Linux & Bash Scripting

Guida Completa al Sistema Operativo e Shell Scripting

15 novembre 2025

Indice

Prefazione

Benvenuti in Linux & Bash Scripting

Questo manuale rappresenta una guida completa all'utilizzo del sistema operativo Linux e alla programmazione in Bash, pensata sia per chi si avvicina per la prima volta a questo mondo, sia per chi desidera approfondire le proprie conoscenze e competenze.

Linux è molto più di un semplice sistema operativo: è una filosofia, un ecosistema di software libero e open source che ha rivoluzionato il modo in cui concepiamo l'informatica. Nato nel 1991 come progetto personale di Linus Torvalds, oggi Linux alimenta la maggior parte dei server web mondiali, tutti i supercomputer della Top500, milioni di dispositivi Android, e innumerevoli sistemi embedded.

A chi è rivolto questo manuale

Questo testo è stato concepito per diverse categorie di lettori:

- **Studenti universitari** di informatica, ingegneria informatica e discipline affini che necessitano di una solida base nell'uso di sistemi Unix-like
- **Professionisti IT** che desiderano ampliare le proprie competenze in ambito system administration e DevOps
- **Sviluppatori** che vogliono automatizzare task ripetitivi e migliorare il proprio workflow
- **Appassionati** di tecnologia che desiderano comprendere a fondo il funzionamento dei sistemi operativi
- **Utenti Windows/macOS** che vogliono esplorare un sistema operativo alternativo

Non sono richieste conoscenze pregresse di Linux, ma è necessaria una buona familiarità con i concetti base dell'informatica.

Struttura del manuale

Il testo è organizzato in capitoli progressivi che guidano il lettore dall'installazione del sistema fino alla scrittura di script avanzati:

1. **Introduzione a Linux**: storia, filosofia, distribuzioni e architettura del sistema
2. **Comandi Base**: navigazione nel filesystem, manipolazione file e directory
3. **Filesystem e Permessi**: struttura delle directory, gestione permessi e proprietà
4. **Gestione Processi**: monitoraggio, controllo e schedulazione dei processi

5. **Bash Scripting**: programmazione shell, variabili, strutture di controllo
6. **Text Processing**: elaborazione testi con sed, awk e altri strumenti
7. **Networking**: configurazione rete, troubleshooting e strumenti di diagnostica
8. **Amministrazione Sistema**: gestione utenti, pacchetti, servizi e log
9. **SSH e Sicurezza**: accesso remoto sicuro, chiavi SSH, best practices
10. **Automatizzazione**: cron, systemd timer, gestione backup e deployment

Ogni capitolo include:

- Spiegazioni teoriche chiare e concise
- Esempi pratici con codice commentato
- Esercizi di difficoltà crescente
- Best practices evidenziate in box colorati
- Riferimenti a documentazione approfondita

Metodologia di apprendimento

L'approccio didattico di questo manuale si basa sui seguenti principi:

Learning by Doing

La migliore modalità per imparare Linux è **usarlo attivamente**. Ogni concetto teorico è immediatamente seguito da esempi pratici che il lettore è invitato a sperimentare direttamente sul proprio sistema. Non limitatevi a leggere: aprite un terminale e provate ogni comando!

Progressione Graduale

I contenuti sono organizzati secondo una difficoltà crescente. I primi capitoli introducono concetti fondamentali che saranno ripresi e approfonditi nei capitoli successivi. Consigliamo di seguire l'ordine proposto, specialmente se siete alle prime armi.

Approccio Problem-Solving

Ogni sezione presenta problemi reali e mostra come risolverli con gli strumenti Linux. Questo approccio orientato alla risoluzione di problemi concreti favorisce un apprendimento più efficace e memorabile.

Comprensione Profonda

Non ci limitiamo a mostrare "cosa" fare, ma spieghiamo anche "perché" e "come" funziona. Comprendere i meccanismi sottostanti vi renderà utenti più consapevoli e capaci di risolvere problemi nuovi.

Obiettivi di apprendimento

Al termine di questo percorso sarete in grado di:

Competenze Tecniche

- Navigare efficacemente nel filesystem Linux
- Gestire file, directory e permessi con padronanza
- Monitorare e controllare processi di sistema
- Scrivere script Bash per automatizzare task complessi
- Elaborare file di testo con sed, awk e regex
- Configurare e risolvere problemi di rete
- Amministrare utenti, gruppi e servizi di sistema
- Implementare procedure di backup e recovery
- Utilizzare SSH per accesso remoto sicuro
- Schedulare job automatici con cron e systemd

Competenze Trasversali

- **Pensiero computazionale**: capacità di scomporre problemi complessi in task elementari
- **Automazione**: identificare processi ripetitivi e automatizzarli
- **Debugging**: diagnosticare e risolvere problemi in modo sistematico
- **Documentazione**: leggere e interpretare man pages e documentazione tecnica
- **Best practices**: scrivere codice pulito, manutenibile e sicuro
- **Command-line efficiency**: lavorare produttivamente senza interfaccia grafica

Prerequisiti e strumenti

Conoscenze Richieste

Per affrontare proficuamente questo manuale è consigliabile avere:

- Familiarità con concetti informatici di base (file, directory, processi)
- Capacità di utilizzo di un computer a livello utente
- Conoscenze elementari di logica e algoritmi (utile ma non indispensabile)
- Buona conoscenza della lingua inglese per consultare documentazione

Ambiente di Lavoro

Per seguire gli esempi e svolgere gli esercizi avrete bisogno di:

Setup Consigliato

- **Sistema Linux installato**: distribuzioni consigliate
  - Ubuntu/Debian: eccellenti per principianti, vasta documentazione
  - Fedora/CentOS: orientate al mondo enterprise
  - Arch Linux: per utenti avanzati che vogliono controllare ogni aspetto
- **Macchina virtuale**: VirtualBox o VMware se volete provare Linux senza modificare il vostro sistema principale
- **WSL2**: Windows Subsystem for Linux se usate Windows 10/11
- **Editor di testo**: vim, nano, o Visual Studio Code con estensioni per Bash
- **Connessione internet**: per installare pacchetti e consultare documentazione

Convenzioni tipografiche

Per facilitare la lettura, in questo manuale adottiamo le seguenti convenzioni:

- `comando`: comandi, file e directory sono in carattere monospazio
- `$`: indica il prompt della shell come utente normale
- `#`: indica il prompt della shell come root (amministratore)
- `<parametro>`: parametro da sostituire con valore effettivo
- `[opzione]`: parametro opzionale

I blocchi di codice sono presentati così:

```
# Questo è un commento esplicativo
comando argomento1 argomento2

# Output atteso:
# risultato dell'esecuzione
```

Le note importanti sono evidenziate in box colorati:

Attenzione!

Informazioni critiche, warning su operazioni potenzialmente pericolose

Suggerimento

Tips, trucchi e best practices per lavorare più efficacemente

Approfondimento

Collegamenti a risorse esterne, approfondimenti teorici, curiosità

Filosofia Unix

Prima di iniziare il viaggio pratico in Linux, è fondamentale comprendere la filosofia che sottende i sistemi Unix e Unix-like. Questi principi, enunciati dai pionieri di Unix negli anni '70, guidano ancora oggi lo sviluppo e l'utilizzo di questi sistemi.

I Principi Fondamentali

1. **Scrivi programmi che fanno una cosa sola e la fanno bene**

   Ogni comando Unix è specializzato in un compito specifico. Piuttosto che creare programmi monolitici, si preferisce avere tanti piccoli tool specializzati.

2. **Scrivi programmi che collaborano**

   I programmi dovrebbero essere progettati per lavorare insieme, con interfacce standardizzate (input/output testuale). Questo è il fondamento delle pipeline.

3. **Scrivi programmi che gestiscono flussi di testo**

   Il testo è un'interfaccia universale. I dati testuali possono essere facilmente ispezionati, modificati e processati da umani e programmi.

4. **Evita l'interfaccia utente captive**

   I programmi dovrebbero poter essere scriptati e automatizzati, non richiedere necessariamente interazione umana.

5. **Tutto è un file**

   Dispositivi hardware, processi, socket di rete: tutto è rappresentato come file nel filesystem, fornendo un'interfaccia uniforme.

Implicazioni Pratiche

Questi principi si traducono in caratteristiche concrete:

- **Composibilità**: concatenare comandi semplici per ottenere risultati complessi
- **Automazione**: script che combinano tool esistenti invece di riscrivere tutto da zero
- **Riusabilità**: lo stesso comando utilizzabile in contesti diversi
- **Trasparenza**: comportamento prevedibile e documentato

Esempio pratico della filosofia Unix:

```
# Invece di un comando monolitico per "trova i 10 utenti più attivi"
# Si combinano tool specializzati:

# 1. Estrai nomi utenti dal log
cat access.log | cut -d' ' -f1 \
    # 2. Conta occorrenze
    | sort | uniq -c \
    # 3. Ordina per frequenza
    | sort -rn \
    # 4. Prendi i primi 10
    | head -10

# Ogni tool fa una cosa sola, ma insieme risolvono problemi complessi
```

Open Source e Software Libero

Linux è parte del movimento del Software Libero e Open Source. È importante comprendere questa dimensione, che va oltre gli aspetti puramente tecnici.

Le Quattro Libertà

Secondo la Free Software Foundation, un software è "libero" se garantisce queste libertà:

0. Libertà di eseguire il programma per qualsiasi scopo
1. Libertà di studiare come funziona il programma e modificarlo
2. Libertà di redistribuire copie
3. Libertà di distribuire copie modificate

Vantaggi Pratici

Questi principi si traducono in vantaggi concreti:

Perché Open Source

- **Trasparenza**: puoi verificare cosa fa realmente il software
- **Sicurezza**: migliaia di occhi possono individuare vulnerabilità
- **Personalizzazione**: adatti il software alle tue esigenze
- **Apprendimento**: puoi studiare codice scritto da esperti
- **Sostenibilità**: non dipendi da un singolo vendor
- **Comunità**: ampio supporto da community globale
- **Costo**: quasi sempre gratuito

Come utilizzare questo manuale

Percorso Lineare

Se siete principianti, il consiglio è di procedere in ordine sequenziale:

1. Leggete attentamente la teoria di ogni sezione
2. Provate gli esempi sul vostro sistema
3. Svolgete gli esercizi proposti
4. Consultate le risorse di approfondimento
5. Solo dopo, passate al capitolo successivo

Consultazione Selettiva

Se avete già esperienza con Linux, potete usare il manuale come reference:

- Consultate l'indice per trovare l'argomento di interesse
- Focalizzatevi sui capitoli che colmano le vostre lacune
- Utilizzate gli esempi come template per i vostri script
- Fate riferimento ai box delle best practices

Apprendimento Attivo

Suggerimenti per lo Studio

- **Praticate quotidianamente**: dedicate almeno 30 minuti al giorno
- **Prendete appunti**: create il vostro cheatsheet personale
- **Sperimentate**: modificate gli esempi, provate varianti
- **Fate errori**: è il modo migliore per imparare
- **Leggete i messaggi di errore**: contengono informazioni preziose
- **Consultate man pages**: abituatevi a cercare informazioni
- **Partecipate a community**: forum, IRC, Discord, Reddit
- **Contribuite a progetti**: il modo migliore per consolidare le competenze

Risorse complementari

Questo manuale è un punto di partenza, ma l'apprendimento non finisce qui:

Documentazione Ufficiale

- **Man pages**: `man comando` - documentazione locale sempre disponibile
- **Info pages**: `info comando` - documentazione estesa per tool GNU
- `/usr/share/doc/`: documentazione installata con i pacchetti
- –**help**: quasi tutti i comandi supportano `comando -help`

Risorse Online

- **The Linux Documentation Project**: tldp.org
- **Arch Wiki**: wiki.archlinux.org (valida per tutte le distribuzioni)
- **Stack Overflow**: stackoverflow.com (domande e risposte)
- **GitHub**: milioni di repository con script di esempio
- **ExplainShell**: explainshell.com (spiega comandi complessi)

Libri Consigliati

- *The Linux Command Line* - William Shotts
- *UNIX and Linux System Administration Handbook* - Nemeth et al.
- *Classic Shell Scripting* - Robbins & Beebe
- *Advanced Bash-Scripting Guide* - Mendel Cooper

Esercizi di riscaldamento

Prima di iniziare il primo capitolo, provate questi esercizi per familiarizzare con il terminale:

Esercizio 0.1: Aprire il terminale

Trovate e aprite l'applicazione terminale sul vostro sistema:

- Ubuntu/Debian: `Ctrl+Alt+T` o cercate "Terminal" nel menu
- Fedora: cercate "Terminal" nelle applicazioni
- macOS: `Cmd+Space`, poi digitate "Terminal"
- WSL: cercate "Ubuntu" o "Debian" nel menu Start di Windows

Esercizio 0.2: Primi comandi

Provate questi comandi e osservate l'output:

```
# Visualizza data e ora corrente
date

# Visualizza il nome dell'utente
whoami

# Visualizza la directory corrente
pwd

# Visualizza informazioni sul sistema
uname -a

# Visualizza un calendario
cal
```

Esercizio 0.3: Navigazione base

```
# Lista file nella directory corrente
ls

# Lista dettagliata con file nascosti
ls -la

# Vai alla home directory
cd ~

# Torna alla directory precedente
```

```
cd -

# Visualizza contenuto di un file di testo
cat /etc/os-release
```

Parole finali

L'apprendimento di Linux e Bash è un viaggio, non una destinazione. Non scoraggiatevi se inizialmente alcuni concetti sembrano oscuri: con la pratica costante diventeranno familiari e naturali.

La comunità Linux è accogliente e pronta ad aiutare. Non abbiate paura di fare domande, ma prima provate a risolvere i problemi autonomamente: questa capacità di problem-solving è la competenza più preziosa che svilupperete.

Ricordate: ogni esperto di Linux è stato un principiante. La differenza sta nella perseveranza e nella curiosità. Siete pronti a iniziare questo viaggio affascinante nel mondo del pinguino?

Buon Apprendimento!

"The best way to predict the future is to invent it."
— **Alan Kay**

"Unix is simple. It just takes a genius to understand its simplicity."
— **Dennis Ritchie**

Iniziamo il nostro viaggio con il Capitolo 1: Introduzione a Linux!

Capitolo 1

Introduzione a Linux

1.1 Storia di Linux e Unix

1.1.1 Le Origini: Unix

La storia di Linux inizia con Unix. Nel 1969, presso i Bell Labs di AT&T, Ken Thompson e Dennis Ritchie svilupparono Unix, un sistema operativo rivoluzionario che avrebbe cambiato per sempre il panorama informatico.

Curiosità Storica

Unix nacque inizialmente come progetto personale di Ken Thompson, che voleva continuare a giocare al videogioco "Space Travel" dopo la cancellazione del progetto Multics. Per farlo, riscrisse il gioco per un computer PDP-7 inutilizzato, e insieme creò un sistema operativo minimale. Il nome "Unix" è un gioco di parole su "Multics" (Multiplexed Information and Computing Service).

Caratteristiche innovative di Unix:

- Sistema multitasking e multiutente
- Filesystem gerarchico
- File di testo per configurazione
- Shell come interfaccia utente programmabile
- Pipe per concatenare comandi
- Portabilità (dopo la riscrittura in C nel 1973)

1.1.2 La Nascita di Linux

Nel 1991, uno studente finlandese di nome **Linus Torvalds** iniziò a sviluppare un kernel Unix-like come hobby:

```
1 # Messaggio originale di Linus Torvalds su comp.os.minix
2 # 25 agosto 1991
3
4 From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
5 Newsgroups: comp.os.minix
6 Subject: What would you like to see most in minix?
7 Date: 25 Aug 91 20:57:08 GMT
8
```

```
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be
big and professional like gnu) for 386(486) AT clones. This
has been brewing since april, and is starting to get ready.
I'd like any feedback on things people like/dislike in minix,
as my OS resembles it somewhat...
```

Timeline evolutiva:

- **1991**: Prima release (0.01) - solo 10.239 righe di codice
- **1992**: Linux adotta licenza GPL
- **1994**: Versione 1.0 con pieno supporto networking
- **1996**: Tux (il pinguino) diventa la mascotte ufficiale
- **1999**: Kernel 2.2 - supporto per sistemi enterprise
- **2001**: Kernel 2.4 - supporto per 64 CPU
- **2011**: Versione 3.0 per il 20° anniversario
- **2019**: Kernel 5.0 con oltre 27 milioni di righe di codice
- **2024**: Kernel 6.x - supporto per hardware moderno

1.1.3 Il Progetto GNU

Linux da solo è solo un kernel. Il sistema completo include migliaia di tool sviluppati dal **Progetto GNU** (GNU's Not Unix), fondato da Richard Stallman nel 1983.

GNU/Linux

Tecnicamente, il sistema operativo completo dovrebbe essere chiamato "GNU/Linux":

- **Linux**: il kernel
- **GNU**: compiler (gcc), librerie (glibc), shell (bash), coreutils (ls, cp, mv...)
- Insieme formano un sistema operativo completo e funzionale

1.2 Architettura del Sistema Linux

1.2.1 Struttura a Livelli

Linux è organizzato in livelli gerarchici:

```
# Visualizzazione schematica dell'architettura

+-----------------------------------+
|    Applicazioni Utente            |  <- Browser, editor, games
+-----------------------------------+
|    Shell e Utility GNU            |  <- bash, ls, grep, sed
+-----------------------------------+
|    Librerie di Sistema            |  <- glibc, libssl, libcurl
+-----------------------------------+
|    System Call Interface          |  <- open(), read(), fork()
```

```
+----------------------------------+
|    KERNEL LINUX                  |
|  +---------------------------+   |
|  | Process Scheduler         |   |  <- Gestione processi
|  | Memory Management         |   |  <- Gestione memoria
|  | Virtual File System       |   |  <- Astrazione filesystem
|  | Network Stack             |   |  <- TCP/IP, routing
|  | Device Drivers            |   |  <- Hardware support
|  +---------------------------+   |
+----------------------------------+
|    Hardware                      |  <- CPU, RAM, disk, network
+----------------------------------+
```

1.2.2 Il Kernel

Il **kernel** è il cuore del sistema operativo. Responsabilità principali:

1. **Gestione Processi**
   - Scheduling: quale processo eseguire e quando
   - Context switching tra processi
   - Creazione e terminazione processi
   - Comunicazione inter-processo (IPC)
2. **Gestione Memoria**
   - Allocazione e deallocazione memoria
   - Memoria virtuale e paginazione
   - Protezione memoria tra processi
   - Caching e buffering
3. **Gestione File System**
   - Astrazione dei filesystem (VFS)
   - Supporto multi-filesystem (ext4, XFS, Btrfs, NFS...)
   - Caching dei metadati
   - Gestione permessi e ownership
4. **Gestione Dispositivi**
   - Driver per hardware
   - Interfaccia unificata via /dev/
   - Gestione interrupt
   - Direct Memory Access (DMA)
5. **Networking**
   - Stack TCP/IP completo
   - Routing e firewall (netfilter/iptables)
   - Socket API
   - Supporto protocolli multipli

1.2.3 Kernel Space vs User Space

```
# Verifica versione kernel
uname -r
# Output: 6.5.0-28-generic

# Visualizza informazioni dettagliate
uname -a
# Output: Linux hostname 6.5.0-28-generic #29-Ubuntu SMP x86_64 GNU/
    Linux

# Visualizza messaggi del kernel
dmesg | head -20
# Output: log di boot e eventi hardware
```

Kernel vs User Space

Kernel Space:

- Esecuzione privilegiata (ring 0)
- Accesso diretto all'hardware
- Memoria protetta
- Bug possono causare kernel panic

User Space:

- Esecuzione non privilegiata (ring 3)
- Accesso hardware via system call
- Isolamento tra processi
- Crash di un processo non affetta il sistema

1.3 Distribuzioni Linux

Una **distribuzione** (o "distro") è un sistema operativo completo basato sul kernel Linux, che include:

- Kernel Linux
- Software GNU e altre utility
- Package manager
- Desktop environment (opzionale)
- Applicazioni preinstallate
- Installer e configurazione

1.3.1 Famiglie di Distribuzioni

Debian-based

Debian: una delle distro più antiche e stabili

```
# Package manager: APT (Advanced Package Tool)
sudo apt update                     # Aggiorna lista pacchetti
sudo apt upgrade                    # Aggiorna pacchetti installati
sudo apt install nome-pacchetto     # Installa pacchetto
sudo apt remove nome-pacchetto      # Rimuove pacchetto
sudo apt search termine             # Cerca pacchetti

# File di configurazione repository
cat /etc/apt/sources.list
```

Ubuntu: basata su Debian, focus su usabilità

- Release regolari ogni 6 mesi (YY.MM: es. 24.04)
- LTS (Long Term Support) ogni 2 anni con 5 anni di supporto
- Varianti: Ubuntu Desktop, Server, Cloud
- Derivate: Kubuntu (KDE), Xubuntu (XFCE), Lubuntu (LXQt)

Linux Mint: basata su Ubuntu, ancora più user-friendly

Red Hat-based

Red Hat Enterprise Linux (RHEL): orientata al mondo enterprise

```
# Package manager: DNF/YUM (Yellowdog Updater Modified)
sudo dnf update                     # Aggiorna sistema
sudo dnf install nome-pacchetto     # Installa pacchetto
sudo dnf remove nome-pacchetto      # Rimuove pacchetto
sudo dnf search termine             # Cerca pacchetti
sudo dnf list installed             # Lista pacchetti installati
```

Fedora: versione community, tecnologie all'avanguardia
CentOS: clone gratuito di RHEL (ora CentOS Stream)
Rocky Linux / AlmaLinux: alternative a CentOS

Arch-based

Arch Linux: rolling release, massima personalizzazione

```
# Package manager: pacman
sudo pacman -Syu                    # Aggiorna sistema
sudo pacman -S nome-pacchetto       # Installa pacchetto
sudo pacman -R nome-pacchetto       # Rimuove pacchetto
sudo pacman -Ss termine             # Cerca pacchetti

# AUR (Arch User Repository) con helper yay
yay -S nome-pacchetto-aur           # Installa da AUR
```

Manjaro: basata su Arch ma più accessibile

Altre Distribuzioni Notevoli

- **openSUSE**: distro europea, ottimi tool amministrazione (YaST)
- **Gentoo**: compilazione source, massima ottimizzazione
- **Slackware**: una delle più antiche, filosofia KISS

- **Alpine Linux**: minimale, usata in container Docker
- **Kali Linux**: penetration testing e sicurezza
- **Raspberry Pi OS**: per dispositivi Raspberry Pi

1.3.2 Come Scegliere una Distribuzione

Guida alla Scelta

Per Principianti:

- Ubuntu / Linux Mint: massima compatibilità, vasta documentazione
- Fedora: se preferite software più recente

Per Server:

- Ubuntu Server LTS: supporto lungo, ampia community
- RHEL / Rocky Linux: ambiente enterprise, certificazioni
- Debian: massima stabilità

Per Utenti Avanzati:

- Arch Linux: controllo totale, sempre aggiornato
- Gentoo: ottimizzazione massima

Per Container/Cloud:

- Alpine Linux: footprint minimo
- Ubuntu Cloud: supporto cloud integrato

1.4 La Shell

La **shell** è l'interfaccia a riga di comando che permette di interagire con il sistema operativo.

1.4.1 Tipi di Shell

- **sh** (Bourne Shell): shell originale Unix
- **bash** (Bourne Again Shell): shell standard su Linux
- **zsh** (Z Shell): estensioni avanzate, popolare su macOS
- **fish** (Friendly Interactive Shell): syntax highlighting, autosuggestions
- **dash** (Debian Almquist Shell): leggera, usata per script di sistema

1.4.2 Bash: La Shell Standard

Bash è la shell più diffusa su Linux. Caratteristiche principali:

```
# Verifica shell corrente
echo $SHELL
# Output: /bin/bash
```

```

# Verifica versione bash
bash --version
# Output: GNU bash, version 5.2.15(1)-release

# Lista shell disponibili
cat /etc/shells
# Output:
# /bin/sh
# /bin/bash
# /bin/zsh
# /bin/dash
```

Anatomia del Prompt

Il prompt della shell fornisce informazioni utili:

```
# Prompt tipico
user@hostname:~/directory$

# Dove:
# user      = nome utente
# hostname  = nome del computer
# ~         = home directory (abbreviazione di /home/user)
# $         = utente normale (# se root)

# Personalizzare il prompt (variabile PS1)
echo $PS1
# Output: \u@\h:\w\$

# Prompt personalizzato con colori
PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
```

1.4.3 Funzionalità Avanzate della Shell

History dei Comandi

```
# Visualizza history comandi
history

# Esegui comando dalla history
!numero          # Esegui comando numero N
!!               # Ripeti ultimo comando
!stringa         # Esegui ultimo comando che inizia con stringa

# Ricerca nella history
Ctrl+R           # Ricerca incrementale all'indietro
Ctrl+S           # Ricerca incrementale in avanti

# Gestione history
history -c       # Pulisci history sessione corrente
history -w       # Salva history su file

# Configurazione history
export HISTSIZE=10000          # Numero comandi in memoria
export HISTFILESIZE=20000      # Numero comandi nel file
export HISTCONTROL=ignoredups:ignorespace  # Non salvare duplicati
```

Tab Completion

```
# Autocompletamento con Tab
cd /ho[Tab]               # Completa a /home/
ls /etc/sys[Tab]          # Completa a /etc/systemd/
sudo apt install fir[Tab][Tab]  # Mostra tutte le opzioni che iniziano
    con fir

# Double-Tab per mostrare tutte le opzioni
git [Tab][Tab]            # Mostra tutti i sottocomandi git
```

Job Control

```
# Esegui comando in background
comando &

# Sospendi processo in foreground
Ctrl+Z

# Riprendi processo in background
bg

# Porta processo in foreground
fg

# Lista job correnti
jobs

# Esempio pratico
sleep 100 &               # Avvia in background
# [1] 12345

jobs                      # Visualizza job
# [1]+  Running    sleep 100 &

fg 1                      # Porta in foreground
Ctrl+Z                    # Sospendi
bg 1                      # Riprendi in background
```

1.4.4 Variabili d'Ambiente

```
# Visualizza tutte le variabili d'ambiente
env
printenv

# Visualizza variabile specifica
echo $HOME               # Home directory utente
echo $USER               # Nome utente
echo $PATH               # Percorsi per eseguibili
echo $PWD                # Directory corrente
echo $SHELL              # Shell corrente

# Definire variabili d'ambiente
export VARIABILE=valore

# Variabile solo per sessione corrente
```

```
TEMP_VAR="valore temporaneo"

# Variabile permanente (aggiungi a ~/.bashrc)
echo 'export MY_VAR="valore"' >> ~/.bashrc
source ~/.bashrc        # Ricarica configurazione

# PATH: dove bash cerca gli eseguibili
echo $PATH
# /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Aggiungere directory al PATH
export PATH=$PATH:/nuova/directory
```

1.4.5 File di Configurazione

Bash legge diversi file all'avvio:

```
# File di configurazione globali (per tutti gli utenti)
/etc/profile            # Eseguito al login
/etc/bash.bashrc        # Eseguito per shell interattive

# File di configurazione utente
~/.bash_profile         # Login shell (alternativa: ~/.profile)
~/.bashrc               # Shell interattiva non-login
~/.bash_logout          # Eseguito al logout
~/.bash_history         # History comandi

# Esempio ~/.bashrc
cat ~/.bashrc
```

Contenuto tipico di `~/.bashrc`:

```
# ~/.bashrc

# Alias utili
alias ll='ls -lh'
alias la='ls -lAh'
alias l='ls -CF'
alias grep='grep --color=auto'
alias ..='cd ..'
alias ...='cd ../..'

# Prompt personalizzato
PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '

# History
HISTSIZE=10000
HISTFILESIZE=20000
HISTCONTROL=ignoreboth

# Editor predefinito
export EDITOR=vim
export VISUAL=vim

# PATH personalizzato
export PATH=$PATH:$HOME/bin:$HOME/.local/bin

# Funzioni personalizzate
```

```
mkcd() {
    mkdir -p "$1" && cd "$1"
}

# Source completamento personalizzato
if [ -f ~/.bash_completion ]; then
    . ~/.bash_completion
fi
```

1.5 Esercizi Pratici

1.5.1 Esercizio 1.1: Esplorazione Sistema

Eseguite i seguenti comandi e annotate i risultati:

```
# 1. Identificate la vostra distribuzione
cat /etc/os-release

# 2. Verificate la versione del kernel
uname -r
uname -a

# 3. Verificate la shell corrente
echo $SHELL
bash --version

# 4. Controllate variabili d'ambiente importanti
echo "Home: $HOME"
echo "User: $USER"
echo "Path: $PATH"

# 5. Visualizzate informazioni hardware
lscpu                    # Informazioni CPU
free -h                  # Memoria RAM
df -h                    # Spazio disco
```

1.5.2 Esercizio 1.2: Personalizzazione Bash

Create un file `~/.bash_aliases` con alias utili:

```
# 1. Create il file
touch ~/.bash_aliases

# 2. Aggiungete alcuni alias
cat >> ~/.bash_aliases << 'EOF'
# Navigazione
alias ..='cd ..'
alias ...='cd ../..'
alias ....='cd ../../..'

# Listing migliorato
alias ll='ls -lh'
alias la='ls -lAh'
alias lt='ls -lht'       # Ordinato per tempo

# Sicurezza
alias rm='rm -i'         # Chiedi conferma
```

```
alias cp='cp -i'
alias mv='mv -i'

# Utilità
alias h='history'
alias c='clear'
alias ports='netstat -tulanp'
EOF

# 3. Source il file in ~/.bashrc
echo '[ -f ~/.bash_aliases ] && source ~/.bash_aliases' >> ~/.bashrc

# 4. Ricaricate la configurazione
source ~/.bashrc

# 5. Testate gli alias
ll
..
```

1.5.3 Esercizio 1.3: History e Ricerca

Praticate con la history dei comandi:

```
# 1. Eseguite alcuni comandi
pwd
ls -l
date
whoami
uname -a

# 2. Visualizzate la history
history

# 3. Ripetete l'ultimo comando
!!

# 4. Eseguite un comando specifico dalla history
!numero

# 5. Ricerca interattiva (Ctrl+R, poi digitate parte del comando)
# Provate a cercare "ls" premendo Ctrl+R e digitando "ls"

# 6. Eseguite ultimo comando che inizia con "d"
!d
```

1.5.4 Esercizio 1.4: Variabili d'Ambiente

Lavorate con variabili d'ambiente:

```
# 1. Create variabile temporanea
MY_VAR="Hello Linux"
echo $MY_VAR

# 2. Create variabile d'ambiente
export MY_ENV_VAR="Environment Variable"
echo $MY_ENV_VAR

```

```
# 3. Visualizzate tutte le variabili
env | grep MY

# 4. Modificate il PATH temporaneamente
export PATH=$PATH:$HOME/scripts
echo $PATH

# 5. Create funzione personalizzata
sayHello() {
    echo "Hello, $USER! Today is $(date +%A)"
}
sayHello
```

1.6 Best Practices

Gestione Shell

1. **Mantenete pulito /.bashrc**
   - Separate alias, funzioni e configurazioni in file diversi
   - Commentate ogni personalizzazione non ovvia
   - Fate backup prima di modifiche importanti
2. **Usate alias con cautela**
   - Non fate override di comandi standard (es: non alias ls='rm -rf')
   - Preferite alias espliciti (ll invece di ridefinire ls)
   - Documentate alias complessi
3. **Gestite la history intelligentemente**
   - Aumentate HISTSIZE per conservare più comandi
   - Usate ignoredups per evitare duplicati
   - Non salvate comandi con password: iniziate con spazio
4. **Proteggete i file di configurazione**
   - Usate git per versionare dotfiles
   - Non includete informazioni sensibili
   - Controllate permessi: chmod 600 /.bashrc

Attenzione!

Non modificate mai /etc/profile o /etc/bash.bashrc a meno che non siate sicuri di cosa state facendo. Questi file affettano tutti gli utenti del sistema. Usate invece i file nella vostra home directory.

1.7 Riepilogo

In questo capitolo abbiamo esplorato:

- La storia di Unix e Linux: dalle origini ai giorni nostri

- L'architettura a livelli del sistema Linux
- Il ruolo del kernel e le sue responsabilità
- Le principali famiglie di distribuzioni e come scegliere
- La shell Bash: prompt, history, job control
- Variabili d'ambiente e PATH
- File di configurazione e personalizzazione

Avete ora una solida comprensione teorica del sistema Linux. Nel prossimo capitolo inizieremo a sporcarci le mani con i comandi fondamentali per navigare e manipolare il filesystem.

Prossimo Capitolo

Nel Capitolo 2 impareremo i comandi base di Linux: come navigare nel filesystem, creare e manipolare file e directory, cercare file e ottenere aiuto dalla documentazione.

Capitolo 2

Comandi Base di Linux

2.1 Introduzione

Padroneggiare i comandi base è fondamentale per lavorare efficacemente in Linux. In questo capitolo esploreremo i comandi essenziali per la navigazione, manipolazione file e ricerca di informazioni.

Filosofia Unix

Ricordate: ogni comando fa una cosa sola, ma la fa bene. La potenza viene dalla capacità di combinarli insieme tramite pipe e redirezioni.

