ক্রিন পরিষ্কার করুন।

অধ্যায় ২

০২ ফাইলসিস্টেম: কোথায় কী আছে বোঝা

লিনাক্সের ফাইলসিস্টেম একটি গাছের মতো। গোড়া `/`। সব ফাইল এরই অধীন। GUI-তে হারানো ডিরেক্টরি টার্মিনালে হারায় না — পাথ মুখস্থ থাকলেই হবে।

মূল পাথসমূহ

- `/` — রুট, ফাইলসিস্টেমের গোড়া। সবকিছু এর নিচে।
- `/home` — সব সাধারণ ইউজারের Home ডিরেক্টরি। যেমন `/home/joy` আপনার।
- `/root` — `root` ইউজারের Home। সাধারণ ইউজার সাধারণত ঢুকতে পারে না।
- `/etc` — গুরুত্বপূর্ণ কনফিগ ফাইল। যেমন `/etc/passwd`, `/etc/hosts`। এখানে হাত দেওয়ার আগে সাবধান।
- `/var` — লগ, স্পুল, ক্যাশ। `/var/log/`-তে সিস্টেম লগ থাকে।
- `/tmp` — অস্থায়ী ফাইল। রিবুটে মুছতে পারে।
- `/usr` — প্রায় সব প্রোগ্রাম আর লাইব্রেরি।
- `/opt` — তৃতীয় পক্ষের সফটওয়্যার। ম্যানুয়ালি ইনস্টল করা।
- `/dev` — ডিভাইস ফাইল।
- `/proc` — চলমান প্রসেসের ভার্চুয়াল ফাইলসিস্টেম।
- `/sys` — হার্ডওয়্যার আর সিস্টেম ইনফো।
- `/mnt` — মাউন্ট পয়েন্ট, সাধারণত বাইরের ড্রাইভ।

পাথ ছাড়া টার্মিনাল অন্ধ। যত দ্রুত পাথ পড়তে পারবেন, তত দ্রুত কাজ করবেন।

পাথ সিম্বল: `/`, `~`, `.`, `..`, `-`

- `/` — মূল ডিরেক্টরি / ডিরেক্টরি সেপারেটর। `/home/joy` মানে root থেকে home থেকে joy।
- `~` — আপনার Home ডিরেক্টরি, যেমন `/home/joy`।
- `.` — এখন যে ডিরেক্টরিতে আছেন, সেটি। `cd .` কোনো মানে নেই, তবে `./file` মানে এখানকার ফাইল।
- `..` — এক লেভেল ওপরে। `cd ..` মানে এক স্তর উপরে যান।
- `-` — আগে যে ডিরেক্টরিতে ছিলেন, সেখানে ফেরা। `cd -` মানে ব্যাক।

⊗ সাধারণ ভুল: `cd /` লিখে `/` সেপারেটরের ভূমিকা ভুলে যাওয়া, `cd /home` আর `cd home` গুলিয়ে ফেলা। লিনাক্সে বড়-ছোট হাতের অক্ষর অনেক গুরুত্বপূর্ণ।

pwd — বর্তমান ডিরেক্টরি যাচাই

সহজ কথায়: আপনি কোথায় আছেন দেখায়।

Syntax

```
pwd
```

উদাহরণ

```
pwd
cd Documents
pwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username
/home/username/Documents
```

কখন দরকার হবে: `ls` আর `cd`-র পর নিজের অবস্থান নিশ্চিত করতে।

মনে রাখো: এই অধ্যায়ে প্রতিটি `cd`-র পর `pwd` চালান — অভ্যাস করুন।

ls — ডিরেক্টরির বিষয়বস্তু দেখা

সহজ কথায়: এখনকার ডিরেক্টরিতে কী কী আছে।

Syntax

```
ls [options] [path]
```

উদাহরণ

```
ls
ls /etc
ls Documents
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Desktop  Documents  Downloads  file.txt
```

কখন দরকার হবে: ফাইলের নাম মনে না থাকলে, ডিরেক্টরি খালি কিনা দেখতে।

↳ ls — চলমান

মনে রাখো: `ls` হিডেন ফাইল (`.` দিয়ে শুরু) দেখায় না। হিডেন সহ দেখতে `ls -a`।

ls -l — Long লিস্টিং: পারমিশন, সাইজ, তারিখসহ

সহজ কথায়: প্রতিটি ফাইলের পারমিশন, মালিক, সাইজ আর তারিখসহ দেখাবে।

Syntax

```
ls -l [path]
```

উদাহরণ

```
ls -l
ls -l /var/log
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
total 16
-rw-r--r-- 1 joy joy  220 Nov  6 10:00 file.txt
drwxr-xr-x 2 joy joy 4096 Nov  6 10:00 Documents
```

কখন দরকার হবে: পারমিশন সমস্যা বুঝতে, ফাইল কত পুরনো তা দেখতে।

মনে রাখো: প্রথম অক্ষর `-` মানে সাধারণ ফাইল, `d` মানে ডিরেক্টরি। তারপর `rwx` ৩ ধাপে: মালিক, গ্রুপ, অন্যান্য।

ls -a — হিডেনসহ সব ফাইল দেখা

সহজ কথায়: `.` দিয়ে শুরু হিডেন ফাইলসহ সব দেখায়।

Syntax

```
ls -a [path]
```

উদাহরণ

```
ls -a
ls -a ~/.ssh
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
.  ..  .bashrc  .ssh  Documents  file.txt
```

↳ ls -a — চলমান

কখন দরকার হবে: কনফিগ ফাইল খুঁজতে, `~/.ssh` চেক করতে।

মনে রাখো: `.` আর `..` সবসময় দেখাবে। এগুলো নয় — দিলে ব্যাখ্যা ওই দুটি ছাড়া বাকি হিডেন দেখাবে।

ls -la — হিডেন + Long একসাথে

সহজ কথায়: হিডেন ফাইলসহ সম্পূর্ণ বিস্তারিত তালিকা।

Syntax

```
ls -la [path]
```

উদাহরণ

```
ls -la
ls -la /home
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
total 24
drwxr-xr-x  5 joy joy 4096 Nov  6 10:00 .
drwxr-xr-x 11 root root 4096 Nov  6 10:00 ..
-rw-r--r--  1 joy joy  220 Nov  6 10:00 .bashrc
```

কখন দরকার হবে: কোনো ফাইল কোথায় লুকালো জানতে, `.bashrc` দেখতে।

মনে রাখো: `ls -a` আর `ls -l` আলাদা করে মনে রাখার দরকার নেই — `-la` একসাথে করে ফেলুন।

ls -lh — মানুষের পড়ার সুবিধামত সাইজে দেখা

সহজ কথায়: বাইটের বদলে KB, MB-তে সাইজ দেখায়।

Syntax

```
ls -lh [path]
```

উদাহরণ

```
ls -lh
ls -lh ~/Downloads
```

↳ ls -lh — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
-rw-r--r-- 1 joy joy 1.2M Nov  6 10:00 ebook.pdf
```

কখন দরকার হবে: সাইজ দ্রুত বুঝতে, Disk ভরে যাওয়ার কারণ খুঁজতে।

মনে রাখো: `-h` মানে Human-readable। `ls -l` এর সাইজ বাইটে দেখায়, `-lh` KB/MB-তে।

cd — ডিরেক্টরি বদলানো

সহজ কথায়: এক ডিরেক্টরি থেকে আরেক ডিরেক্টরিতে যাওয়া।

Syntax

```
cd <path>
cd
```

উদাহরণ

```
cd Documents
cd
pwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username
```

কখন দরকার হবে: Home ফিরে আসতে, প্রজেক্ট ডিরেক্টরিতে ঢুকতে।

মনে রাখো: আর্গুমেন্ট ছাড়া `cd` Home ডিরেক্টরিতে নিয়ে যায়।

cd ~ — Home ডিরেক্টরিতে ফেরা

সহজ কথায়: আপনার ব্যক্তিগত ডিরেক্টরিতে ফিরে আসা।

Syntax

```
cd ~
cd ~/Documents
```

↳ cd ~ — চলমান

উদাহরণ

```
cd /etc
cd ~
pwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username
```

কখন দরকার হবে: হারিয়ে গেলে, `/var/log` থেকে দ্রুত ফেরার টিপস।

মনে রাখো: `cd /home/username`-এর সংক্ষেপ। `cd ~anotheruser` দিয়ে অন্যের Home-এও যাওয়া যায়।

cd .. — এক লেভেল ওপরে যাওয়া

সহজ কথায়: বর্তমান ডিরেক্টরির Parent-এ যাওয়া।

Syntax

```
cd ..
cd ../..
```

উদাহরণ

```
cd ~/Documents/projects
cd ..
pwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username/Documents
```

কখন দরকার হবে: ডিরেক্টরি থেকে সরে আসতে, Backup নেওয়ার পর উপরে ফেরতে।

মনে রাখো: `..` কখনোই `/` এর ওপরে যায় না। `/`-এ গিয়ে `..` চাপলে সেখানেই থাকে।

`cd /` — রুট ডিরেক্টরিতে যাওয়া

সহজ কথায়: ফাইলসিস্টেমের গোড়ায় (রুট) পৌঁছানো।

Syntax

```
cd /
```

উদাহরণ

```
cd /
pwd
ls
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/
bin   boot   dev   etc   home   lib   proc   root   sbin   sys   tmp   usr   var
```

কখন দরকার হবে: `/etc`, `/var/log`, `/usr` ঘুরতে, সিস্টেম স্ট্রাকচার বুঝতে।

মনে রাখো: `/`-এ থেকে `pwd` চালালে শুধু `/` দেখাবে। এটা স্বাভাবিক।

`cd -` — আগের ডিরেক্টরিতে ফেরা

সহজ কথায়: যেখান থেকে এসেছিলেন সেখানে ফেরা।

Syntax

```
cd -
```

উদাহরণ

```
cd ~
cd /etc
cd -
pwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username
/home/username
```

↳ cd - — চলমান

কখন দরকার হবে: দুটি ডিরেক্টরির মধ্যে বারবার যাওয়া-আসা, Log দেখা শেষে প্রজেক্টে ফেরা।

মনে রাখো: `cd -` আলাদা লাইন দেখায় — আউটপুটে আগের পথ প্রিন্ট হয়। এটা সফলতার নিশ্বাস।

mkdir — নতুন ডিরেক্টরি তৈরি

সহজ কথায়: নতুন ফোল্ডার বানাওয়া।

Syntax

```
mkdir <dirname>
mkdir <dir1> <dir2>
```

উদাহরণ

```
mkdir myproject
mkdir backup logs
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# সফল হলে কোনো আউটপুট আসে না
$ ls
myproject  backup  logs
```

কখন দরকার হবে: প্রজেক্ট ফোল্ডার, Backup ডিরেক্টরি তৈরি করতে।

মনে রাখো: নামে স্পেস থাকলে কোট দিয়ে দেবেন, যেমন `mkdir "my project"`।

mkdir -p — Parent সহ ডিরেক্টরি তৈরি

সহজ কথায়: মাঝখানের ডিরেক্টরি না থাকলেও পুরো পথ বানাবে, এরর দেবে না।

Syntax

```
mkdir -p <path>
```

উদাহরণ

```
mkdir -p ~/projects/web/app
mkdir -p logs/2026/11
```

↳ mkdir -p — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# আবার চালালে Error: File exists আসবে না
```

কখন দরকার হবে: স্ক্রিপ্টে পথ নিশ্চিত করতে, কয়েক স্তরের ফোল্ডার একসাথে বানাতে।

মনে রাখো: `-p` মানে Parents। না দিলে `projects` না থাকলে `mkdir projects/web` এরর দেবে।

rmdir — খালি ডিরেক্টরি মুছে ফেলা

সহজ কথায়: শুধুমাত্র খালি ফোল্ডার মুছবে। ভেতরে ফাইল থাকলে মুছবে না।

Syntax

```
rmdir <dirname>
```

উদাহরণ

```
mkdir empty
rmdir empty
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# খালি হলে চুপচাপ মোছা, আওয়াজ নেই
$ rmdir notempty
rmdir: failed to remove 'notempty': Directory not empty
```

কখন দরকার হবে: পরীক্ষার পর খালি ফোল্ডার পরিষ্কার করতে, নিরাপদে ডিরেক্টরি মুছতে।

মনে রাখো: ভেতরে ফাইল থাকলেই `rmdir` ব্যর্থ হবে — এটা নিরাপদ জানালা।

tree — গাছের মতো করে ডিরেক্টরি কাঠামো দেখা

সহজ কথায়: কোন ফোল্ডারের ভেতর কী লুকিয়ে আছে তা আঁকিয়ে দেখায়।

Syntax

```
tree [path]
tree -a -L 2 [path]
```

↳ tree — চলমান

উদাহরণ

```
tree
tree ~/projects
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
.
├── Documents
│   └── file.txt
└── Downloads
```

কখন দরকার হবে: নতুন প্রজেক্টের স্ট্রাকচার বুঝতে, নিজের ডিরেক্টরি মনে রাখতে।

মনে রাখো: কিছু Ubuntu-তে `tree` ইনস্টল করা থাকে না। না থাকলে (শুধু Ubuntu/Debian) চালান: `sudo apt install tree -y`।

find — ফাইল খোঁজা

সহজ কথায়: পথ, নাম, সাইজ, টাইপ দিয়ে ফাইল বের করা।

Syntax

```
find <path> -name "<pattern>"
find <path> -type f -name "*.pdf"
```

উদাহরণ

```
find ~ -name "*.txt"
find /var/log -name "*.log" 2>/dev/null
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username/notes.txt
/home/username/Documents/notes.txt
```

কখন দরকার হবে: হারিয়ে যাওয়া ফাইল খুঁজতে, `.log` ফাইল গনতে।

মনে রাখো: নামে স্পেস থাকলে নিয়ম: `find` একটানে খোঁজে। Permission এরর এড়াতে `2>/dev/null` যোগ করুন।

`locate` — ডেটাবেস থেকে দ্রুত খোঁজা

সহজ কথায়: `find`-এর চেয়ে দ্রুত, কিন্তু আপডেট করা ডেটাবেসে খোঁজে।

Syntax

```
locate <pattern>
sudo updatedb
```

উদাহরণ

```
locate notes.txt
sudo updatedb
locate notes.txt
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username/notes.txt
```

কখন দরকার হবে: পুরো সিস্টেমে নাম দিয়ে ফাইল খুঁজতে, বিস্তারিত ডিরেক্টরি গলিঘাটিতে ঢুকে খুঁজতে।

মনে রাখো: `locate` নতুন তৈরি ফাইল দেখায় না যতক্ষণ `sudo updatedb` চালানো না হয়। কিছু সিস্টেমে `locate` ইনস্টল করতেও হতে পারে। না থাকলে `apt install mlocate` (Ubuntu/Debian) দেখুন।

দ্রুত পুনরালোচনা (Quick recap) — অধ্যায় ২

- `/` রুট, `/home` ইউজার, `/etc` কনফিগ, `/var` লগ, `/tmp` অস্থায়ী।
- Sym: `/` সেপারেটর, `~` Home, `.` এখন, `..` উপরে, `-` আগের।
- `ls -la` + `ls -lh` দৈনন্দিন তালিকা দেখায়, `cd` পাথ বদলায়, `pwd` যাচাই করে।
- `tree` কাঠামো দেখায়, `find` / `locate` খুঁজতে ব্যবহার হয়।

অনুশীলনী (Practice Task)

1. `cd ~`, `mkdir ~/practice` (যদি না থাকে) আর `ls -la`-তে ধারাবাহিকতা দেখুন।
2. `cd /etc`, `pwd`, `ls -lh | head` চালান (কথা নয় — ভয় নেই, শুধু দেখুন)।
3. `cd /var/log`, প্রম্পট দিয়ে লক্ষ্য করুন, তাড়াহুড়া করবেন না।
4. `cd -` দিয়ে ফিরে আসুন — কতবার ঘুরতে পারলেন দেখুন।
5. `mkdir -p ~/practice/a/b`, `tree ~/practice` দেখুন।

০৩

অধ্যায় ৩

তৈরি, কপি, মুভ, ডিলিট: ফাইল জীবনচক্র

ফাইল তৈরি থেকে শুরু করে ডিলিট করা — এই চক্রই টার্মিনালের মূল কাজ। ভুলে মোছা থেকে বাঁচতে হলে এই অধ্যায়ের সতর্কতা হৃদয়ে রাখুন।

মহা সতর্কতা — `rm -rf` অপরিবর্তনীয়

⚠ সাবধান: `rm -rf` দিয়ে মুছা ফাইল Trash-এ যায় না। ফিরিয়ে আনার পথ নেই। `rm -rf /` বা `rm -rf ~` কোনোদিন চালাবেন না। সবসময় ডিলিট করার আগে `pwd` আর `ls` দিয়ে যাচাই করুন।

নিরাপদ প্র্যাকটিস ডিরেক্টরি

```
mkdir -p ~/practice
cd ~/practice
pwd
ls
```

শুধু এই ডিরেক্টরিতে `rm` প্র্যাকটিস করুন। মূল ডিরেক্টরিতে কখনোই নয়।

touch — খালি ফাইল তৈরি বা টাইম আপডেট

সহজ কথায়: ফাইল না থাকলে বানায়, থাকলে টাইম আপডেট করে।

Syntax

```
touch <filename>
touch <file1> <file2>
```

উদাহরণ

```
touch notes.txt
touch ~/practice/test.txt
```

↳ touch — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# আউটপুট আসে না
$ ls -l notes.txt
-rw-r--r-- 1 joy joy 0 Nov  6 10:00 notes.txt
```

কখন দরকার হবে: খালি জায়গা তৈরি, স্ক্রিপ্টের জন্য ফাইল টোকেন বানাতে।

মনে রাখো: `touch` ভেতরে কিছু লেখে না — শুধু খালি ফাইল।

cp — ফাইল কপি করা

সহজ কথায়: এক ফাইল থেকে আরেক জায়গায় ছাঁকা কপি।

Syntax

```
cp <source> <destination>
cp <source> ~/backup/
```

উদাহরণ

```
cp notes.txt notes2.txt
cp /etc/hosts ~/practice/hosts.bak
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ ls -l
-rw-r--r-- 1 joy joy 120 Nov  6 10:00 notes.txt
-rw-r--r-- 1 joy joy 120 Nov  6 10:00 notes2.txt
```

কখন দরকার হবে: কনফিগ ব্যাকআপ, নাম বদলে কপি রাখতে।

মনে রাখো: একই ডিরেক্টরিতে কপি হলে `<dest>` ফোল্ডার বা নতুন নাম দেবেন। নাহলে কনফিউশন।

cp -r — ডিরেক্টরি রিকার্সিভলি কপি

সহজ কথায়: `-r` মানে ভেতরকার সব ফাইলসহ ফোল্ডার কপি।

Syntax

```
cp -r <source_dir> <destination>
```

↳ cp -r — চলমান

উদাহরণ

```
cp -r ~/projects ~/backup/
cp -r ~/practice ~/practice-copy
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ ls ~/backup/
projects
```

কখন দরকার হবে: প্রজেক্ট ব্যাকআপ, টেম্প্লেট ক্লোন করতে।

মনে রাখো: ডিরেক্টরি কপিতে `-r` ছাড়লে Error দেবে। আর `cp -r src/ dest/` আর `cp -r src dest` — ট্রেইলিং `/` মনে রাখুন।

mv — মুভ বা রিনেম

সহজ কথায়: ফাইল বা ডিরেক্টরি সরানো, অথবা নতুন নাম দেওয়া।

Syntax

```
mv <source> <destination>
mv <oldname> <newname>
```

উদাহরণ

```
mv notes.txt ~/Documents/
mv ~/practice/test.txt ~/practice/renamed.txt
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ ls ~/Documents/
notes.txt
```

কখন দরকার হবে: সংগঠিত করতে, নাম শোধরাতে, ফোল্ডার থেকে ফোল্ডারে নিয়ে যেতে।

মনে রাখো: `mv` এক ফাইলসিস্টেমের ভেতরে হলে কপি+ডিলিট করে। স্টোরেজ আলাদা হলে সময় লাগতে পারে।

rm — ফাইল মোছা

সহজ কথায়: ফাইল চিরতরে মুছে ফেলে। Trash-এ যায় না।

Syntax

```
rm <filename>
rm <file1> <file2>
```

উদাহরণ

```
rm notes2.txt
rm ~/practice/test.txt
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ ls
notes.txt
```

কখন দরকার হবে: অপ্রয়োজনীয় ফাইল পরিষ্কার করতে, ভুল লেখা মুছতে।

মনে রাখো: **সাবধান:** `rm` অপরিবর্তনীয়। ডিলিটের আগে `ls <name>` আর `pwd` দেখুন। `rm *` খুব বিপজ্জনক।

rm -r — ডিরেক্টরি রিকার্সিভলি মোছা

সহজ কথায়: ভেতরকার সব ফাইলসহ ফোল্ডার মুছে ফেলে।

Syntax

```
rm -r <dirname>
rm -rf <dirname>
```

উদাহরণ

```
rm -r ~/practice/empty-dir
rm -rf ~/practice/cache
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ ls ~/practice/
file.txt
```

কখন দরকার হবে: ডিরেক্টরি ভেতর ফাইলসহ মুছে ফেলতে, Cache পরিষ্কার করতে।

↳ rm -r — চলমান

মনে রাখো: `rm -rf` অপরিবর্তনীয়। কখনো `rm -rf /` বা `rm -rf ~` চালাবেন না। `rm -r` চাওয়ার আগে `ls` দেখুন। স্ক্রিপ্টে `rm -rf "$dir"` লিখে ভেরিয়েবল ফাঁকা হলে সব উড়ে যেতে পারে।

rmdir — খালি ডিরেক্টরি

সহজ কথায়: শুধুমাত্র খালি ফোল্ডার নিরাপদে মোছে।

Syntax

```
rmdir <dirname>
rmdir <dir1> <dir2>
```

উদাহরণ

```
rmdir ~/practice/empty
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ rmdir ~/practice/notempty
rmdir: failed to remove '/home/username/practice/notempty': Directory not empty
```

কখন দরকার হবে: পরীক্ষার ডিরেক্টরি খালি করে মুছতে, `rm` এড়িয়ে নিরাপদে পরিষ্কার করতে।

মনে রাখো: ভেতরে ফাইল থাকলে `rmdir` ব্যর্থ হয় — এটাই প্রমাণ যে এটা নিরাপদ জানালা।

install — ফাইল কপি ও ইনস্টল (প্যাকেজ ইনস্টলের থেকে আলাদা)

সহজ কথায়: ফাইল নির্দিষ্ট জায়গায় কপি করে দেবে, পারমিশন সেট করে।

Syntax

```
install <options> <source> <destination>
install -m 755 <source> /usr/local/bin/
```

উদাহরণ

```
install -m 644 notes.txt ~/practice-notes/
install -D config.txt ~/app/conf/config.txt
```

↳ install — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
$ ls -l ~/practice-notes/
-rw-r--r-- 1 joy joy 120 Nov  6 10:00 notes.txt
```

কখন দরকার হবে: স্ক্রিপ্ট `/usr/local/bin`-এ সেট করতে, নির্দিষ্ট পারমিশনে কপি করতে।

মনে রাখো: এটি `apt install` প্যাকেজ ইনস্টলের থেকে একদম আলাদা। `install` শুধু ফাইল কপি করে — প্যাকেজ ম্যানেজার নয়।

file — ফাইলের টাইপ বোঝা

সহজ কথায়: এক্সটেনশন না দেখে ফাইল আসলে কী তা বের করা।

Syntax

```
file <filename>
```

উদাহরণ

```
file notes.txt
file /bin/ls
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
notes.txt: ASCII text
/bin/ls: ELF 64-bit LSB executable, x86-64
```

কখন দরকার হবে: ভুল এক্সটেনশন ধরা, বাইনারি vs টেক্সট বুঝতে।

মনে রাখো: `.txt` হলেও ভেতরে বাইনারি হলে টেক্সট বলবে না। `file` দেখলেই বোঝা যাবে।

stat — ফাইলের বিস্তারিত মেটাডেটা

সহজ কথায়: সাইজ, পারমিশন, মালিক, টাইমসহ সম্পূর্ণ তথ্য।

Syntax

```
stat <filename>
stat -c "%n %s" <filename>
```

↳ stat — চলমান

উদাহরণ

```
stat notes.txt
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
  File: notes.txt
  Size: 120  Blocks: 8  IO Block: 4096 regular file
Access: (0644/-rw-r--r--)  Uid: 1000/ joy   Gid: 1000/ joy
```

কখন দরকার হবে: পারমিশন সমস্যা যাচাই, স্ক্রিপ্টে ফাইলের টাইম/সাইজ দেখতে।

মনে রাখো: `-c` দিলে শুধু নির্দিষ্ট ফরম্যাটে দেখাবে — স্ক্রিপ্টিংয়ের জন্য উপকারী।

দ্রুত পুনরালোচনা (Quick recap) — অধ্যায় ৩

- `touch` তৈরি করে, `cp` কপি করে, `cp -r` ডিরেক্টরি কপি করে, `mv` মুভ/রিনেম করে।
- `rm` অপরিবর্তনীয়, `rm -rf` ডিরেক্টরি মুছে — কখনো `/` বা `~` এ নয়। `rmdir` শুধু খালি ফোল্ডার।
- `install` প্যাকেজ ইনস্টলের নয়, ফাইল কপি ও পারমিশন সেট করে। `file`/`stat` যাচাইয়ের জন্য।

অনুশীলনী (Practice Task)

1. `mkdir -p ~/practice`, `cd ~/practice`, `touch a.txt`, `cp a.txt b.txt`, `ls -l` করে দেখুন।
2. `mv b.txt c.txt`, `ls`, `file c.txt`, `stat c.txt` চালান।
3. `mkdir dir1`, `touch dir1/x.txt`, `cp -r dir1 dir2`, `ls -R` দেখুন।
4. `rm c.txt`, `rmdir dir1` — Error আসবে (কারণ ভেতরে ফাইল আছে)। বুঝুন কেন Error এলো।
5. `rm -r dir2`, `ls` — শুধু `c.txt`-এর জায়গা মনে রাখুন? এই ধাপে `rm` চালানোর আগে `pwd` আর `ls` দেখান তো।