2.2 Navigazione nel Filesystem

2.2.1 pwd - Print Working Directory

Il comando `pwd` mostra la directory corrente (dove vi trovate):

```
# Visualizza directory corrente
pwd
# Output: /home/user

# pwd mostra sempre il percorso assoluto
cd /var/log
pwd
# Output: /var/log

# Opzione -P: mostra percorso fisico (risolve symlink)
pwd -P

# Opzione -L: mostra percorso logico (default, mantiene symlink)
pwd -L
```

2.2.2 cd - Change Directory

Il comando `cd` cambia la directory corrente:

```
# Vai a una directory specifica
cd /var/log

# Vai alla home directory (tre modi equivalenti)
cd
cd ~
```

```
cd $HOME

# Vai alla directory precedente
cd -
# Output: /var/log (e cambia alla directory precedente)

# Vai alla directory del padre
cd ..

# Vai due livelli sopra
cd ../..

# Percorso relativo
cd Documents/Projects

# Percorso assoluto
cd /usr/local/bin

# Directory con spazi (usare quote)
cd "My Documents"
cd 'My Documents'
cd My\ Documents
```

Trucchi per cd

```
# Crea alias per directory frequenti in ~/.bashrc
alias proj='cd ~/Projects'
alias docs='cd ~/Documents'
alias down='cd ~/Downloads'

# Funzione per creare directory e entrarci
mkcd() {
    mkdir -p "$1" && cd "$1"
}

# Uso
mkcd ~/Projects/new-project
```

2.2.3 ls - List Directory Contents

Il comando `ls` elenca il contenuto di directory:

```
# Lista semplice
ls
# Output: file1.txt  file2.txt  dir1  dir2

# Lista dettagliata (long format)
ls -l
# Output:
# -rw-r--r-- 1 user group 1024 Nov 15 10:30 file1.txt
# drwxr-xr-x 2 user group 4096 Nov 15 09:15 dir1

# Mostra file nascosti (iniziano con .)
ls -a
# Output: .  ..  .bashrc  .profile  file1.txt

```

```
# Combina opzioni: long + all
ls -la
ls -l -a    # Equivalente

# Human-readable file sizes
ls -lh
# Output: -rw-r--r-- 1 user group 1.0K Nov 15 10:30 file1.txt

# Ordina per tempo di modifica (più recente prima)
ls -lt

# Ordina per tempo di modifica (inverso)
ls -ltr

# Ordina per dimensione
ls -lS

# Ricorsivo (mostra anche subdirectory)
ls -R

# Solo directory
ls -d */

# Con indicatori tipo (/ per dir, * per eseguibili)
ls -F

# Mostra inode numbers
ls -i

# Una colonna per file (utile in script)
ls -1
```

Interpretazione output ls -l:

```
# -rw-r--r-- 1 user group 1024 Nov 15 10:30 file.txt
#

#
          > Nome file
#                                                          > Data
    modifica
#                                                  > Dimensione (
    byte)
#                                           > Gruppo
#                                      > Proprietario
#                                 > Numero hard link
#
#                             > Altri: esecuzione
#                             > Altri: scrittura
#                             > Altri: lettura
#                             > Gruppo: esecuzione
#                             > Gruppo: scrittura
#                             > Gruppo: lettura
#                             > Utente: esecuzione
#                             > Utente: scrittura
#                             > Utente: lettura
#
# Primo carattere:
# - = file normale
```

```
# d = directory
# l = symbolic link
# b = block device
# c = character device
# p = named pipe
# s = socket
```

Alias Utili per ls

```
# Aggiungi a ~/.bash_aliases
alias ll='ls -lh'              # Long format human-readable
alias la='ls -lAh'             # Tutto tranne . e ..
alias lt='ls -lhtr'            # Ordinato per tempo
alias lsize='ls -lhS'          # Ordinato per dimensione
alias tree='ls -R'             # Vista ad albero semplice
```

2.3 Manipolazione File e Directory

2.3.1 mkdir - Make Directory

Crea nuove directory:

```
# Crea singola directory
mkdir mydir

# Crea multiple directory
mkdir dir1 dir2 dir3

# Crea directory con subdirectory (parent)
mkdir -p projects/linux/scripts

# Senza -p darebbe errore se projects/ non esiste
mkdir projects/linux/scripts
# mkdir: cannot create directory 'projects/linux/scripts':
# No such file or directory

# Crea con permessi specifici
mkdir -m 755 public_dir
mkdir -m 700 private_dir

# Verboso (mostra cosa viene creato)
mkdir -pv parent/child/grandchild
# Output:
# mkdir: created directory 'parent'
# mkdir: created directory 'parent/child'
# mkdir: created directory 'parent/child/grandchild'
```

2.3.2 touch - Create Empty File / Update Timestamp

Crea file vuoti o aggiorna timestamp:

```
# Crea file vuoto
touch newfile.txt

# Crea multipli file
```

```
touch file1.txt file2.txt file3.txt

# Crea file con timestamp specifico
touch -t 202311151030 oldfile.txt
# Formato: [[CC]YY]MMDDhhmm[.ss]

# Aggiorna solo access time
touch -a file.txt

# Aggiorna solo modification time
touch -m file.txt

# Non creare file se non esiste
touch -c file.txt

# Usa timestamp di altro file
touch -r reference.txt newfile.txt
```

2.3.3 cp - Copy Files and Directories

Copia file e directory:

```
# Copia file
cp source.txt destination.txt

# Copia file in directory
cp file.txt /path/to/directory/

# Copia multipli file in directory
cp file1.txt file2.txt file3.txt /destination/

# Copia directory ricorsivamente
cp -r source_dir/ destination_dir/

# Preserva attributi (timestamp, ownership, permessi)
cp -p file.txt backup/

# Preserva tutto + ricorsivo
cp -rp source_dir/ backup/

# Interattivo (chiedi conferma prima di sovrascrivere)
cp -i file.txt existing_file.txt

# Forza sovrascrittura senza chiedere
cp -f file.txt existing_file.txt

# Verboso (mostra cosa viene copiato)
cp -v file.txt backup/
# Output: 'file.txt' -> 'backup/file.txt'

# Copia solo se source è più recente di destination
cp -u source.txt destination.txt

# Crea hard link invece di copiare
cp -l file.txt hardlink.txt

# Crea symbolic link invece di copiare
cp -s /path/to/file.txt symlink.txt
```

```
37
38 # Backup automatico se destination esiste
39 cp --backup=numbered file.txt existing.txt
40 # Crea existing.txt.~1~, existing.txt.~2~, etc.
```

Attenzione con cp

- `cp` sovrascrive silenziosamente i file esistenti (usa `-i` per conferma)
- Per directory, `-r` o `-R` è obbligatorio
- `cp dir1 dir2` si comporta diversamente se dir2 esiste o no
- Se dir2 esiste: crea dir2/dir1
- Se dir2 non esiste: crea dir2 con contenuto di dir1

2.3.4 mv - Move/Rename Files and Directories

Sposta o rinomina file e directory:

```
1  # Rinomina file
2  mv oldname.txt newname.txt
3
4  # Sposta file in directory
5  mv file.txt /destination/directory/
6
7  # Sposta multipli file in directory
8  mv file1.txt file2.txt file3.txt /destination/
9
10 # Sposta directory
11 mv old_dir/ new_location/
12
13 # Interattivo
14 mv -i file.txt existing_file.txt
15
16 # Forza sovrascrittura
17 mv -f file.txt existing_file.txt
18
19 # Verboso
20 mv -v source.txt destination.txt
21 # Output: 'source.txt' -> 'destination.txt'
22
23 # Non sovrascrivere file esistenti
24 mv -n file.txt existing_file.txt
25
26 # Aggiorna solo se source è più recente
27 mv -u source.txt destination.txt
28
29 # Backup prima di sovrascrivere
30 mv --backup=numbered file.txt existing.txt
```

2.3.5 rm - Remove Files and Directories

Rimuove file e directory:

```
1  # Rimuovi file
2  rm file.txt
```

```
3
4  # Rimuovi multipli file
5  rm file1.txt file2.txt file3.txt
6
7  # Rimuovi con pattern
8  rm *.txt
9  rm backup_*
10
11 # Interattivo (chiedi conferma per ogni file)
12 rm -i file.txt
13
14 # Rimuovi directory vuota
15 rmdir empty_dir/
16
17 # Rimuovi directory e contenuto ricorsivamente
18 rm -r directory/
19
20 # Forza rimozione senza conferma
21 rm -rf directory/
22
23 # Verboso
24 rm -v file.txt
25 # Output: removed 'file.txt'
26
27 # Rimuovi solo se directory vuota
28 rmdir directory/
```

PERICOLO: rm -rf

Il comando `rm -rf` è estremamente pericoloso!

- Non chiede conferma (`-f`)
- Cancella ricorsivamente (`-r`)
- Non c'è cestino: i file sono persi per sempre
- Fate SEMPRE doppio check prima di eseguire
- Mai eseguire: `rm -rf /` o `rm -rf /*`
- Considerate alias: `alias rm='rm -i'`

Esempio catastrofico da evitare:

```
1  # PERICOLOSISSIMO - Non eseguite mai!
2  sudo rm -rf /      # Cancella tutto il sistema
3  rm -rf ~/*         # Cancella tutta la home
4  rm -rf ./*         # Cancella directory corrente
5
6  # Controllate SEMPRE la variabile prima
7  DIR="/tmp/backup"
8  rm -rf $DIR        # OK se $DIR è valorizzata
9  rm -rf $DIRR       # PERICOLO! $DIRR è vuota, diventa rm -rf /
```

2.4 Visualizzazione Contenuti File

2.4.1 cat - Concatenate and Print Files

Visualizza contenuto file:

```
# Visualizza file
cat file.txt

# Visualizza multipli file
cat file1.txt file2.txt

# Concatena file in nuovo file
cat file1.txt file2.txt > combined.txt

# Mostra numeri di riga
cat -n file.txt

# Mostra numeri di riga solo per righe non vuote
cat -b file.txt

# Mostra caratteri non stampabili
cat -A file.txt

# Mostra $ alla fine di ogni riga
cat -E file.txt

# Comprimi righe vuote multiple in una sola
cat -s file.txt

# Crea file da stdin (Ctrl+D per terminare)
cat > newfile.txt
This is line 1
This is line 2
^D

# Append a file esistente
cat >> existing.txt
Additional line
^D
```

2.4.2 less - View File Contents (Pager)

Visualizza file con navigazione:

```
# Apri file con less
less file.txt

# Comandi all'interno di less:
# Space         Pagina successiva
# b             Pagina precedente
# /pattern      Cerca pattern
# n             Prossima occorrenza
# N             Occorrenza precedente
# g             Vai all'inizio
# G             Vai alla fine
# q             Esci

# Mostra numeri di riga
```

```
less -N file.txt

# Non wrappare righe lunghe
less -S file.txt

# Monitoraggio file in tempo reale (come tail -f)
less +F logfile.txt

# Apri su pattern specifico
less +/ERROR logfile.txt
```

2.4.3 head e tail - View Beginning/End of Files

```
# Mostra prime 10 righe (default)
head file.txt

# Mostra prime N righe
head -n 20 file.txt
head -20 file.txt     # Sintassi breve

# Mostra prime N byte
head -c 100 file.txt

# Multiple files
head file1.txt file2.txt

# Ultime 10 righe (default)
tail file.txt

# Ultime N righe
tail -n 20 file.txt
tail -20 file.txt

# Mostra file in tempo reale (log monitoring)
tail -f /var/log/syslog

# Segue file anche se rinominato/ruotato
tail -F /var/log/syslog

# Mostra ultime N righe e poi segue
tail -n 50 -f logfile.txt

# Mostra da riga N in poi
tail -n +10 file.txt  # Dalla riga 10 fino alla fine
```

2.5 Ricerca e Ricerca Pattern

2.5.1 find - Search for Files

Potente comando per cercare file nel filesystem:

```
# Sintassi base
find [path] [options] [expression]

# Trova tutti i file in directory corrente
find .
```

```
6
7  # Trova file per nome
8  find . -name "*.txt"
9
10 # Case-insensitive
11 find . -iname "*.TXT"
12
13 # Trova directory
14 find . -type d
15
16 # Trova file regolari
17 find . -type f
18
19 # Trova symbolic link
20 find . -type l
21
22 # Trova per dimensione
23 find . -size +100M          # Maggiori di 100MB
24 find . -size -1k            # Minori di 1KB
25 find . -size 50M            # Esattamente 50MB
26
27 # Trova per tempo di modifica
28 find . -mtime -7            # Modificati ultimi 7 giorni
29 find . -mtime +30           # Modificati più di 30 giorni fa
30 find . -mmin -60            # Modificati ultimi 60 minuti
31
32 # Trova per tempo di accesso
33 find . -atime -1            # Acceduti ultimo giorno
34
35 # Trova per permessi
36 find . -perm 644            # Esattamente 644
37 find . -perm -644           # Almeno 644
38 find . -perm /644           # Qualunque bit di 644
39
40 # Trova per proprietario
41 find . -user username
42 find . -group groupname
43
44 # Trova file vuoti
45 find . -empty
46
47 # Combina condizioni (AND implicito)
48 find . -name "*.log" -size +10M
49
50 # OR condition
51 find . \( -name "*.txt" -o -name "*.log" \)
52
53 # NOT condition
54 find . -not -name "*.txt"
55 find . ! -name "*.txt"       # Equivalente
56
57 # Esegui comando sui risultati
58 find . -name "*.tmp" -delete                    # Cancella
59 find . -name "*.log" -exec gzip {} \;          # Comprimi
60 find . -type f -exec chmod 644 {} \;           # Cambia permessi
61 find . -name "*.txt" -exec cp {} /backup/ \;   # Copia
62
63 # Esecuzione più efficiente con +
```

```
find . -name "*.txt" -exec chmod 644 {} +

# Conferma prima di eseguire
find . -name "*.tmp" -ok rm {} \;

# Limita profondità ricerca
find . -maxdepth 2 -name "*.txt"
find . -mindepth 1 -maxdepth 1  # Solo primo livello
```

Esempi Pratici find

```
# Trova e cancella file .tmp più vecchi di 7 giorni
find /tmp -name "*.tmp" -mtime +7 -delete

# Trova file grandi (>100MB) e mostra dimensioni
find . -type f -size +100M -exec ls -lh {} \; | awk '{print $5, $9}'

# Trova file modificati oggi
find . -type f -mtime 0

# Trova e comprimi log vecchi
find /var/log -name "*.log" -mtime +30 -exec gzip {} \;

# Trova file senza proprietario valido (orphaned)
find / -nouser -o -nogroup 2>/dev/null

# Trova file eseguibili
find . -type f -executable

# Trova file con permessi problematici
find . -type f -perm 777  # World-writable
```

2.5.2 grep - Search Pattern in Files

Cerca pattern testuali nei file:

```
# Sintassi base
grep pattern file.txt

# Case-insensitive
grep -i pattern file.txt

# Mostra numero di riga
grep -n pattern file.txt

# Mostra solo count delle corrispondenze
grep -c pattern file.txt

# Mostra righe che NON matchano
grep -v pattern file.txt

# Cerca in multipli file
grep pattern file1.txt file2.txt

# Cerca ricorsivamente in directory
```

```
grep -r pattern directory/

# Segui symbolic link
grep -R pattern directory/

# Mostra solo nomi file con match
grep -l pattern *.txt

# Mostra solo nomi file SENZA match
grep -L pattern *.txt

# Contesto: righe prima e dopo match
grep -C 3 pattern file.txt    # 3 righe prima e dopo
grep -B 2 pattern file.txt    # 2 righe prima
grep -A 5 pattern file.txt    # 5 righe dopo

# Regex avanzate (Extended regex)
grep -E 'pattern1|pattern2' file.txt

# Evidenzia match con colori
grep --color=auto pattern file.txt

# Match intera parola
grep -w word file.txt

# Match intera riga
grep -x "exact line" file.txt

# Escludi file binari
grep -I pattern *

# Mostra solo la parte che matcha
grep -o pattern file.txt
```

Esempi con Regular Expression:

```
# Trova indirizzi email
grep -E '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' file.txt

# Trova indirizzi IP
grep -E '\b([0-9]{1,3}\.){3}[0-9]{1,3}\b' file.txt

# Trova righe che iniziano con parola
grep '^ERROR' logfile.txt

# Trova righe che finiscono con parola
grep 'failed$' logfile.txt

# Trova righe vuote
grep '^$' file.txt

# Trova righe NON vuote
grep -v '^$' file.txt

# Trova numeri
grep -E '[0-9]+' file.txt

# Combina pattern
grep -E '^(ERROR|WARNING|CRITICAL)' logfile.txt
```

2.6 Documentazione e Aiuto

2.6.1 man - Manual Pages

Il sistema di documentazione integrato di Linux:

```
# Visualizza manuale comando
man ls

# Sezioni del manuale:
# 1: Programmi utente
# 2: System calls
# 3: Librerie C
# 4: Dispositivi (/dev)
# 5: Formati file e convenzioni
# 6: Giochi
# 7: Miscellanea
# 8: Comandi amministrazione

# Specifica sezione
man 5 passwd    # File /etc/passwd
man 1 passwd    # Comando passwd

# Cerca nelle man page
man -k keyword
apropos keyword  # Equivalente

# Cerca exact match
man -f command
whatis command   # Equivalente

# Aggiorna database man
sudo mandb

# All'interno di man:
# /pattern      Cerca pattern
# n             Prossima occorrenza
# N             Occorrenza precedente
# q             Esci
# h             Help
```

2.6.2 help e –help

```
# Per comandi builtin della shell
help cd
help alias

# Per comandi esterni
ls --help
grep --help
find --help

# Formato compatto
command -h
```

2.6.3 info - Info Pages

Documentazione GNU più dettagliata:

```
# Apri info page
info ls
info coreutils

# Navigazione in info:
# Space          Pagina successiva
# Backspace      Pagina precedente
# n              Prossimo nodo
# p              Nodo precedente
# u              Nodo superiore
# q              Esci
```

2.7 Esercizi Pratici

2.7.1 Esercizio 2.1: Navigazione e Listing

```
# 1. Trovate la vostra directory corrente
pwd

# 2. Andate nella directory /etc
cd /etc

# 3. Lista tutti i file inclusi i nascosti
ls -la

# 4. Trovate i 5 file più grandi in /etc
ls -lhS | head -6

# 5. Tornate alla directory precedente
cd -

# 6. Andate alla vostra home
cd ~

# 7. Create questa struttura:
#    ~/projects/
#               linux-practice/
#                   docs/
#                   scripts/
mkdir -p ~/projects/linux-practice/{docs,scripts}

# 8. Verificate la struttura
ls -R ~/projects/
```

2.7.2 Esercizio 2.2: Manipolazione File

```
# 1. Create directory di lavoro
mkdir ~/practice
cd ~/practice

# 2. Create alcuni file
touch file1.txt file2.txt file3.txt
```

```

# 3. Create file con contenuto
cat > notes.txt
This is my first note
This is my second note
^D

# 4. Copiate file1.txt come file1_backup.txt
cp file1.txt file1_backup.txt

# 5. Rinominate file2.txt in renamed.txt
mv file2.txt renamed.txt

# 6. Create subdirectory backup/
mkdir backup

# 7. Copiate tutti i .txt in backup/
cp *.txt backup/

# 8. Verificate contenuto backup/
ls -l backup/

# 9. Rimuovete file3.txt con conferma
rm -i file3.txt
```

2.7.3 Esercizio 2.3: Ricerca File

```
# 1. Trovate tutti i file .conf in /etc
find /etc -name "*.conf" 2>/dev/null | head -20

# 2. Trovate file modificati nelle ultime 24 ore nella home
find ~ -type f -mtime 0

# 3. Trovate directory nella home
find ~ -maxdepth 2 -type d

# 4. Trovate file più grandi di 10MB
find ~ -type f -size +10M 2>/dev/null

# 5. Trovate file vuoti
find ~ -type f -empty
```

2.7.4 Esercizio 2.4: Pattern Matching con grep

```
# 1. Create file di test
cat > testfile.txt << 'EOF'
ERROR: Failed to connect
WARNING: Low memory
INFO: Service started
ERROR: Timeout occurred
DEBUG: Variable x = 10
INFO: User logged in
ERROR: Permission denied
EOF

# 2. Trovate tutte le righe con ERROR
```

```
grep ERROR testfile.txt

# 3. Trovate ERROR o WARNING
grep -E 'ERROR|WARNING' testfile.txt

# 4. Trovate righe che NON contengono INFO
grep -v INFO testfile.txt

# 5. Conta occorrenze di ERROR
grep -c ERROR testfile.txt

# 6. Mostra numero riga per ogni match
grep -n ERROR testfile.txt

# 7. Cerca case-insensitive
grep -i error testfile.txt
```

2.8 Best Practices

Sicurezza e Buone Abitudini

1. **Conferme per operazioni distruttive**

```
# Usa sempre -i con rm, cp, mv su file importanti
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

2. **Backup prima di modifiche importanti**

```
# Backup prima di modificare
cp important.conf important.conf.backup
```

3. **Test con echo prima di esecuzione**

```
# Test pattern prima di rm
find . -name "*.tmp"              # Verifica cosa verrà
    cancellato
find . -name "*.tmp" -delete      # Poi cancella
```

4. **Usa path assoluti in script**

```
# Evita ambiguità
/bin/rm file.txt                  # Non rm file.txt
```

5. **Quote sempre i path con spazi**

```
cp "file with spaces.txt" "/destination/path/"
```

2.9 Riepilogo

In questo capitolo abbiamo imparato:

- **Navigazione**: pwd, cd, ls con tutte le opzioni

- **Manipolazione**: mkdir, touch, cp, mv, rm
- **Visualizzazione**: cat, less, head, tail
- **Ricerca**: find con criteri multipli
- **Pattern matching**: grep e regular expressions
- **Documentazione**: man, info, –help

Questi comandi formano la base del lavoro quotidiano in Linux. Padroneggiandoli sarete in grado di navigare, gestire e cercare file efficacemente.

Prossimo Capitolo

Nel Capitolo 3 esploreremo in profondità il filesystem Linux: la sua struttura gerarchica, i permessi, ownership e i comandi per gestirli (chmod, chown, umask).

Capitolo 3

Filesystem e Permessi

3.1 Gerarchia del Filesystem Linux

3.1.1 Filesystem Hierarchy Standard (FHS)

Linux organizza tutti i file in un'unica gerarchia ad albero che parte dalla **root directory** (/). Ogni directory ha uno scopo specifico definito dal Filesystem Hierarchy Standard.

```
# Visualizza struttura radice
ls -l /

# Output tipico:
# drwxr-xr-x   2 root root  4096 /bin
# drwxr-xr-x   4 root root  4096 /boot
# drwxr-xr-x  20 root root  3840 /dev
# drwxr-xr-x 135 root root 12288 /etc
# drwxr-xr-x   3 root root  4096 /home
# drwxr-xr-x  14 root root  4096 /lib
# drwxr-xr-x   2 root root  4096 /mnt
# drwxr-xr-x   3 root root  4096 /opt
# drwxr-xr-x 261 root root     0 /proc
# drwx------   8 root root  4096 /root
# drwxr-xr-x  28 root root   880 /run
# drwxr-xr-x   2 root root 12288 /sbin
# drwxr-xr-x   2 root root  4096 /srv
# drwxr-xr-x  13 root root     0 /sys
# drwxrwxrwt  20 root root  4096 /tmp
# drwxr-xr-x  11 root root  4096 /usr
# drwxr-xr-x  14 root root  4096 /var
```

3.1.2 Directory Principali

/bin - Binari Essenziali

Comandi fondamentali disponibili a tutti gli utenti:

```
# Esplora /bin
ls /bin

# Comandi in /bin: ls, cp, mv, rm, cat, bash, echo, grep, etc.
# Necessari per boot e single-user mode

# Verifica dove si trova un comando
which ls
```

```
# Output: /bin/ls

which bash
# Output: /bin/bash
```

/sbin - Binari di Sistema

Comandi amministrativi (di solito richiedono root):

```
ls /sbin

# Comandi in /sbin: ifconfig, iptables, fdisk, mkfs, shutdown, etc.
# Usati per amministrazione sistema
```

/etc - Configurazione

File di configurazione del sistema:

```
# Configurazioni importanti
/etc/passwd          # Database utenti
/etc/group           # Database gruppi
/etc/shadow          # Password criptate (solo root)
/etc/fstab           # Filesystem da montare al boot
/etc/hosts           # Mapping hostname-IP locale
/etc/hostname        # Nome del sistema
/etc/ssh/            # Configurazione SSH
/etc/nginx/          # Configurazione Nginx
/etc/apache2/        # Configurazione Apache

# Esempi
cat /etc/hostname
cat /etc/hosts
ls -la /etc/ssh/
```

/home - Home Directory Utenti

Directory personali degli utenti:

```
# Ogni utente ha la sua directory
/home/user1/
/home/user2/
/home/alice/

# Variabile $HOME punta alla tua home
echo $HOME
# Output: /home/username

# Shortcut tilde (~)
cd ~               # Vai alla tua home
cd ~alice          # Vai alla home di alice (se hai permessi)
```

/root - Home dell'Amministratore

Home directory dell'utente root:

```
# Separata da /home per motivi di sicurezza
# Accessibile solo a root
sudo ls -la /root
```

/tmp - File Temporanei

Directory per file temporanei:

```
# Writable da tutti
# Solitamente pulita al reboot
# Spesso con sticky bit per sicurezza

ls -ld /tmp
# drwxrwxrwt  20 root root  4096 Nov 15 10:30 /tmp
#          ^
#                 sticky bit (t)

# Creazione file temporanei
mktemp
# Output: /tmp/tmp.xYz123aBc

# Directory temporanea
mktemp -d
# Output: /tmp/tmp.dir.xYz123aBc
```

/var - Dati Variabili

Dati che cambiano durante operazioni normali:

```
/var/log/           # File di log
/var/mail/          # Mail degli utenti
/var/spool/         # Code di stampa, cron, etc.
/var/tmp/           # File temporanei persistenti tra reboot
/var/www/           # Contenuti web server
/var/lib/           # Dati applicazioni

# Log importanti
/var/log/syslog     # Log di sistema (Debian/Ubuntu)
/var/log/messages   # Log di sistema (RHEL/CentOS)
/var/log/auth.log   # Log autenticazione
/var/log/kern.log   # Log kernel

# Visualizza log recenti
sudo tail -f /var/log/syslog
```

/usr - Unix System Resources

Programmi e librerie utente:

```
/usr/bin/           # Binari applicazioni utente
/usr/sbin/          # Binari amministrativi non essenziali
/usr/lib/           # Librerie
/usr/local/         # Software installato localmente
/usr/share/         # Dati condivisi (documentazione, icone)
/usr/include/       # Header file C/C++
/usr/src/           # Codice sorgente
```

```
8
9  # Differenza /bin vs /usr/bin:
10 # /bin:     comandi essenziali per boot
11 # /usr/bin: comandi applicativi aggiuntivi
```

/opt - Software Opzionale

Pacchetti software di terze parti:

```
1 # Software non gestito dal package manager
2 /opt/google/chrome/
3 /opt/teamviewer/
4 /opt/custom-app/
```

/proc - Processo Information Filesystem

Filesystem virtuale con informazioni su processi e sistema:

```
1  # Informazioni CPU
2  cat /proc/cpuinfo
3
4  # Informazioni memoria
5  cat /proc/meminfo
6
7  # Informazioni kernel
8  cat /proc/version
9
10 # Informazioni processo specifico
11 ls /proc/$$          # $$ = PID della shell corrente
12 cat /proc/$$/cmdline
13
14 # System limits
15 cat /proc/sys/fs/file-max
```

/sys - Informazioni Sistema

Filesystem virtuale per informazioni hardware e kernel:

```
1 # Informazioni dispositivi
2 ls /sys/class/
3 ls /sys/block/       # Dispositivi a blocchi
4
5 # Temperatura CPU (se disponibile)
6 cat /sys/class/thermal/thermal_zone0/temp
```

/dev - Device Files

File speciali che rappresentano dispositivi hardware:

```
1 # Dispositivi a blocchi (dischi)
2 /dev/sda            # Primo disco SATA
3 /dev/sda1           # Prima partizione
4 /dev/sdb            # Secondo disco
5
6 # Dispositivi a caratteri
7 /dev/tty            # Terminal
```

```
/dev/null           # Black hole (scarta tutto)
/dev/zero           # Sorgente infinita di zeri
/dev/random         # Generatore numeri casuali

# Esempi pratici
echo "test" > /dev/null    # Scarta output
dd if=/dev/zero of=file bs=1M count=100  # Crea file 100MB di zeri
```

/mnt e /media - Mount Points

```
# /mnt: mount manuali temporanei
# /media: mount automatici (USB, CD, etc.)

# Lista dispositivi montati
mount | column -t

# Oppure
df -h
```

Visualizzazione Grafica

```
# Installa tree per visualizzazione grafica
sudo apt install tree    # Debian/Ubuntu
sudo dnf install tree    # Fedora

# Visualizza struttura directory
tree -L 2 /etc           # Due livelli di profondità
tree -d /usr             # Solo directory
tree -h /var             # Con dimensioni human-readable
```

3.2 Permessi dei File

3.2.1 Il Modello di Permessi Unix

Ogni file e directory in Linux ha:

- Un **proprietario** (owner/user)
- Un **gruppo** (group)
- Tre set di **permessi**: owner, group, others

3.2.2 Tipi di Permessi

```
# Visualizza permessi
ls -l file.txt
# -rw-r--r-- 1 user group 1024 Nov 15 10:30 file.txt

# I tre tipi di permessi:
# r (read)    = 4  = lettura
# w (write)   = 2  = scrittura
# x (execute) = 1  = esecuzione

# Per FILE:
```

```
# r: leggere contenuto
# w: modificare contenuto
# x: eseguire come programma

# Per DIRECTORY:
# r: listare contenuto (ls)
# w: creare/cancellare file dentro
# x: entrare nella directory (cd)
```

3.2.3 Interpretazione Permessi

```
# -rw-r--r--
#
#                                > Altri (others): r-- (4)
#                                > Gruppo (group):  r-- (4)
#                                > Owner (user):    rw- (6)
#
# Primo carattere: tipo file
# -: file normale
# d: directory
# l: symbolic link
# b: block device
# c: character device
# p: named pipe (FIFO)
# s: socket

# Esempi
drwxr-xr-x  # Directory: 755
-rwxr-xr-x  # File eseguibile: 755
-rw-------  # File privato: 600
-rw-r--r--  # File normale: 644
lrwxrwxrwx  # Symbolic link (sempre 777)
```

3.2.4 chmod - Change Mode

Modifica i permessi di file e directory:

```
# METODO NUMERICO (ottale)
chmod 755 script.sh
# 7 = 4+2+1 = rwx (owner)
# 5 = 4+0+1 = r-x (group)
# 5 = 4+0+1 = r-x (others)

chmod 644 file.txt
# 6 = 4+2 = rw- (owner)
# 4 = 4   = r-- (group)
# 4 = 4   = r-- (others)

chmod 600 private.key
# 6 = rw- (owner)
# 0 = --- (group)
# 0 = --- (others)

# METODO SIMBOLICO
chmod u+x script.sh      # User: aggiungi execute
chmod g+w file.txt       # Group: aggiungi write
chmod o-r secret.txt     # Others: rimuovi read
```

```
chmod a+x program        # All: aggiungi execute

# Combinazioni
chmod u+x,g+x,o-rwx file
chmod ug+rw,o-rwx file
chmod a=r file.txt       # Imposta uguale per tutti

# Ricorsivo
chmod -R 755 directory/

# Permessi speciali comuni
chmod 700 ~/.ssh                    # Directory SSH privata
chmod 600 ~/.ssh/id_rsa             # Chiave privata SSH
chmod 644 ~/.ssh/id_rsa.pub         # Chiave pubblica SSH
chmod 644 ~/.ssh/authorized_keys    # Chiavi autorizzate
```

Permessi Comuni

Valore	Significato
777	rwxrwxrwx - Tutti i permessi (EVITARE!)
755	rwxr-xr-x - Eseguibili, directory pubbliche
700	rwx—— - Eseguibili/directory private
666	rw-rw-rw- - File writable da tutti (EVITARE!)
644	rw-r–r– - File normali leggibili
600	rw——- - File privati
444	r–r–r– - File read-only

3.2.5 Permessi Speciali

SetUID (SUID) - 4000

Esegue con permessi del proprietario del file:

```
# Visualizza file con SUID
find /usr/bin -perm -4000 -ls 2>/dev/null

# Esempio: passwd
ls -l /usr/bin/passwd
# -rwsr-xr-x 1 root root 68208 /usr/bin/passwd
#    ^
#           s invece di x = SUID bit

# passwd deve scrivere /etc/shadow (solo root)
# SUID permette di eseguirlo come root

# Impostare SUID
chmod u+s file
chmod 4755 file
```

SetGID (SGID) - 2000

Per file: esegue con permessi del gruppo. Per directory: file creati ereditano il gruppo della directory:

```
# Su directory: file creati ereditano gruppo
mkdir shared
```

```
chmod g+s shared
chmod 2775 shared

ls -ld shared
# drwxrwsr-x 2 user group 4096 shared
#       ^
#               s invece di x = SGID bit

# File creati in shared/ avranno gruppo "group"
```

Sticky Bit - 1000

Su directory: solo proprietario può cancellare i propri file:

```
# Esempio: /tmp
ls -ld /tmp
# drwxrwxrwt 20 root root 4096 /tmp
#          ^
#                 t invece di x = sticky bit

# Tutti possono creare file in /tmp
# Ma solo il proprietario può cancellare i propri file

# Impostare sticky bit
chmod +t directory
chmod 1777 directory
```

3.2.6 chown - Change Owner

Cambia proprietario e gruppo di file:

```
# Cambia solo owner
sudo chown newuser file.txt

# Cambia owner e group
sudo chown newuser:newgroup file.txt

# Cambia solo group
sudo chown :newgroup file.txt
# Oppure
sudo chgrp newgroup file.txt

# Ricorsivo
sudo chown -R user:group directory/

# Esempi pratici
sudo chown www-data:www-data /var/www/html
sudo chown -R $USER:$USER ~/projects

# Cambia owner usando reference file
sudo chown --reference=reference.txt file.txt

# Verbose
sudo chown -v user:group file.txt
# Output: changed ownership of 'file.txt' from root:root to user:group
```

Attenzione!

chown richiede privilegi root (sudo). Non potete dare via la proprietà di vostri file o prendere proprietà di file altrui senza essere root. Questo previene escalation di privilegi.

3.2.7 umask - Default Permissions

Il comando `umask` definisce i permessi di default per nuovi file:

```
# Visualizza umask corrente
umask
# Output: 0022

# Visualizza in formato simbolico
umask -S
# Output: u=rwx,g=rx,o=rx

# Come funziona umask:
# File:      666 (rw-rw-rw-)
# - umask:   022 (----w--w-)
# = result:  644 (rw-r--r--)

# Directory: 777 (rwxrwxrwx)
# - umask:   022 (----w--w-)
# = result:  755 (rwxr-xr-x)

# Imposta nuovo umask
umask 027
# File:      666 - 027 = 640 (rw-r-----)
# Directory: 777 - 027 = 750 (rwxr-x---)

# Umask comuni:
umask 022   # Default: file 644, dir 755
umask 002   # Gruppo può scrivere: file 664, dir 775
umask 077   # Privato: file 600, dir 700

# Test pratico
umask 022
touch test1.txt
mkdir test1dir
ls -ld test1*
# -rw-r--r-- test1.txt
# drwxr-xr-x test1dir

umask 077
touch test2.txt
mkdir test2dir
ls -ld test2*
# -rw------- test2.txt
# drwx------ test2dir

# Rendere permanente: aggiungi a ~/.bashrc
echo "umask 027" >> ~/.bashrc
```

3.3 Attributi Estesi

3.3.1 lsattr e chattr - Extended Attributes

Linux supporta attributi aggiuntivi oltre ai permessi standard:

```
# Visualizza attributi
lsattr file.txt
# --------------e------- file.txt

# Attributi importanti:
# i: immutable - non può essere modificato, cancellato, rinominato
# a: append-only - si può solo aggiungere in coda
# e: extent format (default su ext4)
# s: secure deletion - sovrascrive con zero alla cancellazione

# Rendi file immutabile (richiede root)
sudo chattr +i important.conf

# Ora nemmeno root può modificarlo/cancellarlo
sudo rm important.conf
# rm: cannot remove 'important.conf': Operation not permitted

# Rimuovi attributo immutable
sudo chattr -i important.conf

# Append-only (utile per log)
sudo chattr +a logfile.log

# File può solo crescere, non può essere troncato
echo "new line" >> logfile.log   # OK
echo "overwrite" > logfile.log   # ERRORE
```

3.4 ACL - Access Control Lists

Per permessi più granulari del modello standard:

```
# Verifica supporto ACL
mount | grep acl

# Visualizza ACL
getfacl file.txt

# Imposta ACL: permetti a user2 di leggere
setfacl -m u:user2:r file.txt

# Imposta ACL: permetti a group2 di scrivere
setfacl -m g:group2:rw file.txt

# Rimuovi ACL specifica
setfacl -x u:user2 file.txt

# Rimuovi tutte le ACL
setfacl -b file.txt

# ACL ricorsive su directory
setfacl -R -m u:user2:rx directory/

```

```
# ACL di default (ereditate da nuovi file)
setfacl -d -m g:developers:rwx /shared/project/

# Esempio completo
getfacl file.txt
# Output:
# # file: file.txt
# # owner: user1
# # group: group1
# user::rw-
# user:user2:r--
# group::r--
# group:group2:rw-
# mask::rw-
# other::r--
```

3.5 Esercizi Pratici

3.5.1 Esercizio 3.1: Esplorazione Filesystem

```
# 1. Esplora la struttura principale
ls -l /

# 2. Trova la tua distribuzione
cat /etc/os-release

# 3. Quanti file in /etc?
ls /etc | wc -l

# 4. Trova i 5 file più grandi in /var/log
sudo du -sh /var/log/* | sort -hr | head -5

# 5. Dispositivi montati
df -h

# 6. Informazioni CPU
cat /proc/cpuinfo | grep "model name" | head -1

# 7. Memoria totale
free -h

# 8. Processi attivi
ls /proc | grep "^[0-9]" | wc -l
```

3.5.2 Esercizio 3.2: Gestione Permessi

```
# 1. Create directory di lavoro
mkdir ~/permissions_test
cd ~/permissions_test

# 2. Create file con diversi permessi
touch public.txt private.txt executable.sh

# 3. Imposta permessi
chmod 644 public.txt          # rw-r--r--
```

```
chmod 600 private.txt        # rw-------
chmod 755 executable.sh      # rwxr-xr-x

# 4. Verifica
ls -l

# 5. Create directory con permessi
mkdir public_dir private_dir shared_dir
chmod 755 public_dir         # rwxr-xr-x
chmod 700 private_dir        # rwx------
chmod 775 shared_dir         # rwxrwxr-x

# 6. Test permessi
# Come puoi leggere/scrivere/eseguire?
cat public.txt               # OK
cat private.txt              # OK (sei owner)
./executable.sh              # OK se ha shebang

# 7. Imposta SGID su shared_dir
chmod g+s shared_dir

# 8. Verifica
ls -ld shared_dir
# drwxrwsr-x
```

3.5.3 Esercizio 3.3: Ownership

```
# 1. Verifica owner corrente
ls -l ~/permissions_test

# 2. Create file temporaneo come root
sudo touch /tmp/rootfile.txt
ls -l /tmp/rootfile.txt
# -rw-r--r-- 1 root root 0 /tmp/rootfile.txt

# 3. Cambia ownership a te stesso
sudo chown $USER:$USER /tmp/rootfile.txt
ls -l /tmp/rootfile.txt

# 4. Create struttura
mkdir -p ~/project/{src,docs,bin}
touch ~/project/src/main.c
touch ~/project/docs/README.md

# 5. Imposta ownership ricorsivo
sudo chown -R $USER:$USER ~/project

# 6. Verifica ricorsivamente
ls -lR ~/project
```

3.5.4 Esercizio 3.4: umask

```
# 1. Verifica umask corrente
umask
umask -S

```

```
# 2. Test con umask 022
umask 022
touch file_022.txt
mkdir dir_022
ls -l file_022.txt    # -rw-r--r--
ls -ld dir_022        # drwxr-xr-x

# 3. Test con umask 077
umask 077
touch file_077.txt
mkdir dir_077
ls -l file_077.txt    # -rw-------
ls -ld dir_077        # drwx------

# 4. Test con umask 002
umask 002
touch file_002.txt
mkdir dir_002
ls -l file_002.txt    # -rw-rw-r--
ls -ld dir_002        # drwxrwxr-x

# 5. Ripristina umask default
umask 022

# 6. Cleanup
rm file_* 2>/dev/null
rm -rf dir_* 2>/dev/null
```

3.5.5 Esercizio 3.5: Scenario Reale

Configurare un progetto web condiviso:

```
# 1. Create gruppo developers
sudo groupadd developers

# 2. Aggiungi utenti al gruppo
sudo usermod -aG developers $USER
# (necessario logout/login per applicare)

# 3. Create directory progetto
sudo mkdir -p /var/www/project
sudo chown root:developers /var/www/project

# 4. Imposta permessi con SGID
sudo chmod 2775 /var/www/project
# 2: SGID - nuovi file ereditano gruppo "developers"
# 775: rwxrwxr-x

# 5. Verifica
ls -ld /var/www/project
# drwxrwsr-x 2 root developers 4096 /var/www/project
#       ^
#               SGID bit

# 6. Test: crea file
touch /var/www/project/test.html
ls -l /var/www/project/test.html
# -rw-rw-r-- 1 yourusername developers 0 test.html
```

```
#                              ^
#                                    gruppo ereditato da SGID

# 7. Imposta umask per gruppo writable
echo "umask 002" >> ~/.bashrc
```

3.6 Best Practices

Sicurezza Permessi

1. **Principio del minimo privilegio**

```
# Dai solo i permessi necessari, non di più
chmod 600 ~/.ssh/id_rsa          # Chiave privata: solo owner
chmod 644 public_file.txt        # File pubblico: read per tutti
chmod 700 ~/bin/                 # Script personali: solo owner
```

2. **Mai 777 su file critici**

```
# MALE - troppo permissivo
chmod 777 config.php             # Chiunque può modificare!

# BENE - solo necessario
chmod 640 config.php             # Owner rw, group r
```

3. **Proteggi directory .ssh**

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
chmod 600 ~/.ssh/authorized_keys
```

4. **Usa gruppi per condivisione**

```
# Invece di permessi 777, usa gruppi + SGID
sudo groupadd projectteam
sudo chown -R :projectteam /shared/project
sudo chmod -R 2775 /shared/project
```

5. **Audit regolare permessi**

```
# Trova file world-writable (potenziale rischio)
find /home -type f -perm -002 2>/dev/null

# Trova file con SUID (potenziale rischio escalation)
find / -perm -4000 -type f 2>/dev/null

# Trova file senza owner (orphaned)
find / -nouser -o -nogroup 2>/dev/null
```

Errori Comuni da Evitare

1. **chmod -R 777**: mai su directory importanti
2. **chown senza backup**: sempre backup prima di cambiare ownership massivo

3. **Modificare /etc senza capire**: può rompere il sistema
4. **Dimenticare sticky bit su /tmp**: permette cancellazione file altrui
5. **SUID su script shell**: rischio sicurezza, molte shell lo ignorano

3.7 Riepilogo

In questo capitolo abbiamo esplorato:

- **Filesystem Hierarchy Standard**: organizzazione directory Linux
- **Directory principali**: /, /etc, /var, /home, /usr, /proc, /sys, /dev
- **Permessi**: lettura, scrittura, esecuzione per owner/group/others
- **chmod**: modifica permessi (numerico e simbolico)
- **chown/chgrp**: modifica ownership
- **umask**: permessi di default per nuovi file
- **Permessi speciali**: SUID, SGID, sticky bit
- **Attributi estesi**: chattr, lsattr
- **ACL**: permessi granulari

La comprensione del filesystem e dei permessi è fondamentale per la sicurezza e l'amministrazione di sistemi Linux.

Prossimo Capitolo

Nel Capitolo 4 impareremo a gestire i processi: monitorarli con ps e top, controllarli con kill, e gestire job in foreground e background.

Capitolo 4

Gestione dei Processi

4.1 Introduzione ai Processi

Un **processo** è un'istanza di un programma in esecuzione. Linux è un sistema multitasking che esegue centinaia di processi simultaneamente.

4.1.1 Concetti Fondamentali

- **PID (Process ID)**: identificatore numerico univoco
- **PPID (Parent PID)**: PID del processo genitore
- **UID/GID**: utente e gruppo che eseguono il processo
- **Stato**: running, sleeping, stopped, zombie
- **Priorità**: nice value (-20 a 19, minore = maggior priorità)

```
# Visualizza PID della shell corrente
echo $$
# Output: 1234 (esempio)

# Visualizza PID dell'ultimo processo in background
echo $!

# Il processo con PID 1 è sempre init/systemd
ps -p 1
# PID TTY      TIME CMD
#   1 ?    00:00:02 systemd
```

4.1.2 Gerarchia dei Processi

I processi in Linux formano un albero:

```
# Visualizza albero processi
pstree

# Output esempio:
#   s   y   s   t   e   m   d   NetworkManager2   *[{NetworkManager}]
#             accounts    - d a e m o n 2 *[{accounts-daemon}]
#             cron
#             dbus    -daemon
#
                            sshdsshdsshdbashpstree
```

```
#                   systemd         (sd-pam)
#                         pulseaudio

# Con PID
pstree -p

# Solo per utente specifico
pstree username

# Compatto
pstree -c
```

4.2 Visualizzazione Processi: ps

Il comando `ps` (process status) visualizza informazioni sui processi.

4.2.1 Sintassi Base

```
# Processi nella shell corrente
ps
# PID TTY      TIME CMD
# 1234 pts/0   00:00:00 bash
# 5678 pts/0   00:00:00 ps

# Tutti i processi dell'utente corrente
ps x
ps -x

# Tutti i processi di tutti gli utenti
ps aux
ps -ef

# Formato BSD (ps aux):
# USER   PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
# root     1  0.0  0.1 169868 11840 ?      Ss   10:00   0:02 /sbin/init

# Formato Unix (ps -ef):
# UID  PID  PPID  C STIME TTY      TIME CMD
# root   1     0  0 10:00 ?    00:00:02 /sbin/init
```

4.2.2 Opzioni Utili

```
# Tutti i processi, formato esteso
ps aux

# Tutti i processi, formato gerarchico
ps auxf

# Solo processi specifici
ps -p 1234
ps -p 1234,5678,9012

# Processi di un utente
ps -u username
ps -U username
```

```
14
15 # Processi associati a un terminale
16 ps -t pts/0
17
18 # Custom format
19 ps -eo pid,ppid,user,cmd
20 ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm
21
22 # Ordinamento
23 ps aux --sort=-%cpu        # Per CPU usage (decrescente)
24 ps aux --sort=-%mem        # Per memoria
25 ps aux --sort=-rss         # Per RSS (resident set size)
26
27 # Top 10 processi per CPU
28 ps aux --sort=-%cpu | head -11
29
30 # Top 10 per memoria
31 ps aux --sort=-%mem | head -11
32
33 # Cerca processo per nome
34 ps aux | grep firefox
35 ps -C firefox              # Metodo migliore
```

4.2.3 Comprensione Output ps aux

```
1  # USER   PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
2  # root     1  0.0  0.1 169868 11840 ?     Ss   10:00   0:02 /sbin/init
3
4  # USER:    proprietario processo
5  # PID:     process ID
6  # %CPU:    percentuale CPU
7  # %MEM:    percentuale memoria
8  # VSZ:     virtual memory size (KB)
9  # RSS:     resident set size - memoria fisica (KB)
10 # TTY:     terminal associato (? = nessuno)
11 # STAT:    stato processo
12 # START:   quando è stato avviato
13 # TIME:    tempo CPU consumato
14 # COMMAND: comando completo
15
16 # Codici STAT:
17 # R: running
18 # S: sleeping (interruptible)
19 # D: sleeping (uninterruptible, usually I/O)
20 # T: stopped
21 # Z: zombie
22 # <: high priority
23 # N: low priority
24 # L: has pages locked into memory
25 # s: is a session leader
26 # l: is multi-threaded
27 # +: is in the foreground process group
```

4.3 Monitoraggio Interattivo: top

`top` fornisce una vista dinamica dei processi in esecuzione.

4.3.1 Uso Base

```
# Avvia top
top

# Output:
# top - 14:25:33 up  4:25,  2 users,  load average: 0.23, 0.45, 0.38
# Tasks: 285 total,   1 running, 284 sleeping,   0 stopped,   0 zombie
# %Cpu(s):  2.3 us,  0.8 sy,  0.0 ni, 96.5 id,  0.3 wa,  0.0 hi,  0.1 si
# MiB Mem :  15891.2 total,   8234.5 free,   4123.8 used,   3532.9 buff/cache
# MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.  11234.5 avail Mem
#
#   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
#  1234 user      20   0 4567890 234567  89012 S   5.3   1.5   2:34.56 firefox
#  5678 user      20   0 1234567  98765  45678 S   2.0   0.6   0:45.23 chrome
```

4.3.2 Interpretazione Header

```
# Prima riga: uptime e load average
# up 4:25          = sistema acceso da 4 ore e 25 minuti
# 2 users          = 2 utenti loggati
# load average     = carico medio ultimi 1, 5, 15 minuti
#                    < 1.0 = OK su single-core
#                    < numero_core = OK su multi-core

# Tasks: numero processi per stato
# total, running, sleeping, stopped, zombie

# %Cpu(s): utilizzo CPU
# us = user space
# sy = system (kernel)
# ni = nice (low priority)
# id = idle
# wa = wait I/O
# hi = hardware interrupts
# si = software interrupts
# st = steal (virtualizzazione)

# Memoria:
# total = totale
# free = libera
# used = usata
# buff/cache = buffer e cache
# avail Mem = disponibile per nuove applicazioni
```

4.3.3 Comandi Interattivi in top

```
# Mentre top è in esecuzione:

# Ordinamento
P       # Ordina per %CPU (default)
```

```
M       # Ordina per %MEM
T       # Ordina per TIME+
N       # Ordina per PID

# Filtri
u       # Filtra per utente (chiede username)
k       # Kill processo (chiede PID)
r       # Renice processo (cambia priorità)

# Visualizzazione
1       # Toggle visualizzazione singoli core
t       # Toggle visualizzazione task/CPU bar
m       # Toggle visualizzazione memoria bar
c       # Toggle comando completo/basename
V       # Vista albero (forest)
i       # Nascondi processi idle

# Aggiornamento
s       # Cambia delay aggiornamento (secondi)
d       # Alias per s
Space   # Aggiorna immediatamente

# Altro
h o ?   # Help
q       # Quit
W       # Salva configurazione corrente
```

4.3.4 top con Opzioni

```
# Aggiorna ogni 2 secondi (default: 3)
top -d 2

# Mostra solo processi di utente specifico
top -u username

# Batch mode (utile per logging)
top -b -n 1 > top_snapshot.txt

# Top 10 processi per CPU
top -b -n 1 | head -17

# Evidenzia differenze (delay 1 secondo)
top -d 1 -b -n 2 | tail -20
```

4.3.5 Alternative a top

```
# htop - versione migliorata di top (installazione necessaria)
sudo apt install htop
htop
# - Interfaccia colorata
# - Mouse support
# - Scroll verticale/orizzontale
# - Kill/renice più facile
# - Visualizzazione albero integrata

# atop - advanced monitoring
```

```
sudo apt install atop
atop
# - Statistiche disco dettagliate
# - Network statistics
# - Logging automatico

# glances - monitoring multi-piattaforma
sudo apt install glances
glances
# - Overview sistema completo
# - Auto-adattivo (mostra info più rilevanti)
# - Modalità client-server
# - Export in vari formati
```

4.4 Gestione Processi: kill

Il comando `kill` invia segnali ai processi.

4.4.1 Segnali Comuni

```
# Lista tutti i segnali
kill -l

# Segnali più comuni:
#  1) SIGHUP   - Hangup (ricarica configurazione)
#  2) SIGINT   - Interrupt (Ctrl+C)
#  3) SIGQUIT  - Quit
#  9) SIGKILL  - Kill forzato (non può essere ignorato)
# 15) SIGTERM  - Terminate gracefully (default)
# 18) SIGCONT  - Continue se stopped
# 19) SIGSTOP  - Stop processo (non può essere ignorato)
# 20) SIGTSTP  - Stop (Ctrl+Z)

# Equivalenze
kill -1 PID  = kill -HUP PID  = kill -SIGHUP PID
kill -9 PID  = kill -KILL PID = kill -SIGKILL PID
kill -15 PID = kill -TERM PID = kill -SIGTERM PID = kill PID
```

4.4.2 Uso di kill

```
# Termina processo gracefully (default: SIGTERM)
kill 1234

# Kill forzato (se SIGTERM non funziona)
kill -9 1234
kill -KILL 1234
kill -SIGKILL 1234

# Ricarica configurazione (molti daemon supportano SIGHUP)
sudo kill -HUP $(cat /var/run/nginx.pid)

# Stop processo (pausa)
kill -STOP 1234

# Resume processo
```

```
kill -CONT 1234

# Kill multipli processi
kill 1234 5678 9012

# Kill tutti i processi di un comando
killall firefox
killall -9 firefox

# Kill per nome con pattern matching
pkill firefox
pkill -9 firefox

# Kill per utente
pkill -u username

# Kill con pattern più complesso
pgrep -f "python.*server.py"     # Prima trova PID
pkill -f "python.*server.py"     # Poi killa

# Chiedi conferma prima di kill
pkill -i firefox
```

SIGTERM vs SIGKILL

Sempre provare SIGTERM prima di SIGKILL:

```
# BENE - processo può pulire e chiudere correttamente
kill 1234          # SIGTERM
sleep 5            # Aspetta
kill -9 1234       # SIGKILL solo se necessario

# MALE - kill forzato immediato
kill -9 1234       # Può lasciare file corrotti, lock, etc.
```

SIGTERM permette al processo di:

- Salvare dati
- Chiudere file aperti
- Rilasciare risorse
- Terminare connessioni correttamente

SIGKILL termina immediatamente senza cleanup.

4.4.3 killall e pkill

```
# killall - kill per nome esatto
killall firefox                  # Tutti i processi "firefox"
killall -u username process      # Solo di utente specifico
killall -i process               # Chiedi conferma
killall -w process               # Aspetta fino a terminazione
killall -v process               # Verbose

# pkill - kill per pattern
pkill firefox                    # Match substring
```

```
pkill -f "python script.py"          # Match full command line
pkill -u username                    # Tutti i processi di username
pkill -t pts/0                       # Tutti in terminal pts/0
pkill -x firefox                     # Exact match

# pgrep - trova PID (non killa)
pgrep firefox                        # Lista PID
pgrep -l firefox                     # Con nome
pgrep -a firefox                     # Con full command
pgrep -u username firefox            # Filtra per utente
```

4.5 Job Control

La shell bash permette di gestire job in foreground e background.

4.5.1 Concetti Base

```
# Foreground: processo che controlla il terminal
# - Riceve input da tastiera
# - Blocca il prompt
# - Ctrl+C per interrompere
# - Ctrl+Z per sospendere

# Background: processo che gira senza bloccare terminal
# - Non riceve input da tastiera
# - Prompt disponibile per altri comandi
# - & alla fine del comando
```

4.5.2 Esecuzione in Background

```
# Esegui in background
sleep 100 &
# [1] 12345
# [job_number] PID

# Multipli job in background
sleep 100 &
sleep 200 &
sleep 300 &

# Comando lungo in background
find / -name "*.log" > results.txt 2>&1 &

# Previeni terminazione alla chiusura shell
nohup command &
nohup ./long_script.sh &
# Output rediretto a nohup.out

# Background con priorità bassa
nice -n 19 command &
```

4.5.3 jobs - Lista Job Correnti

```
# Lista job della shell corrente
jobs
# [1]   Running     sleep 100 &
# [2]-  Running     sleep 200 &
# [3]+  Stopped     vim file.txt

# Con PID
jobs -l
# [1]  12345 Running     sleep 100 &
# [2]  12346 Running     sleep 200 &
# [3]  12347 Stopped     vim file.txt

# Solo running jobs
jobs -r

# Solo stopped jobs
jobs -s

# Simboli:
# +  = job corrente (default per fg/bg senza argomenti)
# -  = job precedente
```

4.5.4 fg - Foreground

```
# Porta job in foreground
fg                  # Porta job corrente (+)
fg %1               # Porta job numero 1
fg %2               # Porta job numero 2

# Alternative notation
fg %%               # Job corrente
fg %-               # Job precedente
fg %?string         # Job il cui comando contiene string
fg %command         # Job il cui comando inizia con command

# Esempio
sleep 100 &
# [1] 12345

fg %1
# sleep 100 (ora in foreground)
# Ctrl+C per terminare
```

4.5.5 bg - Background

```
# Scenario: processo in foreground che vogliamo mettere in background

# 1. Avvia comando
sleep 100

# 2. Sospendi con Ctrl+Z
^Z
# [1]+  Stopped    sleep 100

# 3. Riprendi in background
```

```
bg
# [1]+ sleep 100 &

# Oppure job specifico
bg %1
bg %2

# Esempio completo
vim file.txt        # Editing file
# Ctrl+Z            # Sospendi vim
# [1]+  Stopped     vim file.txt
bg %1               # Continua vim in background (NON ha senso per vim!)
fg %1               # Torna a vim

# Esempio sensato
find / -name "*.log" 2>/dev/null
# Operazione lenta...
# Ctrl+Z
# [1]+  Stopped     find / -name "*.log" 2>/dev/null
bg
# [1]+ find / -name "*.log" 2>/dev/null &
# Ora continua in background
```

4.5.6 disown - Scollega Job dalla Shell

```
# Job normalmente terminano quando chiudi la shell
# disown li rende indipendenti

# Avvia job
sleep 1000 &
# [1] 12345

# Rimuovi da job list
disown %1

# Ora jobs non lo mostra più
jobs

# Il processo continua anche dopo logout
# Ma non potrai più controllarlo con fg/bg

# Disown tutti i job
disown -a

# Disown ma mantieni in jobs (non riceve SIGHUP)
disown -h %1
```

4.5.7 nohup - No Hangup

```
# nohup previene terminazione alla chiusura terminal

# Sintassi base
nohup command &

# Output rediretto automaticamente a nohup.out
nohup ./long_script.sh &
```

```
# Output: nohup: ignoring input and appending output to 'nohup.out'

# Specifica file output
nohup ./script.sh > output.log 2>&1 &

# Verifica output
tail -f nohup.out

# Differenza nohup vs disown:
# nohup: deve essere usato all'avvio del comando
# disown: può essere usato dopo che il comando è già avviato

# Combinazione (massima protezione)
nohup ./critical_script.sh &
disown
```

4.6 Priorità dei Processi

4.6.1 nice - Avvia con Priorità

```
# Nice value: -20 (massima priorità) a 19 (minima priorità)
# Default: 0
# Utenti normali: possono solo diminuire priorità (0-19)
# Root: può impostare qualsiasi valore

# Visualizza nice value corrente
nice
# 0

# Avvia con priorità bassa
nice -n 10 command
nice -10 command          # Sintassi alternativa

# Massima priorità (richiede root)
sudo nice -n -20 important_process

# Esempi pratici

# Backup notturno (bassa priorità, non rallenta sistema)
nice -n 19 tar czf backup.tar.gz /home

# Compilazione (priorità normale-bassa)
nice -n 10 make -j4

# Processo critico (alta priorità, root)
sudo nice -n -10 critical_daemon
```

4.6.2 renice - Cambia Priorità

```
# Cambia nice value di processo running

# Per PID
renice -n 10 -p 1234
renice 10 1234              # Sintassi breve

```

```
# Per tutti i processi di un utente
renice -n 5 -u username

# Per tutti i processi di un gruppo
renice -n 5 -g groupname

# Diminuisci priorità (aumenta nice)
renice +5 -p 1234

# Aumenta priorità (diminuisci nice) - richiede root
sudo renice -5 -p 1234

# Esempio: rallenta processo che consuma troppa CPU
# 1. Identifica processo
top
# PID 1234 sta usando 99% CPU

# 2. Riduci priorità
renice 19 -p 1234

# Verifica
ps -o pid,ni,cmd -p 1234
# PID  NI CMD
# 1234 19 ./cpu_intensive_process
```

4.7 Esercizi Pratici

4.7.1 Esercizio 4.1: Esplorazione Processi

```
# 1. Visualizza tutti i tuoi processi
ps x

# 2. Trova processi più pesanti
ps aux --sort=-%mem | head -10
ps aux --sort=-%cpu | head -10

# 3. Trova processi del tuo utente
ps -u $USER

# 4. Albero processi completo
pstree -p $USER

# 5. Informazioni su processo specifico (es. bash)
ps -C bash
ps -C bash -o pid,ppid,user,cmd,%cpu,%mem

# 6. Processi zombie (spero nessuno!)
ps aux | grep 'Z'
```

4.7.2 Esercizio 4.2: Monitoring con top

```
# 1. Avvia top
top

# 2. Mentre in top, prova:
```

```
# - Premi '1' per vedere singoli core
# - Premi 'P' per ordinare per CPU
# - Premi 'M' per ordinare per memoria
# - Premi 'u' e inserisci il tuo username
# - Premi 'c' per vedere comando completo
# - Premi 'i' per nascondere idle processes

# 3. Trova processo più pesante
# - Ordina per CPU (P)
# - Annota PID e nome

# 4. Cattura snapshot
top -b -n 1 > top_snapshot_$(date +%Y%m%d_%H%M%S).txt

# 5. Monitor specifico utente
top -u $USER
```

4.7.3 Esercizio 4.3: Job Control

```
# 1. Avvia processo in background
sleep 300 &
# Annota job number e PID

# 2. Avvia altro processo in background
sleep 400 &

# 3. Lista job
jobs
jobs -l

# 4. Avvia processo in foreground
sleep 500

# 5. Sospendi con Ctrl+Z
# Premi Ctrl+Z

# 6. Lista job (dovresti vedere 3 job)
jobs

# 7. Riprendi ultimo in background
bg

# 8. Porta primo job in foreground
fg %1

# 9. Termina con Ctrl+C

# 10. Verifica job rimanenti
jobs

# 11. Kill tutti i job rimanenti
kill %1 %2
# Oppure
jobs -p | xargs kill
```

4.7.4 Esercizio 4.4: nohup e Priorità

```
# 1. Create script di test
cat > long_task.sh << 'EOF'
#!/bin/bash
for i in {1..100}; do
    echo "Step $i of 100"
    sleep 2
done
echo "Completed!"
EOF

chmod +x long_task.sh

# 2. Esegui con nohup
nohup ./long_task.sh &

# 3. Verifica output
tail -f nohup.out
# Ctrl+C per uscire

# 4. Verifica che processo esiste
jobs -l
ps aux | grep long_task

# 5. Disown il job
disown

# 6. Avvia altro processo con bassa priorità
nice -n 15 ./long_task.sh &
NICE_PID=$!

# 7. Verifica nice value
ps -o pid,ni,cmd -p $NICE_PID

# 8. Cambia priorità
renice 5 -p $NICE_PID

# 9. Verifica cambio
ps -o pid,ni,cmd -p $NICE_PID

# 10. Cleanup
pkill -f long_task.sh
```

4.7.5 Esercizio 4.5: Scenario Reale

Simulazione: processo impazzito che consuma troppa CPU

```
# 1. Create processo CPU-intensive
cat > cpu_hog.sh << 'EOF'
#!/bin/bash
# CPU intensive loop
while true; do
    x=$((x + 1))
done
EOF

chmod +x cpu_hog.sh
```

```
11
12 # 2. Avvia processo
13 ./cpu_hog.sh &
14 HOG_PID=$!
15 echo "Started CPU hog with PID: $HOG_PID"
16
17 # 3. Monitor in top (altro terminal)
18 top -p $HOG_PID
19
20 # 4. Riduci priorità per limitare impatto
21 renice 19 -p $HOG_PID
22
23 # 5. Verifica riduzione impatto in top
24
25 # 6. Se ancora problematico, sospendi
26 kill -STOP $HOG_PID
27
28 # 7. Verifica stato
29 ps -o pid,stat,cmd -p $HOG_PID
30 # STAT dovrebbe essere T (stopped)
31
32 # 8. Riprendi
33 kill -CONT $HOG_PID
34
35 # 9. Termina gracefully
36 kill $HOG_PID
37
38 # 10. Se non termina, forza
39 sleep 2
40 kill -9 $HOG_PID 2>/dev/null
41
42 # 11. Verifica terminazione
43 ps -p $HOG_PID
44 # Dovrebbe dare errore (processo non esiste)
```

4.8 Best Practices

1. **Sempre SIGTERM prima di SIGKILL**

```
1 # Dai al processo tempo di chiudere correttamente
2 kill $PID
3 sleep 5
4 kill -0 $PID 2>/dev/null && kill -9 $PID
```

2. **Usa nohup per processi lunghi**

```
1 # Previeni terminazione accidentale
2 nohup ./long_running_task.sh > task.log 2>&1 &
```

3. **Monitora risorse regolarmente**

```
1 # Check giornaliero top consumers
2 ps aux --sort=-%cpu | head -10
3 ps aux --sort=-%mem | head -10
```

4. **Usa nice per task non critici**

```
# Backup, compilazione, compressione
nice -n 15 tar czf backup.tar.gz /data
```

5. **Cleanup job periodicamente**

```
# Verifica job dimenticati
jobs
# Kill quelli non necessari
```

4.9 Comandi Utili Rapidi

```
# Trova e killa processo per nome
pkill -f "process_name"

# Trova processi che usano file specifico
lsof /path/to/file
fuser /path/to/file

# Trova processi che usano porta
sudo lsof -i :8080
sudo netstat -tulpn | grep :8080

# Monitor continuo top 5 CPU users
watch -n 2 "ps aux --sort=-%cpu | head -6"

# Conta processi per utente
ps aux | awk '{print $1}' | sort | uniq -c | sort -rn

# Kill tutti i processi di uno script
pkill -f script_name.sh

# Verifica se processo è ancora running
kill -0 $PID && echo "Running" || echo "Not running"

# CPU usage totale per comando
ps aux | grep [p]rocess_name | awk '{sum+=$3} END {print sum "%"}'
```

4.10 Riepilogo

In questo capitolo abbiamo imparato:

- **Concetti**: PID, PPID, stati processi, gerarchia
- **ps**: visualizzare processi con varie opzioni
- **top/htop**: monitoring interattivo
- **kill**: inviare segnali (SIGTERM, SIGKILL, etc.)
- **killall/pkill**: terminare processi per nome

- **jobs**: gestire job della shell
- **fg/bg**: spostare processi tra foreground e background
- **nohup/disown**: processi persistenti
- **nice/renice**: gestire priorità

La gestione efficace dei processi è essenziale per amministrare sistemi Linux e ottimizzare le performance.

Prossimo Capitolo

Nel Capitolo 5 entreremo nel mondo del Bash Scripting: variabili, strutture di controllo, loop, funzioni e parametri per automatizzare task complessi.

Capitolo 5

Gestione dei Processi

5.1 Introduzione ai Processi

Un **processo** è un'istanza di un programma in esecuzione. Linux è un sistema multitasking che esegue centinaia di processi simultaneamente.

5.1.1 Concetti Fondamentali

- **PID (Process ID)**: identificatore numerico univoco
- **PPID (Parent PID)**: PID del processo genitore
- **UID/GID**: utente e gruppo che eseguono il processo
- **Stato**: running, sleeping, stopped, zombie
- **Priorità**: nice value (-20 a 19, minore = maggior priorità)

```
# Visualizza PID della shell corrente
echo $$
# Output: 1234 (esempio)

# Visualizza PID dell'ultimo processo in background
echo $!

# Il processo con PID 1 è sempre init/systemd
ps -p 1
# PID TTY      TIME CMD
#   1 ?    00:00:02 systemd
```

5.1.2 Gerarchia dei Processi

I processi in Linux formano un albero:

```
# Visualizza albero processi
pstree

# Output esempio:
#   s y s t e m d  NetworkManager2  *[{NetworkManager}]
#             accounts    - d a e m o n 2 *[{accounts-daemon}]
#             cron
#             dbus    -daemon
#
                            sshdsshdsshdbashpstree
```

```
#                   systemd         (sd-pam)
#                         pulseaudio

# Con PID
pstree -p

# Solo per utente specifico
pstree username

# Compatto
pstree -c
```

5.2 Visualizzazione Processi: ps

Il comando ps (process status) visualizza informazioni sui processi.