অধ্যায় ৪

০৪ পড়া, এডিট, খোঁজা: ফাইল ও Log-এর গল্প

কনফিগ, লগ, কোড — পড়া, এডিট আর খোঁজা ছাড়া কোনো ওয়ার্কফ্লো এগোয় না।

Log Reading Example — লগ পড়ার বাস্তব উদাহরণ

```
ls -lh /var/log/
sudo tail -50 /var/log/syslog
sudo grep -i "error" /var/log/syslog | tail -30
sudo tail -f /var/log/syslog
```

এই চক্রোই DevOps-এর দৈনন্দিন কাজ। `tail -f` দিয়ে লাইভ দেখুন, `grep` দিয়ে Filter করুন।

SSH কী সতর্কতা

⚠ সাবধান: `cat ~/.ssh/id_ed25519.pub` — এই ফাইলটি **পাবলিক কী**, শেয়ার করা নিরাপদ। কিন্তু কখনোই `~/.ssh/id_ed25519` (প্রাইভেট কী) শেয়ার করবেন না। প্রাইভেট কী = আপনার পাসওয়ার্ডের চেয়েও গুরুত্বপূর্ণ।

`cat` — ফাইল সম্পূর্ণ প্রিন্ট

সহজ কথায়: পুরো ফাইল স্ক্রিনে ছাপিয়ে দেয়।

Syntax

```
cat <file>
cat <file1> <file2>
```

↳ cat — চলমান

উদাহরণ

```
cat notes.txt
cat ~/.bashrc
cat ~/.ssh/id_ed25519.pub
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# notes.txt এর ভেতরে যা আছে তা দেখাবে
ssh-ed25519 AAAAC3... user@host
```

কখন দরকার হবে: ছোট কনফিগ দেখতে, পাবলিক কী কপি করতে (প্রাইভেট নয়)।

মনে রাখো: `cat` বড় ফাইলে স্ক্রিন ভরিয়ে ফেলে। বড় ফাইলে `less` ব্যবহার করুন। আর কখনোই `cat ~/.ssh/id_ed25519` শেয়ার করবেন না।

tac — ফাইল উল্টে লাইনে প্রিন্ট

সহজ কথায়: `cat`-এর উল্টো — শেষ লাইন আগে দেখাবে।

Syntax

```
tac <file>
```

উদাহরণ

```
tac /var/log/syslog | head -20
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# /var/log/syslog-এর শেষ লাইন সবার প্রথমে আসবে
```

কখন দরকার হবে: Log-এ সবচেয়ে সাম্প্রতিক তথ্য আগে দেখতে।

মনে রাখো: `tac` লাইন উল্টায়, শব্দ উল্টায় না।

`less` — পেজার, বের হতে q

সহজ কথায়: বড় ফাইল ধীরে ধীরে, স্ক্রল করে দেখা।

Syntax

```
less <file>
```

উদাহরণ

```
less /var/log/syslog
less /etc/passwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# পেজার খুলবে, Space = পরের পেজা, q = বের হোন, /foo = খুঁজুন
```

কখন দরকার হবে: বড় লগ, বড় কনফিগ পড়তে, স্ক্রল ও খুঁজতে।

মনে রাখো: বের হতে `q` চাপুন। `/` দিয়ে খুঁজুন, `n` দিয়ে পরের, `N` দিয়ে আগের।

`more` — ঐতিহ্যবাহী পেজার (সংক্ষেপে)

সহজ কথায়: `less`-এর পুরনো ভাই — স্ক্রল করে পেজ বদলায়।

Syntax

```
more <file>
```

উদাহরণ

```
more /var/log/syslog
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# Space = পরের পেজা, Enter = পরের লাইন, q = বের হোন
```

কখন দরকার হবে: শুধু স্ক্রল করলেই হবে এমন ক্ষেত্রে, `less` না থাকলে ব্যাকআপ।

মনে রাখো: `less`-এ ফিরে খুঁজতে খুব সুবিধা — দৈনন্দিন কাজে `less`-ই ব্যবহার করুন।

head — শুরুর লাইন দেখা

সহজ কথায়: ফাইলের প্রথম ১০ লাইন (ডিফল্ট)।

Syntax

```
head <file>
head -n 20 <file>
```

উদাহরণ

```
head notes.txt
head -n 5 /var/log/syslog
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# notes.txt-এর প্রথম ১০ লাইন
```

কখন দরকার হবে: ফাইল শুরুতে কী আছে দেখতে, CSV-এর হেডার রাতে।

মনে রাখো: `head` শেষ লাইন দেখায় না — তার জন্য `tail`।

tail — শেষের লাইন দেখা

সহজ কথায়: ফাইলের শেষ ১০ লাইন।

Syntax

```
tail <file>
tail -n 50 <file>
```

উদাহরণ

```
tail /var/log/syslog
tail -n 50 /var/log/nginx/error.log
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# লগের শেষ ১০ লাইন — সেখানেই সর্বশেষ এরর
```

কখন দরকার হবে: Log-এর সাম্প্রতিক ঘটনা দেখতে, Crash-এর কারণ বের করতে।

মনে রাখো: Log দেখার প্রথম অভ্যাসই হলো `tail -50`।

tail -f — Log Follow: লাইভ লগ দেখা

সহজ কথায়: নতুন লাইন আসলেই ছাপিয়ে দেয়, Ctrl+C দিয়ে আটকাতে হয়।

Syntax

```
tail -f <file>
tail -F <file>
```

উদাহরণ

```
tail -f /var/log/syslog
sudo tail -f /var/log/auth.log
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# নতুন লাইন আসলেই ছাপবে, থামাতে Ctrl+C
```

কখন দরকার হবে: সার্ভারে লাইভ এরর দেখতে, ডিবাগ করতে, DevOps-এ লাইভ মনিটর করতে।

মনে রাখো: `tail -F` গতকালের রোটেটেড লগও ফলো করে, `-f` শুধু বর্তমান ফাইল। আটকাতে `Ctrl+C`।

nano — টার্মিনাল টেক্সট এডিটর

সহজ কথায়: টার্মিনালে ফাইল এডিটের সহজ উপায়।

Syntax

```
nano <file>
```

উদাহরণ

```
nano notes.txt
nano ~/.bashrc
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# এডিটর খুলবে, Ctrl+O = Save, Enter = নিশ্চিত, Ctrl+X = বের হোন
```

কখন দরকার হবে: কনফিগ এডিটে, `~/.bashrc`-তে জিনিস যোগ করতে।

↳ nano — চলমান

মনে রাখো: সেভ: `Ctrl+O` তারপর `Enter`, বের হোন: `Ctrl+X`। উপরে সাহায্য লেখা আছে। আর `nano ~/.bashrc`-এর উদাহরণ: খুলে আপনার নামের অ্যালিয়াস যোগ করুন।

grep — লাইনে খোঁজা

সহজ কথায়: ফাইলের ভেতরে লাইনে লাইনে প্যাটার্ন খোঁজে।

Syntax

```
grep "<pattern>" <file>
grep -r "<pattern>" <directory>
```

উদাহরণ

```
grep "error" notes.txt
grep -r "TODO" ~/projects
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
line 12: error in config file
```

কখন দরকার হবে: কনফিগে মান খুঁজতে, লগে এরর বাছতে, কোডে TODO দেখতে।

মনে রাখো: প্যাটার্নে স্পেস থাকলে কোট দিয়ে দেবেন। `grep` ছাড়া ফাইল খোঁজা অসম্ভব হয়।

grep -i — Case-insensitive খোঁজা

সহজ কথায়: বড়-ছোট হাতের অক্ষর না দেখে খুঁজবে।

Syntax

```
grep -i "<pattern>" <file>
grep -i "error" /var/log/syslog
```

উদাহরণ

```
grep -i "error" /var/log/syslog
grep -i "home" /etc/passwd
```

↳ grep -i — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
ERROR: ...
Error in file ...
```

কখন দরকার হবে: Log-এ `ERROR`, `Error`, `error` সব মেলাতে।

মনে রাখো: `-i` মানে Ignore case। লগে এটা না দিলে অনেক ম্যাচ হারিয়ে যাবে।

grep -n — লাইন নম্বরসহ দেখা

সহজ কথায়: কোন লাইনে পাওয়া গেল তার নম্বরসহ দেখাবে।

Syntax

```
grep -n "<pattern>" <file>
```

উদাহরণ

```
grep -n "error" notes.txt
grep -n "TODO" ~/projects/file.sh
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
12:error in config file
45:TODO fix this
```

কখন দরকার হবে: কনফিগে ঠিক লাইন খুলতে, এডিটর দ্বারা লাল দাগে পৌঁছাতে।

মনে রাখো: `-n` এর বদলে `nano +12 file` দিয়ে সরাসরি লাইন ১২-এ গিয়ে খুলুন।

grep -r — ডিরেক্টরি রিকার্সিভলি খোঁজা

সহজ কথায়: `-r` মানে ভেতর-বাইরের সব ফাইলে খোঁজে।

Syntax

```
grep -r "<pattern>" <directory>
grep -rn "<pattern>" <directory>
```

↳ grep -r — চলমান

উদাহরণ

```
grep -r "error" /var/log
grep -rn "TODO" ~/projects
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/var/log/syslog:ERROR ...
/var/log/auth.log:failed ...
```

কখন দরকার হবে: পুরো প্রজেক্টে খুঁজতে, লগে এরর বের করতে।

মনে রাখো: `-rn` একসাথে লাইন নম্বর দেবে। `-r` অনেকে `-R` ব্যবহার করে লিংক ফলো করে।

grep -v — Invert: না-থাকা লাইন দেখা

সহজ কথায়: `-v` মানে যেসব লাইনে প্যাটার্ন নেই, সেগুলোই দেখোলো।

Syntax

```
grep -v "<pattern>" <file>
```

উদাহরণ

```
grep -v "^#" config.txt
grep -v "error" /var/log/syslog
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# কমেন্ট ছাড়া কনফিগ
```

কখন দরকার হবে: কমেন্ট বাদ দিতে, লগে শুধু সাকসেস দেখতে।

মনে রাখো: `grep -v "^#"` কমেন্ট ছাড়া লাইন দেবে — কনফিগ পরিষ্কার করার টিপস।

`find` — পাথ-নাম দিয়ে খোঁজা (ফাইলসিস্টেম ব্যাপক)

সহজ কথায়: প্যাটার্ন, টাইপ, সময় দিয়ে ফাইল বের করা।

Syntax

```
find <path> -name "<pattern>" -type f
find <path> -mtime -1
```

উদাহরণ

```
find ~ -name "*.log" -type f
find /var/log -mtime -2 -type f
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/home/username/app.log
/var/log/syslog
```

কখন দরকার হবে: `.log`, `.sh`, কনফিগ একগুচ্ছ বের করতে, ব্যাকআপ স্ক্রিপ্ট লিখতে।

মনে রাখো: `-type f` ফাইল, `-type d` ডিরেক্টরি। নিচের ফিল্ডে প্র্যাকটিস করুন।

`wc` — লাইন, শব্দ, ক্যারেক্টার গোনা

সহজ কথায়: ফাইলে কত লাইন, কত শব্দ, কত বাইট।

Syntax

```
wc <file>
wc -l <file>
wc -w <file>
```

উদাহরণ

```
wc notes.txt
wc -l /var/log/syslog
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
  120  450  3000 notes.txt
  5000
```

↳ wc — চলমান

কখন দরকার হবে: Log-এর লাইন সংখ্যা দেখতে, লেখার দৈর্ঘ্য মাপতে।

মনে রাখো: `wc -l` দিয়ে লাইন সস্তা হিসাব। `-c` বাইট, `-w` শব্দ।

sort — সাজানো

সহজ কথায়: লাইনগুলো Alphabet/সংখ্যা অনুযায়ী সাজায়।

Syntax

```
sort <file>
sort -n <file>
sort -r <file>
```

উদাহরণ

```
sort notes.txt
sort -t: -k3 -n /etc/passwd
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
apple
banana
orange
```

কখন দরকার হবে: লগ সাজাতে, ইউজার লিস্ট সাজাতে, ডিফারেন্স আগে প্রস্তুতি নিতে।

মনে রাখো: `-n` সংখ্যা, `-r` বিপরীত, `-t:` ডেলিমিটার, `-k3` কলাম।

uniq — ডুপ্লিকেট সরানো

সহজ কথায়: পাশাপাশি একই লাইন একটাই রাখে।

Syntax

```
sort <file> | uniq
uniq -c
```

উদাহরণ

```
cat notes.txt | sort | uniq
sort notes.txt | uniq -c
```

↳ uniq — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
3 error
1 success
```

কখন দরকার হবে: লগে একই এরর কতবার দেখতে, ডুপ্লিকেট সরাতে।

মনে রাখো: `uniq` শুধু পাশাপাশি সাজালেই কাজ করে — তাই আগে `sort` করতে হয়।

cut — কলাম/ফিল্ড বাছা

সহজ কথায়: `:` বা `,` দিয়ে আলাদা ফিল্ড বের করা।

Syntax

```
cut -d: -f1 <file>
cut -d, -f2 file.csv
cut -c1-10 file
```

উদাহরণ

```
cut -d: -f1 /etc/passwd
cut -d, -f2 users.csv
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
root
joy
devops
```

কখন দরকার হবে: `/etc/passwd` থেকে শুধু ইউজার বের করতে, CSV-র কলাম বাছতে।

মনে রাখো: `-d:` ডেলিমিটার, `-f1` প্রথম ফিল্ড। `-f1,3` মানে ১ আর ৩।

tr — ক্যারেক্টার ট্রান্সলেট

সহজ কথায়: ছোট থেকে বড়, ব্র্যাকেট বদল, ডিলিট।

Syntax

```
tr 'a-z' 'A-Z' < <file>
tr -d '\n' <file>
```

↳ tr — চলমান

উদাহরণ

```
echo "hello" | tr 'a-z' 'A-Z'
tr -d ' ' < notes.txt
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
HELLO
```

কখন দরকার হবে: Case বদলাতে, স্পেস/নিউলাইন সরাতে।

মনে রাখো: `tr` ক্যাশে ইন-প্লেস বদলায় না — পাইপ বা রিডাইরেক্ট লাগবে।

diff — দুটি ফাইলের পার্থক্য

সহজ কথায়: কোন লাইনে কী বদলেছে তা তুলে ধরবে।

Syntax

```
diff <file1> <file2>
diff -u <file1> <file2>
```

উদাহরণ

```
diff notes.txt notes2.txt
diff -u /etc/hosts ~/backup/hosts.bak
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
< old line
---
> new line
```

কখন দরকার হবে: কনফিগ বদলের ট্র্যাক, জিত-এর জন্য বেসলাইন দেখতে।

মনে রাখো: `-u` unified ফরম্যাটে দেখাবে — পড়তে সবচেয়ে সহজ।

দ্রুত পুনরালোচনা (Quick recap) — অধ্যায় ৪

- `cat` / `tac` সম্পূর্ণ, `less` পেজার, `head` / `tail` সীমিত, `tail -f` লাইভ।
- `nano` এডিটর: `Ctrl+O` সেভ, `Ctrl+X` বের হোন। `nano ~/.bashrc` উদাহরণ।
- `grep` সবচেয়ে শক্তিশালী খোঁজার কমান্ড — `-i`, `-n`, `-r`, `-v` দিয়ে টিউন করুন।
- `wc`, `sort`, `uniq`, `cut`, `tr`, `diff` প্রসেসিংয়ের জন্য।

অনুশীলনী (Practice Task)

1. `ls -la ~/.ssh`, আপনি `.pub` ফাইলটি দেখলেন — `cat ~/.ssh/id_ed25519.pub` চালান। ⚠ সাবধান: প্রাইভেট কী শেয়ার করবেন না।
2. `sudo tail -50 /var/log/syslog`, `sudo grep -i "error" /var/log/syslog | tail -20` চালান।
3. `nano ~/practice/note.txt` খুলে লিখে `Ctrl+O`, `Enter`, `Ctrl+X` করুন।
4. `grep -n "bash" ~/.bashrc`, `wc -l ~/.bashrc` চালান।
5. `echo -e "a\nb\na" | sort | uniq -c` — ডুপ্লিকেট কতটা বোঝা গেল দেখুন।

০৫

অধ্যায় ৫

প্যাকেজ ম্যানেজমেন্ট (শুধু Ubuntu/Debian)

স্পষ্ট নোট: এই অধ্যায়ের সব কমান্ড — `sudo`, `apt`, `dpkg` — **শুধু Ubuntu/Debian-এর জন্য**। Arch-এ `pacman`, Fedora-তে `dnf` চলে। ধারণা সব ডিস্ট্রোতে একই, কিন্তু কমান্ড এক না। `apt` সার্বজনীন নয়।

Repositories, Indexes, Dependencies, .deb

- **Repositories (রিপোজিটরিস):** প্যাকেজের ভান্ডার, অনলাইন সংগ্রহ।
- **Indexes (ইনডেক্সেস):** কোন প্যাকেজ আছে তার ক্যাটালগ। `apt update` এই ক্যাটালগ আপডেট করে।
- **Dependencies (ডিপেন্ডেন্সিস):** প্যাকেজ চলতে আরো প্যাকেজ লাগে। ম্যানেজার সেটা নিজেই বুঝে নেয়।
- `.deb` : লিনাক্সের ইনস্টল ফাইল, `dpkg` / `apt` দিয়ে ইনস্টল হয়।
- **প্যাকেজ ম্যানেজার vs স্টোর:** স্টোর GUI, ক্লিক করে ইনস্টল। প্যাকেজ ম্যানেজার টার্মিনালে স্ক্রিপ্ট করে ব্যবহার করা যায় — DevOps-এ এটাই হাতের হাতিয়ার।

সাধারণ ভুল: "`apt update` দিলেই আপগ্রেড হয়ে গেল" — ভুল। আপডেট ইনডেক্স রিফ্রেশ করে, আপগ্রেড আলাদা ধাপ। ভুল ধারণা থেকে নিরাপত্তা ফাঁকা থাকে।

sudo — উন্নত সুযোগে কমান্ড চালানো

সহজ কথায়: ৫ মিনিটের জন্য `root`-এর মতো কাজ করার অনুমতি চায়।

Syntax

```
sudo <command>
```

উদাহরণ

```
sudo apt update
sudo systemctl restart nginx
```

↳ sudo — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
[sudo] password for joy:
# পাসওয়ার্ড টাইপের সময় স্ক্রিনে কিছু দেখাবে না — এটা নিরাপত্তা ফিচার
```

কখন দরকার হবে: সিস্টেম ফাইল লিখতে, সার্ভিস রিস্টার্টে, প্যাকেজ ইনস্টলে।

মনে রাখো: `sudo` **নিরাপদ করে তোলে না** — শুধু Elevate করে। পাসওয়ার্ড টাইপের সময় ইঙ্গিত আসে না, ভুল হলে আবার চাইবে। `whoami` = `joy` থাকলে `sudo` না দিলে পারমিশন এরর পাবেন।

apt update — ইনডেক্স রিফ্রেশ (শুধু)

সহজ কথায়: রিপোজিটরির ক্যাটালগ আপডেট করে, প্যাকেজ ইনস্টল/আপগ্রেড কিছুই করে না।

Syntax

```
sudo apt update
```

উদাহরণ

```
sudo apt update
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Hit:1 http://archive.ubuntu.com/ubuntu jammy...
Fetched 50 MB in 5s
Reading package lists... Done
All packages are up to date.
```

কখন দরকার হবে: `apt install` / `upgrade` -এর আগে সবসময় একবার চালাতে।

মনে রাখো: `apt update` ভার্সন বদলায় না, শুধু কোন ভার্সন আছে তা রিফ্রেশ করে। ভুল ধারণা ভেঙে দিন।

apt upgrade — প্যাকেজ আপগ্রেড (নিরাপদ)

সহজ কথায়: ইতিমধ্যে ইনস্টল প্যাকেজ আপডেট করবে, নতুন প্যাকেজ যোগ না করে।

Syntax

```
sudo apt upgrade
```

↳ apt upgrade — চলমান

উদাহরণ

```
sudo apt upgrade
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
The following packages will be upgraded:
  git libcurl4
0 upgraded, 2 newly installed, 0 to remove
```

কখন দরকার হবে: নিয়মিত নিরাপত্তা আপডেটে।

মনে রাখো: `upgrade` আপগ্রেড করতে ব্যর্থ হলে `full-upgrade` লাগতে পারে। আগে `apt list --upgradable` দেখুন।

apt full-upgrade — পূর্ণ আপগ্রেড (যোগ/বিয়োগ হতে পারে)

সহজ কথায়: `upgrade`-এর মতো, তবে প্রয়োজন বুঝে নতুন প্যাকেজ যোগ বা অপসারণ করতে পারে।

Syntax

```
sudo apt full-upgrade
```

উদাহরণ

```
sudo apt full-upgrade
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
The following packages will be REMOVED:
  libold
The following NEW packages will be installed:
  libnew
```

কখন দরকার হবে: ডেস্কটপ/সার্ভারে নিয়মিত সিস্টেম আপগ্রেডে।

মনে রাখো: **Review করুন** — কোন প্যাকেজ যোগ/বিয়োগ হবে তা মনোযোগে পড়ুন। সন্দেহ থাকলে Ctrl+C করে বের হয়ে যান।

apt install — প্যাকেজ ইনস্টল

সহজ কথায়: নাম দিয়ে প্যাকেজ ইনস্টল করে।

Syntax

```
sudo apt install <package_name>
sudo apt install <package_name> -y
sudo apt install ./filename.deb
```

উদাহরণ

```
sudo apt install git
sudo apt install git curl wget unzip htop
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Reading package lists... Done
Setting up git (2.34.1-1ubuntu1) ...
```

কখন দরকার হবে: নতুন টুল বসাতে, ডিপেন্ডেন্সিসহ পুরো সেট, সার্ভারে টুলচেইন তৈরি করতে।

মনে রাখো: `<package_name>` আপনার প্রয়োজন অনুযায়ী বসবে। Local .deb-এর জন্য `sudo apt install ./filename.deb` — এখানে `./` মানে এখানকার ফাইল।

apt reinstall / --reinstall — প্যাকেজ পুনঃইনস্টল

সহজ কথায়: একই প্যাকেজ আবার ইনস্টল করে, ভাঙা ফাইল ঠিক করে।

Syntax

```
sudo apt reinstall <package_name>
sudo apt install --reinstall <package_name>
```

উদাহরণ

```
sudo apt reinstall git
sudo apt install --reinstall curl -y
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Setting up git (2.34.1-1ubuntu1) ...
```

↳ apt reinstall / --reinstall — চলমান

কখন দরকার হবে: প্যাকেজ ভেঙে গেলে, আপডেটে সমস্যা হলে।

মনে রাখো: দুটি ফরমই একই কাজ করে। `--reinstall` স্ক্রিপ্টে স্পষ্ট করে লেখে।

apt remove — প্যাকেজ মোছে, কনফিগ রাখে

সহজ কথায়: প্রোগ্রাম মুছবে, কিন্তু কনফিগ ফাইল রেখে যাবে।

Syntax

```
sudo apt remove <package_name>
```

উদাহরণ

```
sudo apt remove git
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Removing git (2.34.1...) ...
```

কখন দরকার হবে: অপ্রয়োজনীয় টুল সরাতে, স্টোরেজ বাঁচাতে।

মনে রাখো: কনফিগ রেখে যায়। আবার ইনস্টল করলে আগের সেটিং ফিরে আসতে পারে — চাওয়ার মতো হতে পারে।

apt purge — প্যাকেজ ও কনফিগ সহ মোছে

সহজ কথায়: প্রোগ্রাম ও তার কনফিগ পুরো মুছবে।

Syntax

```
sudo apt purge <package_name>
```

উদাহরণ

```
sudo apt purge git
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Removing git ...
Purging configuration files for git ...
```

↳ apt purge — চলমান

কখন দরকার হবে: টুল পুরোপুরি মুছে ফেলতে, স্ক্র্যাচ শুরু।

মনে রাখো: ইউজার ডেটা (~/.gitconfig) কখনো মুছবে না। শুধু সিস্টেম কনফিগ মুছে। `purge` বোঝার আগে `remove` দেখুন।

apt autoremove — অপ্রয়োজনীয় ডিপেন্ডেন্সি পরিষ্কার

সহজ কথায়: আর লাগে না এমন প্যাকেজ সাফ করে।

Syntax

```
sudo apt autoremove
sudo apt autoremove --purge
```

উদাহরণ

```
sudo apt autoremove
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
The following packages will be REMOVED:
  libfoo libbar
0 upgraded, 0 newly installed, 2 to remove
```

কখন দরকার হবে: আপগ্রেডের পর ডিস্ক বাঁচাতে, প্যাকেজ মোছার পর পরিষ্কার করতে।

মনে রাখো: **Review list করুন।** ভুল করে দরকারি প্যাকেজ মুছে যেতে পারে। সন্দেহ থাকলে `Ctrl+C`।

apt clean — আর্কাইভ ক্যাশ পরিষ্কার

সহজ কথায়: ডাউনলোডেড .deb আর্কাইভের ক্যাশ মুছে == Disk বাঁচায়।

Syntax

```
sudo apt clean
```

উদাহরণ

```
sudo apt clean
```

↳ apt clean — চলমান

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
# আউটপুট আসে না, ডিস্ক বাঁচে
```