5.2.1 Sintassi Base

```
# Processi nella shell corrente
ps
# PID TTY      TIME CMD
# 1234 pts/0   00:00:00 bash
# 5678 pts/0   00:00:00 ps

# Tutti i processi dell'utente corrente
ps x
ps -x

# Tutti i processi di tutti gli utenti
ps aux
ps -ef

# Formato BSD (ps aux):
# USER   PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
# root     1  0.0  0.1 169868 11840 ?      Ss   10:00   0:02 /sbin/init

# Formato Unix (ps -ef):
# UID  PID  PPID  C STIME TTY      TIME CMD
# root   1     0  0 10:00 ?    00:00:02 /sbin/init
```

5.2.2 Opzioni Utili

```
# Tutti i processi, formato esteso
ps aux

# Tutti i processi, formato gerarchico
ps auxf

# Solo processi specifici
ps -p 1234
ps -p 1234,5678,9012

# Processi di un utente
ps -u username
ps -U username
```

```
14
15 # Processi associati a un terminale
16 ps -t pts/0
17
18 # Custom format
19 ps -eo pid,ppid,user,cmd
20 ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm
21
22 # Ordinamento
23 ps aux --sort=-%cpu        # Per CPU usage (decrescente)
24 ps aux --sort=-%mem        # Per memoria
25 ps aux --sort=-rss         # Per RSS (resident set size)
26
27 # Top 10 processi per CPU
28 ps aux --sort=-%cpu | head -11
29
30 # Top 10 per memoria
31 ps aux --sort=-%mem | head -11
32
33 # Cerca processo per nome
34 ps aux | grep firefox
35 ps -C firefox              # Metodo migliore
```

5.2.3 Comprensione Output ps aux

```
1  # USER   PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMAND
2  # root     1  0.0  0.1 169868 11840 ?     Ss   10:00   0:02 /sbin/init
3
4  # USER:    proprietario processo
5  # PID:     process ID
6  # %CPU:    percentuale CPU
7  # %MEM:    percentuale memoria
8  # VSZ:     virtual memory size (KB)
9  # RSS:     resident set size - memoria fisica (KB)
10 # TTY:     terminal associato (? = nessuno)
11 # STAT:    stato processo
12 # START:   quando è stato avviato
13 # TIME:    tempo CPU consumato
14 # COMMAND: comando completo
15
16 # Codici STAT:
17 # R: running
18 # S: sleeping (interruptible)
19 # D: sleeping (uninterruptible, usually I/O)
20 # T: stopped
21 # Z: zombie
22 # <: high priority
23 # N: low priority
24 # L: has pages locked into memory
25 # s: is a session leader
26 # l: is multi-threaded
27 # +: is in the foreground process group
```

5.3 Monitoraggio Interattivo: top

`top` fornisce una vista dinamica dei processi in esecuzione.

5.3.1 Uso Base

```
# Avvia top
top

# Output:
# top - 14:25:33 up  4:25,  2 users,  load average: 0.23, 0.45, 0.38
# Tasks: 285 total,   1 running, 284 sleeping,   0 stopped,   0 zombie
# %Cpu(s):  2.3 us,  0.8 sy,  0.0 ni, 96.5 id,  0.3 wa,  0.0 hi,  0.1 si
# MiB Mem :  15891.2 total,   8234.5 free,   4123.8 used,   3532.9 buff/cache
# MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.  11234.5 avail Mem
#
#   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
#  1234 user      20   0 4567890 234567  89012 S   5.3   1.5   2:34.56 firefox
#  5678 user      20   0 1234567  98765  45678 S   2.0   0.6   0:45.23 chrome
```

5.3.2 Interpretazione Header

```
# Prima riga: uptime e load average
# up 4:25          = sistema acceso da 4 ore e 25 minuti
# 2 users          = 2 utenti loggati
# load average     = carico medio ultimi 1, 5, 15 minuti
#                    < 1.0 = OK su single-core
#                    < numero_core = OK su multi-core

# Tasks: numero processi per stato
# total, running, sleeping, stopped, zombie

# %Cpu(s): utilizzo CPU
# us = user space
# sy = system (kernel)
# ni = nice (low priority)
# id = idle
# wa = wait I/O
# hi = hardware interrupts
# si = software interrupts
# st = steal (virtualizzazione)

# Memoria:
# total = totale
# free = libera
# used = usata
# buff/cache = buffer e cache
# avail Mem = disponibile per nuove applicazioni
```

5.3.3 Comandi Interattivi in top

```
# Mentre top è in esecuzione:

# Ordinamento
P       # Ordina per %CPU (default)
```

```
M       # Ordina per %MEM
T       # Ordina per TIME+
N       # Ordina per PID

# Filtri
u       # Filtra per utente (chiede username)
k       # Kill processo (chiede PID)
r       # Renice processo (cambia priorità)

# Visualizzazione
1       # Toggle visualizzazione singoli core
t       # Toggle visualizzazione task/CPU bar
m       # Toggle visualizzazione memoria bar
c       # Toggle comando completo/basename
V       # Vista albero (forest)
i       # Nascondi processi idle

# Aggiornamento
s       # Cambia delay aggiornamento (secondi)
d       # Alias per s
Space   # Aggiorna immediatamente

# Altro
h o ?   # Help
q       # Quit
W       # Salva configurazione corrente
```

5.3.4 top con Opzioni

```
# Aggiorna ogni 2 secondi (default: 3)
top -d 2

# Mostra solo processi di utente specifico
top -u username

# Batch mode (utile per logging)
top -b -n 1 > top_snapshot.txt

# Top 10 processi per CPU
top -b -n 1 | head -17

# Evidenzia differenze (delay 1 secondo)
top -d 1 -b -n 2 | tail -20
```

5.3.5 Alternative a top

```
# htop - versione migliorata di top (installazione necessaria)
sudo apt install htop
htop
# - Interfaccia colorata
# - Mouse support
# - Scroll verticale/orizzontale
# - Kill/renice più facile
# - Visualizzazione albero integrata

# atop - advanced monitoring
```

```
sudo apt install atop
atop
# - Statistiche disco dettagliate
# - Network statistics
# - Logging automatico

# glances - monitoring multi-piattaforma
sudo apt install glances
glances
# - Overview sistema completo
# - Auto-adattivo (mostra info più rilevanti)
# - Modalità client-server
# - Export in vari formati
```

5.4 Gestione Processi: kill

Il comando `kill` invia segnali ai processi.

5.4.1 Segnali Comuni

```
# Lista tutti i segnali
kill -l

# Segnali più comuni:
#  1) SIGHUP   - Hangup (ricarica configurazione)
#  2) SIGINT   - Interrupt (Ctrl+C)
#  3) SIGQUIT  - Quit
#  9) SIGKILL  - Kill forzato (non può essere ignorato)
# 15) SIGTERM  - Terminate gracefully (default)
# 18) SIGCONT  - Continue se stopped
# 19) SIGSTOP  - Stop processo (non può essere ignorato)
# 20) SIGTSTP  - Stop (Ctrl+Z)

# Equivalenze
kill -1 PID  = kill -HUP PID  = kill -SIGHUP PID
kill -9 PID  = kill -KILL PID = kill -SIGKILL PID
kill -15 PID = kill -TERM PID = kill -SIGTERM PID = kill PID
```

5.4.2 Uso di kill

```
# Termina processo gracefully (default: SIGTERM)
kill 1234

# Kill forzato (se SIGTERM non funziona)
kill -9 1234
kill -KILL 1234
kill -SIGKILL 1234

# Ricarica configurazione (molti daemon supportano SIGHUP)
sudo kill -HUP $(cat /var/run/nginx.pid)

# Stop processo (pausa)
kill -STOP 1234

# Resume processo
```

```
kill -CONT 1234

# Kill multipli processi
kill 1234 5678 9012

# Kill tutti i processi di un comando
killall firefox
killall -9 firefox

# Kill per nome con pattern matching
pkill firefox
pkill -9 firefox

# Kill per utente
pkill -u username

# Kill con pattern più complesso
pgrep -f "python.*server.py"     # Prima trova PID
pkill -f "python.*server.py"     # Poi killa

# Chiedi conferma prima di kill
pkill -i firefox
```

SIGTERM vs SIGKILL

Sempre provare SIGTERM prima di SIGKILL:

```
# BENE - processo può pulire e chiudere correttamente
kill 1234          # SIGTERM
sleep 5            # Aspetta
kill -9 1234       # SIGKILL solo se necessario

# MALE - kill forzato immediato
kill -9 1234       # Può lasciare file corrotti, lock, etc.
```

SIGTERM permette al processo di:

- Salvare dati
- Chiudere file aperti
- Rilasciare risorse
- Terminare connessioni correttamente

SIGKILL termina immediatamente senza cleanup.

5.4.3 killall e pkill

```
# killall - kill per nome esatto
killall firefox                  # Tutti i processi "firefox"
killall -u username process      # Solo di utente specifico
killall -i process               # Chiedi conferma
killall -w process               # Aspetta fino a terminazione
killall -v process               # Verbose

# pkill - kill per pattern
pkill firefox                    # Match substring
```

```
pkill -f "python script.py"          # Match full command line
pkill -u username                    # Tutti i processi di username
pkill -t pts/0                       # Tutti in terminal pts/0
pkill -x firefox                     # Exact match

# pgrep - trova PID (non killa)
pgrep firefox                        # Lista PID
pgrep -l firefox                     # Con nome
pgrep -a firefox                     # Con full command
pgrep -u username firefox            # Filtra per utente
```

5.5 Job Control

La shell bash permette di gestire job in foreground e background.

5.5.1 Concetti Base

```
# Foreground: processo che controlla il terminal
# - Riceve input da tastiera
# - Blocca il prompt
# - Ctrl+C per interrompere
# - Ctrl+Z per sospendere

# Background: processo che gira senza bloccare terminal
# - Non riceve input da tastiera
# - Prompt disponibile per altri comandi
# - & alla fine del comando
```

5.5.2 Esecuzione in Background

```
# Esegui in background
sleep 100 &
# [1] 12345
# [job_number] PID

# Multipli job in background
sleep 100 &
sleep 200 &
sleep 300 &

# Comando lungo in background
find / -name "*.log" > results.txt 2>&1 &

# Previeni terminazione alla chiusura shell
nohup command &
nohup ./long_script.sh &
# Output rediretto a nohup.out

# Background con priorità bassa
nice -n 19 command &
```

5.5.3 jobs - Lista Job Correnti

```
# Lista job della shell corrente
jobs
# [1]   Running     sleep 100 &
# [2]-  Running     sleep 200 &
# [3]+  Stopped     vim file.txt

# Con PID
jobs -l
# [1]  12345 Running     sleep 100 &
# [2]  12346 Running     sleep 200 &
# [3]  12347 Stopped     vim file.txt

# Solo running jobs
jobs -r

# Solo stopped jobs
jobs -s

# Simboli:
# +  = job corrente (default per fg/bg senza argomenti)
# -  = job precedente
```

5.5.4 fg - Foreground

```
# Porta job in foreground
fg                  # Porta job corrente (+)
fg %1               # Porta job numero 1
fg %2               # Porta job numero 2

# Alternative notation
fg %%               # Job corrente
fg %-               # Job precedente
fg %?string         # Job il cui comando contiene string
fg %command         # Job il cui comando inizia con command

# Esempio
sleep 100 &
# [1] 12345

fg %1
# sleep 100 (ora in foreground)
# Ctrl+C per terminare
```

5.5.5 bg - Background

```
# Scenario: processo in foreground che vogliamo mettere in background

# 1. Avvia comando
sleep 100

# 2. Sospendi con Ctrl+Z
^Z
# [1]+  Stopped    sleep 100

# 3. Riprendi in background
```

```
bg
# [1]+ sleep 100 &

# Oppure job specifico
bg %1
bg %2

# Esempio completo
vim file.txt        # Editing file
# Ctrl+Z            # Sospendi vim
# [1]+  Stopped     vim file.txt
bg %1               # Continua vim in background (NON ha senso per vim!)
fg %1               # Torna a vim

# Esempio sensato
find / -name "*.log" 2>/dev/null
# Operazione lenta...
# Ctrl+Z
# [1]+  Stopped     find / -name "*.log" 2>/dev/null
bg
# [1]+ find / -name "*.log" 2>/dev/null &
# Ora continua in background
```

5.5.6 disown - Scollega Job dalla Shell

```
# Job normalmente terminano quando chiudi la shell
# disown li rende indipendenti

# Avvia job
sleep 1000 &
# [1] 12345

# Rimuovi da job list
disown %1

# Ora jobs non lo mostra più
jobs

# Il processo continua anche dopo logout
# Ma non potrai più controllarlo con fg/bg

# Disown tutti i job
disown -a

# Disown ma mantieni in jobs (non riceve SIGHUP)
disown -h %1
```

5.5.7 nohup - No Hangup

```
# nohup previene terminazione alla chiusura terminal

# Sintassi base
nohup command &

# Output rediretto automaticamente a nohup.out
nohup ./long_script.sh &
```

```
# Output: nohup: ignoring input and appending output to 'nohup.out'

# Specifica file output
nohup ./script.sh > output.log 2>&1 &

# Verifica output
tail -f nohup.out

# Differenza nohup vs disown:
# nohup: deve essere usato all'avvio del comando
# disown: può essere usato dopo che il comando è già avviato

# Combinazione (massima protezione)
nohup ./critical_script.sh &
disown
```

5.6 Priorità dei Processi

5.6.1 nice - Avvia con Priorità

```
# Nice value: -20 (massima priorità) a 19 (minima priorità)
# Default: 0
# Utenti normali: possono solo diminuire priorità (0-19)
# Root: può impostare qualsiasi valore

# Visualizza nice value corrente
nice
# 0

# Avvia con priorità bassa
nice -n 10 command
nice -10 command          # Sintassi alternativa

# Massima priorità (richiede root)
sudo nice -n -20 important_process

# Esempi pratici

# Backup notturno (bassa priorità, non rallenta sistema)
nice -n 19 tar czf backup.tar.gz /home

# Compilazione (priorità normale-bassa)
nice -n 10 make -j4

# Processo critico (alta priorità, root)
sudo nice -n -10 critical_daemon
```

5.6.2 renice - Cambia Priorità

```
# Cambia nice value di processo running

# Per PID
renice -n 10 -p 1234
renice 10 1234              # Sintassi breve

```

```
# Per tutti i processi di un utente
renice -n 5 -u username

# Per tutti i processi di un gruppo
renice -n 5 -g groupname

# Diminuisci priorità (aumenta nice)
renice +5 -p 1234

# Aumenta priorità (diminuisci nice) - richiede root
sudo renice -5 -p 1234

# Esempio: rallenta processo che consuma troppa CPU
# 1. Identifica processo
top
# PID 1234 sta usando 99% CPU

# 2. Riduci priorità
renice 19 -p 1234

# Verifica
ps -o pid,ni,cmd -p 1234
# PID  NI CMD
# 1234 19 ./cpu_intensive_process
```

5.7 Esercizi Pratici

5.7.1 Esercizio 4.1: Esplorazione Processi

```
# 1. Visualizza tutti i tuoi processi
ps x

# 2. Trova processi più pesanti
ps aux --sort=-%mem | head -10
ps aux --sort=-%cpu | head -10

# 3. Trova processi del tuo utente
ps -u $USER

# 4. Albero processi completo
pstree -p $USER

# 5. Informazioni su processo specifico (es. bash)
ps -C bash
ps -C bash -o pid,ppid,user,cmd,%cpu,%mem

# 6. Processi zombie (spero nessuno!)
ps aux | grep 'Z'
```

5.7.2 Esercizio 4.2: Monitoring con top

```
# 1. Avvia top
top

# 2. Mentre in top, prova:
```

```
# - Premi '1' per vedere singoli core
# - Premi 'P' per ordinare per CPU
# - Premi 'M' per ordinare per memoria
# - Premi 'u' e inserisci il tuo username
# - Premi 'c' per vedere comando completo
# - Premi 'i' per nascondere idle processes

# 3. Trova processo più pesante
# - Ordina per CPU (P)
# - Annota PID e nome

# 4. Cattura snapshot
top -b -n 1 > top_snapshot_$(date +%Y%m%d_%H%M%S).txt

# 5. Monitor specifico utente
top -u $USER
```

5.7.3 Esercizio 4.3: Job Control

```
# 1. Avvia processo in background
sleep 300 &
# Annota job number e PID

# 2. Avvia altro processo in background
sleep 400 &

# 3. Lista job
jobs
jobs -l

# 4. Avvia processo in foreground
sleep 500

# 5. Sospendi con Ctrl+Z
# Premi Ctrl+Z

# 6. Lista job (dovresti vedere 3 job)
jobs

# 7. Riprendi ultimo in background
bg

# 8. Porta primo job in foreground
fg %1

# 9. Termina con Ctrl+C

# 10. Verifica job rimanenti
jobs

# 11. Kill tutti i job rimanenti
kill %1 %2
# Oppure
jobs -p | xargs kill
```

5.7.4 Esercizio 4.4: nohup e Priorità

```
# 1. Create script di test
cat > long_task.sh << 'EOF'
#!/bin/bash
for i in {1..100}; do
    echo "Step $i of 100"
    sleep 2
done
echo "Completed!"
EOF

chmod +x long_task.sh

# 2. Esegui con nohup
nohup ./long_task.sh &

# 3. Verifica output
tail -f nohup.out
# Ctrl+C per uscire

# 4. Verifica che processo esiste
jobs -l
ps aux | grep long_task

# 5. Disown il job
disown

# 6. Avvia altro processo con bassa priorità
nice -n 15 ./long_task.sh &
NICE_PID=$!

# 7. Verifica nice value
ps -o pid,ni,cmd -p $NICE_PID

# 8. Cambia priorità
renice 5 -p $NICE_PID

# 9. Verifica cambio
ps -o pid,ni,cmd -p $NICE_PID

# 10. Cleanup
pkill -f long_task.sh
```

5.7.5 Esercizio 4.5: Scenario Reale

Simulazione: processo impazzito che consuma troppa CPU

```
# 1. Create processo CPU-intensive
cat > cpu_hog.sh << 'EOF'
#!/bin/bash
# CPU intensive loop
while true; do
    x=$((x + 1))
done
EOF

chmod +x cpu_hog.sh
```

```
11
12 # 2. Avvia processo
13 ./cpu_hog.sh &
14 HOG_PID=$!
15 echo "Started CPU hog with PID: $HOG_PID"
16
17 # 3. Monitor in top (altro terminal)
18 top -p $HOG_PID
19
20 # 4. Riduci priorità per limitare impatto
21 renice 19 -p $HOG_PID
22
23 # 5. Verifica riduzione impatto in top
24
25 # 6. Se ancora problematico, sospendi
26 kill -STOP $HOG_PID
27
28 # 7. Verifica stato
29 ps -o pid,stat,cmd -p $HOG_PID
30 # STAT dovrebbe essere T (stopped)
31
32 # 8. Riprendi
33 kill -CONT $HOG_PID
34
35 # 9. Termina gracefully
36 kill $HOG_PID
37
38 # 10. Se non termina, forza
39 sleep 2
40 kill -9 $HOG_PID 2>/dev/null
41
42 # 11. Verifica terminazione
43 ps -p $HOG_PID
44 # Dovrebbe dare errore (processo non esiste)
```

5.8 Best Practices

1. **Sempre SIGTERM prima di SIGKILL**

```
1 # Dai al processo tempo di chiudere correttamente
2 kill $PID
3 sleep 5
4 kill -0 $PID 2>/dev/null && kill -9 $PID
```

2. **Usa nohup per processi lunghi**

```
1 # Previeni terminazione accidentale
2 nohup ./long_running_task.sh > task.log 2>&1 &
```

3. **Monitora risorse regolarmente**

```
1 # Check giornaliero top consumers
2 ps aux --sort=-%cpu | head -10
3 ps aux --sort=-%mem | head -10
```

4. **Usa nice per task non critici**

```
# Backup, compilazione, compressione
nice -n 15 tar czf backup.tar.gz /data
```

5. **Cleanup job periodicamente**

```
# Verifica job dimenticati
jobs
# Kill quelli non necessari
```

5.9 Comandi Utili Rapidi

```
# Trova e killa processo per nome
pkill -f "process_name"

# Trova processi che usano file specifico
lsof /path/to/file
fuser /path/to/file

# Trova processi che usano porta
sudo lsof -i :8080
sudo netstat -tulpn | grep :8080

# Monitor continuo top 5 CPU users
watch -n 2 "ps aux --sort=-%cpu | head -6"

# Conta processi per utente
ps aux | awk '{print $1}' | sort | uniq -c | sort -rn

# Kill tutti i processi di uno script
pkill -f script_name.sh

# Verifica se processo è ancora running
kill -0 $PID && echo "Running" || echo "Not running"

# CPU usage totale per comando
ps aux | grep [p]rocess_name | awk '{sum+=$3} END {print sum "%"}'
```

5.10 Riepilogo

In questo capitolo abbiamo imparato:

- **Concetti**: PID, PPID, stati processi, gerarchia
- **ps**: visualizzare processi con varie opzioni
- **top/htop**: monitoring interattivo
- **kill**: inviare segnali (SIGTERM, SIGKILL, etc.)
- **killall/pkill**: terminare processi per nome
- **jobs**: gestire job della shell

- **fg/bg**: spostare processi tra foreground e background
- **nohup/disown**: processi persistenti
- **nice/renice**: gestire priorità

La gestione efficace dei processi è essenziale per amministrare sistemi Linux e ottimizzare le performance.

Prossimo Capitolo

Nel Capitolo 5 entreremo nel mondo del Bash Scripting: variabili, strutture di controllo, loop, funzioni e parametri per automatizzare task complessi.

Capitolo 6

Bash Scripting

6.1 Introduzione allo Scripting

Uno script Bash è un file di testo contenente comandi che vengono eseguiti sequenzialmente. Gli script permettono di automatizzare task ripetitivi, creare tool personalizzati e gestire configurazioni complesse.

6.1.1 Primo Script

```
# Create file hello.sh
cat > hello.sh << 'EOF'
#!/bin/bash
# Questo è un commento
echo "Hello, World!"
EOF

# Rendi eseguibile
chmod +x hello.sh

# Esegui
./hello.sh
# Output: Hello, World!
```

6.1.2 Shebang

La prima riga `#!/bin/bash` è chiamata **shebang** e indica quale interprete usare:

```
#!/bin/bash              # Bash script
#!/bin/sh                # POSIX shell (più portabile)
#!/usr/bin/env bash      # Trova bash nel PATH (più portabile)
#!/usr/bin/env python3   # Python script
#!/usr/bin/perl          # Perl script

# Senza shebang, devi eseguire esplicitamente:
bash script.sh

# Con shebang, puoi eseguire direttamente:
./script.sh
```

Best Practice: Shebang

Usa `#!/usr/bin/env bash` invece di `#!/bin/bash` per massima portabilità. Questo trova bash nel PATH invece di assumere una posizione fissa.

6.2 Variabili

6.2.1 Definizione e Uso

```bash
#!/bin/bash

# Definizione variabile (NO spazi intorno a =)
NAME="Alice"
AGE=25
PI=3.14159

# Uso variabile (con $)
echo "Hello, $NAME"
echo "You are $AGE years old"

# Sintassi alternativa (più sicura)
echo "Hello, ${NAME}"
echo "Age: ${AGE} years"

# ERRORE: spazi intorno a =
NAME = "Alice"      # ERRORE!
NAME= "Alice"       # ERRORE!
NAME ="Alice"       # ERRORE!
```

6.2.2 Tipi di Variabili

```bash
#!/bin/bash

# Stringhe
STRING="Hello World"
EMPTY_STRING=""

# Numeri (in Bash sono trattati come stringhe)
NUMBER=42
FLOAT=3.14    # Bash non supporta nativamente float

# Array
FRUITS=("apple" "banana" "orange")
NUMBERS=(1 2 3 4 5)

# Array associativi (hash/dictionary)
declare -A CAPITALS
CAPITALS["Italy"]="Rome"
CAPITALS["France"]="Paris"
CAPITALS["Spain"]="Madrid"

# Variabili di sola lettura
readonly CONSTANT="Cannot change"
CONSTANT="new value"    # Errore!

```

```
# Variabili d'ambiente (disponibili ai processi figli)
export GLOBAL_VAR="visible to child processes"
```

6.2.3 Espansione Variabili

```
#!/bin/bash

NAME="Alice"

# Espansione base
echo $NAME              # Alice
echo "$NAME"            # Alice (preferito)
echo '$NAME'            # $NAME (letterale, no espansione)

# Protezione con graffe
FILE="document"
echo $FILEtxt           # Errore: variabile FILEtxt non esiste
echo ${FILE}txt         # documenttxt

# Lunghezza stringa
echo ${#NAME}           # 5

# Substring
STRING="Hello World"
echo ${STRING:0:5}      # Hello (da indice 0, lunghezza 5)
echo ${STRING:6}        # World (da indice 6 fino alla fine)
echo ${STRING: -5}      # World (ultimi 5 caratteri, nota lo spazio)

# Sostituzione pattern
PATH_STRING="/home/user/documents/file.txt"
echo ${PATH_STRING/user/admin}          # Sostituisci prima occorrenza
echo ${PATH_STRING//\//\\}              # Sostituisci tutte / con \

# Rimozione pattern
echo ${PATH_STRING%.txt}                # Rimuovi .txt dalla fine
echo ${PATH_STRING#/home/}              # Rimuovi /home/ dall'inizio
echo ${PATH_STRING##*/}                 # file.txt (rimuovi tutto fino a
    ultimo /)
echo ${PATH_STRING%%/*}                 # (vuoto, rimuovi tutto da primo
    /)

# Default values
echo ${UNDEFINED:-"default"}            # default (se UNDEFINED non
   esiste)
echo ${UNDEFINED:="default"}            # default (e imposta UNDEFINED)
echo ${DEFINED:+alternative}            # alternative (se DEFINED esiste
   )
echo ${UNDEFINED:?"Error message"}      # Errore se non esiste
```

6.2.4 Variabili Speciali

```
#!/bin/bash

# Script name e parametri
$0          # Nome dello script
$1, $2...   # Parametri posizionali
```

```
$#          # Numero di parametri
$@          # Tutti i parametri (come array)
$*          # Tutti i parametri (come stringa)

# Exit status
$?          # Exit code dell'ultimo comando
            # 0 = successo, non-zero = errore

# Process IDs
$$          # PID dello script corrente
$!          # PID dell'ultimo processo in background

# Esempi
echo "Script name: $0"
echo "First argument: $1"
echo "Number of arguments: $#"
echo "All arguments: $@"
echo "PID: $$"

# Exit status
ls /existing_dir
echo $?     # 0 (successo)

ls /nonexistent_dir
echo $?     # 2 (errore)
```

6.3 Input/Output

6.3.1 Echo e Printf

```
#!/bin/bash

# echo - semplice output
echo "Simple text"
echo "Line 1\nLine 2"          # \n non interpretato di default
echo -e "Line 1\nLine 2"       # -e abilita escape sequences
echo -n "No newline"           # -n sopprime newline finale

# printf - formattazione avanzata (come C)
printf "Hello, %s\n" "World"
printf "Number: %d\n" 42
printf "Float: %.2f\n" 3.14159
printf "Hex: %x\n" 255

# Formattazione tabulare
printf "%-10s %-10s %s\n" "Name" "Age" "City"
printf "%-10s %-10d %s\n" "Alice" 25 "Rome"
printf "%-10s %-10d %s\n" "Bob" 30 "Milan"
```

6.3.2 Input Utente

```
#!/bin/bash

# read - leggi input
read -p "Enter your name: " NAME
```

```
echo "Hello, $NAME"

# Leggi multipli valori
read -p "Enter name and age: " NAME AGE
echo "Name: $NAME, Age: $AGE"

# Leggi array
read -a ARRAY -p "Enter numbers: "
echo "First number: ${ARRAY[0]}"

# Timeout (secondi)
read -t 5 -p "Quick! Answer in 5 seconds: " ANSWER

# Silent (per password)
read -s -p "Enter password: " PASSWORD
echo  # Newline dopo input silenzioso

# Leggi singolo carattere
read -n 1 -p "Press any key to continue..."
echo

# Default value
read -p "Enter name [default: User]: " NAME
NAME=${NAME:-User}
echo "Name: $NAME"

# Leggi da file
while read LINE; do
    echo "Line: $LINE"
done < input.txt
```

6.3.3 Redirezioni

```
#!/bin/bash

# Output redirection
echo "text" > file.txt          # Sovrascrive
echo "more" >> file.txt         # Append

# Input redirection
wc -l < file.txt                # Leggi da file

# Error redirection
command 2> errors.txt           # Solo stderr
command > output.txt 2>&1       # stdout e stderr insieme
command &> all_output.txt       # Shortcut (Bash 4+)

# Scarta output
command > /dev/null             # Scarta stdout
command 2> /dev/null            # Scarta stderr
command &> /dev/null            # Scarta tutto

# Here document
cat << EOF
Multiple lines
of text
EOF
```

```
25
26 # Here string
27 grep "pattern" <<< "text to search"
28
29 # Pipeline
30 cat file.txt | grep "pattern" | sort | uniq
```

6.4 Strutture di Controllo

6.4.1 Condizionali: if-then-else

```
1  #!/bin/bash
2
3  # Sintassi base
4  if [ condition ]; then
5      # commands
6  fi
7
8  # Con else
9  if [ condition ]; then
10     # commands se true
11 else
12     # commands se false
13 fi
14
15 # Con elif
16 if [ condition1 ]; then
17     # commands
18 elif [ condition2 ]; then
19     # commands
20 else
21     # commands
22 fi
23
24 # Esempi pratici
25 AGE=25
26
27 if [ $AGE -gt 18 ]; then
28     echo "Adult"
29 else
30     echo "Minor"
31 fi
32
33 # Multipli elif
34 SCORE=75
35
36 if [ $SCORE -ge 90 ]; then
37     echo "Grade: A"
38 elif [ $SCORE -ge 80 ]; then
39     echo "Grade: B"
40 elif [ $SCORE -ge 70 ]; then
41     echo "Grade: C"
42 elif [ $SCORE -ge 60 ]; then
43     echo "Grade: D"
44 else
45     echo "Grade: F"
46 fi
```

6.4.2 Test Conditions

```
#!/bin/bash

# OPERATORI NUMERICI
-eq      # uguale
-ne      # diverso
-lt      # minore di
-le      # minore o uguale
-gt      # maggiore di
-ge      # maggiore o uguale

# Esempi
[ 5 -eq 5 ]       # true
[ 5 -gt 3 ]       # true
[ 10 -le 20 ]     # true

# OPERATORI STRINGA
=        # uguale
!=       # diverso
-z       # stringa vuota
-n       # stringa non vuota
<        # minore (lessicografico)
>        # maggiore (lessicografico)

# Esempi
[ "$STR1" = "$STR2" ]       # uguale
[ "$STR" != "" ]            # non vuota
[ -z "$STR" ]               # vuota
[ -n "$STR" ]               # non vuota

# FILE TEST
-e       # esiste
-f       # è un file regolare
-d       # è una directory
-r       # è leggibile
-w       # è scrivibile
-x       # è eseguibile
-s       # dimensione > 0
-L       # è symbolic link

# Esempi
if [ -f "/etc/passwd" ]; then
    echo "File exists"
fi

if [ -d "/home" ]; then
    echo "Directory exists"
fi

if [ -x "./script.sh" ]; then
    echo "Script is executable"
fi

# OPERATORI LOGICI
```

```
&&      # AND
||      # OR
!       # NOT

# Esempi
if [ $AGE -gt 18 ] && [ $AGE -lt 65 ]; then
    echo "Working age"
fi

if [ -f "$FILE" ] || [ -d "$FILE" ]; then
    echo "Path exists"
fi

if [ ! -f "$FILE" ]; then
    echo "File does not exist"
fi

# Sintassi alternativa (doppia parentesi quadra)
if [[ $NAME == "Alice" ]]; then
    echo "Hello Alice"
fi

# Supporta pattern matching
if [[ $FILENAME == *.txt ]]; then
    echo "Text file"
fi

# Supporta regex
if [[ $EMAIL =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$ ]];
   then
    echo "Valid email"
fi
```

6.4.3 Case Statement

```
#!/bin/bash

# Sintassi
case $VARIABLE in
    pattern1)
        # commands
        ;;
    pattern2)
        # commands
        ;;
    *)
        # default
        ;;
esac

# Esempio: menu
echo "Select option:"
echo "1. List files"
echo "2. Show date"
echo "3. Exit"
read -p "Choice: " CHOICE

```

```
case $CHOICE in
    1)
        ls -l
        ;;
    2)
        date
        ;;
    3)
        echo "Goodbye!"
        exit 0
        ;;
    *)
        echo "Invalid option"
        ;;
esac

# Pattern matching
FILE="document.txt"

case $FILE in
    *.txt)
        echo "Text file"
        ;;
    *.jpg|*.png|*.gif)
        echo "Image file"
        ;;
    *.sh)
        echo "Shell script"
        ;;
    *)
        echo "Unknown type"
        ;;
esac

# Multipli statement per pattern
case $ANSWER in
    [Yy]|[Yy][Ee][Ss])
        echo "Confirmed"
        ;;
    [Nn]|[Nn][Oo])
        echo "Declined"
        ;;
    *)
        echo "Invalid answer"
        ;;
esac
```

6.5 Loop

6.5.1 For Loop

```
#!/bin/bash

# Iterazione su lista
for ITEM in apple banana orange; do
    echo "Fruit: $ITEM"
done
```

```

# Iterazione su range
for i in {1..10}; do
    echo "Number: $i"
done

# Con step
for i in {0..100..10}; do
    echo $i     # 0, 10, 20, ..., 100
done

# C-style for loop
for ((i=0; i<10; i++)); do
    echo "Count: $i"
done

# Iterazione su file
for FILE in *.txt; do
    echo "Processing: $FILE"
    # processa file
done

# Iterazione su output comando
for USER in $(cat /etc/passwd | cut -d: -f1); do
    echo "User: $USER"
done

# Iterazione su array
FRUITS=("apple" "banana" "orange")
for FRUIT in "${FRUITS[@]}"; do
    echo "Fruit: $FRUIT"
done

# Con indice
for i in "${!FRUITS[@]}"; do
    echo "Index $i: ${FRUITS[$i]}"
done
```

6.5.2 While Loop

```
#!/bin/bash

# Sintassi base
while [ condition ]; do
    # commands
done

# Esempio: contatore
COUNT=0
while [ $COUNT -lt 10 ]; do
    echo "Count: $COUNT"
    COUNT=$((COUNT + 1))
done

# Leggi file riga per riga
while read LINE; do
    echo "Line: $LINE"
```

```
done < input.txt

# Leggi con IFS personalizzato
while IFS=: read USERNAME PASSWORD UID GID COMMENT HOME SHELL; do
    echo "User: $USERNAME, Home: $HOME"
done < /etc/passwd

# Loop infinito
while true; do
    echo "Press Ctrl+C to stop"
    sleep 1
done

# Condizione su comando
while ps aux | grep -q "[m]yprocess"; do
    echo "Process still running..."
    sleep 5
done
echo "Process terminated"
```

6.5.3 Until Loop

```
#!/bin/bash

# until: esegue finché condizione è FALSE
until [ condition ]; do
    # commands
done

# Esempio
COUNT=0
until [ $COUNT -ge 10 ]; do
    echo "Count: $COUNT"
    COUNT=$((COUNT + 1))
done

# Aspetta che file esista
until [ -f "/tmp/ready.flag" ]; do
    echo "Waiting for ready flag..."
    sleep 2
done
echo "Ready flag found!"

# Aspetta che servizio risponda
until curl -s http://localhost:8080 > /dev/null; do
    echo "Waiting for service..."
    sleep 5
done
echo "Service is up!"
```

6.5.4 Break e Continue

```
#!/bin/bash

# break: esce dal loop
for i in {1..10}; do
```

```
    if [ $i -eq 5 ]; then
        break    # Esce quando i=5
    fi
    echo $i
done
# Output: 1 2 3 4

# continue: salta all'iterazione successiva
for i in {1..10}; do
    if [ $i -eq 5 ]; then
        continue    # Salta 5
    fi
    echo $i
done
# Output: 1 2 3 4 6 7 8 9 10

# Loop nidificati
for i in {1..3}; do
    for j in {1..3}; do
        if [ $i -eq 2 ] && [ $j -eq 2 ]; then
            break 2    # Esce da entrambi i loop
        fi
        echo "$i,$j"
    done
done
```

6.6 Funzioni

6.6.1 Definizione e Chiamata

```
#!/bin/bash

# Definizione funzione
function greet() {
    echo "Hello, World!"
}

# Sintassi alternativa (preferita)
greet() {
    echo "Hello, World!"
}

# Chiamata funzione
greet

# Funzione con parametri
greet_user() {
    echo "Hello, $1!"
}

greet_user "Alice"      # Output: Hello, Alice!
greet_user "Bob"        # Output: Hello, Bob!

# Multipli parametri
add() {
    local RESULT=$(($1 + $2))
    echo $RESULT
```

```
}

SUM=$(add 5 3)
echo "5 + 3 = $SUM"

# Return value (exit code)
is_even() {
    if [ $(($1 % 2)) -eq 0 ]; then
        return 0    # true
    else
        return 1    # false
    fi
}

if is_even 4; then
    echo "4 is even"
fi

if ! is_even 5; then
    echo "5 is not even"
fi
```

6.6.2 Variabili Locali

```
#!/bin/bash

# Variabili globali (default)
GLOBAL="I'm global"

test_scope() {
    GLOBAL="Modified inside function"
    LOCAL_VAR="I'm only in function"
}

echo $GLOBAL            # I'm global
test_scope
echo $GLOBAL            # Modified inside function
echo $LOCAL_VAR         # (vuoto - non esiste fuori dalla funzione)

# Variabili locali (best practice)
better_scope() {
    local INSIDE="I'm local"
    echo $INSIDE
}

better_scope            # I'm local
echo $INSIDE            # (vuoto - locale alla funzione)
```

6.6.3 Funzioni Avanzate

```
#!/bin/bash

# Return multipli valori (via echo)
get_user_info() {
    local NAME="Alice"
    local AGE=25
```

```
    local CITY="Rome"
    echo "$NAME|$AGE|$CITY"
}

USER_INFO=$(get_user_info)
IFS='|' read -r NAME AGE CITY <<< "$USER_INFO"
echo "Name: $NAME, Age: $AGE, City: $CITY"

# Funzione ricorsiva: fattoriale
factorial() {
    local NUM=$1
    if [ $NUM -le 1 ]; then
        echo 1
    else
        local PREV=$(factorial $((NUM - 1)))
        echo $((NUM * PREV))
    fi
}

echo "5! = $(factorial 5)"    # 120

# Funzione con default parameters
greet_with_default() {
    local NAME=${1:-"Guest"}
    echo "Hello, $NAME!"
}

greet_with_default           # Hello, Guest!
greet_with_default "Alice"   # Hello, Alice!

# Funzione con validazione parametri
divide() {
    if [ $# -ne 2 ]; then
        echo "Error: Need exactly 2 arguments" >&2
        return 1
    fi

    if [ $2 -eq 0 ]; then
        echo "Error: Division by zero" >&2
        return 1
    fi

    echo $(($1 / $2))
}

divide 10 2    # 5
divide 10 0    # Error: Division by zero
```

6.7 Array

6.7.1 Array Indicizzati

```
#!/bin/bash

# Definizione
FRUITS=("apple" "banana" "orange")
NUMBERS=(1 2 3 4 5)
```

```
MIXED=("text" 123 "more text")

# Accesso elementi
echo ${FRUITS[0]}         # apple (primo elemento)
echo ${FRUITS[1]}         # banana
echo ${FRUITS[2]}         # orange
echo ${FRUITS[-1]}        # orange (ultimo elemento)

# Tutti gli elementi
echo ${FRUITS[@]}         # apple banana orange
echo ${FRUITS[*]}         # apple banana orange

# Numero elementi
echo ${#FRUITS[@]}        # 3

# Lunghezza elemento specifico
echo ${#FRUITS[0]}        # 5 (length of "apple")

# Aggiungere elementi
FRUITS+=("grape")
FRUITS[4]="mango"

# Modificare elemento
FRUITS[1]="pear"

# Rimuovere elemento
unset FRUITS[2]

# Iterare
for FRUIT in "${FRUITS[@]}"; do
    echo "Fruit: $FRUIT"
done

# Iterare con indice
for i in "${!FRUITS[@]}"; do
    echo "Index $i: ${FRUITS[$i]}"
done

# Slice (subset)
ARRAY=(0 1 2 3 4 5 6 7 8 9)
echo ${ARRAY[@]:2:3}      # 2 3 4 (da indice 2, prendi 3 elementi)
echo ${ARRAY[@]:5}        # 5 6 7 8 9 (da indice 5 fino alla fine)
```

6.7.2 Array Associativi

```
#!/bin/bash

# Dichiarazione (richiesta per array associativi)
declare -A CAPITALS

# Assegnazione
CAPITALS["Italy"]="Rome"
CAPITALS["France"]="Paris"
CAPITALS["Spain"]="Madrid"
CAPITALS["Germany"]="Berlin"

# Accesso
```

```
13 echo ${CAPITALS["Italy"]}        # Rome
14
15 # Tutte le chiavi
16 echo ${!CAPITALS[@]}             # Italy France Spain Germany
17
18 # Tutti i valori
19 echo ${CAPITALS[@]}              # Rome Paris Madrid Berlin
20
21 # Numero elementi
22 echo ${#CAPITALS[@]}             # 4
23
24 # Iterare
25 for COUNTRY in "${!CAPITALS[@]}"; do
26     echo "$COUNTRY: ${CAPITALS[$COUNTRY]}"
27 done
28
29 # Controllare se chiave esiste
30 if [ ${CAPITALS["Italy"]+_} ]; then
31     echo "Italy exists in array"
32 fi
33
34 # Rimuovere elemento
35 unset CAPITALS["Spain"]
```

6.8 Esercizi Pratici

6.8.1 Esercizio 5.1: Script Base

```
1  #!/bin/bash
2  # Esercizio: Create uno script che accetta nome e età,
3  # poi stampa un messaggio personalizzato
4
5  read -p "Enter your name: " NAME
6  read -p "Enter your age: " AGE
7
8  echo "Hello, $NAME!"
9
10 if [ $AGE -lt 18 ]; then
11     echo "You are a minor."
12 elif [ $AGE -lt 65 ]; then
13     echo "You are an adult."
14 else
15     echo "You are a senior."
16 fi
17
18 echo "In 10 years, you will be $((AGE + 10)) years old."
```

6.8.2 Esercizio 5.2: Calcolatrice

```
1  #!/bin/bash
2  # Calcolatrice semplice
3
4  echo "Simple Calculator"
5  read -p "Enter first number: " NUM1
6  read -p "Enter operator (+, -, *, /): " OP
```

```
read -p "Enter second number: " NUM2

case $OP in
    +)
        RESULT=$((NUM1 + NUM2))
        ;;
    -)
        RESULT=$((NUM1 - NUM2))
        ;;
    \*)
        RESULT=$((NUM1 * NUM2))
        ;;
    /)
        if [ $NUM2 -eq 0 ]; then
            echo "Error: Division by zero"
            exit 1
        fi
        RESULT=$((NUM1 / NUM2))
        ;;
    *)
        echo "Error: Invalid operator"
        exit 1
        ;;
esac

echo "$NUM1 $OP $NUM2 = $RESULT"
```

6.8.3 Esercizio 5.3: Backup Script

```
#!/bin/bash
# Backup di directory

# Configurazione
SOURCE_DIR="$HOME/documents"
BACKUP_DIR="$HOME/backups"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="backup_${DATE}.tar.gz"

# Verifica esistenza directory source
if [ ! -d "$SOURCE_DIR" ]; then
    echo "Error: Source directory does not exist: $SOURCE_DIR"
    exit 1
fi

# Crea directory backup se non esiste
mkdir -p "$BACKUP_DIR"

# Esegui backup
echo "Creating backup..."
tar czf "${BACKUP_DIR}/${BACKUP_FILE}" "$SOURCE_DIR"

if [ $? -eq 0 ]; then
    echo "Backup successful: ${BACKUP_DIR}/${BACKUP_FILE}"

    # Mostra dimensione
    SIZE=$(du -h "${BACKUP_DIR}/${BACKUP_FILE}" | cut -f1)
    echo "Size: $SIZE"
```

```
else
    echo "Backup failed!"
    exit 1
fi

# Cleanup: mantieni solo ultimi 5 backup
cd "$BACKUP_DIR"
ls -t backup_*.tar.gz | tail -n +6 | xargs -r rm
echo "Old backups cleaned up"
```

6.8.4 Esercizio 5.4: System Monitor

```
#!/bin/bash
# Monitor sistema con funzioni

# Funzione: info CPU
show_cpu() {
    echo "=== CPU INFO ==="
    grep "model name" /proc/cpuinfo | head -1
    echo "CPU Usage:"
    top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk
        '{print 100 - $1"%"}'
    echo
}

# Funzione: info memoria
show_memory() {
    echo "=== MEMORY INFO ==="
    free -h
    echo
}

# Funzione: info disco
show_disk() {
    echo "=== DISK INFO ==="
    df -h / | tail -1
    echo
}

# Funzione: top processi
show_top_processes() {
    echo "=== TOP 5 PROCESSES (CPU) ==="
    ps aux --sort=-%cpu | head -6
    echo
}

# Menu
while true; do
    echo "System Monitor"
    echo "1. CPU Info"
    echo "2. Memory Info"
    echo "3. Disk Info"
    echo "4. Top Processes"
    echo "5. All Info"
    echo "6. Exit"
    read -p "Choice: " CHOICE

```

```
    case $CHOICE in
        1) show_cpu ;;
        2) show_memory ;;
        3) show_disk ;;
        4) show_top_processes ;;
        5)
            show_cpu
            show_memory
            show_disk
            show_top_processes
            ;;
        6)
            echo "Goodbye!"
            exit 0
            ;;
        *)
            echo "Invalid choice"
            ;;
    esac

    read -p "Press Enter to continue..."
    clear
done
```

6.9 Best Practices

Scripting Best Practices

1. **Sempre quote le variabili**

```
# MALE
if [ $VAR = "value" ]; then    # Errore se VAR è vuota

# BENE
if [ "$VAR" = "value" ]; then  # Sicuro
```

2. **Usa set per debug e sicurezza**

```
#!/bin/bash
set -euo pipefail
# -e: exit on error
# -u: error on undefined variable
# -o pipefail: error se comando in pipe fallisce
```

3. **Validare input**

```
if [ $# -ne 2 ]; then
    echo "Usage: $0 <arg1> <arg2>" >&2
    exit 1
fi
```

4. **Usa funzioni per codice riutilizzabile**

```
error() {
    echo "ERROR: $1" >&2
    exit 1
```

```
}

[ -f "$FILE" ] || error "File not found: $FILE"
```

5. **Commenta il codice**

```
# Spiega PERCHÉ, non cosa fa il codice
# MALE: Loop through files
# BENE: Process each log file to extract errors
```

6.10 Riepilogo

In questo capitolo abbiamo esplorato:

- **Script base**: shebang, esecuzione, commenti
- **Variabili**: definizione, tipi, espansione
- **I/O**: echo, printf, read, redirezioni
- **Condizionali**: if-then-else, test conditions, case
- **Loop**: for, while, until, break, continue
- **Funzioni**: definizione, parametri, return values
- **Array**: indicizzati e associativi

Bash scripting è uno strumento potente per automatizzare task e gestire sistemi Linux.

Prossimo Capitolo

Nel Capitolo 6 impareremo il text processing con sed, awk e altri strumenti per manipolare file di testo in modo efficiente.

Capitolo 7

Text Processing

7.1 Introduzione

L'elaborazione di testi è una delle competenze più potenti in Linux. La filosofia Unix di usare file di testo per configurazioni e dati rende gli strumenti di text processing essenziali per amministratori e sviluppatori.

Filosofia Unix

"Everything is a text stream" - I tool Unix sono progettati per lavorare insieme tramite pipe, processando flussi di testo. Questa composibilità è la chiave della potenza della command line.

7.2 Strumenti Base

7.2.1 wc - Word Count

Conta righe, parole e caratteri:

```
# Sintassi
wc [options] file

# Conta tutto (righe, parole, caratteri)
wc file.txt
# Output: 10  50  300 file.txt
#
#                           > caratteri (bytes)
#                     > parole
#               > righe

# Solo righe
wc -l file.txt

# Solo parole
wc -w file.txt

# Solo caratteri
wc -c file.txt
wc -m file.txt    # Multi-byte characters

# Multipli file
wc *.txt
# Mostra total alla fine
```

```
25
26 # Da stdin
27 cat file.txt | wc -l
28 ls | wc -l         # Conta file in directory
29
30 # Esempi pratici
31 # Conta utenti nel sistema
32 wc -l /etc/passwd
33
34 # Conta file .txt
35 find . -name "*.txt" | wc -l
36
37 # Conta righe di codice
38 find . -name "*.sh" -exec cat {} \; | wc -l
```

7.2.2 cut - Estrai Colonne

Estrae porzioni di testo da ogni riga:

```
1  # Estrai caratteri specifici
2  echo "Hello World" | cut -c 1-5
3  # Output: Hello
4
5  # Estrai da carattere N fino alla fine
6  echo "Hello World" | cut -c 7-
7  # Output: World
8
9  # Estrai campi (delimiter default: tab)
10 echo -e "name\tage\tcity" | cut -f 2
11 # Output: age
12
13 # Specifica delimiter
14 echo "Alice:25:Rome" | cut -d ':' -f 1
15 # Output: Alice
16
17 echo "Alice:25:Rome" | cut -d ':' -f 2
18 # Output: 25
19
20 # Multipli campi
21 echo "Alice:25:Rome" | cut -d ':' -f 1,3
22 # Output: Alice:Rome
23
24 # Range di campi
25 echo "a:b:c:d:e" | cut -d ':' -f 2-4
26 # Output: b:c:d
27
28 # Tutti i campi da N in poi
29 echo "a:b:c:d:e" | cut -d ':' -f 3-
30 # Output: c:d:e
31
32 # Esempi pratici con /etc/passwd
33 # Estrai solo usernames
34 cut -d ':' -f 1 /etc/passwd
35
36 # Estrai username e home directory
37 cut -d ':' -f 1,6 /etc/passwd
38
39 # Estrai solo shell
```

```
cut -d ':' -f 7 /etc/passwd | sort | uniq

# CSV processing
cat users.csv
# name,age,city
# Alice,25,Rome
# Bob,30,Milan

cut -d ',' -f 1 users.csv
# name
# Alice
# Bob
```

7.2.3 sort - Ordina Righe

Ordina righe di testo:

```
# Ordine alfabetico (default)
sort file.txt

# Ordine inverso
sort -r file.txt

# Ordine numerico
sort -n numbers.txt

# Esempio: differenza alfabetico vs numerico
cat > numbers.txt << EOF
100
20
3
1000
EOF

sort numbers.txt
# 100
# 1000
# 20
# 3

sort -n numbers.txt
# 3
# 20
# 100
# 1000

# Ordina per colonna specifica
# File: names.txt
# Alice 25 Rome
# Bob 30 Milan
# Charlie 20 Naples

sort -k 2 -n names.txt    # Ordina per seconda colonna (età), numerico
sort -k 3 names.txt       # Ordina per terza colonna (città)

# Delimiter personalizzato
cat users.csv
# Alice,25,Rome
```

```
# Bob,30,Milan

sort -t ',' -k 2 -n users.csv    # Ordina per età

# Unique: rimuovi duplicati durante sort
sort -u file.txt

# Case-insensitive
sort -f file.txt

# Human-numeric sort (per dimensioni: 1K, 2M, 3G)
du -h | sort -h

# Month sort
cat > months.txt << EOF
Mar
Jan
Dec
Feb
EOF

sort -M months.txt
# Jan
# Feb
# Mar
# Dec

# Check se file è già ordinato
sort -c file.txt
# sort: file.txt:3: disorder: line3
```

7.2.4 uniq - Rimuovi Duplicati

Rimuove righe duplicate adiacenti (richiede sort prima):

```
# Rimuovi duplicati
cat > file.txt << EOF
apple
banana
apple
banana
banana
orange
EOF

sort file.txt | uniq
# apple
# banana
# orange

# Conta occorrenze
sort file.txt | uniq -c
#   2 apple
#   3 banana
#   1 orange

# Ordina per frequenza
sort file.txt | uniq -c | sort -rn
```

```
#    3 banana
#    2 apple
#    1 orange

# Solo duplicati
sort file.txt | uniq -d
# apple
# banana

# Solo unique (no duplicati)
sort file.txt | uniq -u
# orange

# Ignora primi N campi
cat > data.txt << EOF
1 apple
2 apple
3 banana
4 banana
EOF

sort data.txt | uniq -f 1
# 1 apple
# 3 banana

# Case-insensitive
cat > words.txt << EOF
Apple
apple
APPLE
Banana
EOF

sort words.txt | uniq -i
# Apple
# Banana

# Esempi pratici
# Top 10 comandi più usati nella history
history | awk '{print $2}' | sort | uniq -c | sort -rn | head -10

# Trova IP duplicati in log
cat access.log | cut -d ' ' -f 1 | sort | uniq -c | sort -rn
```

7.3 sed - Stream Editor

sed è un editor di flusso per filtrare e trasformare testo.

7.3.1 Sostituzione Base

```
# Sintassi base: sed 's/pattern/replacement/'
echo "Hello World" | sed 's/World/Universe/'
# Output: Hello Universe

# Solo prima occorrenza per riga
echo "foo bar foo" | sed 's/foo/baz/'
```

```
# Output: baz bar foo

# Tutte le occorrenze (flag g = global)
echo "foo bar foo" | sed 's/foo/baz/g'
# Output: baz bar baz

# Case-insensitive (flag i)
echo "Hello WORLD" | sed 's/world/Universe/i'
# Output: Hello Universe

# N-esima occorrenza
echo "foo foo foo" | sed 's/foo/bar/2'
# Output: foo bar foo

# Da file
sed 's/old/new/g' input.txt

# Modifica file in-place (ATTENZIONE!)
sed -i 's/old/new/g' file.txt

# Backup prima di modificare in-place
sed -i.bak 's/old/new/g' file.txt
# Crea file.txt.bak con originale

# Delimiter alternativo (utile per path)
sed 's|/home/user|/home/admin|g' file.txt
sed 's#/old/path#/new/path#g' file.txt
```

7.3.2 Indirizzi e Range

```
# Applica solo a riga specifica
sed '3s/old/new/' file.txt          # Solo riga 3

# Range di righe
sed '2,5s/old/new/' file.txt        # Righe 2-5
sed '2,$s/old/new/' file.txt        # Riga 2 fino alla fine

# Pattern matching
sed '/pattern/s/old/new/' file.txt  # Solo righe che contengono pattern

# Range con pattern
sed '/start/,/end/s/old/new/' file.txt

# Negazione
sed '/pattern/!s/old/new/' file.txt  # Solo righe che NON matchano
```

7.3.3 Comandi sed

```
# d = delete
sed '3d' file.txt                   # Cancella riga 3
sed '2,5d' file.txt                 # Cancella righe 2-5
sed '/pattern/d' file.txt           # Cancella righe che matchano

# p = print (utile con -n)
sed -n '3p' file.txt                # Stampa solo riga 3
sed -n '2,5p' file.txt              # Stampa righe 2-5
```

```
sed -n '/pattern/p' file.txt          # Stampa righe che matchano (come
    grep)

# a = append (dopo riga)
sed '3a\New line after line 3' file.txt

# i = insert (prima di riga)
sed '3i\New line before line 3' file.txt

# c = change (sostituisci intera riga)
sed '3c\This replaces line 3' file.txt

# Multipli comandi
sed -e 's/foo/bar/g' -e 's/hello/world/g' file.txt
sed 's/foo/bar/g; s/hello/world/g' file.txt

# Script sed da file
cat > script.sed << EOF
s/foo/bar/g
s/hello/world/g
/^$/d
EOF

sed -f script.sed input.txt
```

7.3.4 Esempi Pratici sed

```
# Rimuovi righe vuote
sed '/^$/d' file.txt

# Rimuovi righe che iniziano con #
sed '/^#/d' file.txt

# Rimuovi spazi iniziali e finali
sed 's/^[ \t]*//; s/[ \t]*$//' file.txt

# Sostituisci tab con spazi
sed 's/\t/    /g' file.txt

# Numera righe
sed = file.txt | sed 'N; s/\n/\t/'

# Estrai range di righe (come head/tail)
sed -n '10,20p' file.txt

# Sostituisci solo in righe specifiche
sed '/pattern/s/foo/bar/g' file.txt

# Converti Windows line endings (CRLF) a Unix (LF)
sed 's/\r$//' file.txt

# Aggiungi testo all'inizio di ogni riga
sed 's/^/PREFIX: /' file.txt

# Aggiungi testo alla fine di ogni riga
sed 's/$/ SUFFIX/' file.txt

```

```
# Rimuovi tag HTML (semplice)
sed 's/<[^>]*>//g' file.html

# Estrai email da testo
sed -n 's/.*\([a-zA-Z0-9._%+-]\+@[a-zA-Z0-9.-]\+\.[a-zA-Z]\{2,\}\).*/\1/
    p' file.txt
```

7.4 awk - Pattern Scanning and Processing

`awk` è un linguaggio completo per processare testi strutturati.

7.4.1 Sintassi Base

```
# Sintassi: awk 'pattern { action }' file

# Stampa intera riga
awk '{print}' file.txt
awk '{print $0}' file.txt    # Equivalente

# Stampa colonna specifica
awk '{print $1}' file.txt    # Prima colonna
awk '{print $2}' file.txt    # Seconda colonna
awk '{print $NF}' file.txt   # Ultima colonna

# Multipli campi
awk '{print $1, $3}' file.txt

# Con separatore personalizzato
awk '{print $1 " -> " $3}' file.txt

# Esempio con /etc/passwd
awk -F ':' '{print $1}' /etc/passwd              # Usernames
awk -F ':' '{print $1, $6}' /etc/passwd          # Username e home
awk -F ':' '{print $1 " -> " $7}' /etc/passwd    # Username -> shell
```

7.4.2 Field Separator

```
# Default: whitespace (space e tab)
echo "one two three" | awk '{print $2}'
# Output: two

# Delimiter personalizzato (-F)
echo "one:two:three" | awk -F ':' '{print $2}'
# Output: two

# Multipli delimiter possibili
awk -F '[,:]' '{print $2}' file.txt

# Regex come delimiter
awk -F '[,;:|]' '{print $2}' file.txt

# Cambia field separator durante esecuzione
awk 'BEGIN {FS=":"} {print $1}' /etc/passwd

# Output field separator
```

```
awk 'BEGIN {OFS=" | "} {print $1, $2}' file.txt
```

7.4.3 Pattern Matching

```
# Stampa righe che matchano pattern
awk '/pattern/' file.txt
awk '/pattern/ {print}' file.txt    # Equivalente

# Pattern in colonna specifica
awk '$1 == "value"' file.txt
awk '$2 ~ /pattern/' file.txt       # Regex match
awk '$2 !~ /pattern/' file.txt      # NOT match

# Condizioni numeriche
awk '$3 > 100' file.txt             # Terza colonna > 100
awk '$2 >= 50 && $2 <= 100' file.txt

# Condizioni logiche
awk '$1 == "Alice" || $1 == "Bob"' file.txt
awk '$3 > 50 && $4 < 100' file.txt

# Esempi con /etc/passwd
# Utenti con UID > 1000
awk -F ':' '$3 > 1000 {print $1}' /etc/passwd

# Utenti con bash come shell
awk -F ':' '$7 ~ /bash/ {print $1}' /etc/passwd
```

7.4.4 BEGIN e END

```
# BEGIN: eseguito prima di processare file
awk 'BEGIN {print "Starting..."} {print} END {print "Done"}' file.txt

# Esempio: header e footer
awk 'BEGIN {print "Name\tAge"} {print $1, $2} END {print "Total:", NR}'
    file.txt

# Calcola somma
cat > numbers.txt << EOF
10
20
30
40
EOF

awk '{sum += $1} END {print "Total:", sum}' numbers.txt
# Output: Total: 100

# Media
awk '{sum += $1; count++} END {print "Average:", sum/count}' numbers.txt
# Output: Average: 25
```

7.4.5 Variabili Built-in

```
# NR = Number of Records (numero riga totale)
awk '{print NR, $0}' file.txt

# NF = Number of Fields (numero campi nella riga)
awk '{print NF, $0}' file.txt

# FNR = File Number of Records (numero riga nel file corrente)
awk '{print FNR, $0}' file1.txt file2.txt

# FS = Field Separator
awk 'BEGIN {FS=":"} {print $1}' /etc/passwd

# OFS = Output Field Separator
awk 'BEGIN {OFS=" | "} {print $1, $2}' file.txt

# RS = Record Separator (default: newline)
awk 'BEGIN {RS=";"} {print}' file.txt

# ORS = Output Record Separator
awk 'BEGIN {ORS="\n---\n"} {print}' file.txt

# FILENAME = nome file corrente
awk '{print FILENAME, $0}' file1.txt file2.txt
```

7.4.6 Operazioni Avanzate

```
# Aritmetica
awk '{print $1 * $2}' file.txt
awk '{print ($1 + $2) / 2}' file.txt

# Funzioni matematiche
awk '{print sqrt($1)}' numbers.txt
awk '{print int($1)}' floats.txt

# Concatenazione stringhe
awk '{print $1 $2}' file.txt          # Senza spazio
awk '{print $1 " " $2}' file.txt      # Con spazio

# Length
awk '{print length($1)}' file.txt

# Substring
awk '{print substr($1, 1, 5)}' file.txt

# toupper/tolower
awk '{print toupper($1)}' file.txt
awk '{print tolower($1)}' file.txt

# Condizionale ternario
awk '{print ($1 > 50) ? "High" : "Low"}' numbers.txt

# If-else in awk
awk '{if ($1 > 50) print "High"; else print "Low"}' numbers.txt

# For loop
awk '{for (i=1; i<=NF; i++) print $i}' file.txt
```

7.4.7 Esempi Pratici awk

```
# 1. Analisi log Apache
cat access.log | awk '{print $1}' | sort | uniq -c | sort -rn | head -10
# Top 10 IP più frequenti

# 2. Calcola dimensioni totali
ls -l | awk '{sum += $5} END {print "Total:", sum, "bytes"}'

# 3. Filtra e trasforma CSV
awk -F ',' '$3 > 1000 {print $1, $2}' sales.csv

# 4. Pretty print /etc/passwd
awk -F ':' '{printf "%-15s %-30s %s\n", $1, $6, $7}' /etc/passwd

# 5. Conta parole per lunghezza
cat text.txt | tr ' ' '\n' | awk '{len[length($0)]++} END {for (i in len
    ) print i, len[i]}'

# 6. Statistiche file di log
awk '
    {total++}
    /ERROR/ {errors++}
    /WARNING/ {warnings++}
    END {
        print "Total:", total
        print "Errors:", errors
        print "Warnings:", warnings
    }
' logfile.txt

# 7. Converti CSV in JSON
awk -F ',' '
    NR == 1 {for (i=1; i<=NF; i++) header[i]=$i; next}
    {
        print "{"
        for (i=1; i<=NF; i++)
            printf "  \"%s\": \"%s\"%s\n", header[i], $i, (i<NF ? "," :
                "")
        print "}"
    }
' data.csv

# 8. Report formattato
awk '
    BEGIN {
        print "========================================="
        print "           SALES REPORT"
        print "========================================="
        printf "%-15s %-15s %10s\n", "Product", "Quantity", "Revenue"
        print "-----------------------------------------"
    }
    {
        printf "%-15s %-15d $%9.2f\n", $1, $2, $3
        total += $3
    }
    END {
        print "========================================="
```

```
        printf "%-30s $%9.2f\n", "TOTAL", total
    }
' sales.txt
```

7.5 Combinare Strumenti

La vera potenza emerge combinando strumenti via pipe:

```
# 1. Top 10 comandi più usati
history | awk '{print $2}' | sort | uniq -c | sort -rn | head -10

# 2. Trova file grandi e ordina per dimensione
find /var/log -type f -exec du -h {} \; | sort -hr | head -20

# 3. Analisi access log
cat access.log \
    | awk '{print $1}' \
    | sort \
    | uniq -c \
    | sort -rn \
    | head -10 \
    | awk '{print $2, $1}'

# 4. Estrai email uniche da file
grep -Eo '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' file.txt \
    | sort -u

# 5. Statistiche codice
find . -name "*.sh" -type f \
    | xargs cat \
    | sed '/^$/d; /^#/d' \
    | wc -l

# 6. Report processi per utente
ps aux \
    | tail -n +2 \
    | awk '{user[$1]++; cpu[$1]+=$3; mem[$1]+=$4} END {
        for (u in user)
            printf "%-15s %5d processes, CPU: %6.2f%%, MEM: %6.2f%%\n",
                u, user[u], cpu[u], mem[u]
    }' \
    | sort -k3 -rn

# 7. Cleanup log files
find /var/log -name "*.log" -mtime +30 \
    | xargs gzip \
    | tee -a cleanup.log

# 8. Monitor in tempo reale
tail -f /var/log/syslog \
    | grep --line-buffered "ERROR" \
    | awk '{print strftime("%Y-%m-%d %H:%M:%S"), $0}'
```

7.6 Esercizi Pratici

7.6.1 Esercizio 6.1: Analisi File di Testo

```
# Create file di test
cat > books.txt << EOF
1984,George Orwell,1949,328
To Kill a Mockingbird,Harper Lee,1960,281
The Great Gatsby,F. Scott Fitzgerald,1925,180
Pride and Prejudice,Jane Austen,1813,432
Animal Farm,George Orwell,1945,112
EOF

# 1. Estrai solo titoli
cut -d ',' -f 1 books.txt

# 2. Ordina per anno
sort -t ',' -k 3 -n books.txt

# 3. Trova libri di Orwell
grep "Orwell" books.txt

# 4. Media pagine
awk -F ',' '{sum += $4; count++} END {print "Average:", sum/count}'
    books.txt

# 5. Libri prima del 1950
awk -F ',' '$3 < 1950 {print $1}' books.txt

# 6. Formatta output
awk -F ',' '{printf "%-30s by %-20s (%d)\n", $1, $2, $3}' books.txt
```

7.6.2 Esercizio 6.2: Log Analysis

```
# Create log di esempio
cat > server.log << EOF
2024-11-15 10:00:00 INFO User alice logged in
2024-11-15 10:05:00 ERROR Failed to connect to database
2024-11-15 10:10:00 WARNING High memory usage: 85%
2024-11-15 10:15:00 INFO User bob logged in
2024-11-15 10:20:00 ERROR Disk space critical
2024-11-15 10:25:00 INFO User alice logged out
2024-11-15 10:30:00 ERROR Failed to connect to database
EOF

# 1. Conta livelli di log
awk '{print $3}' server.log | sort | uniq -c

# 2. Solo errori
grep "ERROR" server.log

# 3. Conta errori per ora
awk '/ERROR/ {print substr($2, 1, 2)}' server.log | sort | uniq -c

# 4. Utenti unici
grep "logged in" server.log | awk '{print $5}' | sort -u

```

```
# 5. Report formattato
awk '
    BEGIN {print "Log Level Summary\n-----------------"}
    {levels[$3]++}
    END {for (l in levels) print l ":", levels[l]}
' server.log
```

7.6.3 Esercizio 6.3: CSV Processing

```
# Create CSV
cat > sales.csv << EOF
Product,Quantity,Price,Date
Laptop,5,1200,2024-11-01
Mouse,50,25,2024-11-02
Keyboard,30,75,2024-11-02
Monitor,10,300,2024-11-03
Laptop,3,1200,2024-11-04
EOF

# 1. Calcola revenue per prodotto
awk -F ',' 'NR > 1 {revenue[$1] += $2 * $3}
    END {for (p in revenue) print p ":", revenue[p]}' sales.csv

# 2. Total revenue
awk -F ',' 'NR > 1 {sum += $2 * $3} END {print "Total:", sum}' sales.csv

# 3. Prodotto più venduto
awk -F ',' 'NR > 1 {qty[$1] += $2}
    END {for (p in qty) print qty[p], p}' sales.csv \
    | sort -rn | head -1

# 4. Sales per data
awk -F ',' 'NR > 1 {date[$4] += $2 * $3}
    END {for (d in date) print d ":", date[d]}' sales.csv | sort
```

7.7 Best Practices

Text Processing Best Practices

1. **Usa tool giusto per il task**

```
# Semplice estrazione colonne: cut
cut -d ',' -f 1 file.csv

# Pattern matching semplice: grep
grep "ERROR" logfile.txt

# Trasformazioni complesse: awk
awk '{complex logic}' file.txt

# Sostituzioni: sed
sed 's/old/new/g' file.txt
```

2. **Test su sample prima di file grandi**

```
# Test su prime 10 righe
```

```
head -10 bigfile.txt | sed 's/old/new/g'

# Poi applica a tutto il file
sed 's/old/new/g' bigfile.txt > output.txt
```

3. **Backup prima di modifiche in-place**

```
# SEMPRE crea backup
sed -i.bak 's/old/new/g' important.txt
```

4. **Quote correttamente**

```
# Usa single quote per literal
awk '{print $1}' file.txt

# Use double quote se serve espansione variabile
awk "{print \$1, \"$VAR\"}" file.txt
```

5. **Commenta regex complesse**

```
# Email regex (explained)
# [a-zA-Z0-9._%+-]+   : local part
# @                   : literal @
# [a-zA-Z0-9.-]+      : domain
# \.                  : literal dot
# [a-zA-Z]{2,}        : TLD (2+ letters)
grep -E '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' file.
    txt
```

Performance Tips

1. Evita parsing ripetuto: salva risultati intermedi

```
# MALE - parse /etc/passwd 3 volte
cat /etc/passwd | grep bash
cat /etc/passwd | grep nologin
cat /etc/passwd | wc -l

# BENE - parse una volta, salva risultato
PASSWD_DATA=$(cat /etc/passwd)
echo "$PASSWD_DATA" | grep bash
echo "$PASSWD_DATA" | grep nologin
echo "$PASSWD_DATA" | wc -l
```

2. awk è più veloce di multipli pipe per operazioni complesse

```
# Meno efficiente
cat file.txt | cut -d ',' -f 1 | sort | uniq -c | sort -rn

# Più efficiente
awk -F ',' '{count[$1]++} END {for (i in count) print count[i],
    i}' file.txt | sort -rn
```

7.8 Riepilogo

In questo capitolo abbiamo esplorato:

- **Strumenti base**: wc, cut, sort, uniq per operazioni semplici
- **sed**: editing stream, sostituzioni, trasformazioni
- **awk**: linguaggio completo per text processing
- **Combinazioni**: pipe per operazioni complesse
- **Pattern pratici**: analisi log, CSV processing, reporting

La padronanza del text processing è essenziale per lavorare efficacemente in Linux, permettendo di analizzare log, processare dati, generare report e automatizzare task complessi.