কখন দরকার হবে: ডিস্ক ভরে যাচ্ছে মনে হলে, মাঝেমাঝে পরিষ্কার করতে।

মনে রাখো: `apt clean` **সব ক্যাশ মুছে না।** ইনডেক্স সাফ করতে `apt clean` নয় — আরেক কমান্ডের দরকার পড়ে না, ইনডেক্সে থাকা বড় ব্যাখ্যা আমি বদলাব না। `dpkg` নয়।

apt autoclean — শুধু অব্যবহৃত আর্কাইভ সাফ

সহজ কথায়: আর দরকার নেই এমন .deb ক্যাশ মুছবে।

Syntax

```
sudo apt autoclean
```

উদাহরণ

```
sudo apt autoclean
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Del package.deb
```

কখন দরকার হবে: Disk বাঁচাতে, নিয়মিত সাফাই।

মনে রাখো: এতটাই হালকা পরিষ্কার, ঝুঁকি নেই। `clean`-এর মতো আক্রমণাত্মক নয়।

apt search — প্যাকেজ খোঁজা

সহজ কথায়: নাম বা বিবরণে মিলিয়ে প্যাকেজ তালিকা দেয়।

Syntax

```
apt search <keyword>
sudo apt search <keyword>
```

↳ apt search — চলমান

উদাহরণ

```
apt search git
apt search python3 | head -20
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
git - fast version control system
git-doc - documentation ...
```

কখন দরকার হবে: সঠিক নাম খুঁজতে, বিকল্প টুল দেখতে।

মনে রাখো: `apt search` সাধারণত `sudo` ছাড়া চলে। `man apt` দেখলে আরো অপশন পাবেন।

apt show — প্যাকেজের বিস্তারিত তথ্য

সহজ কথায়: কোন প্যাকেজ কী করে, দৈর্ঘ্য, নির্ভরতা, রক্ষণাবেক্ষক।

Syntax

```
apt show <package_name>
```

উদাহরণ

```
apt show git
apt show curl | head -30
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Package: git
Version: 2.34.1-1ubuntu1
Installed-Size: 25000
Description: ...
```

কখন দরকার হবে: ইনস্টল করার আগে যাচাই, দৈর্ঘ্য দেখতে, বিবরণ পড়তে।

মনে রাখো: `apt show`-তে Maintainer দেখুন — কার উপর আস্থা রাখছেন।

apt list --installed — ইনস্টল প্যাকেজের তালিকা

সহজ কথায়: কী কী ইনস্টল আছে সংক্ষেপে।

Syntax

```
apt list --installed
apt list --installed | grep <pattern>
```

উদাহরণ

```
apt list --installed | head -20
apt list --installed | grep git
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
git/jammy-updates,now 2.34.1-1ubuntu1 amd64 [installed]
```

কখন দরকার হবে: মেশিনে কী আছে যাচাই, ইনস্টল সফল হয়েছে দেখতে।

মনে রাখো: তালিকা বড় হয় — `| head` দিলে সূচনা পাবেন।

apt list --upgradable — আপগ্রেডযোগ্য তালিকা

সহজ কথায়: কোন প্যাকেজ আপডেট পাবে তা দেখাবে।

Syntax

```
apt list --upgradable
```

উদাহরণ

```
apt list --upgradable
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
git/jammy-updates 2.34.1-1ubuntu1 amd64 [upgradable from: 2.30.0]
```

কখন দরকার হবে: আপগ্রেড করার আগে কী আপডেট পাবে দেখতে।

মনে রাখো: `apt upgrade`-এর আগে এই কমান্ড চালাবেন। Review ছাড়া আপগ্রেড করবেন না।

dpkg -i — Local .deb ইনস্টল (নিচের স্তর)

সহজ কথায়: .deb ফাইল সরাসরি ইনস্টল করে।

Syntax

```
sudo dpkg -i <filename.deb>
```

উদাহরণ

```
sudo dpkg -i ./filename.deb
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
Setting up package-name (1.0) ...
```

কখন দরকার হবে: আর্কাইভ থেকে ম্যানুয়াল ইনস্টল, অফলাইনে বসাতে।

মনে রাখো: `dpkg -i` ডিপেন্ডেন্সি নিজে থেকে বসায় না। তারজন্য `sudo apt install ./filename.deb` উচিত — এখানে `apt` ডিপেন্ডেন্সি মিলিয়ে নেয়।

dpkg -l — ইনস্টল প্যাকেজ তালিকা (নিচের স্তর)

সহজ কথায়: ইনস্টল ডেটাবেসে কী কী আছে।

Syntax

```
dpkg -l | grep <pattern>
dpkg -l | head -20
```

উদাহরণ

```
dpkg -l | grep git
dpkg -l | head -20
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
ii  git  2.34.1-1ubuntu1  all  fast...
```

কখন দরকার হবে: প্যাকেজ ইনস্টল আছে দেখতে, `apt` থেকে অপেক্ষা করতে।

মনে রাখো: `dpkg -l` (`ii` মানে ইনস্টলড) আর `apt list --installed` দুটোই তালিকা দেয়।

dpkg -L — কোন প্যাকেজ কোন ফাইল বসালো

সহজ কথায়: প্যাকেজের ভেতরে কী কী ফাইল/ডিরেক্টরি কোথায় বসেছে।

Syntax

```
dpkg -L <package_name>
```

উদাহরণ

```
dpkg -L git
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
/usr/bin/git
/usr/share/doc/git
```

কখন দরকার হবে: ফাইল কোথায় গেল দেখতে, কনফিগ কপি করতে।

মনে রাখো: `-L` বড় হাতের। ছোট হাতের `-l` তালিকা, বড় হাতের `-L` ফাইলের তালিকা।

dpkg -S — ফাইল কোন প্যাকেজের

সহজ কথায়: ফাইলের পথ দিয়ে প্যাকেজ খোঁজা।

Syntax

```
dpkg -S <filepath>
```

উদাহরণ

```
dpkg -S /usr/bin/git
dpkg -S /etc/gitconfig
```

সম্ভাব্য Output

উদাহরণ Output — মেশিনভেদে ভিন্ন হতে পারে

```
git: /usr/bin/git
```

কখন দরকার হবে: এক্সিকিউটেবল কোন প্যাকেজের বুঝতে, ভাঙা কমান্ডের উৎস খুঁজতে।

মনে রাখো: `-S` বড় হাতের। ফাইলের নাম স্পষ্ট জানা থাকলে এই কমান্ড দ্রুত হয়।

নিরাপদ ওয়ার্কফ্লো (Safe Workflow) — Ubuntu/Debian

```
sudo apt update
sudo apt install git curl wget unzip htop -y
```

- উপরে `-y` মানে "প্রশ্ন না করে হ্যাঁ ধরে নাও"। **বিগিনার্স: `-y` খানিকটা না বোঝা পর্যন্ত Omit করুন** — মানে প্রথম দিকে `-y` দেবেন না। প্রশ্ন পড়া শিখুন।
- Local `.deb`-র জন্য:

```
sudo apt install ./filename.deb
```

- এখানে `./` মনে রাখুন — বর্তমান ডিরেক্টরিতে ফাইল আছে বোঝায়। `dpkg -i`-তে ডিপেন্ডেন্সি বসে না, `apt install ./file.deb`-তে বসে।

দ্রুত পুনরালোচনা (Quick recap) — অধ্যায় ৫

- সব কমান্ড Ubuntu/Debian-এর জন্য। `apt` সার্বজনীন নয়।
- `sudo` Elevate করে, নিরাপদ করে না। পাসওয়ার্ড টাইপে কিছু দেখায় না।
- `update` = ইনডেক্স রিফ্রেশ শুধু, `upgrade`/`full-upgrade` আপগ্রেড, `install` ইনস্টল, `remove` কনফিগ রাখে, `purge` কনফিগ মুছে।
- `autoremove`/`clean`/`autoclean` পরিষ্কার — Review লিস্ট দেখে করুন।
- Local .deb: `sudo apt install ./file.deb` শ্রেষ্ঠ।

অনুশীলনী (Practice Task — শুধু Ubuntu/Debian)

1. `sudo apt update` চালান — আউটপুট পড়ুন, কোনো ইনস্টল হয়নি বুঝুন।
2. `apt list --installed | grep git`, `apt show git | head -20` দেখুন।
3. `sudo apt install git curl wget unzip htop` — `-y` ছাড়া চালিয়ে প্রশ্ন দেখুন। শুধু `y` চাপুন।
4. `dpkg -l | grep git`, `dpkg -L git | head`, `dpkg -S /usr/bin/git` চালিয়ে পার্থক্য বুঝুন।
5. `sudo apt remove git` আর `sudo apt purge htop` — আপস মধ্যে পার্থক্য দেখুন। ভুল কমান্ড চালাবেন না, আপনারা আটকাবেন না তো?

অধ্যায় ৬

০৬

ইউজার, গ্রুপ ও পারমিশন — কে কী করতে পারবে

আপনি ফাইল খুলতে চান, চালাতে চান, বা মুছে ফেলতে চান — এসবের পেছনে আছে পারমিশন। প্রতিটি ফাইল ও ডিরেক্টরির একজন **মালিক (owner)** থাকে, একটি **গ্রুপ (group)** থাকে, আর **অন্যরা (others)** থাকে। কার কী অধিকার আছে, সেটাই এখানে শিখব।

তিনটি মৌলিক অনুমতি আছে:

- **r (read / পড়া)** = `4` — ফাইল পড়া যায়, ডিরেক্টরি দেখা যায়
- **w (write / লেখা)** = `2` — ফাইল বদলানো যায়, ডিরেক্টরিতে ফাইল যোগ করা/মোছা যায়
- **x (execute / চালানো)** = `1` — ফাইল চালানো যায়, ডিরেক্টরিতে প্রবেশ করা যায়

সবগুলো যোগ করে সংখ্যা হয়। যেমন: `7` = `4+2+1` = rwx (সব অনুমতি), `6` = `4+2` = rw- (পড়া + লেখা, চালানো নয়), `5` = `4+1` = r-x (পড়া + চালানো), `4` = r--, `0` = --- (কোনো অনুমতি নেই)।

তিন ঘণ্টার অবস্থান সবসময় একই ক্রমে থাকে: **owner / group / others**। তাই `755` মানে owner-এর 7, group-এর 5, others-এর 5। `644` মানে owner-এর 6, group-এর 4, others-এর 4।

দুইভাবে পারমিশন দেওয়া যায়:

- **সংখ্যা দিয়ে (numeric):** `chmod 644 file.txt`, `chmod 755 script.sh`
- **প্রতীক দিয়ে (symbolic):** `chmod +x script.sh`, `chmod u=rw,g=r,o=r file.txt`

এই বইয়ে `<username>`, `<server_ip>`-এর মতো লেখা দেখলে বুঝবেন — ওখানে আপনার আসল নাম বা ঠিকানা বসাবেন। যেমন `<username>` মানে আপনার লিনাক্স ইউজারনেম। আসল পাসওয়ার্ড, আসল সার্ভার ঠিকানা বা সিক্রেট এই বইয়ে নেই।

লিনাক্সে দুই ধরনের ডিস্ট্রো দেখা যায় — Debian/Ubuntu-ভিত্তিক এবং RHEL/Fedora-ভিত্তিক। বেশিরভাগ কমান্ড দুই ডিস্ট্রোতেই একইভাবে চলে। যেখানে আলাদা, সেখানে নোট দেওয়া আছে।

`sudo` ব্যবহারের সময় পাসওয়ার্ড লেখা টার্মিনালে দেখায় না — না ডট, না তারকা (*)। এটা স্বাভাবিক, ভয় পাবেন না।

`ls -l` — ফাইল ও পারমিশনের বিস্তারিত তালিকা

সহজ কথায়: কোন ফাইলের পারমিশন কী, মালিক কে, সাইজ কত — এক নজরে দেখা।

Syntax

```
ls -l [পথ]
```

↳ ls -l — চলমান

উদাহরণ

```
ls -l
ls -l ~/documents
```

সম্ভাব্য Output

```
-rw-r--r-- 1 <username> <username> 1200 Mar 10 14:30 file.txt
drwxr-xr-x 2 <username> <username> 4096 Mar 10 14:00 project
-rwxr-xr-x 1 <username> <username>  450 Mar 10 09:12 script.sh
```

প্রথম অক্ষর `-` মানে সাধারণ ফাইল, `d` মানে ডিরেক্টরি। এরপর তিন ঘণ্টা করে owner/group/others-এর `rwx` অনুমতি। উপরের `script.sh` এ `rwxr-xr-x` মানে owner পড়তে, লিখতে, চালাতে পারে; group ও others পড়ে চালাতে পারে।

কখন দরকার হবে: পারমিশন এরর পেলে আগে এটা দিয়ে যাচাই করবেন।

মনে রাখো: `ls -l` হলো পারমিশন চেকের প্রথম কমান্ড।

chmod — পারমিশন বদলানো

সহজ কথায়: ফাইল পড়া, লেখা বা চালানোতে কারা কী করতে পারবে, সেটা বদলে দেওয়া।

Syntax

```
chmod <সংখ্যা-অথবা-প্রতীক> <ফাইল>
```

উদাহরণ

```
chmod +x script.sh
chmod 644 file.txt
chmod 755 script.sh
```

সম্ভাব্য Output

কোনো আউটপুট আসে না। কাজ সফল হলে টার্মিনাল চুপ থাকে। পরের লাইনে `ls -l` দিয়ে যাচাই করুন।

কখন দরকার হবে: স্ক্রিপ্ট চালাতে `Permission denied` এলে, কনফিগ ফাইল সুরক্ষিত করতে চাইলে।

সাবধান: `chmod 777` সাধারণত খারাপ অভ্যাস — সবাইকে পড়া, লেখা, চালানো দিয়ে দিলে নিরাপত্তা কমে। সবসময় **least privilege** মেনে চলুন — যে যতটুকু দরকার, শুধু ততটুকু দিন। `-R` দিয়ে রিকার্সিভ বদল খুব ঝুঁকিপূর্ণ, না বুঝে চালাবেন না।

মনে রাখো: `755` সাধারণত স্ক্রিপ্ট ও ডিরেক্টরির জন্য ভালো, `644` সাধারণ ফাইলের জন্য ভালো।

chown — ফাইলের মালিক বদলানো

সহজ কথায়: ফাইলের মালিক কে হবে, সেটা ঠিক করে দেওয়া।

Syntax

```
sudo chown <username>:<group> <ফাইল>
```

উদাহরণ

```
sudo chown <username>:<username> file.txt
sudo chown <username>:<username> ~/project
```

সম্ভাব্য Output

সাধারণত আউটপুট আসে না। `ls -l` দিয়ে মালিক বদলেছে কিনা দেখুন।

`<username>`-এর জায়গায় আপনার আসল ইউজারনেম বসাবেন।

কখন দরকার হবে: `sudo` দিয়ে তৈরি করা ফাইল নিজের নামে আনার জন্য, ওয়েব সার্ভারের ফাইলের মালিক ঠিক করার জন্য।

সাবধান: সাধারণত `sudo` ছাড়া `chown` চলে না। ভুল পথে রিকার্সিভ `chown` দিলে সিস্টেম নষ্ট হতে পারে।

মনে রাখো: চাইল্ড প্রসেসের মতো, ফাইলের মালিকও অ্যাকাউন্টের কাজকে সহজ করে।

chgrp — ফাইলের গ্রুপ বদলানো

সহজ কথায়: ফাইল কোন গ্রুপের অধীনে থাকবে, সেটা বদলানো।

Syntax

```
sudo chgrp <group> <ফাইল>
```

উদাহরণ

```
sudo chgrp <group> file.txt
sudo chgrp <group> ~/project
```

সম্ভাব্য Output

আউটপুট আসে না। `ls -l` দিয়ে যাচাই করুন।

কখন দরকার হবে: টিমের সবাই যাতে একই ফোল্ডারে কাজ করতে পারে, সেজন্য।

মনে রাখো: `chgrp` শুধু গ্রুপ বদলায়, মালিক বদলায় না।

umask — নতুন ফাইলের ডিফল্ট পারমিশন

সহজ কথায়: নতুন ফাইল বা ডিরেক্টরি তৈরি হওয়ার সময় সেটার পারমিশন কেমন হবে, সেটার নিয়ম।

Syntax

```
umask
umask <সংখ্যা>
```

উদাহরণ

```
umask
umask 022
```

সম্ভাব্য Output

```
0022
```

সংখ্যাটি হলো কাটার নিয়ম — অর্থাৎ কোন অনুমতিগুলো কেটে ফেলা হবে। `022` মানে group ও others-এর `w` কেটে ফেলা হয়, তাই নতুন ফাইল প্রায়ই `644` হয়।

কখন দরকার হবে: নতুন ফাইল নিজের অজান্তে বেশি খোলামেলা হচ্ছে কিনা দেখার জন্য।

মনে রাখো: বেশিরভাগ সাধারণ ইউজারের জন্য `022` ঠিক আছে।

groups — আমি কোন কোন গ্রুপে আছি

সহজ কথায়: আপনার অ্যাকাউন্ট কোন কোন গ্রুপের সদস্য, দেখায়।

Syntax

```
groups
groups <username>
```

উদাহরণ

```
groups
groups <username>
```

সম্ভাব্য Output

```
<username> : <username> sudo docker
```

কখন দরকার হবে: গ্রুপ পারমিশন এরর পেলে, Docker ব্যবহারের পারমিশন যাচাই করতে।

↳ groups — চলমান

মনে রাখো: গ্রুপে যোগ হয়নি বলে পারমিশন এরর হলে অ্যাডমিনকে জানান।

id — ইউজার ও গ্রুপ ID দেখা

সহজ কথায়: আপনার ইউজার ID (uid), গ্রুপ ID (gid) ও গ্রুপগুলো দেখা।

Syntax

```
id
id <username>
```

উদাহরণ

```
id
id <username>
```

সম্ভাব্য Output

```
uid=1000(<username>) gid=1000(<username>) groups=1000(<username>),27(sudo),999(docker)
```

কখন দরকার হবে: পারমিশন সমস্যা ডিবাগ করতে, স্ক্রিপ্টে ইউজার যাচাই করতে।

মনে রাখো: `groups`-এর চেয়ে বেশি বিস্তারিত তথ্য দেয়।

whoami — আমি কে

সহজ কথায়: এখন টার্মিনালে কোন ইউজার হিসেবে আছেন, জানায়।

Syntax

```
whoami
```

উদাহরণ

```
whoami
```

সম্ভাব্য Output

```
<username>
```

কখন দরকার হবে: ভুল অ্যাকাউন্টে কাজ করছেন কিনা দ্রুত বুঝতে, `sudo whoami`-এর সঙ্গে তুলনা করতে।

মনে রাখো: স্ক্রিপ্টে অ্যাকাউন্ট যাচাইয়ের সবচেয়ে সহজ উপায়।

sudo — অ্যাডমিন অধিকার দিয়ে কমান্ড চালানো

সহজ কথায়: সাময়িকভাবে সুপারইউজারের ক্ষমতা দিয়ে কাজ করানো।

Syntax

```
sudo <কমান্ড>
```

উদাহরণ

```
sudo chown <username>:<username> file.txt
sudo apt update
```

সম্ভাব্য Output

পাসওয়ার্ড চাইবে। পাসওয়ার্ড লেখা দেখায় না। ভুল হলে `try again` আসবে। কমান্ড সফল হলে প্রায়ই আউটপুট আসে না।

কখন দরকার হবে: প্যাকেজ ইনস্টল, সিস্টেম ফোল্ডারে লেখা, মালিক বদলানো, সার্ভিস পরিচালনা।

সাবধান: `sudo` মানে পুরো ক্ষমতা — ভুল কমান্ডে সিস্টেম ভাঙতে পারে। সবসময় আগে ভেবে নিন, ফাইনাল কমান্ডটা ঠিক আছে কিনা। অপ্রয়োজনীয় জায়গায় `sudo` ব্যবহার করবেন না।

মনে রাখো: `sudo` টার্মিনাল ছাড়া কাজ শেষে ক্ষমতা নিজে থেকেই ফিরে যায়।

দ্রুত পুনরালোচনা

- r = 4 (পড়া), w = 2 (লেখা), x = 1 (চালানো) — owner / group / others ক্রমে।
- `644` সাধারণ ফাইল, `755` স্ক্রিপ্ট/ডিরেক্টরির জন্য ভালো ডিফল্ট।
- `777` সাধারণত এড়িয়ে চলুন — least privilege মেনে চলুন।
- `ls -l` দিয়ে পারমিশন দেখুন, `chown` / `chgrp` দিয়ে মালিক/গ্রুপ বদলান।
- `umask` নতুন ফাইলের ডিফল্ট নিয়ম ঠিক করে।

অনুশীলন

একটি ডামি ফাইল আর স্ক্রিপ্ট বানান, `ls -l` দিয়ে দেখুন, `chmod 644 file.txt` ও `chmod +x script.sh` দিয়ে পারমিশন বদলে আবার `ls -l` দিয়ে যাচাই করুন। `whoami`, `id`, `groups`, `umask` চালিয়ে নিজের অ্যাকাউন্টের তথ্য লিখে রাখুন।

০৭

অধ্যায় ৭

প্রসেস ও মনিটরিং — কম্পিউটারের নাড়ি টিপুন

কম্পিউটার একসঙ্গে অনেক কাজ চালায়। প্রতিটি কাজই একটি **প্রসেস** — যার একটি ID (PID) থাকে, CPU বা মেমরি ব্যবহার করে। আমরা প্রসেস দেখতে, আটকাতে, বন্ধ করতে পারি। আর সিস্টেমের সামগ্রিক অবস্থা — CPU, RAM, ডিস্ক — মনিটর করতে পারি।

দুইটি সংকেত (signal) মনে রাখা জরুরি: **SIGTERM** মানে ভদ্রভাবে বন্ধ হওয়ার অনুরোধ (নম্বর `15`), **SIGKILL** মানে জোর করে মেরে ফেলা (নম্বর `9`)। সবসময় আগে SIGTERM চেষ্টা করুন, কাজ না হলে তবেই SIGKILL। সেটা প্রসেস নিজের কাজ গুছিয়ে বন্ধ হতে পারে।

কিছু মূল কথা:

- **প্রসেস** = চলমান প্রোগ্রাম, প্রতিটির PID আছে।
- **fg/bg** = একই টার্মিনালে প্রসেস চালানোতে রাখা (foreground) বা পেছনে পাঠানো (background)। `Ctrl+Z` দিয়ে থামিয়ে `bg` বা `fg` দিয়ে আনা যায়।
- **df বনাম du** = `df` ডিস্কে মোট কত জায়গা ফাঁকা আছে দেখায়, `du` কোন ডিরেক্টরি বা ফাইল কত জায়গা নিচ্ছে দেখায়।

ps — চলমান প্রসেসের তালিকা

সহজ কথায়: এখন কোন প্রসেস চলছে, দেখায়।

Syntax

```
ps
ps [অপশন]
```

উদাহরণ

```
ps
ps aux
```

সম্ভাব্য Output

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
<username>  1234  0.0  0.1  12345  6789 pts/0    Ss   10:00    0:00 bash
```

আউটপুট মেশিন ও সময় অনুযায়ী বদলাবে।

কখন দরকার হবে: কোনো প্রোগ্রাম চলছে কিনা, PID কী, দেখার জন্য।

↳ ps — চলমান

মনে রাখো: `ps aux` সব প্রসেসের বেশি তথ্য দেয়।

ps aux — সব প্রসেসের পূর্ণ তালিকা

সহজ কথায়: সিস্টেমের সব ইউজারের সব প্রসেস একসঙ্গে দেখা।

Syntax

```
ps aux
```

উদাহরণ

```
ps aux
ps aux | grep node
```

সম্ভাব্য Output

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          567  0.0  0.0   4567   123 ?        Ss   09:00    0:01 sshd
<username>   1234  0.5  0.2  23456  8901 pts/0    Sl   10:00    0:02 node server.js
```

আউটপুট সিস্টেম অনুযায়ী বদলাবে।

কখন দরকার হবে: PID খুঁজতে, কোন প্রসেস CPU বেশি খাচ্ছে দেখতে, অন্য ইউজারের প্রসেস দেখতে।

মনে রাখো: `grep` দিয়ে নির্দিষ্ট প্রসেস খুঁজুন, যেমন `ps aux | grep node`।

top — লাইভ সিস্টেম মনিটর

সহজ কথায়: CPU ও RAM কে বেশি ব্যবহার করছে — লাইভ দেখা।

Syntax

```
top
```

উদাহরণ

```
top
```

↳ top — চলমান

সম্ভাব্য Output

```
top - 10:12:01 up 2 days, 1 user, load average: 0.05, 0.03, 0.01
Tasks: 120 total, 1 running
%A: 100.0 * 1 Cpu(s):  2.0 us,  1.0 sy, 97.0 id
MiB Mem : 7986.5 total, 4000.0 free, 2500.0 used
```

লাইভ স্ক্রিন — আউটপুট প্রতি মুহূর্তে বদলাবে। `q` চাপলে বের হবে।

কখন দরকার হবে: CPU/RAM সমস্যা ডিবাগ করতে, কোন প্রসেস বেশি খাচ্ছে বুঝতে।

মনে রাখো: ডিস্ট্রো অনুযায়ী রঙ ও বিন্যাস আলাদা হতে পারে।

htop — সুন্দর ইন্টারঅ্যাক্টিভ মনিটর

সহজ কথায়: `top`-এর চেয়ে সুন্দর ও সহজে নেভিগেট করার মতো মনিটর।

Syntax

```
htop
```

উদাহরণ

```
sudo apt install htop
htop
```

সম্ভাব্য Output

রঙিন CPU বার ও প্রসেসের তালিকা সহ ইন্টারঅ্যাক্টিভ স্ক্রিন। আউটপুট লাইভ বদলাবে। q চাপলে বের হবেন।

কখন দরকার হবে: প্রসেস খুঁজে স্ক্রল করা, কাউন্ট/সর্ট করে CPU বেশি খাওয়া প্রসেস দেখা।