Prossimi Passi

Con le competenze acquisite fino a qui, siete in grado di:

- Navigare e gestire il filesystem
- Controllare permessi e ownership
- Gestire processi e risorse
- Scrivere script Bash complessi
- Processare e analizzare file di testo

Nei capitoli successivi esplorerete networking, amministrazione sistema, sicurezza e automazione avanzata.

7.9 Cheatsheet Rapido

```
# === CUT ===
cut -d ':' -f 1 /etc/passwd              # Estrai username
cut -c 1-10 file.txt                     # Primi 10 caratteri

# === SORT ===
sort file.txt                            # Alfabetico
sort -n numbers.txt                      # Numerico
sort -r file.txt                         # Reverse
sort -k 2 -n file.txt                    # Per colonna 2, numerico
sort -u file.txt                         # Unique (rimuovi duplicati)

# === UNIQ ===
sort file.txt | uniq                     # Rimuovi duplicati adiacenti
sort file.txt | uniq -c                  # Conta occorrenze
sort file.txt | uniq -d                  # Solo duplicati

# === SED ===
sed 's/old/new/' file.txt                # Sostituisci (prima occorrenza
    )
sed 's/old/new/g' file.txt               # Sostituisci (tutte)
sed -i 's/old/new/g' file.txt            # Modifica in-place
sed '/pattern/d' file.txt                # Cancella righe che matchano
```

```
sed -n '10,20p' file.txt                    # Stampa righe 10-20

# === AWK ===
awk '{print $1}' file.txt                   # Prima colonna
awk -F ':' '{print $1}' /etc/passwd         # Con delimiter :
awk '$3 > 100' file.txt                     # Filtra: colonna 3 > 100
awk '{sum += $1} END {print sum}' num.txt # Somma colonna 1

# === COMBINAZIONI ===
# Top 10 IP in log
cat access.log | cut -d ' ' -f 1 | sort | uniq -c | sort -rn | head -10

# Conta righe codice (no commenti/vuote)
find . -name "*.sh" -exec cat {} \; | sed '/^#/d; /^$/d' | wc -l

# Converti CSV in TSV
sed 's/,/\t/g' input.csv > output.tsv
```

Capitolo 8

Networking in Linux

8.1 Introduzione

La configurazione e gestione della rete è una competenza fondamentale per ogni amministratore di sistema Linux. In questo capitolo esploreremo i comandi essenziali per diagnosticare, configurare e trasferire dati attraverso la rete.

Dalla verifica della connettività alla configurazione di interfacce, dal trasferimento sicuro di file alla download di risorse web, questi strumenti costituiscono il fondamento delle operazioni di rete quotidiane.

Mappa del capitolo

Sezioni: Configurazione interfacce (ifconfig, ip), Diagnostica rete (netstat, ss, ping), Connessioni remote (ssh, scp), Sincronizzazione (rsync), Download (curl, wget), Esempi pratici, Best practice, Troubleshooting.

8.2 Obiettivi di Apprendimento

- Configurare e visualizzare interfacce di rete con `ip` e `ifconfig`.
- Diagnosticare problemi di rete con `netstat`, `ss`, `ping`, `traceroute`.
- Gestire connessioni remote sicure con SSH.
- Trasferire file con `scp`, `rsync`.
- Scaricare e interagire con API tramite `curl` e `wget`.

8.3 Configurazione Interfacce di Rete

8.3.1 Il comando `ifconfig` (deprecato ma ancora usato)

`ifconfig` era lo strumento tradizionale per configurare interfacce di rete. Sebbene deprecato in favore di `ip`, è ancora presente in molti sistemi legacy.

```
# Visualizzare tutte le interfacce
ifconfig

# Visualizzare una interfaccia specifica
ifconfig eth0

# Attivare/disattivare un'interfaccia
```

```
sudo ifconfig eth0 up
sudo ifconfig eth0 down

# Assegnare indirizzo IP
sudo ifconfig eth0 192.168.1.100 netmask 255.255.255.0

# Abilitare modalità promiscua (sniffing)
sudo ifconfig eth0 promisc
```

Spiegazione del codice

- **ifconfig**: senza parametri mostra tutte le interfacce attive.
- **up/down**: attiva o disattiva un'interfaccia di rete.
- **netmask**: specifica la maschera di sottorete.
- **promisc**: modalità promiscua per catturare tutto il traffico.

8.3.2 Il comando `ip` (moderno e raccomandato)

`ip` è il comando moderno per la gestione di rete in Linux, parte del pacchetto *iproute2*.

```
# Visualizzare tutte le interfacce
ip link show
ip addr show

# Interfaccia specifica
ip addr show dev eth0

# Attivare/disattivare interfaccia
sudo ip link set eth0 up
sudo ip link set eth0 down

# Assegnare indirizzo IP
sudo ip addr add 192.168.1.100/24 dev eth0

# Rimuovere indirizzo IP
sudo ip addr del 192.168.1.100/24 dev eth0

# Visualizzare routing table
ip route show

# Aggiungere route
sudo ip route add 192.168.2.0/24 via 192.168.1.1 dev eth0

# Route di default
sudo ip route add default via 192.168.1.1

# Visualizzare statistiche
ip -s link show eth0

# Visualizzare neighbors (ARP)
ip neigh show
```

Confronto ifconfig vs ip

ifconfig	ip
ifconfig -a	ip link show
ifconfig eth0	ip addr show dev eth0
ifconfig eth0 up	ip link set eth0 up
route -n	ip route show
arp -a	ip neigh show

8.4 Diagnostica di Rete

8.4.1 Test di Connettività: `ping`

```
# Ping base
ping google.com

# Limitare numero di pacchetti
ping -c 4 8.8.8.8

# Impostare intervallo tra ping (0.2 secondi)
ping -i 0.2 192.168.1.1

# Dimensione pacchetto personalizzata
ping -s 1000 google.com

# Ping fino a successo
while ! ping -c 1 192.168.1.1 > /dev/null 2>&1; do
    echo "Waiting for network..."
    sleep 2
done
echo "Network is up!"
```

8.4.2 Traceroute: percorso verso destinazione

```
# Tracciare il percorso
traceroute google.com

# Con numeri IP (no DNS)
traceroute -n 8.8.8.8

# Usare ICMP invece di UDP
traceroute -I google.com

# Numero massimo di hop
traceroute -m 20 google.com
```

8.4.3 `netstat`: statistiche di rete (deprecato)

```
# Tutte le connessioni
netstat -a

# Solo connessioni TCP
netstat -at
```

```
6
7  # Solo connessioni UDP
8  netstat -au
9
10 # Porte in ascolto
11 netstat -l
12
13 # Con numeri invece di nomi
14 netstat -n
15
16 # Mostrare PID e nome programma
17 netstat -p
18
19 # Routing table
20 netstat -r
21
22 # Statistiche interfacce
23 netstat -i
24
25 # Combinazione utile: porte in ascolto con PID
26 sudo netstat -tulpn
```

8.4.4 ss: socket statistics (moderno)

ss è il sostituto moderno di **netstat**, più veloce ed efficiente.

```
1  # Tutti i socket
2  ss -a
3
4  # Socket TCP
5  ss -t
6
7  # Socket in ascolto
8  ss -l
9
10 # Combinazione comune: TCP listening con processi
11 sudo ss -tlpn
12
13 # Statistiche dettagliate
14 ss -s
15
16 # Filtrare per porta
17 ss -tn sport = :80
18 ss -tn dport = :443
19
20 # Connessioni stabilite
21 ss -t state established
22
23 # Mostrare timer
24 ss -to
25
26 # Connessioni a un host specifico
27 ss dst 192.168.1.100
28
29 # Contare connessioni per stato
30 ss -tan | awk '{print $1}' | sort | uniq -c
```

Stati comuni delle connessioni TCP

- **LISTEN**: porta in ascolto per connessioni in entrata
- **ESTABLISHED**: connessione attiva e funzionante
- **TIME-WAIT**: attesa dopo chiusura connessione
- **CLOSE-WAIT**: connessione chiusa dal remoto, in attesa di chiusura locale
- **SYN-SENT**: tentativo di connessione in corso

8.4.5 Test avanzati di connettività

```
# Test porta TCP specifica (telnet/nc)
nc -zv google.com 80

# Scansione range di porte
nc -zv 192.168.1.1 20-80

# Test UDP
nc -zvu 192.168.1.1 53

# Ascoltare su porta (server)
nc -l 8080

# Connettersi (client)
nc localhost 8080

# Trasferire file con netcat
# Server
nc -l 8080 > received_file.txt
# Client
nc 192.168.1.100 8080 < file.txt
```

8.5 DNS e Risoluzione Nomi

```
# Interrogare DNS
nslookup google.com

# Interrogare server DNS specifico
nslookup google.com 8.8.8.8

# Query DNS dettagliate (dig)
dig google.com

# Solo la risposta
dig google.com +short

# Record specifici
dig google.com MX
dig google.com TXT
dig google.com AAAA  # IPv6

# Reverse DNS lookup
dig -x 8.8.8.8
```

```
20
21 # Tracciare query DNS
22 dig google.com +trace
23
24 # Host command (semplice)
25 host google.com
26 host -t MX google.com
```

8.6 SSH: Secure Shell

8.6.1 Connessioni Base

```
1  # Connessione base
2  ssh user@hostname
3
4  # Porta custom
5  ssh -p 2222 user@hostname
6
7  # Specificare chiave privata
8  ssh -i ~/.ssh/mykey user@hostname
9
10 # Verbose mode (debugging)
11 ssh -v user@hostname
12 ssh -vv user@hostname  # più dettagli
13 ssh -vvv user@hostname  # massimo dettaglio
14
15 # Eseguire comando remoto
16 ssh user@hostname 'ls -la'
17 ssh user@hostname 'df -h'
18
19 # Eseguire comandi multipli
20 ssh user@hostname 'cd /var/log && tail -n 20 syslog'
21
22 # X11 Forwarding
23 ssh -X user@hostname
```

8.6.2 SSH Configuration

Il file `~/.ssh/config` permette di semplificare le connessioni.

```
1  # File: ~/.ssh/config
2
3  Host myserver
4      HostName 192.168.1.100
5      User admin
6      Port 2222
7      IdentityFile ~/.ssh/myserver_key
8
9  Host jump
10     HostName jump.example.com
11     User jumper
12
13 Host internal
14     HostName 10.0.0.50
15     User admin
16     ProxyJump jump
```

```
17
18 Host *
19     ServerAliveInterval 60
20     ServerAliveCountMax 3
21     Compression yes
```

Uso dopo configurazione

Dopo aver configurato `~/.ssh/config`, puoi connetterti semplicemente con:

```
1 ssh myserver
2 ssh internal  # usa automaticamente jump host
```

8.6.3 Port Forwarding e Tunneling

```
1  # Local port forwarding
2  # Connetti porta locale 8080 a remoto:80
3  ssh -L 8080:localhost:80 user@server
4
5  # Accedere a database remoto
6  ssh -L 3306:localhost:3306 user@dbserver
7  # Ora: mysql -h localhost -P 3306
8
9  # Remote port forwarding
10 # Esporre porta locale 3000 su remoto:8080
11 ssh -R 8080:localhost:3000 user@server
12
13 # Dynamic port forwarding (SOCKS proxy)
14 ssh -D 8080 user@server
15 # Configura browser con SOCKS proxy localhost:8080
16
17 # Combinare forwarding multipli
18 ssh -L 8080:localhost:80 -L 3306:localhost:3306 user@server
19
20 # Background e keep alive
21 ssh -fN -L 8080:localhost:80 user@server
```

Opzioni SSH utili

- **-f**: manda SSH in background
- **-N**: non eseguire comandi remoti (solo tunneling)
- **-L**: local port forwarding
- **-R**: remote port forwarding
- **-D**: dynamic port forwarding (SOCKS)
- **-J**: jump host

8.7 Trasferimento File

8.7.1 SCP: Secure Copy

```
# Copiare file verso server remoto
scp file.txt user@server:/path/to/destination/

# Copiare da server remoto
scp user@server:/path/to/file.txt ./

# Copiare directory ricorsivamente
scp -r mydir/ user@server:/path/to/destination/

# Porta custom
scp -P 2222 file.txt user@server:/path/

# Preservare permessi e timestamp
scp -p file.txt user@server:/path/

# Limitare banda (KB/s)
scp -l 1000 largefile.tar.gz user@server:/path/

# Verbose mode
scp -v file.txt user@server:/path/

# Compressione
scp -C largefile.txt user@server:/path/

# Copia tra due server remoti
scp user1@server1:/path/file.txt user2@server2:/path/

# Specificare chiave
scp -i ~/.ssh/mykey file.txt user@server:/path/
```

8.7.2 RSYNC: Sincronizzazione Efficiente

`rsync` è superiore a `scp` per sincronizzare grandi quantità di dati, trasferendo solo le differenze.

```
# Sincronizzazione base locale
rsync -av source/ destination/

# Sincronizzazione remota
rsync -av source/ user@server:/path/to/destination/

# Download da remoto
rsync -av user@server:/path/to/source/ ./destination/

# Opzioni comuni
# -a: archive mode (preserva tutto)
# -v: verbose
# -z: compressione
# -P: progress + partial (riprende download interrotti)
# --delete: elimina file in dest non presenti in source

# Sincronizzazione completa con progress e delete
rsync -avzP --delete source/ user@server:/backup/

# Dry run (simulazione)
rsync -avzn --delete source/ destination/

# Escludere file/directory
```

```
rsync -av --exclude='*.log' --exclude='tmp/' source/ dest/

# Includere solo certi file
rsync -av --include='*.txt' --exclude='*' source/ dest/

# Limitare banda (KB/s)
rsync -av --bwlimit=1000 source/ user@server:/dest/

# SSH con porta custom
rsync -av -e 'ssh -p 2222' source/ user@server:/dest/

# Backup incrementale con hard links
rsync -av --link-dest=../previous_backup source/ new_backup/

# Mostrare progresso
rsync -av --progress source/ dest/

# Log delle operazioni
rsync -av --log-file=rsync.log source/ dest/
```

Differenza trailing slash in rsync

Con slash: `rsync source/ dest/` copia il *contenuto* di source in dest
Senza slash: `rsync source dest/` copia la *directory* source dentro dest

8.7.3 Script rsync per backup

```
#!/bin/bash
# backup_script.sh - Backup incrementale con rsync

SOURCE="/home/user/documents"
DEST="user@backup-server:/backups/documents"
LOG="/var/log/backup.log"
DATE=$(date +%Y%m%d_%H%M%S)

echo "[$DATE] Starting backup..." >> "$LOG"

rsync -avzP \
    --delete \
    --exclude='*.tmp' \
    --exclude='.cache/' \
    --log-file="$LOG" \
    "$SOURCE/" \
    "$DEST/"

if [ $? -eq 0 ]; then
    echo "[$DATE] Backup completed successfully" >> "$LOG"
else
    echo "[$DATE] Backup failed!" >> "$LOG"
    # Invia notifica email o alert
fi
```

8.8 Download e Web: curl e wget

8.8.1 WGET: Download Files

```
# Download semplice
wget https://example.com/file.zip

# Salvare con nome diverso
wget -O output.zip https://example.com/file.zip

# Continuare download interrotto
wget -c https://example.com/largefile.iso

# Download in background
wget -b https://example.com/file.zip

# Limitare velocità (KB/s)
wget --limit-rate=200k https://example.com/file.zip

# Download ricorsivo di un sito
wget -r -np -k https://example.com/docs/

# Mirror di un sito
wget --mirror --convert-links --page-requisites \
     --no-parent https://example.com/

# Download con autenticazione
wget --user=username --password=pass https://site.com/file

# Retry automatico
wget --tries=10 --retry-connrefused https://example.com/file

# User agent custom
wget --user-agent="Mozilla/5.0" https://example.com/

# Download multipli da file
# File urls.txt contiene un URL per riga
wget -i urls.txt

# Timestamping (scarica solo se più recente)
wget -N https://example.com/file.zip
```

8.8.2 CURL: Swiss Army Knife del Web

`curl` è più versatile di wget, ideale per API e debugging HTTP.

```
# GET request base
curl https://api.example.com/data

# Salvare output su file
curl -o output.json https://api.example.com/data
curl -O https://example.com/file.zip  # usa nome originale

# Seguire redirect
curl -L https://short.url/xyz

# Mostrare headers
curl -I https://example.com
curl -i https://example.com  # headers + body

# Verbose mode (debug)
```

```
curl -v https://example.com

# POST request con dati
curl -X POST https://api.example.com/users \
     -d "name=John&email=john@example.com"

# POST JSON
curl -X POST https://api.example.com/users \
     -H "Content-Type: application/json" \
     -d '{"name":"John","email":"john@example.com"}'

# PUT request
curl -X PUT https://api.example.com/users/123 \
     -H "Content-Type: application/json" \
     -d '{"name":"John Updated"}'

# DELETE request
curl -X DELETE https://api.example.com/users/123

# Autenticazione Basic
curl -u username:password https://api.example.com/data

# Bearer token
curl -H "Authorization: Bearer TOKEN" https://api.example.com/data

# Custom headers
curl -H "X-Custom-Header: value" https://api.example.com/

# Upload file
curl -F "file=@/path/to/file.pdf" https://api.example.com/upload

# Form multipart
curl -F "name=John" -F "file=@photo.jpg" https://api.example.com/

# Cookie
curl -b "session=abc123" https://example.com/
curl -c cookies.txt https://example.com/  # salva cookies

# Timeout
curl --connect-timeout 10 --max-time 30 https://example.com/

# Seguire redirect con limite
curl -L --max-redirs 5 https://example.com/

# Download con progress bar
curl -# -O https://example.com/largefile.zip

# Rate limiting
curl --limit-rate 100K https://example.com/file.zip

# Proxy
curl -x http://proxy.example.com:8080 https://api.example.com/

# Ignorare certificati SSL (non in produzione!)
curl -k https://self-signed.example.com/
```

8.8.3 Script curl per monitoraggio API

```bash
#!/bin/bash
# api_monitor.sh - Monitora availability e response time di API

API_URL="https://api.example.com/health"
THRESHOLD=2  # secondi

# Misurare tempo di risposta
RESPONSE_TIME=$(curl -o /dev/null -s -w '%{time_total}\n' "$API_URL")
HTTP_CODE=$(curl -o /dev/null -s -w '%{http_code}\n' "$API_URL")

echo "API Response Time: ${RESPONSE_TIME}s"
echo "HTTP Status Code: $HTTP_CODE"

if [ "$HTTP_CODE" != "200" ]; then
    echo "ERROR: API returned non-200 status!"
    # Invia alert
elif (( $(echo "$RESPONSE_TIME > $THRESHOLD" | bc -l) )); then
    echo "WARNING: API response time above threshold!"
fi
```

8.8.4 curl: format output avanzato

```bash
# Creare file curl-format.txt:
# time_namelookup:  %{time_namelookup}\n
# time_connect:     %{time_connect}\n
# time_starttransfer: %{time_starttransfer}\n
# time_total:       %{time_total}\n
# speed_download:   %{speed_download}\n
# http_code:        %{http_code}\n

curl -w "@curl-format.txt" -o /dev/null -s https://example.com

# Inline
curl -w "DNS: %{time_namelookup}s\nConnect: %{time_connect}s\nTotal: %{time_total}s\n" \
    -o /dev/null -s https://example.com
```

8.9 Best Practice e Sicurezza

Best Practice Networking

1. **Usa ip invece di ifconfig**: il comando moderno è più potente e consistente.
2. **Preferisci ss a netstat**: più veloce ed efficiente.
3. **Sempre verbose in debug**: usa `-v` o `-vv` per troubleshooting.
4. **Verifica connettività a livelli**: ping → traceroute → nc → curl.
5. **rsync per sincronizzazioni**: più efficiente di scp per file multipli.
6. **Usa SSH config**: semplifica gestione di server multipli.
7. **Limitare banda**: usa `-bwlimit` per non saturare rete.

8. **Dry-run**: testa sempre con `-n` prima di operazioni distruttive.

Sicurezza

- **SSH keys**: mai usare password in produzione, sempre chiavi SSH.
- **Firewall**: configurare iptables/ufw per limitare accessi.
- **Cambia porte default**: SSH su porta diversa da 22.
- **Fail2ban**: protezione contro brute force.
- **VPN**: per accesso a risorse interne, non esporre direttamente.
- **HTTPS sempre**: per API e trasferimenti sensibili.
- **Validare certificati SSL**: non usare `curl -k` in produzione.
- **Limitare utenti SSH**: configurare `AllowUsers` in sshd_config.

8.10 Troubleshooting Comuni

8.10.1 Problemi di connettività

```
# Checklist diagnostica rete

# 1. Verificare interfaccia è up
ip link show

# 2. Verificare indirizzo IP
ip addr show

# 3. Ping localhost
ping -c 2 127.0.0.1

# 4. Ping gateway
ping -c 2 $(ip route | grep default | awk '{print $3}')

# 5. Ping DNS pubblico
ping -c 2 8.8.8.8

# 6. Risoluzione DNS
nslookup google.com

# 7. Traceroute
traceroute google.com

# 8. Verificare firewall
sudo iptables -L -n
sudo ufw status
```

8.10.2 SSH non si connette

```
# Debug SSH connection
ssh -vvv user@host
```

```
# Verificare porta SSH
sudo ss -tlpn | grep ssh

# Test porta SSH da remoto
nc -zv host 22

# Verificare chiavi
ssh-keygen -l -f ~/.ssh/id_rsa.pub

# Permessi corretti per chiavi
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
chmod 600 ~/.ssh/authorized_keys

# Verificare configurazione server
sudo sshd -t  # test configuration
```

8.11 Esercizi Pratici

1. Configurare un indirizzo IP statico su interfaccia eth0 con `ip addr`.
2. Creare uno script che verifichi se un host è raggiungibile e invii email se down.
3. Configurare `~/.ssh/config` per 3 server con configurazioni diverse.
4. Creare uno script rsync per backup incrementale giornaliero con rotazione settimanale.
5. Usare curl per interrogare un'API REST (es. https://api.github.com/users/USERNAME).
6. Scaricare un intero sito web con wget in modalità mirror.
7. Configurare un tunnel SSH per accedere a database remoto via porta locale.
8. Usare netcat per trasferire un file tra due macchine.
9. Creare script che monitora porte aperte e notifica se cambiano.
10. Misurare latenza e throughput di connessione con ping e iperf.

8.12 Script Completo: Network Monitor

```
#!/bin/bash
# network_monitor.sh - Monitor completo stato rete

LOG_FILE="/var/log/network_monitor.log"
HOSTS=("8.8.8.8" "google.com" "192.168.1.1")
EMAIL="admin@example.com"

log_message() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a "$LOG_FILE"
}

check_host() {
    local host=$1
```

```
    if ping -c 3 -W 2 "$host" > /dev/null 2>&1; then
        log_message "OK: $host is reachable"
        return 0
    else
        log_message "ERROR: $host is unreachable!"
        return 1
    fi
}

check_dns() {
    if nslookup google.com > /dev/null 2>&1; then
        log_message "OK: DNS resolution working"
        return 0
    else
        log_message "ERROR: DNS resolution failed!"
        return 1
    fi
}

check_interface() {
    local iface=$1
    if ip link show "$iface" | grep -q "state UP"; then
        log_message "OK: Interface $iface is UP"
        return 0
    else
        log_message "ERROR: Interface $iface is DOWN!"
        return 1
    fi
}

send_alert() {
    local message=$1
    echo "$message" | mail -s "Network Alert" "$EMAIL"
}

main() {
    log_message "Starting network monitor"

    # Check interface
    if ! check_interface "eth0"; then
        send_alert "Interface eth0 is DOWN!"
    fi

    # Check DNS
    if ! check_dns; then
        send_alert "DNS resolution failed!"
    fi

    # Check hosts
    for host in "${HOSTS[@]}"; do
        if ! check_host "$host"; then
            send_alert "Host $host is unreachable!"
        fi
    done

    log_message "Network monitor completed"
}

```

```
72 main
```

8.13 Riepilogo

In questo capitolo hai imparato:

- Configurare interfacce di rete con `ip` e `ifconfig`
- Diagnosticare problemi con `ping`, `traceroute`, `ss`, `netstat`
- Gestire connessioni remote sicure con SSH e tunneling
- Trasferire file efficientemente con `scp` e `rsync`
- Interagire con API e scaricare contenuti con `curl` e `wget`
- Implementare monitoring e automazione delle operazioni di rete

8.14 Riferimenti

- https://man7.org/linux/man-pages/man8/ip.8.html
- https://man.openbsd.org/ssh
- https://rsync.samba.org/
- https://curl.se/docs/
- https://www.gnu.org/software/wget/manual/

Capitolo 9

Amministrazione di Sistema

9.1 Introduzione

L'amministrazione di sistema Linux richiede competenze in gestione utenti, pianificazione task, gestione servizi e analisi log. Questi strumenti sono essenziali per mantenere un sistema sicuro, efficiente e monitorato.

In questo capitolo esploreremo la gestione completa di utenti e gruppi, l'automazione con cron, la gestione moderna dei servizi con systemd, e l'analisi dei log di sistema.

Mappa del capitolo

Sezioni: Gestione utenti e gruppi, Gestione permessi avanzati, Cron e scheduling, Systemd e gestione servizi, Log di sistema, Monitoraggio risorse, Best practice sicurezza.

9.2 Obiettivi di Apprendimento

- Creare e gestire utenti e gruppi di sistema.
- Configurare permessi avanzati (ACL, setuid, setgid).
- Pianificare task automatici con cron e systemd timers.
- Gestire servizi con systemd.
- Analizzare e monitorare log di sistema con journalctl.
- Implementare best practice di sicurezza.

9.3 Gestione Utenti e Gruppi

9.3.1 Gestione Utenti

```
# Creare nuovo utente
sudo useradd username

# Creare utente con home directory e shell
sudo useradd -m -s /bin/bash username

# Creare utente con parametri completi
sudo useradd -m -s /bin/bash -c "John Doe" \
    -G sudo,developers -e 2025-12-31 username

```

```
# Impostare password
sudo passwd username

# Modificare utente esistente
sudo usermod -c "New Comment" username
sudo usermod -s /bin/zsh username
sudo usermod -L username  # Lock account
sudo usermod -U username  # Unlock account

# Aggiungere utente a gruppo
sudo usermod -aG groupname username

# Rimuovere utente da gruppo (solo quello specificato)
sudo gpasswd -d username groupname

# Cambiare home directory
sudo usermod -d /new/home/path -m username

# Eliminare utente
sudo userdel username
sudo userdel -r username  # rimuove anche home directory

# Visualizzare info utente
id username
finger username
getent passwd username

# Elencare utenti loggati
who
w
users
last  # storico login
```

Opzioni comuni useradd

- **-m**: crea home directory
- **-s**: specifica shell di login
- **-c**: commento (nome completo)
- **-G**: gruppi supplementari (separati da virgola)
- **-e**: data scadenza account (YYYY-MM-DD)
- **-d**: home directory custom
- **-u**: UID specifico

9.3.2 Gestione Gruppi

```
# Creare gruppo
sudo groupadd developers

# Creare gruppo con GID specifico
sudo groupadd -g 1500 developers

```

```
# Modificare gruppo
sudo groupmod -n newname oldname

# Eliminare gruppo
sudo groupdel groupname

# Visualizzare gruppi di un utente
groups username
id -Gn username

# Elencare tutti i gruppi
getent group

# Visualizzare membri di un gruppo
getent group groupname
grep groupname /etc/group
```

9.3.3 File di configurazione utenti

```
# /etc/passwd - Database utenti
# Formato: username:x:UID:GID:comment:home:shell
cat /etc/passwd
grep username /etc/passwd

# /etc/shadow - Password criptate (solo root)
# Formato: username:encrypted_password:last_change:min:max:warn:inactive
    :expire
sudo cat /etc/shadow

# /etc/group - Database gruppi
# Formato: groupname:x:GID:members
cat /etc/group

# /etc/sudoers - Configurazione sudo
sudo visudo  # SEMPRE usare visudo per editare!

# Esempio /etc/sudoers
# username ALL=(ALL:ALL) ALL
# %groupname ALL=(ALL:ALL) ALL
# username ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart nginx
```

9.3.4 Script: Creazione utente completo

```
#!/bin/bash
# create_user.sh - Crea utente con configurazione completa

if [ $# -ne 2 ]; then
    echo "Usage: $0 <username> <full_name>"
    exit 1
fi

USERNAME=$1
FULLNAME=$2

# Verifica se utente esiste già
if id "$USERNAME" &>/dev/null; then
```

```
    echo "Error: User $USERNAME already exists"
    exit 1
fi

# Crea utente
sudo useradd -m -s /bin/bash -c "$FULLNAME" "$USERNAME"

# Imposta password
echo "Setting password for $USERNAME"
sudo passwd "$USERNAME"

# Aggiungi a gruppo developers
sudo usermod -aG developers "$USERNAME"

# Crea directory di lavoro
sudo mkdir -p "/home/$USERNAME/projects"
sudo chown "$USERNAME:$USERNAME" "/home/$USERNAME/projects"

# Configura skeleton bash
cat << 'EOF' | sudo tee -a "/home/$USERNAME/.bashrc" > /dev/null

# Custom aliases
alias ll='ls -lah'
alias ..='cd ..'
alias update='sudo apt update && sudo apt upgrade'

# Custom prompt
PS1='\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
EOF

# Fix ownership
sudo chown "$USERNAME:$USERNAME" "/home/$USERNAME/.bashrc"

echo "User $USERNAME created successfully!"
echo "Home: /home/$USERNAME"
echo "Groups: $(groups $USERNAME)"
```

9.4 Permessi Avanzati

9.4.1 ACL: Access Control Lists

Gli ACL permettono permessi granulari oltre i classici owner/group/others.

```
# Visualizzare ACL
getfacl file.txt

# Impostare ACL per utente specifico
setfacl -m u:username:rw file.txt

# Impostare ACL per gruppo
setfacl -m g:groupname:rx directory/

# ACL ricorsivi
setfacl -R -m u:username:rwx directory/

# ACL default (ereditati da nuovi file)
setfacl -d -m u:username:rw directory/
```

```
15
16 # Rimuovere ACL specifico
17 setfacl -x u:username file.txt
18
19 # Rimuovere tutti gli ACL
20 setfacl -b file.txt
21
22 # Copiare ACL da un file all'altro
23 getfacl file1.txt | setfacl --set-file=- file2.txt
24
25 # Esempio completo
26 sudo setfacl -R -m u:webuser:rwx /var/www/mysite
27 sudo setfacl -R -d -m u:webuser:rwx /var/www/mysite
```

9.4.2 Permessi Speciali: SUID, SGID, Sticky Bit

```
1  # SUID (Set User ID) - esegue con permessi owner
2  # Esempio: /usr/bin/passwd
3  chmod u+s file
4  chmod 4755 file
5
6  # SGID (Set Group ID)
7  # Su file: esegue con permessi gruppo
8  # Su directory: nuovi file ereditano gruppo directory
9  chmod g+s directory
10 chmod 2755 directory
11
12 # Sticky Bit (solo su directory)
13 # Solo owner può eliminare i propri file
14 # Esempio: /tmp
15 chmod +t directory
16 chmod 1777 directory
17
18 # Visualizzare permessi speciali
19 ls -l /usr/bin/passwd  # -rwsr-xr-x (SUID)
20 ls -ld /tmp            # drwxrwxrwt (Sticky)
21
22 # Trovare file con SUID/SGID (potenziali rischi sicurezza)
23 find / -perm -4000 -type f 2>/dev/null  # SUID
24 find / -perm -2000 -type f 2>/dev/null  # SGID
```

Notazione permessi speciali

Simbolo	Numero	Significato
u+s	4—	SUID
g+s	2—	SGID
+t	1—	Sticky Bit

Permesso completo: 4755 = SUID + rwxr-xr-x

9.5 Cron: Task Scheduling

9.5.1 Sintassi Crontab

```
1 # Formato crontab:
```

```
#               minuto (0-59)
#                   ora (0-23)
#                       giorno del mese (1-31)
#                           mese (1-12)
#                               giorno della settimana (0-7, 0=domenica)
#
# * * * * * comando da eseguire

# Gestione crontab
crontab -e      # Modifica crontab corrente
crontab -l      # Lista crontab corrente
crontab -r      # Rimuovi crontab
crontab -u user # Gestisci crontab di altro utente (root only)

# Esempi comuni
# Ogni minuto
* * * * * /path/to/script.sh

# Ogni ora al minuto 0
0 * * * * /path/to/script.sh

# Ogni giorno alle 2:30 AM
30 2 * * * /path/to/backup.sh

# Ogni lunedì alle 9:00
0 9 * * 1 /path/to/weekly_report.sh

# Primo giorno del mese alle 00:00
0 0 1 * * /path/to/monthly.sh

# Ogni 15 minuti
*/15 * * * * /path/to/check.sh

# Ogni 6 ore
0 */6 * * * /path/to/script.sh

# Giorni lavorativi alle 8:00
0 8 * * 1-5 /path/to/workday.sh

# Range di ore (9-17)
0 9-17 * * * /path/to/business_hours.sh

# Multipli valori (alle 9, 12, 18)
0 9,12,18 * * * /path/to/script.sh
```

9.5.2 Cron speciali

```
# Shortcut speciali
@reboot   /path/to/script.sh    # All'avvio
@yearly   /path/to/script.sh    # 0 0 1 1 *
@annually /path/to/script.sh    # (identico a @yearly)
@monthly  /path/to/script.sh    # 0 0 1 * *
@weekly   /path/to/script.sh    # 0 0 * * 0
@daily    /path/to/script.sh    # 0 0 * * *
@midnight /path/to/script.sh    # (identico a @daily)
@hourly   /path/to/script.sh    # 0 * * * *

```

```
# Esempio completo crontab
# PATH=/usr/local/bin:/usr/bin:/bin
# SHELL=/bin/bash
# MAILTO=admin@example.com

# Backup giornaliero
@daily /usr/local/bin/backup.sh

# Log cleanup settimanale
@weekly /usr/local/bin/cleanup_logs.sh

# System update mensile
@monthly /usr/local/bin/system_update.sh

# Monitor ogni 5 minuti
*/5 * * * * /usr/local/bin/monitor.sh
```

9.5.3 Cron system-wide

```
# Directory system cron
/etc/cron.hourly/    # Script eseguiti ogni ora
/etc/cron.daily/     # Script eseguiti ogni giorno
/etc/cron.weekly/    # Script eseguiti ogni settimana
/etc/cron.monthly/   # Script eseguiti ogni mese

# Esempio: creare script in /etc/cron.daily/
sudo nano /etc/cron.daily/backup

#!/bin/bash
# Backup script
rsync -av /home/ /backup/home/

# Rendere eseguibile
sudo chmod +x /etc/cron.daily/backup

# File /etc/crontab (system-wide)
# Formato include campo username
# m h dom mon dow user command
0 2 * * * root /usr/local/bin/backup.sh
```

9.5.4 Script cron con logging

```
#!/bin/bash
# cron_backup.sh - Backup script con logging completo

LOGFILE="/var/log/backup.log"
BACKUP_SRC="/home/user/documents"
BACKUP_DEST="/backup/documents"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=30

log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" >> "$LOGFILE"
}

# Redirect output a log
```

```
exec 1>> "$LOGFILE"
exec 2>> "$LOGFILE"

log "=== Starting backup ==="

# Verifica directory sorgente
if [ ! -d "$BACKUP_SRC" ]; then
    log "ERROR: Source directory $BACKUP_SRC does not exist"
    exit 1
fi

# Crea directory destinazione se non esiste
mkdir -p "$BACKUP_DEST"

# Esegui backup
log "Backing up $BACKUP_SRC to $BACKUP_DEST"
if rsync -av --delete "$BACKUP_SRC/" "$BACKUP_DEST/"; then
    log "Backup completed successfully"
else
    log "ERROR: Backup failed with exit code $?"
    exit 1
fi

# Cleanup backup vecchi
log "Cleaning up backups older than $RETENTION_DAYS days"
find "$BACKUP_DEST" -type f -mtime +$RETENTION_DAYS -delete

# Invia notifica se errori
if grep -q ERROR "$LOGFILE"; then
    echo "Backup errors detected!" | mail -s "Backup Alert"
        admin@example.com
fi

log "=== Backup completed ==="
```

9.6 Systemd: Init System Moderno

9.6.1 Gestione Servizi

```
# Stato servizio
systemctl status nginx
systemctl status nginx.service  # equivalente

# Start/Stop/Restart
sudo systemctl start nginx
sudo systemctl stop nginx
sudo systemctl restart nginx
sudo systemctl reload nginx  # ricarica config senza restart

# Enable/Disable (avvio automatico)
sudo systemctl enable nginx   # avvia al boot
sudo systemctl disable nginx  # non avvia al boot
sudo systemctl enable --now nginx  # enable + start

# Verificare se abilitato
systemctl is-enabled nginx
systemctl is-active nginx
```

```
19
20 # Elencare servizi
21 systemctl list-units --type=service
22 systemctl list-units --type=service --state=running
23 systemctl list-units --type=service --state=failed
24
25 # Elencare tutti i unit files
26 systemctl list-unit-files
27
28 # Filtrare servizi
29 systemctl list-units --type=service | grep nginx
30
31 # Dependency tree
32 systemctl list-dependencies nginx
33
34 # Reload systemd daemon (dopo modifiche unit files)
35 sudo systemctl daemon-reload
```

9.6.2 Creare Unit File Custom

```
1  # File: /etc/systemd/system/myapp.service
2
3  [Unit]
4  Description=My Custom Application
5  After=network.target
6  Documentation=https://example.com/docs
7
8  [Service]
9  Type=simple
10 User=appuser
11 Group=appgroup
12 WorkingDirectory=/opt/myapp
13 ExecStart=/opt/myapp/bin/myapp --config /etc/myapp/config.yaml
14 ExecReload=/bin/kill -HUP $MAINPID
15 Restart=on-failure
16 RestartSec=5s
17
18 # Environment
19 Environment="NODE_ENV=production"
20 EnvironmentFile=-/etc/myapp/environment
21
22 # Logging
23 StandardOutput=journal
24 StandardError=journal
25 SyslogIdentifier=myapp
26
27 # Security
28 NoNewPrivileges=true
29 PrivateTmp=true
30
31 [Install]
32 WantedBy=multi-user.target
```

Tipi di servizio systemd

- **simple**: processo principale del servizio (default)
- **forking**: processo che fa fork e termina (daemon tradizionali)
- **oneshot**: eseguito una volta e completa
- **notify**: notifica a systemd quando pronto
- **dbus**: servizio acquisisce nome D-Bus

9.6.3 Systemd Timers (alternativa a cron)

```
# File: /etc/systemd/system/backup.service
[Unit]
Description=Backup Service

[Service]
Type=oneshot
ExecStart=/usr/local/bin/backup.sh

# File: /etc/systemd/system/backup.timer
[Unit]
Description=Backup Timer
Requires=backup.service

[Timer]
# Esegui alle 2:00 ogni giorno
OnCalendar=*-*-* 02:00:00
# Esegui 15 minuti dopo boot se mancato
Persistent=true

[Install]
WantedBy=timers.target

# Attivare timer
sudo systemctl enable backup.timer
sudo systemctl start backup.timer

# Verificare timer
systemctl list-timers
systemctl status backup.timer

# Esempi OnCalendar
OnCalendar=daily                    # 00:00:00 ogni giorno
OnCalendar=weekly                   # 00:00:00 ogni lunedì
OnCalendar=monthly                  # 00:00:00 primo del mese
OnCalendar=*-*-* 04:00:00          # 4 AM ogni giorno
OnCalendar=Mon *-*-* 09:00:00      # Lunedì alle 9 AM
OnCalendar=*-*-01 00:00:00         # Primo del mese
OnCalendar=*-01,07 00:00:00        # 1 Gennaio e 1 Luglio

# Intervalli
OnUnitActiveSec=5min                # 5 minuti dopo ultima attivazione
OnBootSec=10min                     # 10 minuti dopo boot
```

9.6.4 Journal: Log Management

```
# Visualizzare tutti i log
journalctl

# Log di un servizio specifico
journalctl -u nginx
journalctl -u nginx.service

# Follow (tempo reale)
journalctl -u nginx -f

# Ultime N righe
journalctl -u nginx -n 50

# Da timestamp
journalctl --since "2025-01-01 00:00:00"
journalctl --since yesterday
journalctl --since "1 hour ago"
journalctl --since 09:00 --until 17:00

# Range temporale
journalctl --since "2025-01-01" --until "2025-01-31"

# Solo errori
journalctl -p err
journalctl -p warning

# Priorità: emerg, alert, crit, err, warning, notice, info, debug
journalctl -p err -u nginx

# Boot corrente
journalctl -b

# Boot precedenti
journalctl --list-boots
journalctl -b -1  # boot precedente

# Kernel messages
journalctl -k

# Output format
journalctl -o json
journalctl -o json-pretty
journalctl -o verbose

# Disk usage
journalctl --disk-usage

# Cleanup
sudo journalctl --vacuum-time=7d    # più vecchi di 7 giorni
sudo journalctl --vacuum-size=500M  # mantieni max 500MB

# Configurazione persistenza
# File: /etc/systemd/journald.conf
[Journal]
Storage=persistent
SystemMaxUse=500M
```

```
SystemMaxFileSize=50M
MaxRetentionSec=1month
```

9.6.5 Monitoring e Status

```
# System status
systemctl status

# Failed units
systemctl --failed

# Targets (runlevels)
systemctl get-default
systemctl list-units --type=target

# Cambiare target
sudo systemctl isolate multi-user.target
sudo systemctl isolate graphical.target

# Shutdown/reboot
sudo systemctl poweroff
sudo systemctl reboot
sudo systemctl suspend
sudo systemctl hibernate

# Analisi boot time
systemd-analyze
systemd-analyze blame  # servizi più lenti
systemd-analyze critical-chain  # catena critica
systemd-analyze plot > boot.svg  # grafico SVG
```

9.7 Log di Sistema

9.7.1 File di Log Tradizionali

```
# Principali file log
/var/log/syslog       # Log generale sistema (Debian/Ubuntu)
/var/log/messages     # Log generale (RedHat/CentOS)
/var/log/auth.log     # Autenticazione
/var/log/kern.log     # Kernel
/var/log/boot.log     # Boot
/var/log/dmesg        # Device messages
/var/log/apache2/     # Apache logs
/var/log/nginx/       # Nginx logs

# Visualizzare log in real-time
tail -f /var/log/syslog
tail -f /var/log/auth.log

# Cercare in log
grep "error" /var/log/syslog
grep -i "failed" /var/log/auth.log

# Analisi tentativi login falliti
sudo grep "Failed password" /var/log/auth.log
```

```
21
22 # IP con più tentativi falliti
23 sudo grep "Failed password" /var/log/auth.log | \
24     awk '{print $(NF-3)}' | sort | uniq -c | sort -rn
25
26 # Log Apache/Nginx
27 tail -f /var/log/nginx/access.log
28 tail -f /var/log/nginx/error.log
29
30 # Contare status code HTTP
31 awk '{print $9}' /var/log/nginx/access.log | sort | uniq -c | sort -rn
32
33 # Top 10 IP visitatori
34 awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn |
     head -10
```

9.7.2 Logrotate

```
1  # Configurazione: /etc/logrotate.conf
2  # Configurazioni specifiche: /etc/logrotate.d/
3
4  # Esempio: /etc/logrotate.d/myapp
5  /var/log/myapp/*.log {
6      daily
7      rotate 7
8      compress
9      delaycompress
10     missingok
11     notifempty
12     create 0640 appuser appgroup
13     sharedscripts
14     postrotate
15         systemctl reload myapp > /dev/null 2>&1 || true
16     endscript
17 }
18
19 # Opzioni comuni
20 # daily/weekly/monthly: frequenza rotazione
21 # rotate N: mantieni N file ruotati
22 # compress: comprimi file vecchi
23 # delaycompress: comprimi al prossimo ciclo
24 # missingok: non errore se file manca
25 # notifempty: non ruotare se vuoto
26 # create mode owner group: permessi nuovo file
27 # postrotate/endscript: comandi dopo rotazione
28
29 # Test configurazione
30 sudo logrotate -d /etc/logrotate.d/myapp
31
32 # Forzare rotazione
33 sudo logrotate -f /etc/logrotate.conf
```

9.8 Monitoraggio Risorse

```
1  # CPU e processi
```

```
top
htop  # più user-friendly

# Memoria
free -h
vmstat 1  # ogni secondo

# Disco
df -h         # spazio disco
du -sh *      # uso directory
du -sh /* | sort -rh | head -10  # top 10 directory

# I/O disco
iostat
iotop  # richiede root

# Network
iftop   # traffico per connessione
nethogs # traffico per processo

# Processo specifico
ps aux | grep nginx
pgrep nginx
pidof nginx

# Resource usage di processo
ps -p PID -o %cpu,%mem,cmd
top -p PID
```

9.9 Best Practice Amministrazione

Best Practice Sicurezza

1. **Principio minimo privilegio**: dare solo permessi necessari
2. **Sudo invece di root**: non usare account root direttamente
3. **Password forti**: policy di password complesse
4. **SSH key auth**: disabilitare password SSH in produzione
5. **Audit regolari**: verificare utenti, gruppi, permessi SUID
6. **Log centralized**: aggregare log per analisi
7. **Updates automatici**: patch sicurezza tempestive
8. **Backup**: strategia 3-2-1 (3 copie, 2 media, 1 off-site)
9. **Monitoring**: alert proattivi su anomalie
10. **Documentation**: documentare modifiche e configurazioni

9.10 Esercizi Pratici

1. Creare 3 utenti (alice, bob, charlie) con home directory e shell bash.

2. Creare gruppi "developers" e "admins", assegnare utenti appropriatamente.
3. Configurare ACL su directory condivisa accessibile solo a gruppo developers.
4. Creare cron job che esegue backup ogni giorno alle 3 AM.
5. Creare systemd service per applicazione custom.
6. Creare systemd timer che esegue cleanup ogni domenica.
7. Analizzare log per trovare IP con più tentativi login falliti.
8. Configurare logrotate per ruotare log applicazione settimanalmente.
9. Implementare script monitoring che invia alert se disco $> 80\%$.
10. Usare journalctl per debuggare servizio che non si avvia.

9.11 Script Completo: System Health Check

```bash
#!/bin/bash
# system_health.sh - Controllo completo salute sistema

REPORT="/var/log/system_health_$(date +%Y%m%d).log"
ALERT_EMAIL="admin@example.com"
DISK_THRESHOLD=80
CPU_THRESHOLD=80
MEM_THRESHOLD=80

exec > >(tee "$REPORT")
exec 2>&1

echo "=== System Health Check - $(date) ==="

# CPU Usage
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print $2}' | cut -d'%' -f1)
echo "CPU Usage: ${CPU_USAGE}%"
if (( $(echo "$CPU_USAGE > $CPU_THRESHOLD" | bc -l) )); then
    echo "WARNING: CPU usage above threshold!"
fi

# Memory Usage
MEM_USAGE=$(free | grep Mem | awk '{printf("%.0f", $3/$2 * 100.0)}')
echo "Memory Usage: ${MEM_USAGE}%"
if [ "$MEM_USAGE" -gt "$MEM_THRESHOLD" ]; then
    echo "WARNING: Memory usage above threshold!"
fi

# Disk Usage
echo -e "\nDisk Usage:"
df -h | grep -vE '^Filesystem|tmpfs|cdrom' | while read line; do
    usage=$(echo $line | awk '{print $5}' | sed 's/%//')
    partition=$(echo $line | awk '{print $1}')
    if [ "$usage" -gt "$DISK_THRESHOLD" ]; then
        echo "WARNING: $partition at ${usage}% (threshold: ${
            DISK_THRESHOLD}%)"
    fi
done
```

```
38
39 # Failed Services
40 echo -e "\nFailed Services:"
41 FAILED=$(systemctl --failed --no-pager)
42 if [ -n "$FAILED" ]; then
43     echo "$FAILED"
44 fi
45
46 # Recent Login Failures
47 echo -e "\nRecent Failed Logins (last 24h):"
48 journalctl --since "24 hours ago" | grep "Failed password" | tail -5
49
50 # Load Average
51 echo -e "\nLoad Average:"
52 uptime
53
54 # Top 5 CPU processes
55 echo -e "\nTop 5 CPU Processes:"
56 ps aux --sort=-%cpu | head -6
57
58 # Top 5 Memory processes
59 echo -e "\nTop 5 Memory Processes:"
60 ps aux --sort=-%mem | head -6
61
62 echo -e "\n=== Health Check Completed ==="
63
64 # Send alert if warnings found
65 if grep -q "WARNING" "$REPORT"; then
66     mail -s "System Health Alert" "$ALERT_EMAIL" < "$REPORT"
67 fi
```

9.12 Riepilogo

Hai imparato a:

- Gestire utenti e gruppi con useradd, usermod, groupadd
- Configurare permessi avanzati con ACL e permessi speciali
- Automatizzare task con cron e systemd timers
- Gestire servizi con systemd
- Analizzare log con journalctl e file tradizionali
- Monitorare risorse e salute del sistema

9.13 Riferimenti

- https://www.freedesktop.org/software/systemd/man/
- https://man7.org/linux/man-pages/man1/journalctl.1.html
- https://man7.org/linux/man-pages/man8/useradd.8.html
- https://man7.org/linux/man-pages/man5/crontab.5.html
- https://linux.die.net/man/1/setfacl

Capitolo 10

SSH e Sicurezza

10.1 Introduzione

SSH (Secure Shell) è il protocollo standard per l'accesso remoto sicuro ai sistemi Linux. Oltre alla semplice connessione, SSH offre funzionalità avanzate come autenticazione tramite chiavi crittografiche, tunneling, port forwarding e trasferimento file sicuro.

In questo capitolo approfondiremo la generazione e gestione delle chiavi SSH, configurazioni avanzate, tecniche di tunneling e best practice di sicurezza per proteggere i sistemi.

Mappa del capitolo

Sezioni: SSH keys (generazione, gestione), Configurazione client/server, Port forwarding e tunneling, SSH Agent, ProxyJump, Hardening SSH server, Firewall, Fail2ban, Best practice sicurezza.

10.2 Obiettivi di Apprendimento

- Generare e gestire coppie di chiavi SSH (RSA, Ed25519).
- Configurare autenticazione senza password.
- Utilizzare SSH config per gestire connessioni multiple.
- Implementare port forwarding e tunneling SSH.
- Configurare e hardening SSH server.
- Proteggere sistema con firewall e fail2ban.
- Implementare best practice di sicurezza.

10.3 SSH Keys: Autenticazione a Chiave Pubblica

10.3.1 Generazione Chiavi SSH

```
# Generare chiave RSA (4096 bit)
ssh-keygen -t rsa -b 4096 -C "user@email.com"

# Generare chiave Ed25519 (raccomandato, più sicuro e veloce)
ssh-keygen -t ed25519 -C "user@email.com"

# Specificare nome file
```

```
ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_github -C "github@email.com"

# Generare senza passphrase (NON raccomandato)
ssh-keygen -t ed25519 -N "" -f ~/.ssh/id_ed25519_test

# Cambiare passphrase di chiave esistente
ssh-keygen -p -f ~/.ssh/id_ed25519

# Visualizzare fingerprint chiave
ssh-keygen -lf ~/.ssh/id_ed25519.pub
ssh-keygen -lf ~/.ssh/id_ed25519 -E md5
ssh-keygen -lf ~/.ssh/id_ed25519 -E sha256

# Visualizzare formato ASCII art
ssh-keygen -lvf ~/.ssh/id_ed25519.pub
```

Tipi di chiavi SSH

- **RSA**: tradizionale, supportato ovunque (min 2048 bit, raccomandato 4096)
- **Ed25519**: moderno, più sicuro, più veloce, chiavi più piccole (raccomandato)
- **ECDSA**: elliptic curve, buono ma controversie su implementazione
- **DSA**: deprecato, NON usare

Raccomandazione: usare Ed25519 quando possibile, altrimenti RSA 4096 bit.

10.3.2 Copiare Chiave Pubblica su Server

```
# Metodo 1: ssh-copy-id (più semplice)
ssh-copy-id user@server

# Con chiave specifica
ssh-copy-id -i ~/.ssh/id_ed25519.pub user@server

# Con porta custom
ssh-copy-id -i ~/.ssh/id_ed25519.pub -p 2222 user@server

# Metodo 2: manuale
cat ~/.ssh/id_ed25519.pub | ssh user@server "mkdir -p ~/.ssh && \
    chmod 700 ~/.ssh && cat >> ~/.ssh/authorized_keys && \
    chmod 600 ~/.ssh/authorized_keys"

# Metodo 3: copiare direttamente (se hai già accesso)
scp ~/.ssh/id_ed25519.pub user@server:~/temp_key.pub
ssh user@server
mkdir -p ~/.ssh
cat ~/temp_key.pub >> ~/.ssh/authorized_keys
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
rm ~/temp_key.pub
```

Permessi corretti per SSH

IMPORTANTE: SSH è molto rigido sui permessi per sicurezza.

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub
chmod 600 ~/.ssh/authorized_keys
chmod 600 ~/.ssh/config
```

Se i permessi non sono corretti, SSH rifiuterà di usare le chiavi!

10.3.3 Gestione Multiple Chiavi

```
# Struttura tipica ~/.ssh/
~/.ssh/
          id_ed25519          # Chiave privata generale
          id_ed25519.pub      # Chiave pubblica generale
          id_rsa_github       # Chiave privata GitHub
          id_rsa_github.pub   # Chiave pubblica GitHub
          id_ed25519_work     # Chiave privata lavoro
          id_ed25519_work.pub # Chiave pubblica lavoro
          authorized_keys     # Chiavi autorizzate (server)
          known_hosts         # Host fidati
          config              # Configurazione SSH client

# Usare chiave specifica
ssh -i ~/.ssh/id_ed25519_work user@work-server

# Aggiungere chiave all'SSH agent
ssh-add ~/.ssh/id_ed25519_work
ssh-add -l  # Lista chiavi caricate
ssh-add -D  # Rimuovi tutte le chiavi
```

10.4 SSH Config: Configurazione Avanzata

10.4.1 File `~/.ssh/config`

```
# File: ~/.ssh/config
# Configurazione SSH client

# Configurazione di default per tutti gli host
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3
    Compression yes
    AddKeysToAgent yes

# Server personale
Host myserver
    HostName 192.168.1.100
    User admin
    Port 22
    IdentityFile ~/.ssh/id_ed25519
    ForwardAgent yes

```

```
# Server con porta custom
Host vps
    HostName vps.example.com
    User root
    Port 2222
    IdentityFile ~/.ssh/id_ed25519_vps

# Jump host (bastion)
Host jump
    HostName jump.company.com
    User jumper
    IdentityFile ~/.ssh/id_rsa_work

# Server interno accessibile via jump host
Host internal
    HostName 10.0.0.50
    User developer
    ProxyJump jump
    # Alternativa vecchia sintassi
    # ProxyCommand ssh -W %h:%p jump

# GitHub
Host github.com
    User git
    IdentityFile ~/.ssh/id_rsa_github
    IdentitiesOnly yes

# Pattern matching multipli server
Host server-*
    User admin
    IdentityFile ~/.ssh/id_ed25519_work
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null

# Database server con port forwarding automatico
Host db
    HostName db.internal.company.com
    User dbadmin
    LocalForward 3306 localhost:3306
    IdentityFile ~/.ssh/id_ed25519_work
```

Opzioni SSH Config comuni

- **HostName**: indirizzo IP o hostname reale
- **User**: username per login
- **Port**: porta SSH (default 22)
- **IdentityFile**: percorso chiave privata
- **ForwardAgent**: forwarding SSH agent
- **ProxyJump**: server jump/bastion
- **LocalForward**: port forwarding locale
- **RemoteForward**: port forwarding remoto

- **ServerAliveInterval**: keep-alive ping
- **Compression**: abilita compressione

10.4.2 Utilizzo dopo Configurazione

```
# Con config, connessione semplificata
ssh myserver  # invece di: ssh -p 22 admin@192.168.1.100

ssh vps       # invece di: ssh -p 2222 root@vps.example.com

ssh internal  # usa automaticamente jump host

# Copiare file usando alias
scp file.txt myserver:/path/to/destination/

# Rsync con alias
rsync -av documents/ myserver:/backup/
```

10.5 SSH Agent: Gestione Chiavi in Memoria

```
# Avviare SSH agent
eval "$(ssh-agent -s)"

# Verificare se agent è in esecuzione
echo $SSH_AGENT_PID
ssh-add -l

# Aggiungere chiave all'agent
ssh-add ~/.ssh/id_ed25519
# Inserire passphrase una volta

# Aggiungere con timeout (secondi)
ssh-add -t 3600 ~/.ssh/id_ed25519  # 1 ora

# Elencare chiavi caricate
ssh-add -l
ssh-add -L  # mostra chiavi pubbliche complete

# Rimuovere chiave specifica
ssh-add -d ~/.ssh/id_ed25519

# Rimuovere tutte le chiavi
ssh-add -D

# Configurazione permanente in ~/.bashrc o ~/.zshrc
if [ -z "$SSH_AUTH_SOCK" ]; then
   eval "$(ssh-agent -s)"
   ssh-add ~/.ssh/id_ed25519 2>/dev/null
fi

# macOS: usare Keychain
# Aggiungere a ~/.ssh/config
Host *
    AddKeysToAgent yes
```

```
    UseKeychain yes
    IdentityFile ~/.ssh/id_ed25519
```

Agent Forwarding

Agent forwarding permette di usare chiavi locali su server remoto.
ATTENZIONE: abilita solo su server fidati! Root del server remoto può accedere alle tue chiavi.

```
# Connessione con agent forwarding
ssh -A user@server

# In config
Host trusted
    HostName trusted.example.com
    ForwardAgent yes
```

10.6 Port Forwarding e Tunneling

10.6.1 Local Port Forwarding

Redirige porta locale a porta su server remoto.

```
# Sintassi: ssh -L [local_addr:]local_port:destination:destination_port
    server

# Accedere a database remoto
ssh -L 3306:localhost:3306 user@dbserver
# Ora: mysql -h localhost -P 3306 connette a dbserver

# Accedere a web server remoto
ssh -L 8080:localhost:80 user@webserver
# Ora: http://localhost:8080 mostra webserver remoto

# Bind su interfaccia specifica
ssh -L 0.0.0.0:8080:localhost:80 user@server  # accessibile da rete
ssh -L 127.0.0.1:8080:localhost:80 user@server  # solo localhost

# Accedere a host interno via jump server
ssh -L 3389:internal-windows:3389 user@jumpserver
# Connette RDP a internal-windows passando per jumpserver

# Multipli forwarding
ssh -L 8080:localhost:80 -L 3306:localhost:3306 user@server

# Background
ssh -fN -L 8080:localhost:80 user@server
# -f: background
# -N: no remote command

# Script tunneling automatico
#!/bin/bash
# tunnel.sh
ssh -fN -L 8080:localhost:80 \
        -L 3306:localhost:3306 \
        -L 5432:localhost:5432 \
```

```
        user@server
echo "Tunnels established"
```

10.6.2 Remote Port Forwarding

Redirige porta su server remoto a porta locale.

```
# Sintassi: ssh -R [remote_addr:]remote_port:destination:
   destination_port server

# Esporre web server locale su server remoto
ssh -R 8080:localhost:3000 user@publicserver
# Ora: publicserver:8080 mostra localhost:3000

# Esporre servizio dietro NAT
ssh -R 2222:localhost:22 user@publicserver
# Permette SSH reverse: ssh -p 2222 user@publicserver

# Allow remote bind (config server)
# /etc/ssh/sshd_config
# GatewayPorts yes

# Esempio: demo app locale su server pubblico
ssh -R 80:localhost:8000 user@demo-server
# Utile per mostrare lavori in progress
```

10.6.3 Dynamic Port Forwarding (SOCKS Proxy)

Crea proxy SOCKS per tunneling tutto il traffico.