মনে রাখো: Debian/Ubuntu-তে ইনস্টল: `sudo apt install htop`। Fedora/RHEL-এ `sudo dnf install htop`। ইনস্টল ছাড়া `top` দিয়ে কাজ চালিয়ে নিন।

pgrep — নাম দিয়ে PID খোঁজা

সহজ কথায়: প্রসেসের নাম দিলে তার PID বের করে দেয়।

Syntax

```
pgrep <প্রসেস-নাম>
pgrep -a <প্রসেস-নাম>
```

↳ pgrep — চলমান

উদাহরণ

```
pgrep node
pgrep -a python3
```

সম্ভাব্য Output

```
1234
```

`-a` দিলে PID-এর সঙ্গে পুরো কমান্ড লাইনও দেখায়।

কখন দরকার হবে: `kill` চালানোর আগে PID নিশ্চিত করতে।

মনে রাখো: নামের অংশমাত্র দিলেও কাজ করে, কিন্তু একাধিক মিল এলে সব PID-ই আসবে।

pkill — নাম দিয়ে প্রসেস বন্ধ করা

সহজ কথায়: নাম দিয়ে প্রসেস থামানো।

Syntax

```
pkill <প্রসেস-নাম>
pkill -f <পূর্ণ-কম্যান্ড>
```

উদাহরণ

```
pkill node
pkill -f server.js
```

সম্ভাব্য Output

আউটপুট আসে না। কাজ না হলে কোনো `PID` মিলল না এমন ওয়ার্নিং আসতে পারে।

কখন দরকার হবে: ডেভ সার্ভার বন্ধ করতে, PID মনে না থাকলে।

সাবধান: `pkill node` ভুল জায়গে চালালে প্রোডাকশনের প্রসেসও বন্ধ হয়ে যেতে পারে। নামটি নিশ্চিত হয়ে নিন।

`kill` — PID দিয়ে প্রসেস বন্ধ করা

সহজ কথায়: নির্দিষ্ট প্রসেসকে বন্ধ করার অনুরোধ পাঠানো।

Syntax

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

উদাহরণ

```
kill 1234
kill -TERM 1234
kill -9 1234
```

সম্ভাব্য Output

আউটপুট আসে না। প্রসেস বন্ধ না হলে "No such process" বা পারমিশন এরর আসবে।

সাধারণত শুধু `kill <PID>` মানে SIGTERM পাঠায়, সেটাই সবার আগে চেষ্টা করুন।

কখন দরকার হবে: আটকে যাওয়া নিজের প্রসেস বন্ধ করতে, ডেভ সার্ভার থামাতে।

সাবধান: `-9` (SIGKILL) জোর করে মেরে ফেলে — প্রসেস গোছানোর সুযোগ পায় না, ডেটা লস হতে পারে। শেষ উপায় হিসেবে ব্যবহার করুন। অন্য ইউজারের প্রসেসে `sudo` লাগবে।

মনে রাখো: নাম নয়, সবসময় PID দিন — এটাই নিরাপদ।

`killall` — নাম দিয়ে সব প্রসেস বন্ধ করা

সহজ কথায়: একই নামের সব প্রসেস একসঙ্গে বন্ধ করা।

Syntax

```
killall <প্রসেস-নাম>
```

উদাহরণ

```
killall node
killall python3
```

সম্ভাব্য Output

আউটপুট আসে না।

↳ killall — চলমান

কখন দরকার হবে: একাধিক একই প্রসেস একসঙ্গে বন্ধ করতে।

সাবধান: খুবই সাবধানে ব্যবহার করুন — `killall` মানে সব ম্যাচ করা প্রসেসই শেষ। যেমন ভুলে `killall bash` দিলে নিজের টার্মিনালও বন্ধ হয়ে যেতে পারে। সাধারণত `kill <PID>`-ই ভালো।

free -h — মেমরি (RAM) ব্যবহার দেখা

সহজ কথায়: মেমরিতে মোট কত আছে, কত ব্যবহার হয়েছে, কত ফাঁকা — সহজ সংখ্যায়।

Syntax

```
free -h
```

উদাহরণ

```
free -h
```

সম্ভাব্য Output

```
               total        used        free      shared  buff/cache   available
Mem:           7.8Gi       2.5Gi       3.0Gi       200Mi       2.3Gi       5.0Gi
Swap:          2.0Gi       100Mi       1.9Gi
```

কখন দরকার হবে: সিস্টেম স্লো হলে RAM-ই কারণ কিনা দেখতে।

মনে রাখো: `available` কলামটি সবচেয়ে গুরুত্বপূর্ণ — এটা নতুন প্রসেসের জন্য আসল ফাঁকা RAM নির্দেশ করে।

df -h — ডিস্কে ফাঁকা জায়গা

সহজ কথায়: প্রতিটি ডিস্ক/পার্টিশনে মোট কত জায়গা, কত ফাঁকা।

Syntax

```
df -h
df -h <পথ>
```

উদাহরণ

```
df -h
df -h ~
```

↳ `df -h` — চলমান

সম্ভাব্য Output

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       50G   30G   18G  63% /
```

Filesystem free space বলতে এখানকার Avail বোঝায় — মানে ওই ডিস্কে আসলে কত ফাঁকা বাকি।

কখন দরকার হবে: ডিস্ক ফুল হলে, কোন ড্রাইভে জায়গা শেষ হচ্ছে বুঝতে।

মনে রাখো: `df` পুরো ডিস্কের হিসাব দেয়, কোন ডিরেক্টরি জায়গা নিচ্ছে সেটা দেখায় না।

`du -sh` — একটি ডিরেক্টরি/ফাইলের সাইজ

সহজ কথায়: একটি নির্দিষ্ট ডিরেক্টরি বা ফাইল ডিস্কে কত জায়গা নিচ্ছে।

Syntax

```
du -sh <পথ>
```

উদাহরণ

```
du -sh ~
du -sh ~/project
du -sh ~/project/logs
```

সম্ভাব্য Output

```
2.5G    /home/<username>/project
```

কখন দরকার হবে: কোন ফোল্ডার অসামান্য বড় হয়েছে খুঁজতে, জায়গা ফাঁকা করতে।

মনে রাখো: `du` ডিরেক্টরি/ফাইল ব্যবহার দেখায়, মোট ফাঁকা জায়গা দেখায় না (সেটা `df`)।

`du -h` — ডিরেক্টরি ভেতরে ফাইলের সাইজ

সহজ কথায়: একটি ডিরেক্টরির ভেতরে প্রতিটি ফাইল ও সাবডিরেক্টরির সাইজ।

Syntax

```
du -h <পথ>
```

↳ du -h — চলমান

উদাহরণ

```
du -h ~/project
du -h ~/project | sort -h | tail -20
```

সম্ভাব্য Output

```
1.2M    /home/<username>/project/data/file.bin
500K    /home/<username>/project/src
2.5G    /home/<username>/project
```

কখন দরকার হবে: গোপনে ডিস্ক ভরে ফেলা ফাইল বের করতে।

মনে রাখো: আউটপুট দীর্ঘ হতে পারে, `sort`-এর সঙ্গে ব্যবহার করলে বড় ফাইল সহজে দেখা যায়।

uptime — সিস্টেম কতক্ষণ চালু আছে

সহজ কথায়: সিস্টেম শেষ কবে চালু হয়েছে, কতক্ষণ চলছে, লোড কত — এক লাইনে।

Syntax

```
uptime
```

উদাহরণ

```
uptime
```

সম্ভাব্য Output

```
10:12:45 up 15 days, 3:22, 2 users, load average: 0.05, 0.03, 0.01
```

কখন দরকার হবে: সার্ভার কতদিন আপটাইম আছে দেখতে, লোড স্বাভাবিক কিনা বুঝতে।

মনে রাখো: load average 1-এর কাছাকাছি মানে স্থির, বেশি মানে CPU চাপ।

lscpu — CPU-এর তথ্য

সহজ কথায়: কী CPU, কয়টি কোর, কত গতির — হার্ডওয়্যারের পরিচয়।

Syntax

```
lscpu
```

↳ lscpu — চলমান

উদাহরণ

```
lscpu
```

সম্ভাব্য Output

```
CPU(s):              4
Model name:          Example CPU Vendor CPU
Thread(s) per core:  2
```

ঠিক আউটপুট মেশিন অনুযায়ী বদলাবে।

কখন দরকার হবে: সার্ভার কেনার আগে দরকারি CPU যাচাই করতে, সফটওয়্যার সামঞ্জস্য দেখতে।

মনে রাখো: ভার্চুয়াল মেশিনে ভার্চুয়াল CPU-র তথ্যই দেখাবে।

lsblk — ব্লক ডিভাইস (ডিস্ক/পার্টিশন) দেখা

সহজ কথায়: কোন ডিস্ক, কোন পার্টিশন, কীভাবে মাউন্টেড — দেখায়।

Syntax

```
lsblk
```

উদাহরণ

```
lsblk
lsblk -f
```

সম্ভাব্য Output

```
NAME    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda       8:0    0   50G  0 disk
├─sda1    8:1    0   50G  0 part /
```

কখন দরকার হবে: কোন ডিপার্টিশন কোথায় মাউন্টেড, নতুন ডিস্ক এসেছে কিনা বুঝতে।

মনে রাখো: `df -h`-এর সঙ্গে মিলিয়ে পড়লে হার্ডওয়্যার ও ফাঁকা জায়গা দুটোই বোঝা যায়।

lsusb — USB ডিভাইস দেখা

সহজ কথায়: কোন USB ডিভাইস কম্পিউটারে লাগানো আছে।

Syntax

```
lsusb
```

উদাহরণ

```
lsusb
```

সম্ভাব্য Output

```
Bus 001 Device 002: ID 8087:0024 Intel Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

কখন দরকার হবে: USB কী/হার্ড ডিস্ক চেনা যাচ্ছে না, এমন সমস্যা ডিবাগ করতে।

মনে রাখো: সব ডিস্ট্রোতে `usbutils` প্যাকেজ লাগতে পারে — Debian/Ubuntu-তে `sudo apt install usbutils` ।

dmesg — কার্নেল মেসেজ

সহজ কথায়: কার্নেলের শেষ ঘটনাগুলো — ড্রাইভার, হার্ডওয়্যার, মাউন্ট এররের মেসেজ।

Syntax

```
dmesg
dmesg | tail -50
```

উদাহরণ

```
dmesg | tail -50
dmesg | grep -i usb
```

সম্ভাব্য Output

```
[  123.456] usb 1-1: new high-speed USB device number 5 using xhci_hcd
[  123.789] EXT4-fs (sda1): mounted filesystem
```

কখন দরকার হবে: USB লাগিয়ে কী হলো দেখতে, মাউন্ট এরর, ড্রাইভার সমস্যা।

সাবধান: আধুনিক সিস্টেমে সাধারণ ইউজারের সব মেসেজ দেখার অনুমতি নাও থাকতে পারে। পারমিশন এরর হলে সব মেসেজ বন্ধ — `sudo dmesg` ব্যবহারের আগে নিয়ম জেনে নিন।

↳ dmesg — চলমান

মনে রাখো: সিস্টেমে `dmesg` বেশি দ্রুত ঘূর্ণন বলে পুরোনো মেসেজ হারিয়ে যেতে পারে।

journalctl — সিস্টেম লগ

সহজ কথায়: সিস্টেম লগ পড়া — বুট থেকে এখন পর্যন্ত ঘটনার ইতিহাস।

Syntax

```
journalctl
journalctl -u <সার্ভিস>
journalctl -f
```

উদাহরণ

```
journalctl -u nginx --since "1 hour ago"
journalctl -f
```

সম্ভাব্য Output

```
Feb 10 10:00:00 server systemd[1]: Started nginx.
Feb 10 10:00:05 server nginx[1234]: emerg ...
```

আউটপুট সময় ও সার্ভিস অনুযায়ী বদলাবে।

কখন দরকার হবে: সার্ভিসের এরর বুঝতে, বুট সমস্যা দেখতে, সাম্প্রতিক লগ ফলো করতে।

মনে রাখো: বিস্তারিত আলোচনা অধ্যায় ১১-এ।

systemctl — সার্ভিস ও বুট নিয়ন্ত্রণ

সহজ কথায়: সিস্টেম সার্ভিস চালু, বন্ধ, সক্ষম (enable) করা — আরো বিস্তারিত অধ্যায় ১১-এ।

Syntax

```
systemctl status <সার্ভিস>
systemctl list-units --type=service
```

উদাহরণ

```
systemctl status ssh
systemctl list-units --type=service
```

↳ `systemctl` — চলমান

সম্ভাব্য Output

```
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled)
     Active: active (running)
```

কখন দরকার হবে: সার্ভিসের অবস্থা দেখতে, চালু সার্ভিস তালিকা করতে।

মনে রাখো: বিস্তারিত কমান্ডগুলো অধ্যায় ১১-এ আলাদা করে আছে।

দ্রুত পুনরালোচনা

- `ps aux` প্রসেস তালিকা দেখায়, `top` / `htop` লাইভ মনিটর করে।
- PID দিয়ে `kill <PID>` — আগে SIGTERM, নাহলে SIGKILL।
- `df -h` মানে ডিস্কে মোট কত ফাঁকা, `du -h` মানে কোন ফোল্ডার কত জায়গা নিচ্ছে।
- `free -h` RAM, `uptime` + `lscpu` + `lsblk` সিস্টেমের অবস্থা, `dmesg` / `journalctl` লগ দেখায়।

অনুশীলন

ব্যাকগ্রাউন্ডে একটি দীর্ঘ প্রসেস চালান (`sleep 1000` বা আপনার Dev সার্ভার), `ps aux`, `pgrep -a` দিয়ে PID খুঁজুন, `top`-এ দেখুন, তারপর `kill <PID>` দিয়ে বন্ধ করুন। `df -h`, `du -sh ~`, `free -h`, `uptime`, `lsblk` চালিয়ে নোট করুন।

অধ্যায় ৮

০৮ নেটওয়ার্ক ও রিমোট — সংযোগের জাদু

নেটওয়ার্ক ছাড়া সার্ভার মানে বিচ্ছিন্ন দ্বীপ। প্রতিটি ডিভাইসের একটি **IP ঠিকানা** থাকে — যেমন বাড়ির ঠিকানার মতো। **DNS** সেই ঠিকানাকে নামে বদলায় (যেমন `example.com` → সংখ্যায়)। প্রতিটি সার্ভিস নির্দিষ্ট **পোর্টে** শোনে — HTTP সাধারণত পোর্ট `80`, HTTPS পোর্ট `443`, SSH পোর্ট `22`।

HTTP বেসিক: ব্রাউজার `GET` দিয়ে পেজ চায়, সার্ভার `200 OK` বা `404 Not Found` ফেরত দেয়। `curl` / `wget` টার্মিনাল থেকে একই কাজ করে।

SSH হোস্ট যাচাই: প্রথমবার রিমোট সার্ভারে সংযোগ দিলে সার্ভারের ফিঙ্গারপ্রিন্ট দেখাবে। পরিচিত হলে `yes` দিন, না পরিচিত হলে থামুন — ভুল সার্ভারে সংযোগ ঝুঁকিপূর্ণ।

পাবলিক বনাম প্রাইভেট কী: SSH কী-র একটি জোড়া থাকে — **প্রাইভেট কী** গোপন, **পাবলিক কী** শেয়ার করা নিরাপদ। পাবলিক কী সার্ভারে রাখলে প্রাইভেট কী ছাড়া পাসওয়ার্ড ছাড়াই লগইন হয়।

এই অধ্যায়ে `<server_ip>` মানে আপনার সার্ভারের রিয়েল IP, `<username>` মানে আপনার অ্যাকাউন্ট। আসল তথ্য কখনো বইয়ে বা কারো সঙ্গে শেয়ার করবেন না।

ip a — নেটওয়ার্ক ইন্টারফেস ও IP দেখা

সহজ কথায়: মেশিনে কোন নেটওয়ার্ক কার্ড আছে, কী IP পেয়েছে।

Syntax

```
ip a
ip addr show
```

উদাহরণ

```
ip a
```

সম্ভাব্য Output

```
1: lo: <LOOPBACK,UP> mtu 65536
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,UP> mtu 1500
    inet <your_local_ip>/24 brd <broadcast> scope global eth0
```

↳ ip a — চলমান

কখন দরকার হবে: IP ঠিক আছে কিনা, লোকাল নেটওয়ার্কের ঠিকানা দেখতে, DHCP এসেছে কিনা যাচাই করতে।

মনে রাখো: `lo` (127.0.0.1) হলো লোকাল লুপব্যাক — নিজের মেশিন নিজে।

ip route — রাউটিং টেবিল দেখা

সহজ কথায়: নেটওয়ার্ক ট্রাফিক কোথা দিয়ে যাবে, সেই টেবিল।

Syntax

```
ip route
ip route show
```

উদাহরণ

```
ip route
```

সম্ভাব্য Output

```
default via <gateway_ip> dev eth0 proto dhcp metric 100
<your_subnet>/24 dev eth0 proto kernel scope link
```

কখন দরকার হবে: ইন্টারনেট যাচ্ছে না — ডিফল্ট গেটওয়ে আছে কিনা দেখতে।

মনে রাখো: `default via` লাইনটি ইন্টারনেটের দরজা নির্দেশ করে।

ping — সংযোগ আছে কিনা পরীক্ষা

সহজ কথায়: নির্দিষ্ট ঠিকানায় ছোট প্যাকেট পাঠিয়ে উত্তর আসছে কিনা দেখা।

Syntax

```
ping <server_ip>
ping -c 4 <server_ip>
```

উদাহরণ

```
ping -c 4 <server_ip>
ping -c 4 8.8.8.8
```

↳ ping — চলমান

সম্ভাব্য Output

```
PING <server_ip> (<server_ip>) 56(84) bytes of data.
64 bytes from <server_ip>: icmp_seq=1 ttl=64 time=0.12 ms
4 packets transmitted, 4 received, 0% packet loss
```

কখন দরকার হবে: DNS কাজ করছে কিনা, সার্ভার জেগে আছে কিনা, লেটেন্সি দেখতে।

মনে রাখো: আধুনিক ফায়ারওয়াল ping বন্ধ রাখতে পারে — ping না-আসা মানেই সার্ভার ডাউন নয়।

curl — HTTP টেস্ট ও ডাউনলোড

সহজ কথায়: ওয়েব ঠিকানা থেকে ডেটা আনা, API টেস্ট করা।

Syntax

```
curl <URL>
curl -I <URL>
curl -o <ফাইল> <URL>
```

উদাহরণ

```
curl -I https://example.com
curl -o file.txt <URL>
```

সম্ভাব্য Output

```
HTTP/2 200
content-type: text/html
...
```

`example.com` হলো পরীক্ষার জন্য সুপরিচিত, এখানে ব্যবহার সুরক্ষিত।

কখন দরকার হবে: সার্ভার HTTP দিচ্ছে কিনা দেখতে, API রেসপন্স যাচাই করতে, ফাইল দ্রুত ডাউনলোড করতে।

মনে রাখো: `-I` হেডার মাত্র দেখায়, `-o` ফাইলে সেভ করে, `-L` রিডাইরেক্ট অনুসরণ করে।

wget — ফাইল ডাউনলোড

সহজ কথায়: URL থেকে ফাইল ডাউনলোড করা।

Syntax

```
wget <URL>
wget -O <ফাইল> <URL>
```

উদাহরণ

```
wget https://example.com/file.txt
wget -O data.json <URL>
```

সম্ভাব্য Output

```
--2026-02-10 10:15:00--  https://example.com/file.txt
Length: 1234 [text/plain]
file.txt  100%[==========>] 1.23K  --.-KB/s
```

কখন দরকার হবে: লিনাক্স টার্মিনাল থেকে টুল, আর্কাইভ, ডেটা সেট নামাতে।

মনে রাখো: `curl`-এর মতোই কাজ করে, মূলত ফাইল সংরক্ষণে সুবিধাজনক।

ss -tuln — কোন পোর্ট শুনছে

সহজ কথায়: TCP ও UDP-তে কোন পোর্ট খোলা আছে।

Syntax

```
ss -tuln
```

উদাহরণ

```
ss -tuln
```

সম্ভাব্য Output

```
Netid State  Recv-Q Send-Q Local Address:Port Peer Address:Port
tcp   LISTEN 0      4096   0.0.0.0:22    0.0.0.0:*
tcp   LISTEN 0      4096   0.0.0.0:80    0.0.0.0:*
```

কখন দরকার হবে: সার্ভিস কোন পোর্টে শুনছে যাচাই করতে, ফায়ারওয়াল সমস্যা দেখতে।

মনে রাখো: `t`=TCP, `u`=UDP, `l`=listening, `n`=সংখ্যায় (নাম নয়)।

ss -tulnp — পোর্টের সঙ্গে প্রসেসও দেখা

সহজ কথায়: কোন পোর্ট কোন প্রসেস খুলেছে, সেটা দেখা।

Syntax

```
sudo ss -tulnp
```

উদাহরণ

```
sudo ss -tulnp
```

সম্ভাব্য Output

```
tcp   LISTEN 0 4096 0.0.0.0:22 0.0.0.0:* users:(("sshd",pid=1234,fd=3))
```

কখন দরকার হবে: কোন প্রসেস পোর্ট দখল করেছে খুঁজতে।

সাবধান: প্রসেসের নাম দেখতে প্রায়ই `sudo` লাগে — কেননা অন্য ইউজারের প্রসেস তথ্য সুরক্ষিত থাকে। `sudo` ছাড়া প্রসেসের জায়গা খালি দেখাবে।

মনে রাখো: `-p` মানেই process, `sudo` ছাড়া পুরো তথ্য মিলবে না।

ssh — রিমোট সার্ভারে নিরাপদ লগইন

সহজ কথায়: টার্মিনাল থেকে অন্য মেশিনে নিরাপদে ঢোকা।

Syntax

```
ssh <username>@<server_ip>
```

উদাহরণ

```
ssh <username>@<server_ip>
ssh <username>@<server_ip> -p 2222
```

সম্ভাব্য Output

```
The authenticity of host '<server_ip>' can't be established.
ECDSA key fingerprint is SHA256:xxxxx.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

কখন দরকার হবে: রিমোট সার্ভারে ফাইল পরিচালনা, লগ দেখা, সফটওয়্যার চালানো।

সাবধান: পরিচিত সার্ভারের ফিঙ্গারপ্রিন্ট নিশ্চিত না হলে কখনো `yes` দেবেন না।

↳ ssh — চলমান

মনে রাখো: `<username>` ও `<server_ip>`-এর জায়গায় আপনার তথ্য বসাবেন।

ssh-keygen — কী জোড়া তৈরি করা

সহজ কথায়: প্রাইভেট ও পাবলিক কী তৈরি — পাসওয়ার্ডহীন লগইনের ভিত্তি।

Syntax

```
ssh-keygen -t ed25519
```

উদাহরণ

```
ssh-keygen -t ed25519
ssh-keygen -t ed25519 -C "my-work-laptop"
```

সম্ভাব্য Output

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (~/.ssh/id_ed25519):
Your identification has been saved in ~/.ssh/id_ed25519
Your public key has been saved in ~/.ssh/id_ed25519.pub
```

কখন দরকার হবে: প্রথমবার কী সেটআপ করতে, নিরাপদ লগইনের প্রস্তুতি নিতে।

সাবধান: পাসফ্রেজ দিলে প্রাইভেট কী আরো সুরক্ষিত থাকে। প্রাইভেট কীটি `chmod 600` রাখুন।

মনে রাখো: ডিফল্ট কী টাইপ `ed25519` আধুনিক ও নিরাপদ।

ssh-copy-id — পাবলিক কী সার্ভারে কপি

সহজ কথায়: পাবলিক কী রিমোট সার্ভারে পাঠিয়ে Password-এর ঝামেলা শেষ করা।

Syntax

```
ssh-copy-id &lt;username&gt;@&lt;server_ip&gt;
```

উদাহরণ

```
ssh-copy-id &lt;username&gt;@&lt;server_ip&gt;
```

↳ ssh-copy-id — চলমান

সম্ভাব্য Output

```
Number of key(s) added: 1
Now try logging into the machine, with: "ssh '<username>@<server_ip>'"
```

কখন দরকার হবে: একবার সেটআপ করলে বারবার পাসওয়ার্ড ছাড়া ঢুকতে।

মনে রাখো: আপনার পাবলিক কী (`id_ed25519.pub`) নিরাপদে শেয়ার করা যায়। প্রাইভেট কী (`id_ed25519`) কখনো শেয়ার করবেন না।

scp — সুরক্ষিতভাবে ফাইল কপি

সহজ কথায়: SSH দিয়ে লোকাল থেকে রিমোটে বা উল্টো ফাইল পাঠানো।

Syntax

```
scp <লোকাল-পথ> <username>@<server_ip>:<রিমোট-পথ>
scp <username>@<server_ip>:<রিমোট-পথ> <লোকাল-পথ>
```

উদাহরণ

```
scp file.txt <username>@<server_ip>:~/file.txt
scp <username>@<server_ip>:~/logs/app.log ~/app.log
```

সম্ভাব্য Output

```
file.txt  100%  1.2KB   1.0MB/s   00:00
```

কখন দরকার হবে: বড় ফাইল দ্রুত পাঠাতে, ব্যাকআপ আনার জন্য।

মনে রাখো: `<username>` ও `<server_ip>`-এর জায়গায় আপনার তথ্য বসাবেন।

sftp — ইন্টারঅ্যাক্টিভ ফাইল ট্রান্সফার

সহজ কথায়: FTP-র মতো ইন্টারঅ্যাক্টিভ ফাইল সেশন, কিন্তু SSH-এর নিরাপত্তাসহ।

Syntax

```
sftp <username>@<server_ip>
```

উদাহরণ

```
sftp <username>@<server_ip>
```

↳ sftp — চলমান

সম্ভাব্য Output

```
Connected to <server_ip>.
sftp> get ~/file.txt
sftp> put local.txt ~/remote.txt
sftp> bye
```

কখন দরকার হবে: অনেক ফাইল ঘুরে ঘুরে ব্রাউজ করে নেওয়া/দেওয়া।

মনে রাখো: ভেতরে গিয়ে `help` টাইপ করলে কমান্ড তালিকা দেখায়।

dig — DNS ট্রাবলশুট

সহজ কথায়: ডোমেন কোন IP-তে যায়, DNS উত্তরে কী আসে — বিস্তারিত।

Syntax

```
dig <ডোমেইন>
```

উদাহরণ

```
dig example.com
dig -x <server_ip>
```

সম্ভাব্য Output

```
; <<>> DiG 9.18 <<>> example.com
;; ANSWER SECTION:
example.com. 300 IN A <resolved_ip>
```

কখন দরকার হবে: DNS সমস্যা, ডোমেন ঠিক জায়গায় যাচ্ছে কিনা দেখতে।

মনে রাখো: প্যাকেজ: Debian/Ubuntu-তে `bind9-dnsutils` (বা `dnsutils`), Fedora/RHEL-তে `bind-utils`। না থাকলে ইন্সটল করুন।

nslookup — সহজ DNS খোঁজ

সহজ কথায়: ডোমেনের IP দ্রুত খোঁজা।

Syntax

```
nslookup <ডোমেইন>
```

↳ nslookup — চলমান

উদাহরণ

```
nslookup example.com
nslookup <ডোমেইন> 8.8.8.8
```

সম্ভাব্য Output

```
Server:  8.8.8.8
Address: 8.8.8.8#53
Name: example.com
Address: <resolved_ip>
```

কখন দরকার হবে: দ্রুত DNS জানার জন্য।

মনে রাখো: প্যাকেজ: Debian/Ubuntu-তে `bind9-dnsutils` বা `dnsutils`, Fedora/RHEL-তে `bind-utils`।

traceroute — প্যাকেটের পথ দেখা

সহজ কথায়: প্যাকেট আপনার মেশিন থেকে গন্তব্য পর্যন্ত কোন কোন রাউটার পেরিয়ে যায়।

Syntax

```
traceroute <server_ip>
```

উদাহরণ

```
traceroute <server_ip>
```

সম্ভাব্য Output

```
traceroute to <server_ip> (<server_ip>), 30 hops max
 1  <gateway_ip>  1.0 ms
 2  isp-router  5.0 ms
 3  <server_ip>  20.0 ms
```

কখন দরকার হবে: নেটওয়ার্ক কোথায় আটকাচ্ছে, লেটেন্সি বেশি কোথায় ডিবাগ করতে।

মনে রাখো: প্যাকেজ: Debian/Ubuntu-তে `traceroute`, Fedora/RHEL-তে `traceroute` (বা `inetutils`)।

`hostname -I` — নিজের সব IP এক লাইনে

সহজ কথায়: এই মেশিনে কী কী IP লাগানো আছে — দ্রুত দেখা।

Syntax

```
hostname -I
```

উদাহরণ

```
hostname -I
```

সম্ভাব্য Output

```
<your_local_ip> 172.17.0.1
```

কখন দরকার হবে: লোকাল নেটওয়ার্কে মেশিনের ঠিকানা জানতে, SSH-এর জন্য বেসিক চেক।

মনে রাখো: একাধিক IP থাকলে এক লাইনে সব দেখাবে।

দ্রুত পুনরালোচনা

- `ip a` IP, `ip route` গেটওয়ে, `ping` সংযোগ, `hostname -I` লোকাল IP দেখায়।
- `ss -tuln` পোর্ট, `ss -tulnp` প্রসেস — প্রসেস দেখতে প্রায়ই `sudo` লাগে।
- `curl` / `wget` HTTP ফাইল আনে, `dig` / `nslookup` DNS দেখে, `traceroute` পথ দেখায়।
- `ssh` লগইন, `ssh-keygen` + `ssh-copy-id` কী সেটআপ, `scp` / `sftp` ফাইল পাঠায়।
- **CRITICAL:** `~/.ssh/id_ed25519` (প্রাইভেট কী) কখনো শেয়ার করবেন না — প্রাইভেট কেউ পেলেই ঢুকে পড়বে। `id_ed25519.pub` (পাবলিক কী) শেয়ার করা নিরাপদ।

অনুশীলন

`hostname -I`, `ip a`, `ip route`, `ping -c 4 <server_ip>`, `ss -tuln`, `curl -I https://example.com` চালিয়ে নোট করুন। `ssh-keygen -t ed25519` দিয়ে কী বানিয়ে `.ssh` ডিরেক্টরিতে `ls -l` দিয়ে প্রাইভেট কী-র পারমিশন (`600`) যাচাই করুন।

অধ্যায় ৯

০৯

আর্কাইভ, কম্প্রেশন ও ব্যাকআপ — প্যাক করে রাখুন

ফাইল ছড়িয়ে থাকলে ব্যাকআপ বা শেয়ার করা ঝামেলা। আর্কাইভ ও কম্প্রেশন দুটি আলাদা জিনিস:

- **আর্কাইভ (Archive):** অনেক ফাইল/ডিরেক্টরিকে এক প্যাকেটে গুছিয়ে রাখা — যেমন `tar`, `zip`।
- **কম্প্রেশন (Compression):** প্যাকেটকে ছোট করা — যেমন `gzip`, `xz`, `7z`।

বহু টুল দুটোই করে — যেমন `tar -czf` মানে টার আর্কাইভ + গজিপ কম্প্রেশন।

মনে রাখো: কম্প্রেশন মানেই ব্যাকআপ কৌশল নয়! কম্প্রেস করা ফাইল হারিয়ে গেলে ব্যাকআপ নেই। ব্যাকআপ মানে আলাদা কপি, নিয়মিত অনুলিপি, ভিন্ন অবস্থান — আর সেটার জন্য আলাদা পরিকল্পনা লাগে।

`tar -czf` — আর্কাইভ + গজিপ কম্প্রেশন

সহজ কথায়: সব ফাইল এক প্যাকেটে গুছিয়ে ছোট করে রাখা।

Syntax

```
tar -czf <আর্কাইভ-নাম.tar.gz> <ফাইল-বা-ডিরেক্টরি>
```

উদাহরণ

```
tar -czf project-backup.tar.gz ~/project
tar -czf logs-$(date +%F).tar.gz ~/logs
```

সম্ভাব্য Output

আউটপুট আসে না। `ls -lh` দিয়ে `project-backup.tar.gz` হয়েছে কিনা দেখুন।

কখন দরকার হবে: পুরো প্রজেক্টের স্ন্যাপশট নিতে, সার্ভারে আপলোড করতে।

মনে রাখো: `-c` create, `-z` gzip, `-f` file নাম।

`tar -xzf` — গজিপ আর্কাইভ খোলা

সহজ কথায়: `tar.gz` প্যাকেট ভেতরে বের করা।

Syntax

```
tar -xzf <আর্কাইভ-নাম.tar.gz>
tar -xzf <আর্কাইভ-নাম.tar.gz> -C <লক্ষ্য-ডিরেক্টরি>
```

উদাহরণ

```
tar -xzf project-backup.tar.gz
mkdir -p ~/restore && tar -xzf project-backup.tar.gz -C ~/restore
```

সম্ভাব্য Output

আউটপুট আসে না। `ls` দিয়ে যাচাই করুন।

কখন দরকার হবে: ব্যাকআপ ফেরানো, ঐতিহাসিক ফাইল খুঁজে বের করা।

সাবধান: আর্কাইভের নাম ভেতরে পাথসহ থাকলে ফাইল একই জায়গায় ওভাররাইট হতে পারে। অফিসিয়াল ব্যাকআপের আগে `-C` দিয়ে নিরাপদ ফোল্ডারে খুলুন।

`tar -tf` — আর্কাইভ খোলা ছাড়াই তালিকা

সহজ কথায়: প্যাকেট খোলা ছাড়াই ভেতরে কী আছে দেখা।

Syntax

```
tar -tf <আর্কাইভ-নাম.tar.gz>
```

উদাহরণ

```
tar -tf project-backup.tar.gz
tar -tf project-backup.tar.gz | head -20
```

সম্ভাব্য Output

```
project/
project/README.md
project/src/main.js
project/src/utils.js
```

কখন দরকার হবে: আর্কাইভের ভেতরে কী আছে নিরাপদে যাচাই করতে।

↳ tar -tf — চলমান

মনে রাখো: আর্কাইভ খোলার আগে সবসময় `-t` দিয়ে তালিকা দেখুন।

zip — জিপ আর্কাইভ তৈরি

সহজ কথায়: সব ফাইল এক জিপ প্যাকেটে গুছিয়ে রাখা (উইন্ডোজেও খোলা যায়)।

Syntax

```
zip <আর্কাইভ.zip> <ফাইল-বা-ডিরেক্টরি>
```

উদাহরণ

```
zip -r project.zip ~/project
zip -r logs.zip ~/logs/*.log
```

সম্ভাব্য Output

```
  adding: home/<username>/project/README.md (deflated 20%)
```

কখন দরকার হবে: ক্লায়েন্টকে পাঠাতে, অন্য অপারেটিং সিস্টেমের সঙ্গে শেয়ার করতে।

মনে রাখো: ডিরেক্টরির জন্য `-r` (recursive) লাগে।

unzip — জিপ খোলা

সহজ কথায়: জিপ প্যাকেট ভেতরে বের করা।

Syntax

```
unzip <আর্কাইভ.zip> -d <লক্ষ্য-ডিরেক্টরি>
```

উদাহরণ

```
unzip project.zip
unzip project.zip -d ~/unpacked
```

সম্ভাব্য Output

```
Archive:  project.zip
  inflating: project/README.md
  inflating: project/src/main.js
```

কখন দরকার হবে: পাওয়া জিপ আর্কাইভ খুলতে।

↳ unzip — চলমান

⚠ সাবধান: জিপ স্লিপ অ্যাটাক ও ওভাররাইট থেকে বাঁচতে পরিচিত উৎস ছাড়া অন্ধকার ফাইল খুলবেন না।

gzip — এক ফাইল কম্প্রেস করা

সহজ কথায়: একক ফাইলকে ছোট করা (.gz তৈরি)।

Syntax

```
gzip <ফাইল>
```

উদাহরণ

```
gzip data.log
ls -lh data.log.gz
```

সম্ভাব্য Output

মেসেজ আসে না। `data.log` মুছে `.gz` হয়ে যাবে।

কখন দরকার হবে: লগ ফাইল সংরক্ষণে জায়গা বাঁচাতে।

⚠ সাবধান: `gzip` আসল ফাইল মুছে `.gz` বানায় — কপি রাখতে চাইলে `gzip -k` ব্যবহার করুন।

gunzip — গজিপ ডিকম্প্রেস করা

সহজ কথায়: `.gz` ফাইলকে আগের রূপে ফেরানো।

Syntax

```
gunzip <ফাইল.gz>
gunzip -k <ফাইল.gz>
```

উদাহরণ

```
gunzip data.log.gz
gunzip -k data.log.gz
```

সম্ভাব্য Output

আউটপুট আসে না। `ls` দিয়ে যাচাই করুন।

কখন দরকার হবে: সংরক্ষিত লগ আবার পড়তে চাইলে।

↳ gunzip — চলমান

মনে রাখো: `-k` রাখলে `.gz` কপি বজায় থাকে।

xz — শক্তিশালী কম্প্রেশন

সহজ কথায়: gzip-এর চেয়ে ভালো কম্প্রেশন অনুপাত (ধীর কিন্তু ছোট)।

Syntax

```
xz <ফাইল>
xz -d <ফাইল.xz>
```

উদাহরণ

```
xz data.log
xz -d data.log.xz
```

সম্ভাব্য Output

আউটপুট আসে না। `ls -lh data.log.xz` দিয়ে দেখুন।

কখন দরকার হবে: অনেক জায়গা বাঁচাতে হবে, বড় আর্কাইভ পাঠাতে হবে।

মনে রাখো: `xz`-এর অপশন অনেকটা gzip-এর মতো: `-k` কপি রাখে, `-d` ডিকম্প্রেস করে।

7z — বহুমুখী আর্কাইভার

সহজ কথায়: 7-Zip ফরম্যাটসহ অনেক ফরম্যাট খোলা ও তৈরি করা।

Syntax

```
7z a <আর্কাইভ.7z> <ফাইল-বা-ডিরেক্টরি>
7z x <আর্কাইভ.7z>
```

উদাহরণ

```
7z a project.7z ~/project
7z x project.7z -o~/unpacked
```

সম্ভাব্য Output

```
Creating archive: project.7z
Everything is Ok
```

↳ 7z — চলমান

কখন দরকার হবে: উইন্ডোজ-লিনাক্স ক্রস-প্ল্যাটফর্ম প্যাকেট, ভালো কম্প্রেশন দরকারে।

মনে রাখো: ইনস্টল: Debian/Ubuntu-তে `sudo apt install p7zip-full`, Fedora/RHEL-তে `sudo dnf install p7zip`।

ছোট প্রজেক্ট ব্যাকআপ — পুরো ওয়ার্কফ্লো

একটি ছোট প্রজেক্টের ব্যাকআপ নেওয়ার পুরো ধাপ:

```
# ১. ডিরেক্টরিতে ঢুকুন
cd ~
# ২. ব্যাকআপের আগে প্রজেক্টে থাকা বড় ফাইল চেক
du -sh ~/project
# ৩. tar আর্কাইভ বানান
tar -czf project-backup-$(date +%F).tar.gz ~/project
# ৪. তালিকা যাচাই
tar -tf project-backup-$(date +%F).tar.gz | head -20
# ৫. সাইজ যাচাই
ls -lh project-backup-$(date +%F).tar.gz
# ৬. চাইলে সার্ভারে পাঠান
scp project-backup-$(date +%F).tar.gz <username>@<server_ip>:~/backups/
# ৭. সার্ভারে এক্সট্রাক্ট পরীক্ষা (টেস্ট)
mkdir -p ~/restore-test && tar -xzf project-backup-$(date +%F).tar.gz -C ~/restore-test &&
ls ~/restore-test/project
```

কখন দরকার হবে: দিনশেষে, গুরুত্বপূর্ণ প্রজেক্ট ডিপ্লয়েসের আগে, অভ্যাস তৈরি করতে।

মনে রাখো: ব্যাকআপ টেস্ট করাটাই আসল ব্যাকআপ — অটেস্টেড ব্যাকআপ কোনো ব্যাকআপ নয়। কম্প্রেশন ≠ ব্যাকআপ কৌশল — ব্যাকআপ মানে প্রভাবদৃষ্টি: আলাদা মিডিয়া, নিয়মিত অটোমেশন, আর রিস্টোর টেস্ট।

দ্রুত পুনরালোচনা

- `tar -czf` আর্কাইভ + কম্প্রেশন, `-tf` তালিকা, `-xzf` খোলা।
- `zip` / `unzip` শেয়ারের জন্য ভালো, `gzip` / `xz` এক ফাইলের কম্প্রেশন, `7z` বহুমুখী।
- শুধু কম্প্রেস করলেই ব্যাকআপ প্রক্রিয়া শেষ নয় — আলাদা কপি ও রিস্টোর টেস্ট জরুরি।
- আর্কাইভ খোলার আগে `-t` দিয়ে তালিকা দেখা নিরাপদ অভ্যাস।

অনুশীলন

একটি ছোট ফোল্ডার বানিয়ে `tar -czf`, `tar -tf`, `tar -xzf` দিয়ে আর্কাইভ ও রিস্টোর করুন। একই ফাইল `zip`, `gzip -k`, `7z` (ইন্সটল থাকলে) দিয়ে প্যাক করে সাইজ তুলনা করুন।

১০

অধ্যায় ১০

শেল প্রোডাক্টিভিটি — পাওয়ার টুল

শেল মানে শুধু কমান্ড চালানো না — সেসব কমান্ডকে যুক্ত করে কাজ সহজ করা। পাইপ (`|`) দিয়ে এক কমান্ডের আউটপুট অন্য কমান্ডে পাঠানো যায়, রিডাইরেক্ট দিয়ে আউটপুট ফাইলে পাঠানো যায়, অ্যালিয়াস দিয়ে লম্বা কমান্ড ছোট করা যায়। এখানকার টুলগুলো দৈনন্দিন DevOps ওয়ার্কফ্লোতে হাতের কাছে থাকা জরুরি।

মনে রাখুন:

- **Env var + PATH:** পরিবেশের ভেরিয়েবল (`HOME` , `PATH`) নির্ধারণ করে কোন কমান্ড কোথা থেকে খুঁজবে।
- `>` একই ফাইলে ওভাররাইট করে — আগের কনটেন্ট উড়িয়ে দেয়।
- `>>` যোগ করে (append) — আগের কনটেন্ট থাকে।
- সিঙ্গেল কোটে ভেরিয়েবল বদলায় না, ডাবল কোটে বদলায় — সচেতন থাকুন।

echo — টেক্সট প্রিন্ট করা

সহজ কথায়: টার্মিনালে বা ফাইলে লেখা পাঠানো।

Syntax

```
echo <টেক্সট>
echo <টেক্সট> > <ফাইল>
echo <টেক্সট> >> <ফাইল>
```

উদাহরণ

```
echo "hello"
echo "hello" > note.txt
echo "another line" >> note.txt
cat note.txt
```

সম্ভাব্য Output

```
hello
$ cat note.txt
hello
another line
```

কখন দরকার হবে: ফাইল বানাতে, ভেরিয়েবল টেস্ট করতে, ডিবাগে মেসেজ দেখাতে।

↳ echo — চলমান

⚠ সাবধান: `>` দিলে note.txt আগের সব কনটেন্ট মুছে যায়। নিরাপদে যোগ করতে `>>` ব্যবহার করুন।

মনে রাখো: ভেরিয়েবলসহ `echo "Hello $USER"` ডাবল কোটে, সিঙ্গেল কোটে `echo 'Hello $USER'` আক্ষরিকভাবে দেখাবে।

printf — ফরম্যাটেড প্রিন্ট

সহজ কথায়: লেখার বিন্যাস নিয়ন্ত্রণ করে প্রিন্ট করা (ট্যাব, নিউলাইন, সারিবদ্ধ কলাম)।

Syntax

```
printf '<ফরম্যাট>\n' <মান>
printf '%s %d\n' <string> <সংখ্যা>
```

উদাহরণ

```
printf 'Disk: %s used\n' '80%'
printf '%-15s %s\n' 'File' 'Size'
```

সম্ভাব্য Output

```
Disk: 80% used
File            Size
```

কখন দরকার হবে: টেবিল বানাতে, স্ট্যাটাস লগে সুন্দর বিন্যাসে।

মনে রাখো: নিয়মিত লেখার জন্য `echo` সহজ, ফরম্যাট লাগলে `printf`।

env — সব env ভেরিয়েবল দেখা

সহজ কথায়: শেলের সব পরিবেশ ভেরিয়েবলের তালিকা।

Syntax

```
env
```

উদাহরণ

```
env | grep PATH
env | sort
```

↳ env — চলমান

সম্ভাব্য Output

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOME=/home/<username>
USER=<username>
```

কখন দরকার হবে: PATH কেন কমান্ড পাচ্ছে না ডিবাগ করতে, স্ক্রিপ্টে কী ভেরিয়েবল আছে বুঝতে।

মনে রাখো: অনেক ভেরিয়েবল — `grep` দিয়ে খুঁজুন।

printenv — নির্দিষ্ট env ভেরিয়েবল দেখা

সহজ কথায়: একটি ভেরিয়েবলের মান বের করা।

Syntax

```
printenv <ভেরিয়েবল-নাম>
```

উদাহরণ

```
printenv PATH
printenv HOME
printenv USER
```

সম্ভাব্য Output

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

কখন দরকার হবে: স্ক্রিপ্টে ভেরিয়েবল সঠিক আছে কিনা যাচাই করতে।

মনে রাখো: ভেরিয়েবল না থাকলে আউটপুট আসে না, এরর আসে না।

export — ভেরিয়েবল পরের প্রসেসে পাঠানো

সহজ কথায়: ভেরিয়েবল এমনভাবে সেট করা যাতে চালানো কমান্ড/স্ক্রিপ্টও সেটা দেখতে পায়।

Syntax

```
export <NAME>=<মান>
```

↳ export — চলমান

উদাহরণ

```
export PATH=$PATH:~/bin
export APP_ENV=dev
```

সম্ভাব্য Output

কোনো আউটপুট আসে না। `printenv APP_ENV` দিয়ে যাচাই করুন।

কখন দরকার হবে: APP_ENV, DATABASE_URL-এর মতো কনফিগ স্ক্রিপ্টে হস্তান্তর করতে।

সাবধান: সেশনের জন্য সাময়িক — টার্মিনাল বন্ধ হলে হারায়। স্থায়ী করতে `~/.bashrc`-এ যোগ করুন (যদি bash থাকে)।

মনে রাখো: শুধু `NAME=value` দিলে ভেরিয়েবল শুধু শেলে থাকে, `export` দিলে চাইল্ড প্রসেসও পায়।

alias — কমান্ডের শর্টকাট

সহজ কথায়: লম্বা কমান্ডের সহজ নাম দেওয়া, যাতে বারবার টাইপ না করতে হয়।

Syntax

```
alias <শর্টকাট>='<কমান্ড>'
alias
```

উদাহরণ

```
alias ll='ls -l'
alias gs='git status'
```

সম্ভাব্য Output

```
ll='ls -l'
gs='git status'
```

কখন দরকার হবে: প্রতিদিন ব্যবহৃত লম্বা কমান্ড ছোট করতে।

মনে রাখো: সেশনের জন্যই থাকে। স্থায়ী করতে `~/.bashrc`-তে alias লাইন যোগ করুন।

unalias — শর্টকাট মোছা

সহজ কথায়: alias সরিয়ে দেওয়া।

Syntax

```
unalias <শর্টকাট>
```

উদাহরণ

```
unalias ll
```

সম্ভাব্য Output

কোনো আউটপুট আসে না।

কখন দরকার হবে: ভুল alias ঠিক করতে, আসল কমান্ড চালাতে।

মনে রাখো: শুধু বর্তমান সেশনে কাজ করে।

source — ফাইল লোড করা

সহজ কথায়: `~/.bashrc`-এর মতো স্ক্রিপ্ট বর্তমান সেশনে লোড করা।

Syntax

```
source <ফাইল>
```

উদাহরণ

```
source ~/.bashrc
source ~/my_config.sh
```

সম্ভাব্য Output

ভেরিয়েবল ও `alias` সঙ্গে সঙ্গে কার্যকর হয়, আউটপুট সাধারণত আসে না।

কখন দরকার হবে: alias/ভেরিয়েবল না লগআউট করেই আপডেট করতে।

সাবধান: স্ক্রিপ্ট নিরাপদ কিনা জেনে নিন — এটা শেলের ভেতরে চলে।

মনে রাখো: `source .` হলো সংক্ষিপ্ত রূপ।

bash — নতুন শেল খোলা

সহজ কথায়: নতুন Bash শেল সেশন শুরু করা (আগেরটা পেছনে থাকে)।

Syntax

```
bash
```

উদাহরণ

```
bash
echo $BASH_VERSION
exit
```

সম্ভাব্য Output

আগের প্রম্পটের চেয়ে নতুন প্রম্পট দেখা যাবে। `exit` দিলে ফিরে আসবেন।

কখন দরকার হবে: নতুন সেশনে টেস্ট করতে, নিরাপদ পরিবেশে স্ক্রিপ্ট চালাতে।

মনে রাখো: `exit` দিলে আগের সেশনে ফেরা যায়।

which — কমান্ড কোন ফাইল

সহজ কথায়: কমান্ড টাইপ করলে কোন প্রোগ্রাম চলবে — পথ দেখায়।

Syntax

```
which <কমান্ড>
```

উদাহরণ

```
which python3
which git
```

সম্ভাব্য Output

```
/usr/bin/python3
/usr/bin/git
```

কখন দরকার হবে: কমান্ড কেন পাচ্ছে না, কোন ভার্সন চলছে বুঝতে।

মনে রাখো: PATH-এর খোঁজ দেখায়। `type` দিয়ে alias/বিল্টইনও দেখা যায়।

type — কমান্ডের ধরন

সহজ কথায়: কমান্ডটি alias, ফাংশন, বিল্টইন, নাকি আসল প্রোগ্রাম — জানায়।

Syntax

```
type <কমান্ড>
type -a <কমান্ড>
```

উদাহরণ

```
type ls
type -a python3
```

সম্ভাব্য Output

```
ls is aliased to `ls --color=auto'
python3 is /usr/bin/python3
```

কখন দরকার হবে: alias-এর কারণে ভিন্ন আচরণ বুঝতে।

মনে রাখো: `type -a` সব সম্ভাব্য পথ দেখায়।

whereis — কমান্ডের সব অবস্থান

সহজ কথায়: প্রোগ্রাম, ম্যান পেজ, সোর্স কোথায় আছে — একসঙ্গে খোঁজা।

Syntax

```
whereis <কমান্ড>
```

উদাহরণ

```
whereis python3
whereis git
```

সম্ভাব্য Output

```
python3: /usr/bin/python3 /usr/bin/python3.11 /usr/share/man/man1/python3.1.gz
```

কখন দরকার হবে: কম্পিলট ইনস্টল পিকচার বুঝতে।

মনে রাখো: `which` থেকে বিস্তারিত।

xargs — লিস্ট থেকে কমান্ড

সহজ কথায়: পাইপের আউটপুটকে কমান্ডের আর্গুমেন্টে বদলে দেওয়া।

Syntax

```
<আউটপুট> | xargs <কমান্ড>
```

উদাহরণ

```
ls *.txt | xargs wc -l
ps aux | grep node | awk '{print $2}' | xargs kill
```

সম্ভাব্য Output

```
12 file1.txt
45 file2.txt
57 total
```

কখন দরকার হবে: অনেক ফাইলে একই কমান্ড চালাতে, PID লিস্ট দিয়ে কাজ করতে।

সাবধান: Deposit-সেরা সতর্কতা — `xargs kill` খুব বিপজ্জনক। নামের তালিকা খালি হলে কমান্ড চালানো এড়াতে `-r` অপশন ব্যবহার করুন: `xargs -r`। শুধু নিরাপদ ও পরীক্ষিত ব্যবহারই শিখুন — ধ্বংসাত্মক কমান্ডে আগে ড্রাই-রান করুন।

মনে রাখো: beginner-safe ব্যবহার: ফাইলের তালিকা পড়া, লিস্ট প্রিন্ট করা।

tee — পাইপ পথে ফাইলে সেভ

সহজ কথায়: টার্মিনালে দেখানোর পাশাপাশি একই ডেটা ফাইলে সংরক্ষণ করা।

Syntax

```
command | tee <ফাইল>
command | tee -a <ফাইল>
```

উদাহরণ

```
ps aux | grep node | tee output.log
make build 2>&1 | tee build.log
```

সম্ভাব্য Output

```
<username> 1234 ... node server.js
$ ls -l output.log
আউটপুটটা output.log-এও সেভ হয়ে গেছে।
```

↳ tee — চলমান

কখন দরকার হবে: লাইভ লগ দেখতে দেখতে ফাইলে সেভ করতে।

মনে রাখো: `-a` দিলে append, না দিলে ওভাররাইট।

অপারেটর চিটশিট — পাইপ, রিডাইরেক্ট, চেইনিং

অপারেটর	অর্থ	উদাহরণ	টিপস
`|` (pipe)	বাম দিকের আউটপুট ডান কমান্ডে পাঠায়	`ps aux | grep node`	ডান দিয়ে খুঁজে ফিল্টার করুন
`>`	আউটপুট ফাইলে ওভাররাইট লেখে (আগের কনটেন্ট মুছে)	`echo "hello" > note.txt`	সতর্কতা: আগের কনটেন্ট হারায়
`>>`	আউটপুট ফাইলে যোগ করে (append)	`echo "another line" >> note.txt`	নিরাপদ, আগের কনটেন্ট থাকে
`<`	ফাইল থেকে ইনপুট নেয়	`wc -l < note.txt`	ম্যানুয়াল ইনপুটের436 বদলে ফাইল থেকে
`2>`	stderr (এরর) ফাইলে পাঠায়	`command 2> errors.log`	এরর আলাদা লগে
`2>&1`	এরর ও আউটপুট একসঙ্গে পাঠায়	`command > log.txt 2>&1`	পূর্ণ লগ এক ফাইলে
`&&`	বাম সফল হলে ডান চালায়	`make && ./app`	ক্রমবদ্ধ কাজ
`||`	বাম ব্যর্থ হলে ডান চালায়	`ping server || echo "down"`	ফলব্যাক
`;`	ক্রমে চালায় (সফলতা দেখে না)	`cd ~/project; ls`	সবসময় চালেবে
`*`	যেকোনো ক্যারেক্টারের সেট	`ls *.txt`	ওয়াইলডকার্ড: সব .txt ফাইল
`?`	যেকোনো একটি ক্যারেক্টার	`ls file?.log`	ওয়াইলডকার্ড: এক ক্যারেক্টার
`'` (সিঙ্গেল কোট)	ভেরিয়েবল বদলায় না	`echo '$HOME'` → `$HOME`	এখানকার সবকিছু আক্ষরিক
`"` (ডাবল কোট)	ভেরিয়েবল বদলায়	`echo "$HOME"` → `/home/<username>`	ভেরিয়েবল/কমান্ড সাবস্টিটিউশন হয়

কখন দরকার হবে: প্রসেস ফিল্টার, লগ সংরক্ষণ, স্ক্রিপ্ট অটোমেশন।

মনে রাখো: `>`-এর ওভাররাইট ব্যবহার করলে আগের কনটেন্ট ফিরানো যায় না — আগে ব্যাকআপ নিন।

দ্রুত পুনরালোচনা

- `echo` / `printf` লেখা, `env` / `printenv` / `export` পরিবেশ, `alias` ছোট নাম।
- `ps aux | grep node` + `tee output.log` হলো প্রসেস দেখা ও সংরক্ষণের জোড়া।
- `>` ওভাররাইট করে, `>>` append, `2>` এরর, `2>&1` উভয় একসঙ্গে।
- `&&` সফলতায় চেইন, `||` ফলব্যাক, `;` অনিশ্চিত ক্রম।
- পরিবেশ + PATH মানে কমান্ড কোথা থেকে আসবে, সেটার নিশ্চয়তা।

অনুশীলন

`ps aux | grep node | tee output.log` চালান, `echo "line1" > note.txt`, `echo "line2" >> note.txt`, `cat note.txt` করে ওভাররাইট বনাম append যাচাই করুন। `command 2> errors.log` ও `2>&1` টেস্ট করে আলাদা প্যাকেজ দেখুন। `alias today='date +%F'` বানিয়ে ব্যাবহার করুন।

অধ্যায় ১১

১১ সার্ভিস, লগ ও Cron — সিস্টেম চালানো

লিনাক্সে ব্যাকগ্রাউন্ডে চলা নির্দিষ্ট কাজই **সার্ভিস** — যেমন SSH সার্ভার, ওয়েব সার্ভার, ডাটাবেজ। এটি আলাদা **প্রসেস** থেকে আলাদা — প্রসেস একটি চলমান ইনস্ট্যান্স, সার্ভিস হলো সেই কাজের ব্যবস্থাপনা, যেটি বুটের সঙ্গেও চালু হতে পারে।

দুটি ধারণা আলাদা করা জরুরি:

- **enable বনাম start:** `start` মানে এখনই চালু করা, `enable` মানে বুটের সময় নিজে চালু হবে — দুটোই লাগে সার্ভিস স্থায়ী করতে।
- **লগ:** `journalctl` সিস্টেমের ডায়েরি — সার্ভিসের সব ঘটনা এখানে জমা হয়।

Cron ফিল্ড: `crontab -e` দিয়ে সময় নির্ধারণ হয় ক্রমানুসারে: মিনিট (0-59) ঘণ্টা (0-23) মাসের দিন (1-31) মাস (1-12) সপ্তাহের দিন (0-7) কমান্ড।

systemctl status — সার্ভিসের অবস্থা

সহজ কথায়: সার্ভিস চলছে কিনা, এরর কী — এক লাইনে।

Syntax

```
systemctl status <সার্ভিস>
```

উদাহরণ

```
systemctl status ssh
systemctl status nginx
```

সম্ভাব্য Output

```
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled)
     Active: active (running) since ...
```

কখন দরকার হবে: সার্ভিস ডাউন কেন নিশ্চিত করতে।

মনে রাখো: `Active: failed` মানেই লগ দেখা লাগবে।

systemctl start — সার্ভিস চালু

সহজ কথায়: বন্ধ থাকা সার্ভিস এখনই চালু করা।

Syntax

```
sudo systemctl start <সার্ভিস>
```

উদাহরণ

```
sudo systemctl start ssh
sudo systemctl start nginx
```

সম্ভাব্য Output

কোনো আউটপুট আসে না। `systemctl status` দিয়ে যাচাই করুন।

কখন দরকার হবে: ডেভ সার্ভার, SSH, ডাটাবেজ হাত দিয়ে চালু করতে।

মনে রাখো: শুধু `start` দিলে রিবুটের পর চালু নাও হতে পারে — `enable` দরকার।

systemctl stop — সার্ভিস বন্ধ করা

সহজ কথায়: চলমান সার্ভিস এখনই বন্ধ করা।

Syntax

```
sudo systemctl stop <সার্ভিস>
```

উদাহরণ

```
sudo systemctl stop nginx
```

সম্ভাব্য Output

আউটপুট আসে না। `status` দেখুন।

কখন দরকার হবে: সার্ভিস বন্ধ করে কনফিগ বদলাতে, খরচ কমাতে।

সাবধান: কাস্টমার-সংক্রান্ত সার্ভিস বন্ধ করার আগে জানান।

systemctl restart — সার্ভিস রিস্টার্ট

সহজ কথায়: বন্ধ করে আবার চালু করা (সংক্ষেপে)।

Syntax

```
sudo systemctl restart <সার্ভিস>
```

উদাহরণ

```
sudo systemctl restart nginx
sudo systemctl restart ssh
```

সম্ভাব্য Output

আউটপুট আসে না। `status` দিয়ে যাচাই করুন।

কখন দরকার হবে: কনফিগ বদলের পর সার্ভিস রিফ্রেশ করতে, আটকে যাওয়া সার্ভিস ফিরাতে।

মনে রাখো: চলমান ক্লায়েন্ট সংযোগ বিচ্ছিন্ন হতে পারে।

systemctl enable — বুটে চালু করা

সহজ কথায়: সার্ভিসকে বুটের তালিকায় যোগ করা।

Syntax

```
sudo systemctl enable <সার্ভিস>
```

উদাহরণ

```
sudo systemctl enable ssh
sudo systemctl enable nginx
```

সম্ভাব্য Output

```
Created symlink ... → enabled.
```

কখন দরকার হবে: রিবুটের পর সার্ভিস নিজে আসবে নিশ্চিত করতে।

মনে রাখো: `start` ছাড়া `enable` দিলে এখনই চালু হয় না। দুটোই একসঙ্গে: `sudo systemctl enable --now <সার্ভিস>`।

systemctl disable — বুট থেকে সরানো

সহজ কথায়: সার্ভিস বুটে আর চালু হবে না।

Syntax

```
sudo systemctl disable <সার্ভিস>
```

উদাহরণ

```
sudo systemctl disable nginx
```

সম্ভাব্য Output

```
Removed symlink ... → disabled.
```

কখন দরকার হবে: অপ্রয়োজনীয় সার্ভিস বন্ধ রাখতে, নিরাপত্তা কমাতে।

মনে রাখো: disable শুধু বুট নিয়ন্ত্রণ করে — এখনই বন্ধ করতে `stop` দরকার।

journalctl -u — নির্দিষ্ট সার্ভিসের লগ

সহজ কথায়: একটি সার্ভিসের লগ পড়া।

Syntax

```
journalctl -u <সার্ভিস>
journalctl -u <সার্ভিস> --since "1 hour ago"
```

উদাহরণ

```
journalctl -u nginx --since "1 hour ago"
journalctl -u ssh -f
```

সম্ভাব্য Output

```
Feb 10 10:00:00 server systemd[1]: Started nginx.
Feb 10 10:00:05 server nginx[1234]: emerg ...
```

কখন দরকার হবে: সার্ভিস কেন ফেইল করছে, কখন শুরু হলো দেখতে।

মনে রাখো: `--since` দিয়ে নির্দিষ্ট সময়সীমা দিন।

journalctl -f — লাইভ লগ ফলো

সহজ কথায়: লগ লাইভ স্ট্রিম করা — নতুন লাইন এলে সঙ্গে সঙ্গে দেখা।

Syntax

```
journalctl -f
journalctl -u <সার্ভিস> -f
```

উদাহরণ

```
journalctl -f
journalctl -u ssh -f
```

সম্ভাব্য Output

```
Feb 10 10:05:00 server sshd[5678]: Accepted password for <username> from <server_ip>
Feb 10 10:05:12 server sshd[5679]: Failed password ...
```

কখন দরকার হবে: লাইভ সমস্যা ডিবাগ করতে, লগইন অ্যাটেম্পট দেখতে।

মনে রাখো: বের হতে `Ctrl+C`। অনেক লাইন — `less`-এর সঙ্গে ব্যবহারও ভালো।

crontab -e — সময়সূচি সম্পাদন

সহজ কথায়: কোন সময় কী কমান্ড চালাবে — টেবিল সম্পাদন করা।

Syntax

```
crontab -e
```

উদাহরণ

```
crontab -e
```

সম্ভাব্য Output

```
কিসের সম্পাদক (nano/vim) চাইবে। তারপর # লাইনসহ ফিল্ড দেখায়।
```

কখন দরকার হবে: নিয়মিত ব্যাকআপ, লগ পরিষ্কার, ডেটা সিঙ্ক অটোমেট করা।

সাবধান: নিরাপদ, ছোট পরীক্ষা দিয়েই শুরু করুন। ভুল কমান্ডে অসীম সিস্টেম লোড হতে পারে। ডিস্ট্রো অনুযায়ী ডিফল্ট সম্পাদক আলাদা হতে পারে (Debian-এ nano বেশি সাধারণ)।

মনে রাখো: ক্রন ফিল্ড ক্রম — মিনিট ঘণ্টা মাসের-দিন মাস সপ্তাহের-দিন কমান্ড।

`crontab -l` — সময়সূচি তালিকা

সহজ কথায়: সেট করা সব ক্রন জব একসঙ্গে দেখা।

Syntax

```
crontab -l
```

উদাহরণ

```
crontab -l
```

সম্ভাব্য Output

```
*/5 * * * * echo "test" >> ~/cron-test.log
0 3 * * * tar -czf ~/backup.tar.gz ~/project
```

কখন দরকার হবে: কী শিডিউল করা আছে যাচাই করতে, ডুপ্লিকেট এড়াতে।

মনে রাখো: কোনো জব না থাকলে "no crontab" মেসেজ আসে।

নিরাপদ পরীক্ষামূলক ক্রন জব

নিরাপদ প্রশিক্ষণের জন্য ছোট উদাহরণ:

```
*/2 * * * * echo "$(date) - cron test" >> ~/cron-test.log
```

এটি প্রতি ২ মিনিটে `~/cron-test.log`-এ সময় যোগ করে। পরীক্ষা শেষে `crontab -l` দিয়ে দেখুন, লগে লাইন এসেছে কিনা দেখুন, তারপর জবটি মুছে ফেলুন।

কখন দরকার হবে: ক্রন কাজ করছে কিনা নিশ্চিত করতে, সময়ের হিসাব বুঝতে।

সাবধান: প্রোডাকশনে যাওয়ার আগে অবশ্যই টেস্ট লগে যাচাই করুন।

`systemctl list-units` — সব সার্ভিস তালিকা

সহজ কথায়: systemd-এর সব ইউনিট (সার্ভিস সহ) দেখা।

Syntax

```
systemctl list-units --type=service
systemctl list-units --failed
```

↳ systemctl list-units — চলমান

উদাহরণ

```
systemctl list-units --type=service
systemctl list-units --failed
```

সম্ভাব্য Output

```
UNIT                    LOAD   ACTIVE SUB     DESCRIPTION
ssh.service             loaded active running OpenBSD Secure Shell server
nginx.service           loaded active running nginx
```

কখন দরকার হবে: চলমান/ফেইল সার্ভিস চিহ্নিত করতে।

মনে রাখো: `list-units --failed` সমস্যার সবচেয়ে দ্রুত উপায়।

reboot — সিস্টেম রিবুট করা

সহজ কথায়: মেশিন আবার চালু করা।

Syntax

```
sudo reboot
```

উদাহরণ

```
sudo reboot
```

সম্ভাব্য Output

কোনো আউটপুট আসে না। কয়েক সেকেন্ডে সংযোগ বিচ্ছিন্ন হয়ে রিবুট শুরু হয়।

কখন দরকার হবে: আপডেটের পর, সিস্টেম স্থিতিশীল করতে, সার্ভিস বুট-টেস্ট করতে।

সাবধান: কাউকে জানিয়ে করুন, সেভ না করা কাজ শেষ করুন। শিডিউল ছাড়া ইনকারেক্ট রিবুট সার্ভিস ডাউন রাখে।

মনে রাখো: রিবুটের পর `systemctl list-units --failed` দিয়ে সার্ভিস এসেছে কিনা দেখুন।

shutdown — সিস্টেম বন্ধ করা

সহজ কথায়: নির্দিষ্ট সময়ে সিস্টেম শাটডাউন/রিবুট করা।

Syntax

```
sudo shutdown -h now
sudo shutdown -r +10
sudo shutdown -c
```

উদাহরণ

```
sudo shutdown -h now
sudo shutdown -r +10 "আপডেট শেষ হবে"
sudo shutdown -c
```

সম্ভাব্য Output

```
Shutdown scheduled for ..., use 'shutdown -c' to cancel.
Broadcast message from root ... shutdown ...
```

কখন দরকার হবে: সময়সূচি অনুযায়ী বন্ধ করা, কাউন্টডাউনসহ রিবুট।

সাবধান: `-h now` সঙ্গে সঙ্গে বন্ধ করে — সেভ করা ডেটা শেষ আছে কিনা নিশ্চিত করুন। `shutdown -c` সবসময় হাতে রাখুন।

দ্রুত পুনরালোচনা

- সার্ভিস = ব্যবস্থাপনা, প্রসেস = চলমান ইনস্ট্যান্স।
- `start` এখন, `enable` বুটে — স্থায়ী সার্ভিসে দুটোই লাগে।
- `journalctl -u` সার্ভিসের লগ, `-f` লাইভ ফলো।
- `crontab -e` / `-l` দিয়ে সময়সূচি করা, ফিল্ড: মিনিট ঘণ্টা দিন মাস সপ্তাহ-দিন।
- `reboot` / `shutdown` চালানোর আগে জানান ও সেভ শেষ করুন।

অনুশীলন

`systemctl status ssh`, `systemctl list-units --type=service`, `journalctl -u ssh --since "1 hour ago"`, `journalctl -f` চালিয়ে নিজের সার্ভিসের অবস্থা দেখুন। একটি নিরাপদ ক্রন লাইন (`echo "$(date) - test" >> ~/cron-test.log`) যোগ করে দু'মিনিট অপেক্ষা করে লগ যাচাই করুন। শেষে ক্রন থেকে সরিয়ে ফেলুন।

অধ্যায় ১২

১২ ডেভেলপারদের জন্য লিনাক্স

ডেভেলপার হিসেবে লিনাক্সে আপনি শুধু ফাইল ঘাঁটাবেন না — টার্মিনাল ব্যবহার করে কোড টেনে আনবেন, ডিপেন্ডেন্সি ইনস্টল করবেন, অ্যাপ চালাবেন, সার্ভার রিস্টার্ট করবেন আর লগ পড়বেন। এই অধ্যায়ে কোনো টুলের পূর্ণ ম্যানুয়াল নেই। আছে বাস্তব ওয়ার্কফ্লো: Git, Node.js/npm, PM2 আর Nginx।

> মনে রাখুন: Git, Node.js, npm, PM2 আর Nginx আলাদা ইনস্টল করা টুল। এরা লিনাক্সের বিল্ট-ইন কমান্ড নয়। লিনাক্সে `ls`, `cat`, `grep`, `ps`, `ss`, `chmod` — এগুলো বিল্ট-ইন/কোর কমান্ড। Git/Node/pm2/nginx আলাদাভাবে প্যাকেজ ম্যানেজার দিয়ে ইনস্টল করতে হয়। ইনস্টলেশনের নির্দেশ ডিস্ট্রো ও ভার্সন অনুযায়ী আলাদা হতে পারে, তাই প্রয়োজনে অফিসিয়াল ডকুমেন্টেশন দেখুন।

`git status` — Git রিপোজিটরির বর্তমান অবস্থা দেখা

সহজ কথায়: এখন কোন ফাইল বদলেছে, কোনটা স্টেজ হয়েছে, কোনটা নতুন — সব এক নজরে। কোড এডিটের পর প্রথমে এটাই দেখুন।

Syntax

```
git status
git status --short
```

উদাহরণ

```
git status
```

সম্ভাব্য Output

```
On branch main
Changes not staged for commit:
  modified:   app.js
Untracked files:
  .env.example
```

কখন দরকার হবে: কোড লেখা, রিভার্ট, পুল/পুশ করার আগে; কি কমিট করছেন তা যাচাই করতে।

মনে রাখো: এটা কোনো ফাইল বদলায় না, শুধু রিপোর্ট করে — নিরাপদে বারবার চালানো যায়। রিপোজিটরির বাইরে চালালে এরর আসবে।

git clone — রিমোট রিপোজিটরি ডাউনলোড / ক্লোন করা

সহজ কথায়: GitHub/GitLab থেকে প্রোজেক্টের একটি কপি আপনার ডিরেক্টরিতে আনুন।

Syntax

```
git clone <repository_url> [directory]
```

উদাহরণ

```
git clone https://github.com/example/project.git
git clone https://github.com/example/project.git my-project
```

সম্ভাব্য Output

```
Cloning into 'project'...
remote: Enumerating objects: 100, done.
Receiving objects: 100% (150/150), done.
```

কখন দরকার হবে: নতুন প্রোজেক্ট শুরুতে, VPS-এ কোড আনার সময়।

মনে রাখো: প্রাইভেট রিপোর জন্য SSH কী বা টোকেন লাগতে পারে। পাবলিক URL-এ পাসওয়ার্ড বসাবেন না।

git add — ফাইল স্টেজে তোলা

সহজ কথায়: কমিটের জন্য কোন ফাইল নেবেন তা বেছে নেওয়া। সরাসরি কমিট হয় না।

Syntax

```
git add <file>
git add .
```

উদাহরণ

```
git add app.js
git add README.md
git add .
```

সম্ভাব্য Output

```
(সাধারণত কোনো আউটপুট আসে না — এরর না থাকলে সফল।)
```

কখন দরকার হবে: বদল দেখার পর নির্দিষ্ট ফাইল কমিট করতে। `git status` দিয়ে যাচাই করুন।

↳ git add — চলমান

মনে রাখো: `git add .` এক নজরে সব ফাইল নেবে — `.env`, লগ, `node_modules` ভুল করে যেন না ঢোকে। `.gitignore` ব্যবহার করুন।

git commit — পরিবর্তন সংরক্ষণ করা

সহজ কথায়: স্টেজ করা বদল একটি কমিট হিসেবে রিপোজিটরির ইতিহাসে রাখুন।

Syntax

```
git commit -m "<বার্তা>"
git commit
```

উদাহরণ

```
git commit -m "Add login validation"
```

সম্ভাব্য Output

```
[main a1b2c3d] Add login validation
 1 file changed, 20 insertions(+)
```

কখন দরকার হবে: ছোট, অর্থবহ ইউনিট করে কাজ জমা রাখতে।

মনে রাখো: সংক্ষিপ্ত, স্পষ্ট বার্তা লিখুন। কমিট শুধু লোকালে হয় — রিমোটে পাঠাতে আলাদা `git push` লাগে।

git pull — রিমোট থেকে সর্বশেষ কোড টানা

সহজ কথায়: রিমোটের (GitHub/GitLab) নতুন কমিট আপনার লোকাল ব্রাঞ্চে আনা।

Syntax

```
git pull
git pull origin main
```

উদাহরণ

```
git pull
```

↳ git pull — চলমান

সম্ভাব্য Output

```
From https://github.com/example/project.git
 * branch            main       -> FETCH_HEAD
Updating abc1234..def5678
Fast-forward
```

কখন দরকার হবে: কাজ শুরু আগে, `Already up to date` দেখার জন্য।

মনে রাখো: না-কমিট করা বদল থাকলে কনফ্লিক্ট হতে পারে। প্রথমে `git status` দেখুন।

git push — লোকাল কমিট রিমোটে পাঠানো

সহজ কথায়: আপনার লোকাল কমিট GitHub/GitLab-এ পাঠানো যাতে টিম/সার্ভারে পৌঁছায়।

Syntax

```
git push
git push origin main
git push -u origin <new_branch>
```

উদাহরণ

```
git push origin main
```

সম্ভাব্য Output

```
Enumerating objects: 5, done.
To https://github.com/example/project.git
   abc1234..def5678  main -> main
```

কখন দরকার হবে: কাজ কমিট শেষে শেয়ার বা ডিপ্লয় করতে।

মনে রাখো: প্রাইভেট রিপোর জন্য অথেন্টিকেশন লাগে। জোর করে `--force` না-বুঝে ব্যবহার করবেন না।

git ইনস্টলেশন নোট

```
# ডিস্ট্রো অনুযায়ী প্যাকেজ ম্যানেজার আলাদা; উদাহরণ Ubuntu/Debian:
sudo apt update
sudo apt install git
```

ইনস্টলের পর `git --version` দিয়ে যাচাই করুন। প্রথমবার ব্যবহারে নাম/ইমেইল সেট করা থাকতে পারে: `git config --global user.name` ও `user.email`।

node --version — Node.js ইনস্টল আছে কি না যাচাই

সহজ কথায়: Node.js ভার্সন দেখা। না থাকলে ভার্সন লেখা কমান্ডেই এরর আসবে।

Syntax

```
node --version
node -v
```

উদাহরণ

```
node --version
```

সম্ভাব্য Output

```
v20.11.0
```

কখন দরকার হবে: প্রজেক্ট চালানোর আগে, সাপোর্ট টিকেটে ভার্সন জানাতে।

মনে রাখো: Node.js আলাদা টুল; লিনাক্সে বিল্ট-ইন নয়। ইনস্টল না থাকলে `command not found` আসবে।

npm --version — npm প্যাকেজ ম্যানেজার যাচাই

সহজ কথায়: npm (Node প্যাকেজ ম্যানেজার) ইনস্টল ও ভার্সন দেখা।

Syntax

```
npm --version
```

উদাহরণ

```
npm --version
```

সম্ভাব্য Output

```
10.2.4
```

কখন দরকার হবে: ডিপেন্ডেন্সি ইনস্টল, স্ক্রিপ্ট চালানোর আগে।

মনে রাখো: npm সাধারণত Node.js-এর সাথে আসে। ভার্সন Node-এর সাথে সামঞ্জস্য থাকা দরকার।

npm install — প্যাকেজ ইনস্টল

সহজ কথায়: `package.json` অনুযায়ী ডিপেন্ডেন্সি ডাউনলোড করে `node_modules` তৈরি।

Syntax

```
npm install
npm install <package>
npm install -D <package>
```

উদাহরণ

```
npm install
npm install express
```

সম্ভাব্য Output

```
added 50 packages, and audited 51 packages in 2s
```

কখন দরকার হবে: ক্লোনের পর প্রথম সেটআপে, নতুন লাইব্রেরি যোগে।

মনে রাখো: ইনস্টলের পর `package-lock.json` বদলায় — সেটাও Git-এ রাখা ভালো। `node_modules` সাধারণত `.gitignore`-এ থাকে।

npm run — প্রোজেক্ট স্ক্রিপ্ট চালানো

সহজ কথায়: `package.json` এর `scripts` ধরে অ্যাপ/টেস্ট/বিল্ড চালানো।

Syntax

```
npm run <script>
npm run dev
npm run start
```

উদাহরণ

```
npm run dev
npm run build
```

সম্ভাব্য Output

```
> app@1.0.0 dev
> node server.js
Server listening on 3000
```

কখন দরকার হবে: ডেভ সার্ভার, টেস্ট, প্রোডাকশন স্টার্ট — যা-ই `package.json` এ সংজ্ঞায়িত।

↳ npm run — চলমান

মনে রাখো: স্ক্রিপ্টের নাম প্রোজেক্টভেদে আলাদা; প্রথমে `cat package.json` দেখুন।

npx — প্যাকেজ না ইনস্টল করে টুল চালানো

সহজ কথায়: শুধু একবারের জন্য কোনো CLI টুল চালানো, গ্লোবাল ইনস্টল ছাড়া।

Syntax

```
npx <command>
npx --yes <package>
```

উদাহরণ

```
npx create-vite@latest
npx tsc --version
```

সম্ভাব্য Output

(টুল অনুযায়ী আউটপুট; প্রথমবারে ডাউনলোডের নোট আসতে পারে)

কখন দরকার হবে: ওয়ান-অফ টুল, স্ক্যাফোল্ড, পুরনো প্রোজেক্টে ভিন্ন ভার্সন চালাতে।

মনে রাখো: অনন্যশ্রদ্ধেয়/অপরিচিত প্যাকেজে `npx` চালাবেন না — এটা কোড চালায়। আগে সোর্স/সার্টিফিকেট যাচাই করুন।

pm2 list — PM2 প্রসেস তালিকা

সহজ কথায়: PM2-এর অধীনে চলা সব Node প্রসেস, স্ট্যাটাস, মেমরি দেখা।

Syntax

```
pm2 list
pm2 status
```

উদাহরণ

```
pm2 list
```

সম্ভাব্য Output

```
┌────┬──────┬────────┬─────┬────────┐
│ id │ name │ status │ cpu │ memory │
```

↳ pm2 list — চলমান

কখন দরকার হবে: VPS-এ অ্যাপ চলছে কিনা, ক্র্যাশ হয়েছে কিনা দেখতে।

মনে রাখো: PM2 Node-এর প্রসেস ম্যানেজার, লিনাক্সের বিল্ট-ইন নয়। আলাদাভাবে ইনস্টল করতে হয়।

pm2 logs — PM2 অ্যাপের লগ দেখা

সহজ কথায়: অ্যাপ কি বলে গেল — এরর, স্টার্টআপ মেসেজ — টেইল করা।

Syntax

```
pm2 logs <name>
pm2 logs <name> --lines 50
```

উদাহরণ

```
pm2 logs my-app --lines 50
```

সম্ভাব্য Output

```
my-app    | Server listening on 3000
my-app    | SyntaxError: ...
```

কখন দরকার হবে: অ্যাপ ক্র্যাশ, পোর্ট এরর, env মিসিং — কারণ খুঁজতে।

মনে রাখো: লগে পাসওয়ার্ড/সিক্রেট ছেপে যাওয়া বদ অভ্যাস; তারপর `Ctrl+C` দিয়ে বের হও।

pm2 restart — অ্যাপ রিস্টার্ট

সহজ কথায়: চলমান অ্যাপ বন্ধ না-করে নতুন কোড/কনফিগ লোড করে আবার চালানো।

Syntax

```
pm2 restart <name|id>
```

উদাহরণ

```
pm2 restart my-app
```

সম্ভাব্য Output

```
[PM2] Restarting my-app
[PM2] Application my-app restarted
```

↳ pm2 restart — চলমান

কখন দরকার হবে: কোড আপডেট, env বদল, আটকে যাওয়া প্রসেস জাগাতে।

মনে রাখো: ভুল নাম দিলে অন্য অ্যাপ রিস্টার্ট হবে — প্রথমে `pm2 list` দেখুন।

pm2 stop — অ্যাপ বন্ধ করা

সহজ কথায়: PM2 অ্যাপ থামানো (মুছে ফেলা নয়)।

Syntax

```
pm2 stop <name|id>
pm2 stop all
```

উদাহরণ

```
pm2 stop my-app
```

সম্ভাব্য Output

```
[PM2] Stopping my-app
[PM2] Successfully stopped
```

কখন দরকার হবে: ডিপ্লয় রোলব্যাক, মেইন্টেন্যান্স, পোর্ট খালি করতে।

মনে রাখো: `pm2 stop` আর `pm2 delete` আলাদা; ডিলিট করলে লিস্ট থেকে হারে যাবে।

pm2 ইনস্টলেশন নোট

```
npm install -g pm2
pm2 --version
```

PM2 নিজে npm প্যাকেজ; Node থাকতে হবে। রিবুটের পর চালুর জন্য `pm2 startup` সেটআপের কথা অফিসিয়াল ডকুমেন্টে আছে — বুঝে ব্যবহার করুন।

`nginx -t` — Nginx কনফিগ টেস্ট

সহজ কথায়: Nginx কনফিগে সিনট্যাক্স এরর আছে কিনা পরীক্ষা করা, সার্ভিস থামানো ছাড়া।

Syntax

```
nginx -t
sudo nginx -t
```

উদাহরণ

```
nginx -t
```

সম্ভাব্য Output

```
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file test is successful
```

কখন দরকার হবে: কনফিগ বদলানোর পর, রিলোডের আগে।

মনে রাখো: Nginx আলাদা সার্ভার টুল; বিল্ট-ইন নয়। সিস্টেম কনফিগ দেখতে `sudo` লাগতে পারে।

`systemctl status nginx` — Nginx সার্ভিস স্ট্যাটাস

সহজ কথায়: Nginx সার্ভিস চলছে কিনা, আপটাইম, সাম্প্রতিক লগ স্ন্যাপশট।

Syntax

```
systemctl status nginx
sudo systemctl status nginx
```

উদাহরণ

```
systemctl status nginx
```

সম্ভাব্য Output

```
● nginx.service - A high performance web server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled)
   Active: active (running) since ...
```

কখন দরকার হবে: সার্ভিস ক্র্যাশ, স্টার্ট-আপ এরর ডিবাগে।

মনে রাখো: systemd যুক্ত ডিস্ট্রোতে কাজ করে; না-থাকলে এরর আসবে। টিপস: `active (failed)` দেখলে `journalctl -u nginx` দেখুন।

`systemctl reload nginx` — কনফিগ রিলোড (রিস্টার্ট নয়)

সহজ কথায়: Nginx থামানো ছাড়া নতুন কনফিগ লোড। সংযোগে সামান্য হিট কম।

Syntax

```
sudo systemctl reload nginx
```

উদাহরণ

```
sudo nginx -t
sudo systemctl reload nginx
```

সম্ভাব্য Output

```
(এরর না থাকলে সাধারণত আউটপুট আসে না)
```

কখন দরকার হবে: কনফিগ টেস্ট পাসের পর লাইভ আপডেট।

মনে রাখো: আগে অবশ্যই `nginx -t` চালান। `restart` আর `reload` আলাদা — রিলোড নিরাপদ, রিস্টার্টে ডাউনটাইম বেশি।

`journalctl -u nginx` — Nginx-এর systemd লগ

সহজ কথায়: Nginx সার্ভিসের সিস্টেম জার্নাল লগ পড়া — স্টার্ট ফেইল কেন হলো।

Syntax

```
journalctl -u nginx
journalctl -u nginx -n 50
journalctl -u nginx -f
```

উদাহরণ

```
journalctl -u nginx -n 50
```

সম্ভাব্য Output

```
systemd[1]: Started A high performance web server and reverse proxy.
nginx: [emerg] bind() to 0.0.0.0:80 failed (Address already in use)
```

কখন দরকার হবে: `systemctl status` এ failing দেখার পর কারণ খুঁজতে।

মনে রাখো: `-f` লাইভ ফলো; বের হতে `Ctrl+C`। কিছু লগ দেখতে `sudo` লাগতে পারে।

`ps aux | grep node` — Node প্রসেস খুঁজা

সহজ কথায়: চলমান সব প্রসেসের তালিকায় Node-সংক্রান্ত লাইন আনা। `ps aux` পুরো তালিকা, `grep node` ছাঁকা।

Syntax

```
ps aux | grep node
ps aux | grep "[n]ode"
```

উদাহরণ

```
ps aux | grep node
```

সম্ভাব্য Output

```
ubuntu   1234  0.0  2.0  node server.js
ubuntu   2345  0.0  0.1  grep node
```

কখন দরকার হবে: `pm2 list` আর মিলছে না, বা PM2 বাইরে চলা প্রসেস খুঁজতে।

মনে রাখো: `grep` নিজেই মেলে যাবে; `"[n]ode"` ট্রিক দিয়ে নিজেকে বাদ দেওয়া যায়। পাইপ নিরাপদ — কিছু মুছ anywhere না।

`ss -tulnp` — খোলা পোর্ট ও প্রসেস দেখা

সহজ কথায়: কোন পোর্ট শুনছে (listening), কোন প্রসেস ধরেছে — `netstat` এর আধুনিক রিপ্লেসমেন্ট।

Syntax

```
ss -tulnp
sudo ss -tulnp
```

উদাহরণ

```
ss -tulnp
```

সম্ভাব্য Output

```
tcp   LISTEN 0      4096         0.0.0.0:3000  0.0.0.0:*
tcp   LISTEN 0      4096         0.0.0.0:80    0.0.0.0:*
```

কখন দরকার হবে: `Port already in use`, Nginx/Node পোর্ট চেক, ফায়ারওয়াল ডিবাগ।

মনে রাখো: নিজের ইউজার না-হলে প্রসেস নাম/পিড দেখাতে `sudo` লাগতে পারে।

দ্রুত পুনরালোচনা

- Git: `status → add → commit → push/pull` — ছোট ইউনিটে কমিট, আগে `status`।
- Node/npm: `node --version`, `npm --version`, `npm install`, `npm run` — সব আলাদা টুল।
- PM2: `list → logs → restart/stop` — অ্যাপ জীবনের প্রসেস ম্যানেজার।
- Nginx: `nginx -t → systemctl reload` — টেস্ট ছাড়া রিলোড নয়। `journalctl -u nginx` কারণ বলে।
- প্রসেস/পোর্ট: `ps aux | grep node`, `ss -tulnp`।

অনুশীলন

1. একটি নতুন ডিরেক্টরিতে `git init` করে (যদি থাকে টুল) একটি টেস্ট ফাইল বানান, `git status` → `git add` → `git commit` চক্রটি করুন।
2. ভুলে `.env` স্টেজ হওয়ার ঘটনা সিমুলেট করুন — `git status` দেখে `.gitignore`-এ যোগ করুন।
3. `nginx -t` এবং `systemctl status nginx` দুটোই চালিয়ে আউটপুটের পার্থক্য লিখুন।

অধ্যায় ১৩

১৩

Ubuntu VPS — বেসিক ওয়ার্কফ্লো (এন্ড-টু-এন্ড)

এই অধ্যায়ে কোনো রিয়েল IP, পাসওয়ার্ড বা টোকেন নেই। সব জায়গায় প্লেসহোল্ডার ব্যবহার হয়েছে। টার্মিনালে লেখা `<...>` অংশগুলো নিজের আসল মান দিয়ে বদলাবেন, কিন্তু এই বইয়ের উদাহরণ হুবহু কপি না-করে কপি করবেন না।

প্লেসহোল্ডার গাইড: `<server_ip>` — আপনার সার্ভারের public IP; `<username>` — SSH ইউজার (উদাহরণ: `ubuntu`); `<app_dir>` — অ্যাপের ডিরেক্টরি পাথ (উদাহরণ: `/srv/my-app`)।

এই ওয়ার্কফ্লোতে শুধু অ্যাপ চালানো দেখানো হলো। ফায়ারওয়াল (UFW), DNS রেকর্ড, TLS/Let's Encrypt, সিকিউরিটি হার্ডেনিং — এগুলোর জন্য আলাদা কনফিগ ও গাইড লাগে। শুধু Nginx ইনস্টল করে আপনার সাইট সিকিউর হবে — এমন ধরে নেবেন না।

১) SSH দিয়ে কানেক্ট

```
ssh <username>@<server_ip>
ssh ubuntu@<server_ip>
```

সহজ কথায়: সার্ভারে দূরবর্তী টার্মিনাল খোলা। সফল হলে প্রম্পটে ইউজারনেম/হোস্ট দেখাবে।

২) কে আমি আর কোন OS

```
whoami
pwd
lsb_release -a
uname -a
```

সহজ কথায়: লগইন করা ইউজার, বর্তমান ডিরেক্টরি, OS রিলিজ নিশ্চিত করা।

৩) প্যাকেজ তালিকা আপডেট

```
sudo apt update
```

সহজ কথায়: প্যাকেজ ক্যাটালগ রিফ্রেশ। `sudo` এখানে প্রয়োজন, কিন্তু হার্ডকোডেড পাসওয়ার্ড কোনো স্ক্রিপ্টে দেবেন না।

৪) মৌলিক টুল ইনস্টল

```
sudo apt install -y git curl nginx
```

সহজ কথায়: Git, curl, Nginx লাগানো। Node/pm2 থাকতে হবে আপনার অ্যাপের জন্য; প্রয়োজনে Node-এর অফিসিয়াল পদ্ধতি অনুসরণ করুন।

৫) অ্যাপ ডিরেক্টরি তৈরি

```
sudo mkdir -p <app_dir>
sudo chown <username>:<username> <app_dir>
cd <app_dir>
pwd
ls -la
```

সহজ কথায়: `/srv` বা হোমের ভেতরে মালিকানাসহ অ্যাপ ফোল্ডার। পাথ আগে `pwd` / `ls` দিয়ে যাচাই করুন।

৬) কোড ক্লোন / ট্রান্সফার

```
git clone <repository_url> <app_dir>
# বা অফলাইন স্থানান্তর: হোস্ট থেকে
scp -r ./local-app <username>@<server_ip>:<app_dir>
```

সহজ কথায়: সার্ভারে কোড পৌঁছানো। প্রাইভেট রিপোর জন্য SSH কী ব্যবহার করুন।

৭) env ভ্যারিয়েবল নিরাপদে সেট করা

```
cp .env.example .env
nano .env
chmod 600 .env
```

সহজ কথায়: `.env` ফাইলে সিক্রেট রাখা, দলের অন্যদের শেয়ার না-করা। পারমিশন `600` মানে শুধু মালিক পড়তে/লিখতে পারে। `.env` কখনো Git-এ কমিট করবেন না।

৮) ডিপেন্ডেন্সি ইনস্টল

```
cd <app_dir>
npm install
```

সহজ কথায়: `node_modules` তৈরি। এরর হলে `npm --version`, Node ভার্সন আর ডিস্ক স্পেস (`df -h`) দেখুন।

৯) প্রসেস ম্যানেজার দিয়ে চালানো

```
pm2 start npm --name <app_name> -- run start
pm2 list
pm2 logs <app_name>
```

সহজ কথায়: PM2 টার্মিনাল বন্ধ হলেও অ্যাপ চালিয়ে রাখে। `pm2 list` স্ট্যাটাস, `pm2 logs` কারণ।

উদাহরণ (প্লেসহোল্ডার ছাড়া): `pm2 start npm --name my-app -- run start`

১০) পোর্ট যাচাই

```
ss -tulnp
curl http://127.0.0.1:<app_port>
```

সহজ কথায়: অ্যাপ নির্দিষ্ট পোর্টে শুনছে কিনা, লোকালহোস্টে টেস্ট করে ফায়ারওয়াল-এর আগে নিজে যাচাই। `<app_port>` — যেমন `3000`।

১১) Nginx চেক

```
sudo nginx -t
sudo systemctl status nginx
sudo systemctl reload nginx
```

সহজ কথায়: Nginx কনফিগ টেস্ট (সফল না-হলে রিলোড নয়), রিভার্স প্রক্সি সামনে থাকলে লোড।

১২) লগ দেখা

```
pm2 logs <app_name>
journalctl -u nginx -n 50
```

সহজ কথায়: অ্যাপ লগ আর ওয়েব সার্ভার লগ আলাদা জায়গায় — দুটোই দেখতে হয়।

১৩) Minimal restart / reload

```
pm2 restart <app_name>
pm2 list
sudo nginx -t
sudo systemctl reload nginx
```

সহজ কথায়: কোড আপডেটের পর অ্যাপ রিস্টার্ট, Nginx শুধু টেস্ট পাস হলে রিলোড।

দ্রুত পুনরালোচনা

SSH → `whoami/OS check` → `apt update` → টুল → `<app_dir>` → কোড → `.env` → `npm install` → pm2 → `ss -tulnp` → `nginx -t` → logs (`pm2 logs` / `journalctl -u nginx`) → `pm2 restart` + Nginx reload। ফায়ারওয়াল/DNS/TLS/হার্ডেনিং আলাদা দরকার।

অনুশীলন

1. নিজের টেস্ট VPS (আসল সার্ভার নয়, স্ন্যাপশট/কন্টেইনার) এ ১–৪ ধাপ পর্যন্ত অনুসরণ করুন।
2. `<app_dir>` পরিবর্তন করে ভুল পাথ দিয়ে `cd` করে দেখুন কি এরর আসে; তারপর `pwd` / `ls` দিয়ে থামুন।
3. `.env` ফাইলের পারমিশন `ls -l .env` দিয়ে দেখুন কেন `600` প্রয়োজন ব্যাখ্যা করুন।

অধ্যায় ১৪

১৪ ট্রাবলশুটিং — common সমস্যা ও নিরাপদ সমাধান

আতঙ্কিত হবেন না। প্রতিটা সমস্যার ধরণ: সম্ভাব্য কারণ → ডায়াগনস্টিক কমান্ড → নিরাপদ পরবর্তী ধাপ। ভুল ধারণায় সিস্টেম বদলাবেন না।

1. `command not found`

- **সম্ভাব্য কারণ:** টুল ইনস্টল নেই, PATH-এ নেই, টাইপো, ভার্চুয়াল এনভায়রনমেন্ট বন্ধ।
- **ডায়াগনস্টিক:** `which <command>`; `command -v <command>`; `ls /usr/bin/<command>`
- **নিরাপদ পরবর্তী ধাপ:** `sudo apt search <name>` করে প্যাকেজ খুঁজুন, বানান ভুল কিনা `man` / `--help` দেখুন। অন্ধভাবে PATH বদলাবেন না।

2. `Permission denied`

- **সম্ভাব্য কারণ:** ফাইল 644/755/600 বনাম মালিক, ফোল্ডারে execute বিট নেই, `sudo` দরকার।
- **ডায়াগনস্টিক:** `ls -l <path>`; `stat <path>`; `id`
- **নিরাপদ পরবর্তী ধাপ:** মালিক/পারমিশন দেখে `chmod` / `chown` ঠানুন; প্রয়োজনে `sudo` দিয়ে চালান, পুরো রুট স্ক্রিপ্টে `sudo bash` চালাবেন না।

3. `No such file or directory`

- **সম্ভাব্য কারণ:** ভুল ডিরেক্টরি, টাইপো, ফাইল নেই।
- **ডায়াগনস্টিক:** `pwd`; `ls -la`; `ls <parent_dir>`
- **নিরাপদ পরবর্তী ধাপ:** `cd` ঠিক করুন, টাইপো যাচাই করুন; `rm` বা `mkdir -p` দিয়ে আন্দাজে বদলাবেন না।

4. `Could not get lock` / apt lock

- **সম্ভাব্য কারণ:** আরেকটি apt/automatic-update প্রসেস চলছে।
- **ডায়াগনস্টিক:** `ps aux | grep apt`; `sudo fuser -v /var/lib/dpkg/lock-frontend`
- **নিরাপদ পরবর্তী ধাপ:** অন্য প্রসেস শেষ হতে দিন। **লক ফাইল ডিলিট করে সিস্টেম সারাবেন না — ডিফল্ট হিসেবে এটা করতে যাবেন না।** ধৈর্য ধরুন; আসলে অন্য update চলছে, লক নিজে সাফ হবে।

5. DNS / নেটওয়ার্ক ব্যর্থতা

- **সম্ভাব্য কারণ:** ইন্টারনেট নেই, DNS সেটিং ভুল, ফায়ারওয়াল, Wi-Fi/VPN।
- **ডায়াগনস্টিক:** `ping -c 1 1.1.1.1`; `getent hosts example.com`; `ip addr`
- **নিরাপদ পরবর্তী ধাপ:** রাউটার/VPN যাচাই করুন; Wi-Fi রিসেট করার চেয়ে আগে ক্যাবল/Wi-Fi স্ট্যাটাস দেখুন। `/etc/resolv.conf` আন্দাজে বদলাবেন না।

6. `Port already in use`

- **সম্ভাব্য কারণ:** পুরনো অ্যাপ/Nginx ডেভ সার্ভার পোর্ট ছাড়েনি।
- **ডায়াগনস্টিক:** `sudo ss -tulnp | grep :<port>`
- **নিরাপদ পরবর্তী ধাপ:** পোর্ট ধরে রাখা প্রসেস বন্ধ করুন (`pm2 stop` বা `kill <pid>` যাচাই করে), অথবা অন্য পোর্টে অ্যাপ চালান। পোর্ট খালি হলে আবার চেষ্টা করুন।

7. Service failed to start

- **সম্ভাব্য কারণ:** কনফিগ এরর, পোর্ট কনফ্লিক্ট, পারমিশন, ডিপেন্ডেন্সি মিসিং।
- **ডায়াগনস্টিক:** `systemctl status <service>`; `journalctl -u <service> -n 50`; `sudo nginx -t` (Nginx-এর জন্য)
- **নিরাপদ পরবর্তী ধাপ:** লগে প্রথম `error`/`failed` লাইন দেখে সিঙ্গেল কারণ ঠিক করুন। কনফিগ ব্যাকআপ ছাড়া বদলাবেন না।

8. Disk full

- **সম্ভাব্য কারণ:** লগ জমা, `node_modules`, ডকার ইমেজ, পুরনো archive।
- **ডায়াগনস্টিক:** `df -h`; `du -sh /* 2>/dev/null | sort -h`; `du -sh ~/.cache`
- **নিরাপদ পরবর্তী ধাপ:** বড় ডিরেক্টরি দেখে শুধু জানিনভাবে ডিলিট করুন (`.env` নয়)। ক্যাশ ক্লিন করার আগে কি মুছেন লিখুন।

9. Hidden ফাইল দেখা যাচ্ছে না

- **সম্ভাব্য কারণ:** নামে `.` দিয়ে শুরু; `ls` ডিফল্টভাবে লুকায়।
- **ডায়াগনস্টিক:** `ls -a`; `ls -la ~`
- **নিরাপদ পরবর্তী ধাপ:** `-a` দিয়ে দেখুন; `ls -a` দিয়ে দেখা যায়। মুছতে চাইলে আগে `ls -la` প্রিন্ট করে নিন।

10. SSH authentication failure

- **সম্ভাব্য কারণ:** ভুল কী, ভুল ইউজার, ক্লাউড কী এখনো হয়নি, `~/.ssh/authorized_keys` পারমিশন।
- **ডায়াগনস্টিক:** `ssh -v <username>@<server_ip>`; `ls -l ~/.ssh`; `ssh -i <private_key_file> <username>@<server_ip>`
- **নিরাপদ পরবর্তী ধাপ:** সঠিক কী ও ইউজার যাচাই করুন; অন্যদের কী দিয়ে চেষ্টা বন্ধ করুন।

11. ভুল working directory

- **সম্ভাব্য কারণ:** `cd` ভুল পাথে, টাইপো, হোমে নয়।
- **ডায়াগনস্টিক:** `pwd`; `ls -la`
- **নিরাপদ পরবর্তী ধাপ:** `pwd`/`ls` ধরে এক ধাপ এগিয়ে আসুন। রুট পাথ মুছতে `rm -rf` এর আগে অবশ্যই `pwd`/`ls` দিয়ে পাথ নিশ্চিত করুন।

দ্রুত পুনরালোচনা

ডায়াগনস্টিক আগে, পদক্ষেপ পরে। লক ডিলিট নয়, ডিস্ক ফুল শনাক্ত নয়, পোর্ট/ডিরেক্টরি/পারমিশন দেখে বদলান।

অনুশীলন

1. জেনে ভুল এক্সিকিউটেবল নাম চালিয়ে `command not found` তৈরি করুন, `which` দিয়ে সমাধান করুন।
2. `644` এর ফাইলে `chmod +x` না-দিয়ে চালিয়ে `Permission denied` দেখুন, `ls -l` দিয়ে মিলিয়ে নিন।
3. `.env` ফাইল `ls -a` দিয়ে দেখুন; `ls` কেন দেখায় না ব্যাখ্যা করুন।

অধ্যায় ১৫

১৫ প্র্যাকটিস ল্যাব — ১২টি ল্যাব

প্রতিটি ল্যাব টেস্ট পরিবেশে করুন। আসল সিস্টেমে ধ্বংসাত্মক পদক্ষেপ নেবেন না। শুরুতে একটি ডেডিকেটেড ডিরেক্টরি বানান: `mkdir ~/linux-lab && cd ~/linux-lab` — এটি প্র্যাকটিস ডিরেক্টরি, প্রোডাকশন অ্যাপের ডিরেক্টরি নয়।

ল্যাব ১: প্রোজেক্ট ডিরেক্টরিতে নেভিগেট

- **লক্ষ্য:** `pwd`, `cd`, `ls` দিয়ে ডিরেক্টরি যাচাই।
- **ধাপ:** `pwd`, `mkdir demo`, `cd demo`, `pwd`, `ls`
- **প্রত্যাশিত:** শেল প্রম্পট ডিরেক্টরি বদলে; `ls` খালি তালিকা দেখায়।
- **চ্যালেঞ্জ:** `cd ..` আর `cd ~` ও ট্রাই করুন; `pwd` দিয়ে নিশ্চিত করুন।

ল্যাব ২: নেস্টেড ফোল্ডার/ফাইল

- **লক্ষ্য:** গভীরে ফাইল বানানো।
- **ধাপ:** `mkdir -p src/lib`, `touch src/lib/app.txt`, `ls -R`
- **প্রত্যাশিত:** `src/lib/app.txt` দেখা যায়।
- **চ্যালেঞ্জ:** `tree` বা `find . -type f` দেখুন।

ল্যাব ৩: কপি / রিনেম / নিরাপদ ডিলিট

- **লক্ষ্য:** `cp`, `mv`, `rm` নিরাপদে।
- **ধাপ:** `cp app.txt backup.txt`, `mv backup.txt renamed.txt`, `ls`, `rm renamed.txt`
- **প্রত্যাশিত:** ফেজ-বাই-ফেজ লিস্ট বদলায়।
- **চ্যালেঞ্জ:** `cp -r` দিয়ে ফোল্ডার কপি করুন; `rm` এর আগে `ls` বাধ্যতামূলক — সেভাবে অভ্যাস করুন।

ল্যাব ৪: লগে এরর খোঁজা (`grep` + `tail`)

- **লক্ষ্য:** `grep` , `tail` দিয়ে লগ ছাঁকা।
- **ধাপ:** `grep "Error" app.log` , `tail -n 20 app.log` , `grep -i error app.log`
- **প্রত্যাশিত:** শুধু Error লাইনগুলো আসে; টেইল শেষ লাইন দেখায়।
- **চ্যালেঞ্জ:** `grep "Error" | tail -n 5` দিয়ে শেষ ৫টি এরর করুন।

ল্যাব ৫: প্যাকেজ ইনস্টল ও ইনসপেক্ট

- **লক্ষ্য:** apt search/show ব্যবহার।
- **ধাপ:** `apt search curl | head` , `apt show curl | head -n 20`
- **প্রত্যাশিত:** প্যাকেজ নাম, সংস্করণ, সাইজ দেখা যায়।
- **চ্যালেঞ্জ:** `apt-cache policy curl` দেখুন; `sudo apt install -y curl` টেস্ট করুন।

ল্যাব ৬: এক্সিকিউটেবল স্ক্রিপ্ট

- **লক্ষ্য:** `echo` দিয়ে স্ক্রিপ্ট, `chmod +x` , চালানো।
- **ধাপ:** `echo '#!/bin/bash\necho hello' > hello.sh` , `chmod +x hello.sh` , `./hello.sh`
- **প্রত্যাশিত:** সফল হলে `hello` প্রিন্ট হয়।
- **চ্যালেঞ্জ:** `bash hello.sh` আর `./hello.sh` এর পার্থক্য লিখুন।

ল্যাব ৭: পারমিশন ফিক্স (644/755)

- **লক্ষ্য:** `chmod 644/755` ব্যবহার।
- **ধাপ:** `touch memo.txt; chmod 644 memo.txt; ls -l memo.txt`
- **প্রত্যাশিত:** `memo.txt` এ `-rw-r--r--` দেখা যায়; এক্সিকিউট করে এরর আসে।
- **চ্যালেঞ্জ:** স্ক্রিপ্টে `chmod 755` , ডিরেক্টরিতে `chmod 755` এর পার্থক্য বোঝান।

ল্যাব ৮: প্রসেস + পোর্ট মনিটর

- **লক্ষ্য:** `ps aux | grep` , `ss -tulnp` ।
- **ধাপ:** `python3 -m http.server 8000 &` , `ps aux | grep http` , `ss -tulnp | grep 8000` , `pkill -f http.server`
- **প্রত্যাশিত:** 8000 পোর্ট LISTEN দেখা যায়, বন্ধের পর হারায়।
- **চ্যালেঞ্জ:** `lsof -i :8000` বা `ss` দিয়ে পোর্ট ধরুন; `Ctrl+Z` / `fg` বনাম `pkill` দেখুন।

ল্যাব ৯: আর্কাইভ ব্যাকআপ (`tar`)

- **লক্ষ্য:** টার ডিরেক্টরি ব্যাকআপ।
- **ধাপ:** `tar -czf backup.tar.gz myfolder` , `ls -lh backup.tar.gz` , `tar -tzf backup.tar.gz | head`
- **প্রত্যাশিত:** gzip আর্কাইভ বনে; লিস্টে ফাইলগুলো দেখা যায়।
- **চ্যালেঞ্জ:** টেম্প ডিরেক্টরিতে `tar -xzf backup.tar.gz -C /tmp/opencode/test` দিয়ে রিস্টোর টেস্ট।

ল্যাব ১০: (ঐচ্ছিক) টেস্ট সার্ভারে SSH

- **লক্ষ্য:** টেস্ট সার্ভারে `whoami` , `pwd` , `ls` ।
- **ধাপ:** `ssh ubuntu@<test_server_ip>` ; `whoami; pwd; ls -la; exit`
- **প্রত্যাশিত:** টেস্ট সার্ভারে ইউজার/ডিরেক্টরি দেখা যায়।
- **চ্যালেঞ্জ:** চাবি দিয়ে `-i <key_file>` ব্যবহার করুন; `ssh -v` ডিবাগ লগ পড়ুন।

ল্যাব ১১: systemd সার্ভিস + লগস

- **লক্ষ্য:** সিম্পল সার্ভিস ফাইল দেখা, লগ পড়া।
- **ধাপ:** `systemctl status cron` , `journalctl -u cron -n 20` (সিস্টেমে থাকলে)
- **প্রত্যাশিত:** Active status ও সাম্প্রতিক লাইন দেখা যায়।
- **চ্যালেঞ্জ:** নিজের ইউজার সার্ভিস (`systemctl --user status`) বা Nginx জার্নাল দেখুন।

ল্যাব ১২: Node.js ওয়ার্কফ্লো ট্রেস

- **লক্ষ্য:** Node লাইফসাইকেল ট্রেস করা।
- **ধাপ:** `node --version; npm --version; npm init -y; npm install; npm run start` (স্ক্রিপ্ট থাকলে)
- **প্রত্যাশিত:** ভার্সন, `added` মেসেজ, node_modules তৈরি।
- **চ্যালেঞ্জ:** `ls node_modules | wc -l` আর `cat package.json` খুলে স্ক্রিপ্ট পরিবর্তন করুন।

দ্রুত পুনরালোচনা

নেভিগেশন → ফাইল অপ → `grep/tail` → `apt search/show` → `chmod +x` → `644/755` → `ps/ss` → `tar` → SSH টেস্ট → `systemctl/journalctl` → Node ট্রেস। সব নিরাপদে, টেস্ট পরিমাণে, `pwd` / `ls` নিশ্চিত করে।

অনুশীলন

এই ১২টি ল্যাবআই প্র্যাকটিস টাস্ক। নিজে শীতে লিখুন — ধাপ/প্রত্যাশিত/চ্যালেঞ্জ হিসেবে। ল্যাব শেষে `history` দেখে শুধুমাত্র কমান্ড তালিকা তুলুন (সিক্রেট থাকলে বাদ দিন)।

অধ্যায় ১৬

১৬

দ্রুত রেফারেন্স — চিটশিট

এটি মূল অধ্যায়গুলোর কনডেন্সড সারাংশ। বিস্তারিত ব্যাখ্যা আগে আছেই; এখানে শুধু দ্রুত তথ্য। সুবিধার জন্য টেবিলে ভাগ করা হলো।

নেভিগেশন

কাজ	কমান্ড
বর্তমান ডিরেক্টরি	`pwd`
ডিরেক্টরি/ফাইল তালিকা	`ls`, `ls -la`, `ls -lh`
ডিরেক্টরি বদল	`cd <dir>`, `cd ..`, `cd ~`
প্যারেন্ট/হোম	`cd ..`, `cd ~`

ফাইল অপারেশন

কাজ	কমান্ড
তৈরি	`touch <file>`, `mkdir <dir>`, `mkdir -p <dir>`
কপি/রিনেম	`cp <src> <dst>`, `mv <src> <dst>`, `cp -r`
মুছা	`rm <file>`, `rm -r <dir>`; আগে `ls` দিয়ে নিশ্চিত করুন

রিড / এডিট / সার্চ

কাজ	কমান্ড
দেখা	`cat <file>`, `less <file>`, `head -n 20`, `tail -n 20`, `tail -f`
খোঁজ	`find . -name <pattern>`, `grep "text" <file>`
এডিট	`nano`, `vi` — প্রথমে ব্যাকআপ

প্যাকেজ ম্যানেজমেন্ট

কাজ	কমান্ড
আপডেট	`sudo apt update`, `sudo apt upgrade`
সার্চ/ইনসপেক্ট	`apt search <q>`, `apt show <pkg>`, `apt policy <pkg>`
ইনস্টল/সরানো	`sudo apt install <pkg>`, `sudo apt remove <pkg>`

পারমিশন

কাজ	কমান্ড
দেখা	`ls -l <path>`, `stat <path>`
বদলানো	`chmod 644/755`, `chmod +x <file>`, `chown <user>:<group>`

প্রসেস / সিস্টেম

কাজ	কমান্ড
প্রসেস	`ps aux`, `ps aux | grep <name>`, `top`, `htop`
ডিস্ক/মেমরি	`df -h`, `du -sh`, `free -h`
ম্যানেজ	`pm2 list/logs/restart/stop`

নেটওয়ার্কিং / SSH

কাজ	কমান্ড
পোর্ট	`ss -tulnp`, `curl http://127.0.0.1:<port>`
SSH	`ssh <user>@<host>`, `ssh -i <key_file> <user>@<host>`, `scp -r ./dir <user>@<host>:<path>`
DNS	`getent hosts <domain>`

আর্কাইভ / ব্যাকআপ

কাজ	কমান্ড
বানানো	`tar -czf <out>.tar.gz <dir>`
দেখা/খোলা	`tar -tzf <file>.tar.gz`, `tar -xzf <file>.tar.gz`

শেল অপারেটর

অপারেটর	অর্থ	উদাহরণ
`|` (পাইপ)	আগের আউটপুট পরের ইনপুট	`ps aux | grep node`
`>`	আউটপুট ফাইলে (ওভাররাইট)	`tail -n 20 log > recent.log`
`>>`	আপেন্ড	`echo line >> file`
`<`	ফাইল থেকে ইনপুট	`cat < input.txt`
`;`	ক্রমানুসারে	`cd ~/a; ls`
`&&`	সফল হলে	`cd src && npm run build`
`||`	ব্যর্থ হলে	`npm install || echo fail`
`* ?`	গ্লব	`ls *.log`, `ls file?.txt`

সার্ভিস / লগস

কাজ	কমান্ড
স্ট্যাটাস	`systemctl status <svc>`, `systemctl is-active`
নিয়ন্ত্রণ	`sudo systemctl start/stop/restart/reload <svc>`, `enable`
লগ	`journalctl -u <svc> -n 50`, `journalctl -f`, `--since`

ডেভেলপার ওয়ার্কফ্লো

ধাপ	কমান্ড
Git	`git status`, `git clone <url>`, `git add`, `git commit -m ""`, `git pull`, `git push`
Node/npm	`node --version`, `npm --version`, `npm install`, `npm run <script>`, `npx <cmd>`
PM2/Nginx	`pm2 list/logs/restart/stop`, `nginx -t`, `systemctl reload nginx`
প্রসেস/পোর্ট	`ps aux | grep node`, `ss -tulnp`

Enter চাপার আগে — নিরাপত্তা চেকলিস্ট

1. `pwd` — আমি সঠিক ডিরেক্টরিতে আছি?
2. `ls` — `ls` করে পাথ নিশ্চিত করেছি?
3. `echo <path>` — পাথ মুখে পড়ে যাচাই করেছি?
4. `sudo` কমিয়েছি? — অপ্রয়োজনীয় `sudo` নেই?
5. ধ্বংসাত্মক চেক — `rm -rf`/`chmod -R`/`chown -R` আছে কিনা; থাকলে স্থগিত করেছি?
6. সিক্রেট যাচাই — পাসওয়ার্ড/টোকেন/`.env` কমান্ড লাইন বা Git-এ ঢুকছে না?
7. ব্যাকআপ — গুরুত্বপূর্ণ বদলের আগে ব্যাকআপ আছে?

এই ৭টি প্রশ্নের উত্তর না-মিললে Enter চাপবেন না।

দ্রুত পুনরালোচনা

টেবিলগুলোই আপনার ফাস্ট-ডায়াগনস্টিক গাইড। `pwd` → `ls` → কমান্ড → আউটপুট দেখা → নিরাপদ ধাপ — এই লুপ মনে রাখুন।

পরিশিষ্ট

পরিশিষ্ট

গ্লোসারি (Glossary)

বাংলা	English	অর্থ
ডিরেক্টরি	Directory/Folder	ফাইল রাখার পাত্র
পারমিশন	Permission	কে পড়তে/লিখতে/এক্সিকিউট পারে
প্যাকেজ	Package	সফটওয়্যার/লাইব্রেরির প্যাকেট
প্যাকেজ ম্যানেজার	Package Manager	প্যাকেজ খুঁজে ইনস্টল করা টুল
প্রসেস	Process	চলমান প্রোগ্রাম
টার্মিনাল	Terminal	কমান্ড ইন্টারফেস
সার্ভার	Server	সার্ভিস বা ব্যাকগ্রাউন্ড প্রোগ্রাম
ডিপেন্ডেন্সি	Dependency	অ্যাপের লাইব্রেরি প্রয়োজন
রিপোজিটরি	Repository	কোড ও ইতিহাসের স্টোর
ক্লোন	Clone	রিপো ডাউনলোড
স্টেজ	Stage	কমিটের জন্য বাছাই
কমিট	Commit	স্ন্যাপশট সেভ
পুল / পুশ	Pull / Push	আনা / পাঠানো
পোর্ট	Port	নেটওয়ার্ক যোগাযোগ ঠিকানা
সার্ভিস	Service	ব্যাকগ্রাউন্ডে চলা প্রোগ্রাম
লগ	Log	ইভেন্ট রেকর্ড
আর্কাইভ	Archive	ফাইলের প্যাকেট
env ভ্যারিয়েবল	Environment Variable	পরিবেশের কনফিগ
স্ক্রিপ্ট	Script	একাধিক কমান্ডের ফাইল
ফায়ারওয়াল	Firewall	নেটওয়ার্ক ট্রাফিক নিয়ন্ত্রণ

পরিশিষ্ট

পরিশিষ্ট

কমান্ড ইনডেক্স (Commands Index)

- `apt`, `apt search`, `apt show`, `apt policy`, `apt update`, `apt upgrade`, `apt install`, `apt remove`
- `cat`, `cd`, `chmod`, `chown`, `cp`, `curl`, `du`, `df`, `echo`, `find`, `free`, `fuser`, `getent`, `grep`, `head`, `htop`, `id`, `journalctl`, `kill`, `ls`, `less`, `lsof`, `mkdir`, `mv`, `nano`, `nginx -t`, `npm install`, `npm run`, `npm --version`, `npx`, `ping`, `pkill`, `pm2 list`, `pm2 logs`, `pm2 restart`, `pm2 stop`, `pm2 start`, `ps aux`, `pwd`, `rm`, `scp`, `ssh`, `ss -tulnp`, `stat`, `systemctl status`, `systemctl reload`, `systemctl restart`, `systemctl start`, `systemctl stop`, `systemctl enable`, `systemctl is-active`, `tail`, `tar`, `top`, `touch`, `uname`, `vi`, `which`, `whoami`, `lsb_release`
- `git status`, `git clone`, `git add`, `git commit`, `git pull`, `git push`, `git init`, `git config`, `git --version`
- `node --version`, `node -v`
- `tree`
- operators: `|`, `>`, `>>`, `<`, `;`, `&&`, `||`, `*`, `?`, `mv`, `cp`, `rm`

(Previous chapters 1–11 cover coreutils file, permission, process, network, archive, shell basics; এই ইনডেক্সে Ch12–16 এর সব কমান্ডসহ বর্ণমালা-ক্রমিক সারাংশ দেওয়া হলো। ডিস্ট্রো-ভেদে পাথ/ফ্ল্যাগ আলাদা হতে পারে, আগে `--help` / `man` দেখুন।)

পরিশিষ্ট

পরিশিষ্ট

রেফারেন্স ও পরবর্তী শেখা (References)

- **Ubuntu Help / অফিসিয়াল হেল্প** — Canonical / Ubuntu — `https://help.ubuntu.com`
- **Debian Documentation** — Debian Project — `https://www.debian.org/doc`
- **GNU Coreutils ম্যানুয়াল** — GNU Project — `https://www.gnu.org/software/coreutils/manual/coreutils.html` (প্রোজেক্ট সাইট: `https://www.gnu.org/software/coreutils/`)
- **GNU Bash ম্যানুয়াল** — GNU Project — `https://www.gnu.org/software/bash/manual/`
- **man-pages (Linux man 7)** — man7.org — `https://man7.org/linux/man-pages/`
- **systemd ডকুমেন্টেশন** — freedesktop.org — `https://freedesktop.org/software/systemd/man/`
- **OpenSSH ম্যানুয়াল** — openssh.com — `https://www.openssh.com/manual.html`
- **Git ডকুমেন্টেশন** — git-scm.com — `https://git-scm.com/doc`
- **Docker ডকুমেন্টেশন** — Docker Inc. — `https://docs.docker.com`
- **Nginx ডকুমেন্টেশন** — nginx.org — `https://nginx.org/en/docs/`
- **PostgreSQL ডকুমেন্টেশন** — PostgreSQL Global Development Group — `https://www.postgresql.org/docs/`

পরিশিষ্ট

পরিশিষ্ট

লেখক পরিচিতি (About the Author)

জয় চন্দ্র উদয়

পরিশিষ্ট

পরিশিষ্ট

সংস্করণ ও নির্ভুলতা নোট (Version / Accuracy Note)

এই বইয়ের কমান্ড Ubuntu/Debian-ভিত্তিক ডিস্ট্রোর উদাহরণে লেখা। Fedora/RHEL, Arch, Alpine-তে প্যাকেজ ম্যানেজার, পাথ, সার্ভিস ব্যবস্থা আলাদা হতে পারে। `apt`, `dnf`, `pacman`, `systemctl` এর আচরণ ডিস্ট্রো ও ভার্সন অনুযায়ী ভিন্ন হতে পারে। সব কমান্ডের ফ্ল্যাগ/আউটপুট আপডেট হতে পারে; প্রোডাকশনে ব্যবহারের আগে আপনার ডিস্ট্রোর `man` / `--help` ও অফিসিয়াল ডকুমেন্টেশন অফিসিয়াল সোর্স দিয়ে যাচাই করুন। এই বই শিক্ষামূলক — সিস্টেমে পরিবর্তন প্রয়োগের আগে ব্যাকআপ ও টেস্টে চালুন।

পরিশিষ্ট

পরিশিষ্ট

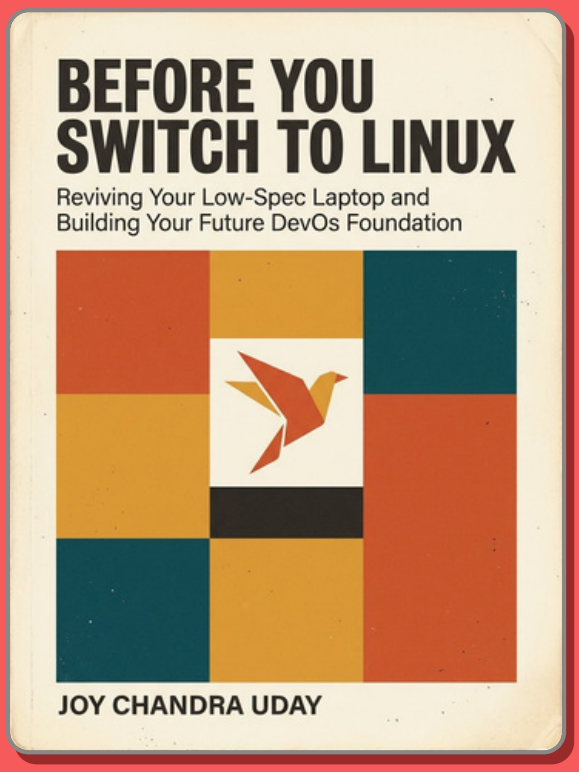
ব্র্যান্ডিং ও প্লেসমেন্ট (Branding & Placement)

এই বইয়ের ভিজ্যুয়াল পরিচয় মূল ড্রাফট (Coral-রেড + সাদা বোল্ড বাংলা + কালো পিল হেডার) ধরে রাখা হয়েছে, তবে ৭০–১০০ পৃষ্ঠার হ্যান্ডবুকের জন্য বডি পেজ সাদা/হলকা রাখা হয়েছে। কোরাল-রেড শুধু কভার, অধ্যায় ওপেনার ও অ্যাকসেন্টে ব্যবহৃত হবে।

- **কভার:** কোরাল-রেড ব্যাকগ্রাউন্ড, বড় বাংলা টাইটেল `লিনাক্স টার্মিনাল মাস্টারি`, সাবটাইটেল `কমান্ডের ব্যাখ্যা, বাস্তব ওয়ার্কফ্লো এবং DevOps চিটশিট`, সফট 3D `>_` টার্মিনাল মোটিফ, নিচে `জয় চন্দ্র উদয়`।
- **ফুটার (প্রতিটি কন্টেন্ট পেজ):** `লিনাক্স টার্মিনাল মাস্টারি | জয় চন্দ্র উদয়` + বাংলা পৃষ্ঠা নম্বর (০১, ০২...), চলমান অধ্যায় টাইটেল হেডারে।
- **অধ্যায় ওপেনার:** বড় অধ্যায় নম্বর + টাইটেল + এক লাইনের লার্নিং আউটকাম।
- **কমান্ড কার্ড:** কালো পিল হেডার (`cd (Change Directory)`) + ডার্ক টার্মিনাল কোড ব্লক (`bash` / `text`) + `সহজ অর্থ / Syntax / উদাহরণ / সম্ভাব্য Output / কখন কাজে লাগবে / মনে রাখুন`।
- **সেফটি নোট:** অ্যাম্বার ওয়ার্নিং বক্স — `rm -rf` অপরিবর্তনযোগ্য, `sudo` মানে সেফ না, `chmod 777` সাধারণত ভুল, `>` ওভাররাইট করে।
- **ক্রস-সেল প্লেসমেন্ট (শেষের খানিক আগে):** আপনার আগের বই `BEFORE YOU SWITCH TO LINUX — Reviving Your Low-Spec Laptop and Building Your Future DevOps Foundation` (লেখক: JOY CHANDRA UDAY) + বাংলা CTA `লিনাক্স এ পুরোপুরি সুইচ করার আগের প্রস্তুতি এবং পরিকল্পনা`। হলুদ হাইলাইট বক্সে — মূল PDF-এর লেআউট অনুসরণ করে। PDF ডিজাইনে এই পেজটি চিটশিটের পরে, ব্যাক কভারের আগে বসবে।
- **ব্যাক কভার:** ২–৩ লাইন ব্লার্ব + লেখক নাম + সংস্করণ নোট। লম্বা বডি কপি কভারে রাখা হবে না।

> টাইপোগ্রাফি সিস্টেম: বাংলা বডির জন্য Noto Sans Bengali / Hind Siliguri, কোডের জন্য DejaVu Sans Mono / JetBrains Mono। কভার ডিসপ্লে: বোল্ড বাংলা + হালকা ড্রপ ছায়া (শুধু কভার/হেডারে)। `টফু` বক্স এড়াতে সব পিল আইকন ফন্ট-ভিত্তিক রাখা হয়েছে।

Linux-এ পুরোপুরি সুইচ করার আগে



BEFORE YOU SWITCH TO LINUX

Reviving Your Low-Spec Laptop
and Building Your Future DevOps Foundation

JOY CHANDRA UDAY

লিনাক্সে পুরোপুরি সুইচ করার আগের প্রস্তুতি আর পরিকল্পনা

বইটি সংগ্রহ করুন →

g2gdigital.com/shop/before-you-switch-to-linux

এই বই শেষ করে পরের ধাপে যেতে চাইলে — পুরনো বা লো-স্পেক ল্যাপটপে Linux বসানো থেকে DevOps ফাউন্ডেশন গোছানোর গল্পটা **Before You Switch To Linux**-এ।

https://g2gdigital.com/shop/before-you-switch-to-linux

টার্মিনালে দেখা হবে!

লিনাক্স টার্মিনাল মাস্টারি — ভয় নয়, অভ্যাসই আসল পথ

```
>_ কমান্ড মুখস্থ নয়, বুঝেই হোক টার্মিনাল মাস্টারি।
```

জয় চন্দ্র উদয়

JOY CHANDRA UDAY

AUTHOR & FOUNDER

সফটওয়্যার ইঞ্জিনিয়ার, টেকনোলজি রাইটার এবং কো-ফাউন্ডার — **G2G Digital**। লিনাক্স ও ক্লাউড সিস্টেম, গ্রোথ ইনফ্রাস্ট্রাকচার ও আধুনিক ওয়েব প্ল্যাটফর্ম তৈরিতে নিবেদিত। সহজ বাংলায় প্র্যাকটিক্যাল লার্নিংকে ছড়িয়ে দেওয়াই এই বইয়ের মূল উদ্দেশ্য।

- joychandrauday.vercel.app
- g2gdigital.com
- linkedin.com/in/joychandraudayy
- x.com/joychandrauday

G2G Digital
Your Digital Growth Partner.

OPERATOR-GRADE EXECUTION INFRASTRUCTURE

https://g2gdigital.com

লিনাক্স টার্মিনাল মাস্টারি — প্রথম সংস্করণ, ২০২৬
কপিরাইট © ২০২৬ জয় চন্দ্র উদয় ও G2G Digital। সর্বস্বত্ব সংরক্ষিত।