```
# Creare SOCKS proxy
ssh -D 8080 user@server

# Configurare browser:
# Proxy SOCKS: localhost:8080

# Con Firefox: Preferences > Network Settings
# Manual proxy: SOCKS Host: localhost, Port: 8080, SOCKS v5

# Con curl
curl --socks5 localhost:8080 https://api.example.com

# Script proxy SSH
#!/bin/bash
# socks_proxy.sh
PORT=8080
SERVER="user@proxy-server"

ssh -fN -D $PORT $SERVER

echo "SOCKS proxy running on localhost:$PORT"
echo "Configure your browser or use:"
echo "curl --socks5 localhost:$PORT https://example.com"

# Chiudere tunnel
pkill -f "ssh -fN -D $PORT"
```

Casi d'uso Port Forwarding

1. **Database access**: accedere a DB in rete privata
2. **Web development**: testare app locale su dominio pubblico
3. **Bypass firewall**: accedere a servizi bloccati
4. **Sicurezza**: crittografare connessioni non sicure
5. **NAT traversal**: esporre servizi dietro NAT/firewall
6. **VPN alternativa**: SOCKS proxy come lightweight VPN

10.7 Configurazione e Hardening SSH Server

10.7.1 File `/etc/ssh/sshd_config`

```
# File: /etc/ssh/sshd_config
# Configurazione SSH server (richiede root)

# Porta SSH (cambiare da default 22)
Port 2222

# Indirizzo bind (specifica interfaccia)
ListenAddress 0.0.0.0
# ListenAddress 192.168.1.100  # solo interfaccia specifica

# Protocol 2 (SSH-2 solo, mai SSH-1)
Protocol 2

# Log level
SyslogFacility AUTH
LogLevel VERBOSE  # per audit, altrimenti INFO

# Autenticazione
PermitRootLogin no  # IMPORTANTE: mai permettere root diretto
PubkeyAuthentication yes
PasswordAuthentication no  # Solo chiavi, no password
PermitEmptyPasswords no
ChallengeResponseAuthentication no

# Limitare utenti/gruppi
AllowUsers user1 user2 admin
# AllowGroups ssh-users
# DenyUsers baduser
# DenyGroups notssh

# Autenticazione key
AuthorizedKeysFile .ssh/authorized_keys

# Timeouts
LoginGraceTime 60
ClientAliveInterval 300
ClientAliveCountMax 2

# Limiti connessioni
```

```
MaxAuthTries 3
MaxSessions 10
MaxStartups 10:30:60

# Forwarding
AllowTcpForwarding yes
GatewayPorts no
X11Forwarding no  # disabilita se non necessario
AllowAgentForwarding yes

# Banner (messaggio pre-login)
Banner /etc/ssh/banner.txt

# Subsystem (per SFTP)
Subsystem sftp /usr/lib/openssh/sftp-server

# Match blocks per configurazioni specifiche
Match User deployer
    AllowTcpForwarding no
    X11Forwarding no
    PermitTTY no
    ForceCommand internal-sftp

Match Address 192.168.1.*
    PasswordAuthentication yes
```

10.7.2 Applicare e Verificare Configurazione

```
# Testare configurazione (non riavvia)
sudo sshd -t

# Verificare sintassi con output dettagliato
sudo sshd -T

# Riavviare SSH service
sudo systemctl restart sshd
sudo systemctl restart ssh  # Debian/Ubuntu

# Verificare status
sudo systemctl status sshd

# Verificare porta in ascolto
sudo ss -tlpn | grep ssh
sudo netstat -tlpn | grep ssh

# Log SSH in real-time
sudo journalctl -u sshd -f
sudo tail -f /var/log/auth.log
```

ATTENZIONE: Testing SSH Config

IMPORTANTE: Prima di chiudere sessione SSH corrente, testa la nuova configurazione in una *nuova* sessione!

1. Modifica `/etc/ssh/sshd_config`
2. Test: `sudo sshd -t`

3. Riavvia: `sudo systemctl restart sshd`
4. Apri **nuova** sessione SSH per testare
5. Solo dopo test OK, chiudi vecchia sessione

Altrimenti rischi di bloccarti fuori dal server!

10.8 Firewall: UFW e iptables

10.8.1 UFW (Uncomplicated Firewall)

```
# Installare UFW (Ubuntu/Debian)
sudo apt install ufw

# Status
sudo ufw status
sudo ufw status verbose
sudo ufw status numbered

# Abilitare/Disabilitare
sudo ufw enable
sudo ufw disable

# Default policy
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Permettere SSH (PRIMA di enable!)
sudo ufw allow ssh
sudo ufw allow 22/tcp

# SSH su porta custom
sudo ufw allow 2222/tcp

# Permettere da IP specifico
sudo ufw allow from 192.168.1.100
sudo ufw allow from 192.168.1.0/24

# Permettere porta da IP specifico
sudo ufw allow from 192.168.1.100 to any port 22

# Permettere servizi comuni
sudo ufw allow http
sudo ufw allow https
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp

# Range porte
sudo ufw allow 6000:6007/tcp

# Negare porta
sudo ufw deny 23/tcp  # telnet

# Eliminare regola
sudo ufw delete allow 80/tcp
```

```
sudo ufw delete 3  # per numero (da status numbered)

# Limitare tentativi connessione (rate limiting)
sudo ufw limit ssh  # max 6 connessioni in 30 secondi

# Reset (elimina tutte le regole)
sudo ufw reset

# Log
sudo ufw logging on
sudo ufw logging medium
sudo tail -f /var/log/ufw.log

# Configurazione completa esempio
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw limit 2222/tcp comment 'SSH custom port'
sudo ufw allow 80/tcp comment 'HTTP'
sudo ufw allow 443/tcp comment 'HTTPS'
sudo ufw allow from 192.168.1.0/24 comment 'Local network'
sudo ufw enable
```

10.8.2 iptables (low-level)

```
# Visualizzare regole
sudo iptables -L -n -v
sudo iptables -L INPUT -n -v
sudo iptables -t nat -L -n -v

# Permettere SSH
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# Permettere traffico locale
sudo iptables -A INPUT -i lo -j ACCEPT

# Permettere connessioni stabilite
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# HTTP/HTTPS
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Da IP specifico
sudo iptables -A INPUT -s 192.168.1.100 -j ACCEPT

# Rate limiting (protezione brute force)
sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW \
    -m recent --set --name SSH
sudo iptables -A INPUT -p tcp --dport 22 -m state --state NEW \
    -m recent --update --seconds 60 --hitcount 4 --name SSH -j DROP

# Default policy
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT ACCEPT

# Eliminare regola
```

```
sudo iptables -D INPUT -p tcp --dport 80 -j ACCEPT

# Flush tutte le regole
sudo iptables -F

# Salvare regole (persistent)
# Debian/Ubuntu
sudo apt install iptables-persistent
sudo netfilter-persistent save

# RedHat/CentOS
sudo service iptables save
```

10.9 Fail2ban: Protezione Brute Force

```
# Installare fail2ban
sudo apt install fail2ban  # Debian/Ubuntu
sudo yum install fail2ban  # RedHat/CentOS

# Avviare servizio
sudo systemctl enable fail2ban
sudo systemctl start fail2ban

# Status
sudo systemctl status fail2ban
sudo fail2ban-client status
sudo fail2ban-client status sshd

# Configurazione: /etc/fail2ban/jail.local
# NON modificare jail.conf, creare jail.local

sudo nano /etc/fail2ban/jail.local
```

```
# File: /etc/fail2ban/jail.local

[DEFAULT]
# Ban per 1 ora
bantime = 3600
# Finestra 10 minuti
findtime = 600
# Max 5 tentativi
maxretry = 5

# Email notifiche
destemail = admin@example.com
sendername = Fail2Ban
mta = sendmail
action = %(action_mwl)s

[sshd]
enabled = true
port = 2222  # porta SSH custom
logpath = /var/log/auth.log
maxretry = 3
bantime = 7200  # 2 ore

```

```
[sshd-ddos]
enabled = true
port = 2222
logpath = /var/log/auth.log
maxretry = 2

[nginx-http-auth]
enabled = true
filter = nginx-http-auth
port = http,https
logpath = /var/log/nginx/error.log

[nginx-noscript]
enabled = true
port = http,https
filter = nginx-noscript
logpath = /var/log/nginx/access.log
maxretry = 6

[nginx-badbots]
enabled = true
port = http,https
filter = nginx-badbots
logpath = /var/log/nginx/access.log
maxretry = 2
```

```
# Applicare configurazione
sudo systemctl restart fail2ban

# Verificare jail attivi
sudo fail2ban-client status

# Status jail specifico
sudo fail2ban-client status sshd

# Unbannare IP
sudo fail2ban-client set sshd unbanip 192.168.1.100

# Bannare IP manualmente
sudo fail2ban-client set sshd banip 1.2.3.4

# Log fail2ban
sudo tail -f /var/log/fail2ban.log

# IP bannati (in iptables)
sudo iptables -L -n | grep -i reject
```

10.10 Best Practice Sicurezza

SSH Security Best Practice

1. **Chiavi invece password**: sempre autenticazione a chiave pubblica
2. **Passphrase forte**: proteggere chiave privata con passphrase robusta
3. **Ed25519**: usare algoritmo moderno ed25519

4. **No root login**: `PermitRootLogin no`
5. **Porta custom**: cambiare da default 22 (riduce rumore log)
6. **Whitelist utenti**: `AllowUsers` per limitare accesso
7. **Fail2ban**: protezione automatica brute force
8. **Firewall**: limitare accesso SSH a IP fidati quando possibile
9. **2FA**: implementare two-factor authentication (Google Authenticator)
10. **Audit regolari**: monitorare `/var/log/auth.log` e `/.ssh/authorized_keys`
11. **Aggiornamenti**: mantenere OpenSSH aggiornato
12. **Backup chiavi**: backup sicuro chiavi private (offline, criptato)

System Security Checklist

- Aggiornamenti automatici sicurezza abilitati
- Firewall configurato e attivo
- SSH: solo chiavi, no password
- SSH: root login disabilitato
- Fail2ban installato e configurato
- Utenti: sudo solo a chi necessario
- Audit file SUID/SGID regolare
- Log monitoring attivo
- Backup regolari e testati
- Intrusion detection (AIDE, Tripwire)
- SELinux o AppArmor abilitato
- Porte non necessarie chiuse

10.11 Two-Factor Authentication (2FA)

```
# Installare Google Authenticator PAM
sudo apt install libpam-google-authenticator

# Configurare per utente
google-authenticator
# Rispondere domande:
# - Time-based tokens: y
# - Update .google_authenticator: y
# - Disallow reuse: y
# - Rate limiting: y
# - Time skew: n (o y se problemi sincronizzazione)

```

```
# Scan QR code con app mobile (Google Authenticator, Authy)

# Configurare SSH PAM
sudo nano /etc/pam.d/sshd

# Aggiungere in fondo:
auth required pam_google_authenticator.so

# Configurare SSH daemon
sudo nano /etc/ssh/sshd_config

# Modificare/aggiungere:
ChallengeResponseAuthentication yes
UsePAM yes

# Restart SSH
sudo systemctl restart sshd

# Test: nuova connessione richiederà verification code
```

10.12 Monitoring e Auditing

```
# Login correnti
who
w
last
lastlog

# Tentativi login falliti
sudo lastb
sudo grep "Failed password" /var/log/auth.log

# IP con più tentativi falliti
sudo grep "Failed password" /var/log/auth.log | \
    awk '{print $(NF-3)}' | sort | uniq -c | sort -rn | head -10

# Sessioni SSH attive
who | grep pts
ss -tnp | grep sshd

# File accessi unauthorized
sudo cat ~/.ssh/authorized_keys  # verificare chiavi
sudo find / -name authorized_keys 2>/dev/null

# File SUID potenzialmente pericolosi
sudo find / -perm -4000 -type f -ls 2>/dev/null

# Port scan detection (monitorare tentativi)
sudo journalctl -u sshd | grep "Connection from"

# Script monitoring SSH
#!/bin/bash
# ssh_monitor.sh
echo "=== SSH Security Check ==="
echo ""
echo "Current SSH Sessions:"
```

```
who | grep pts
echo ""
echo "Recent Failed Logins:"
sudo lastb | head -10
echo ""
echo "Failed Password Attempts (last 24h):"
sudo journalctl --since "24 hours ago" | \
    grep "Failed password" | wc -l
echo ""
echo "Banned IPs (fail2ban):"
sudo fail2ban-client status sshd | grep "Banned IP"
```

10.13 Esercizi Pratici

1. Generare coppia di chiavi Ed25519 e configurare autenticazione senza password.
2. Configurare `/.ssh/config` per 3 server diversi con alias.
3. Implementare jump host per accedere a server interno.
4. Configurare local port forwarding per accedere a database remoto.
5. Creare SOCKS proxy SSH e configurare browser.
6. Hardening SSH server: porta custom, no root, no password.
7. Configurare UFW per permettere solo SSH, HTTP, HTTPS.
8. Installare e configurare fail2ban per SSH.
9. Implementare 2FA con Google Authenticator.
10. Creare script monitoring che invia alert su tentativi brute force.

10.14 Script Completo: SSH Security Audit

```
#!/bin/bash
# ssh_security_audit.sh - Audit completo sicurezza SSH

REPORT="/var/log/ssh_audit_$(date +%Y%m%d).log"

exec > >(tee "$REPORT")
exec 2>&1

echo "=== SSH Security Audit - $(date) ==="

# Check SSH config
echo -e "\n[1] SSH Server Configuration:"
if sudo grep -q "^PermitRootLogin no" /etc/ssh/sshd_config; then
    echo "    Root login disabled"
else
    echo "    WARNING: Root login enabled!"
fi

if sudo grep -q "^PasswordAuthentication no" /etc/ssh/sshd_config; then
    echo "    Password authentication disabled"
```

```
else
    echo "    WARNING: Password authentication enabled!"
fi

if sudo grep -q "^Port [0-9]" /etc/ssh/sshd_config; then
    PORT=$(sudo grep "^Port" /etc/ssh/sshd_config | awk '{print $2}')
    echo "    SSH port: $PORT"
    if [ "$PORT" == "22" ]; then
        echo "      Consider using non-default port"
    fi
else
    echo "    SSH on default port 22"
fi

# Check firewall
echo -e "\n[2] Firewall Status:"
if sudo ufw status | grep -q "Status: active"; then
    echo "    UFW firewall active"
    sudo ufw status numbered | grep -E "22|ssh"
else
    echo "    WARNING: UFW firewall inactive!"
fi

# Check fail2ban
echo -e "\n[3] Fail2ban Status:"
if systemctl is-active --quiet fail2ban; then
    echo "    Fail2ban active"
    sudo fail2ban-client status sshd 2>/dev/null || echo "    SSH jail
       not configured"
else
    echo "    WARNING: Fail2ban not running!"
fi

# Recent failed logins
echo -e "\n[4] Failed Login Attempts (last 24h):"
FAILED_COUNT=$(sudo journalctl --since "24 hours ago" | grep "Failed
   password" | wc -l)
echo "Total failed attempts: $FAILED_COUNT"

if [ "$FAILED_COUNT" -gt 10 ]; then
    echo "    High number of failed attempts detected!"
    echo "Top 5 attacking IPs:"
    sudo journalctl --since "24 hours ago" | \
        grep "Failed password" | \
        awk '{print $(NF-3)}' | \
        sort | uniq -c | sort -rn | head -5
fi

# Check authorized_keys
echo -e "\n[5] Authorized Keys Check:"
for home in /home/*; do
    user=$(basename "$home")
    authkeys="$home/.ssh/authorized_keys"
    if [ -f "$authkeys" ]; then
        count=$(wc -l < "$authkeys")
        echo "User $user: $count authorized key(s)"
        # Check permissions
        perms=$(stat -c %a "$authkeys")
```

```
        if [ "$perms" != "600" ]; then
            echo "      WARNING: Wrong permissions: $perms (should be
                600)"
        fi
    fi
done

# Check SUID files
echo -e "\n[6] SUID Files Check:"
SUID_COUNT=$(sudo find / -perm -4000 -type f 2>/dev/null | wc -l)
echo "Total SUID files: $SUID_COUNT"

# Updates
echo -e "\n[7] System Updates:"
if command -v apt &> /dev/null; then
    UPDATES=$(apt list --upgradable 2>/dev/null | grep -c upgradable)
    echo "Available updates: $UPDATES"
    if [ "$UPDATES" -gt 0 ]; then
        echo "    System updates available"
    fi
fi

echo -e "\n=== Audit Completed ==="
echo "Report saved to: $REPORT"

# Send alert if critical issues
if grep -qE "   |WARNING" "$REPORT"; then
    echo "Critical issues found! Sending alert..."
    # mail -s "SSH Security Alert" admin@example.com < "$REPORT"
fi
```

10.15 Riepilogo

Hai imparato:

- Generare e gestire chiavi SSH (RSA, Ed25519)
- Configurare autenticazione senza password
- Usare SSH config per gestione server multipli
- Implementare port forwarding (local, remote, dynamic)
- Configurare e hardening SSH server
- Proteggere sistema con firewall (UFW, iptables)
- Implementare fail2ban contro brute force
- Best practice sicurezza e auditing

10.16 Riferimenti

- https://www.openssh.com/manual.html
- https://man.openbsd.org/sshd_config

- https://help.ubuntu.com/community/UFW
- https://www.fail2ban.org/
- https://www.ssh.com/academy/ssh/tunneling
- NIST SSH Guidelines: https://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7966.pdf

Capitolo 11

Automazione e Scripting Avanzato

11.1 Introduzione

L'automazione è il cuore dell'amministrazione di sistema efficiente. Script ben progettati eliminano task ripetitivi, riducono errori umani e permettono monitoraggio proattivo dell'infrastruttura.

In questo capitolo esploreremo tecniche avanzate di Bash scripting, pattern comuni di automazione, gestione errori robusta, logging professionale, e integrazione con sistemi di monitoring e alerting.

Mappa del capitolo

Sezioni: Bash scripting avanzato, Gestione errori, Logging, Parsing output, Automazione deployment, Monitoring e alerting, Backup automation, Task scheduling avanzato, Best practice.

11.2 Obiettivi di Apprendimento

- Scrivere script Bash robusti e manutenibili.
- Implementare gestione errori professionale.
- Creare sistemi di logging strutturato.
- Automatizzare deployment e configurazioni.
- Implementare monitoring e alerting automatici.
- Creare pipeline di backup affidabili.
- Applicare best practice DevOps.

11.3 Bash Scripting Avanzato

11.3.1 Template Script Professionale

```
#!/bin/bash
#
# Script: system_backup.sh
# Description: Automated system backup with error handling and logging
# Author: Admin Team
# Version: 1.2.0
# Last Modified: 2025-01-15
```

```
#
# Usage: ./system_backup.sh [options]
# Options:
#   -d, --dir DIR       Backup directory (default: /backup)
#   -v, --verbose       Verbose output
#   -h, --help          Show this help
#

# Strict mode
set -euo pipefail
IFS=$'\n\t'

# Constants
readonly SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
readonly SCRIPT_NAME="$(basename "$0")"
readonly LOG_FILE="/var/log/${SCRIPT_NAME%.sh}.log"
readonly PID_FILE="/var/run/${SCRIPT_NAME%.sh}.pid"

# Default values
BACKUP_DIR="/backup"
VERBOSE=false

# Colors for output
readonly RED='\033[0;31m'
readonly GREEN='\033[0;32m'
readonly YELLOW='\033[1;33m'
readonly NC='\033[0m' # No Color

# Logging functions
log() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"
}

log_info() {
    log "INFO: $*"
    [ "$VERBOSE" = true ] && echo -e "${GREEN}[INFO]${NC} $*"
}

log_warn() {
    log "WARN: $*"
    echo -e "${YELLOW}[WARN]${NC} $*" >&2
}

log_error() {
    log "ERROR: $*"
    echo -e "${RED}[ERROR]${NC} $*" >&2
}

# Error handling
error_exit() {
    log_error "$1"
    cleanup
    exit "${2:-1}"
}

# Cleanup function
cleanup() {
    log_info "Cleaning up..."
```

```
    [ -f "$PID_FILE" ] && rm -f "$PID_FILE"
}

# Trap signals
trap cleanup EXIT
trap 'error_exit "Script interrupted" 130' INT TERM

# Usage function
usage() {
    cat << EOF
Usage: $SCRIPT_NAME [OPTIONS]

Automated system backup script.

OPTIONS:
    -d, --dir DIR       Backup directory (default: /backup)
    -v, --verbose       Verbose output
    -h, --help          Show this help message

EXAMPLES:
    $SCRIPT_NAME -d /mnt/backup -v
    $SCRIPT_NAME --dir /backup

EOF
    exit 0
}

# Parse command line arguments
parse_args() {
    while [[ $# -gt 0 ]]; do
        case $1 in
            -d|--dir)
                BACKUP_DIR="$2"
                shift 2
                ;;
            -v|--verbose)
                VERBOSE=true
                shift
                ;;
            -h|--help)
                usage
                ;;
            *)
                error_exit "Unknown option: $1" 1
                ;;
        esac
    done
}

# Check if script is already running
check_running() {
    if [ -f "$PID_FILE" ]; then
        local pid
        pid=$(cat "$PID_FILE")
        if ps -p "$pid" > /dev/null 2>&1; then
            error_exit "Script already running with PID $pid" 1
        fi
    fi
```

```
    echo $$ > "$PID_FILE"
}

# Main function
main() {
    log_info "=== Starting backup process ==="

    # Your main logic here
    log_info "Backup directory: $BACKUP_DIR"

    # Example operations...

    log_info "=== Backup completed successfully ==="
}

# Entry point
parse_args "$@"
check_running
main
```

Bash Strict Mode Explained

`set -euo pipefail`

- **-e**: exit on error (comando fallisce → script termina)
- **-u**: error su variabili non definite
- **-o pipefail**: pipe fallisce se qualsiasi comando fallisce

`IFS=$'\n\t'`: previene word splitting su spazi

11.3.2 Gestione Argomenti Avanzata: getopts

```
#!/bin/bash
# Advanced argument parsing with getopts

usage() {
    cat << EOF
Usage: $0 [-h] [-v] [-f FILE] [-o OUTPUT] [-n NUM] COMMAND

OPTIONS:
    -h          Show help
    -v          Verbose mode
    -f FILE     Input file
    -o OUTPUT   Output file
    -n NUM      Number of iterations

COMMANDS:
    backup      Perform backup
    restore     Restore from backup
    check       Check integrity

EOF
    exit 1
}

```

```bash
# Default values
VERBOSE=false
INPUT_FILE=""
OUTPUT_FILE=""
ITERATIONS=1

# Parse options
while getopts ":hvf:o:n:" opt; do
    case $opt in
        h)
            usage
            ;;
        v)
            VERBOSE=true
            ;;
        f)
            INPUT_FILE="$OPTARG"
            ;;
        o)
            OUTPUT_FILE="$OPTARG"
            ;;
        n)
            ITERATIONS="$OPTARG"
            ;;
        \?)
            echo "Invalid option: -$OPTARG" >&2
            usage
            ;;
        :)
            echo "Option -$OPTARG requires an argument" >&2
            usage
            ;;
    esac
done

# Shift processed options
shift $((OPTIND-1))

# Get command (remaining argument)
COMMAND="${1:-}"

if [ -z "$COMMAND" ]; then
    echo "Error: Command required" >&2
    usage
fi

# Validate required options
if [ -z "$INPUT_FILE" ]; then
    echo "Error: Input file required (-f)" >&2
    usage
fi

echo "Command: $COMMAND"
echo "Input: $INPUT_FILE"
echo "Output: $OUTPUT_FILE"
echo "Iterations: $ITERATIONS"
echo "Verbose: $VERBOSE"
```

11.3.3 Array e Associative Array

```bash
#!/bin/bash
# Advanced array usage

# Indexed arrays
servers=("web1" "web2" "db1" "cache1")

# Add element
servers+=("web3")

# Iterate
for server in "${servers[@]}"; do
    echo "Processing $server"
done

# Iterate with index
for i in "${!servers[@]}"; do
    echo "Server $i: ${servers[$i]}"
done

# Array length
echo "Total servers: ${#servers[@]}"

# Slice array
echo "First two: ${servers[@]:0:2}"

# Associative arrays (hash/dict)
declare -A config
config[host]="localhost"
config[port]="5432"
config[db]="mydb"
config[user]="admin"

# Access
echo "Host: ${config[host]}"

# Iterate keys
for key in "${!config[@]}"; do
    echo "$key = ${config[$key]}"
done

# Check if key exists
if [ -v config[host] ]; then
    echo "Host configured"
fi

# Example: Server configuration
declare -A servers_config
servers_config[web1]="192.168.1.10:80"
servers_config[web2]="192.168.1.11:80"
servers_config[db1]="192.168.1.20:5432"

for server in "${!servers_config[@]}"; do
    IFS=':' read -r ip port <<< "${servers_config[$server]}"
    echo "Server: $server, IP: $ip, Port: $port"
done
```

11.3.4 Funzioni Avanzate

```bash
#!/bin/bash
# Advanced function patterns

# Function with return value via echo
get_timestamp() {
    echo "$(date +%Y%m%d_%H%M%S)"
}

# Function with return code
check_port() {
    local host=$1
    local port=$2

    if nc -z -w 2 "$host" "$port" 2>/dev/null; then
        return 0  # success
    else
        return 1  # failure
    fi
}

# Function with multiple return values (array)
get_system_info() {
    local -n result=$1  # nameref (pass by reference)

    result[hostname]=$(hostname)
    result[kernel]=$(uname -r)
    result[uptime]=$(uptime -p)
    result[load]=$(uptime | awk -F'load average:' '{print $2}')
}

# Function with error handling
backup_file() {
    local src=$1
    local dest=$2

    # Validate input
    if [ ! -f "$src" ]; then
        echo "Error: Source file not found: $src" >&2
        return 1
    fi

    # Create destination directory
    local dest_dir
    dest_dir=$(dirname "$dest")
    mkdir -p "$dest_dir" || {
        echo "Error: Cannot create directory: $dest_dir" >&2
        return 1
    }

    # Perform backup
    if cp -p "$src" "$dest"; then
        echo "Backup successful: $src -> $dest"
        return 0
    else
        echo "Error: Backup failed" >&2
        return 1
```

```
    fi
}

# Usage examples
timestamp=$(get_timestamp)
echo "Timestamp: $timestamp"

if check_port "localhost" 22; then
    echo "SSH port is open"
else
    echo "SSH port is closed"
fi

declare -A sysinfo
get_system_info sysinfo
for key in "${!sysinfo[@]}"; do
    echo "$key: ${sysinfo[$key]}"
done

if backup_file "/etc/hosts" "/backup/hosts.$timestamp"; then
    echo "File backed up successfully"
fi
```

11.4 Pattern Processing e Parsing

11.4.1 Text Processing con awk

```
#!/bin/bash
# Advanced awk patterns

# Log analysis: count HTTP status codes
awk '{print $9}' /var/log/nginx/access.log | \
    sort | uniq -c | sort -rn

# Calculate average response time
awk '{sum+=$NF; count++} END {print sum/count}' response_times.log

# Filter and format
awk '$9 == 500 {print $1, $7, $9}' /var/log/nginx/access.log

# Complex parsing
awk -F',' '
    BEGIN { total=0; count=0 }
    NR > 1 {  # skip header
        total += $3
        count++
        if ($3 > max) max = $3
        if (NR == 2 || $3 < min) min = $3
    }
    END {
        print "Count:", count
        print "Total:", total
        print "Average:", total/count
        print "Min:", min
        print "Max:", max
    }
' data.csv
```

```

# Process JSON with awk
curl -s https://api.example.com/data | \
    awk -F'"' '/"id":/ {print $4}'

# Process multiple files
awk '{sum+=$1} END {print FILENAME, sum}' file1 file2 file3
```

11.4.2 JSON Processing con jq

```
#!/bin/bash
# JSON processing with jq

# Parse JSON response
response=$(curl -s https://api.github.com/users/octocat)

# Extract field
name=$(echo "$response" | jq -r '.name')
echo "Name: $name"

# Extract multiple fields
echo "$response" | jq -r '.name, .company, .location'

# Array processing
repos=$(curl -s https://api.github.com/users/octocat/repos)

# Get all repository names
echo "$repos" | jq -r '.[].name'

# Filter and format
echo "$repos" | jq -r '.[] | select(.fork == false) | .name'

# Complex query
echo "$repos" | jq -r '.[] |
    select(.stargazers_count > 100) |
    "\(.name): \(.stargazers_count) stars"'

# Create JSON
jq -n \
    --arg name "John" \
    --arg email "john@example.com" \
    --argjson age 30 \
    '{name: $name, email: $email, age: $age}'

# Modify JSON
echo '{"name":"John","age":30}' | \
    jq '.age = 31 | .city = "NYC"'

# Practical example: Monitor API
#!/bin/bash
check_api() {
    local url=$1
    local response

    response=$(curl -s -w "\n%{http_code}" "$url")
    local body=$(echo "$response" | head -n -1)
    local code=$(echo "$response" | tail -n 1)
```

```
48
49      if [ "$code" -ne 200 ]; then
50          echo "API error: HTTP $code"
51          return 1
52      fi
53
54      local status=$(echo "$body" | jq -r '.status')
55      local message=$(echo "$body" | jq -r '.message')
56
57      echo "Status: $status"
58      echo "Message: $message"
59
60      if [ "$status" != "ok" ]; then
61          return 1
62      fi
63
64      return 0
65  }
66
67  check_api "https://api.example.com/health"
```

11.4.3 Parsing XML con xmllint

```
1   #!/bin/bash
2   # XML parsing with xmllint
3
4   xml_file="data.xml"
5
6   # Extract specific element
7   xmllint --xpath "//user/name/text()" "$xml_file"
8
9   # With namespace
10  xmllint --xpath "//*[local-name()='name']/text()" "$xml_file"
11
12  # Format XML
13  xmllint --format unformatted.xml
14
15  # Validate against schema
16  xmllint --schema schema.xsd data.xml --noout
17
18  # Example: Parse RSS feed
19  curl -s "https://example.com/rss" | \
20      xmllint --xpath "//item/title/text()" -
```

11.5 Automation Patterns

11.5.1 Deployment Automation

```
1   #!/bin/bash
2   # deploy.sh - Application deployment automation
3
4   set -euo pipefail
5
6   # Configuration
7   readonly APP_NAME="myapp"
```

```
readonly DEPLOY_USER="deployer"
readonly DEPLOY_DIR="/opt/$APP_NAME"
readonly BACKUP_DIR="/backup/$APP_NAME"
readonly SERVICE_NAME="$APP_NAME.service"

# Colors
readonly GREEN='\033[0;32m'
readonly RED='\033[0;31m'
readonly YELLOW='\033[1;33m'
readonly NC='\033[0m'

log_info() { echo -e "${GREEN}[INFO]${NC} $*"; }
log_warn() { echo -e "${YELLOW}[WARN]${NC} $*"; }
log_error() { echo -e "${RED}[ERROR]${NC} $*" >&2; }

# Pre-deployment checks
pre_deploy_check() {
    log_info "Running pre-deployment checks..."

    # Check user
    if [ "$(whoami)" != "$DEPLOY_USER" ]; then
        log_error "Must run as $DEPLOY_USER"
        return 1
    fi

    # Check directory
    if [ ! -d "$DEPLOY_DIR" ]; then
        log_error "Deploy directory not found: $DEPLOY_DIR"
        return 1
    fi

    # Check service
    if ! systemctl list-unit-files | grep -q "$SERVICE_NAME"; then
        log_error "Service not found: $SERVICE_NAME"
        return 1
    fi

    # Check disk space (need at least 1GB)
    local available
    available=$(df "$DEPLOY_DIR" | awk 'NR==2 {print $4}')
    if [ "$available" -lt 1048576 ]; then
        log_error "Insufficient disk space"
        return 1
    fi

    log_info "Pre-deployment checks passed"
    return 0
}

# Backup current version
backup_current() {
    log_info "Backing up current version..."

    local backup_name="$APP_NAME-$(date +%Y%m%d_%H%M%S)"
    local backup_path="$BACKUP_DIR/$backup_name"

    mkdir -p "$BACKUP_DIR"

```

```
    if tar -czf "$backup_path.tar.gz" -C "$DEPLOY_DIR" . ; then
        log_info "Backup created: $backup_path.tar.gz"
        echo "$backup_path.tar.gz"
        return 0
    else
        log_error "Backup failed"
        return 1
    fi
}

# Deploy new version
deploy() {
    local artifact=$1

    log_info "Deploying $artifact..."

    # Extract artifact
    if ! tar -xzf "$artifact" -C "$DEPLOY_DIR"; then
        log_error "Extraction failed"
        return 1
    fi

    # Set permissions
    chown -R "$DEPLOY_USER:$DEPLOY_USER" "$DEPLOY_DIR"

    # Run migrations if needed
    if [ -f "$DEPLOY_DIR/migrate.sh" ]; then
        log_info "Running migrations..."
        bash "$DEPLOY_DIR/migrate.sh"
    fi

    log_info "Deployment completed"
    return 0
}

# Restart service
restart_service() {
    log_info "Restarting service..."

    if sudo systemctl restart "$SERVICE_NAME"; then
        sleep 5
        if sudo systemctl is-active --quiet "$SERVICE_NAME"; then
            log_info "Service restarted successfully"
            return 0
        else
            log_error "Service failed to start"
            return 1
        fi
    else
        log_error "Failed to restart service"
        return 1
    fi
}

# Health check
health_check() {
    log_info "Performing health check..."

```

```
    local max_attempts=10
    local attempt=1

    while [ $attempt -le $max_attempts ]; do
        if curl -sf http://localhost:8080/health > /dev/null; then
            log_info "Health check passed"
            return 0
        fi

        log_warn "Health check attempt $attempt/$max_attempts failed"
        sleep 5
        ((attempt++))
    done

    log_error "Health check failed after $max_attempts attempts"
    return 1
}

# Rollback to previous version
rollback() {
    local backup_file=$1

    log_warn "Rolling back to $backup_file..."

    # Stop service
    sudo systemctl stop "$SERVICE_NAME"

    # Remove current deployment
    rm -rf "${DEPLOY_DIR:?}"/*

    # Restore backup
    if tar -xzf "$backup_file" -C "$DEPLOY_DIR"; then
        log_info "Backup restored"

        # Restart service
        if restart_service && health_check; then
            log_info "Rollback successful"
            return 0
        fi
    fi

    log_error "Rollback failed"
    return 1
}

# Main deployment flow
main() {
    local artifact=${1:-}

    if [ -z "$artifact" ] || [ ! -f "$artifact" ]; then
        log_error "Usage: $0 <artifact.tar.gz>"
        exit 1
    fi

    log_info "=== Starting deployment of $APP_NAME ==="

    # Pre-checks
    if ! pre_deploy_check; then
```

```
        log_error "Pre-deployment checks failed"
        exit 1
    fi

    # Backup
    backup_file=$(backup_current) || {
        log_error "Backup failed, aborting"
        exit 1
    }

    # Stop service
    sudo systemctl stop "$SERVICE_NAME"

    # Deploy
    if deploy "$artifact"; then
        # Restart and verify
        if restart_service && health_check; then
            log_info "=== Deployment successful ==="
            exit 0
        else
            log_error "Deployment verification failed"
            rollback "$backup_file"
            exit 1
        fi
    else
        log_error "Deployment failed"
        rollback "$backup_file"
        exit 1
    fi
}

main "$@"
```

11.5.2 Multi-Server Deployment

```
#!/bin/bash
# multi_deploy.sh - Deploy to multiple servers

set -euo pipefail

readonly SERVERS=(
    "web1.example.com"
    "web2.example.com"
    "web3.example.com"
)

readonly DEPLOY_SCRIPT="/usr/local/bin/deploy.sh"
readonly ARTIFACT="app-v1.2.3.tar.gz"
readonly SSH_USER="deployer"

log_info() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] INFO: $*"
}

log_error() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] ERROR: $*" >&2
}
```

```
23
24 # Deploy to single server
25 deploy_to_server() {
26     local server=$1
27
28     log_info "Deploying to $server..."
29
30     # Copy artifact
31     if ! scp "$ARTIFACT" "$SSH_USER@$server:/tmp/"; then
32         log_error "Failed to copy artifact to $server"
33         return 1
34     fi
35
36     # Run deployment
37     if ssh "$SSH_USER@$server" "$DEPLOY_SCRIPT /tmp/$ARTIFACT"; then
38         log_info "Deployment to $server successful"
39         return 0
40     else
41         log_error "Deployment to $server failed"
42         return 1
43     fi
44 }
45
46 # Sequential deployment
47 deploy_sequential() {
48     local failed_servers=()
49
50     for server in "${SERVERS[@]}"; do
51         if ! deploy_to_server "$server"; then
52             failed_servers+=("$server")
53         fi
54         sleep 10  # delay between deployments
55     done
56
57     if [ ${#failed_servers[@]} -eq 0 ]; then
58         log_info "All deployments successful"
59         return 0
60     else
61         log_error "Failed servers: ${failed_servers[*]}"
62         return 1
63     fi
64 }
65
66 # Parallel deployment
67 deploy_parallel() {
68     local pids=()
69     local failed_servers=()
70
71     # Start deployments in background
72     for server in "${SERVERS[@]}"; do
73         deploy_to_server "$server" &
74         pids+=($!)
75     done
76
77     # Wait for all to complete
78     for i in "${!pids[@]}"; do
79         if ! wait "${pids[$i]}"; then
80             failed_servers+=("${SERVERS[$i]}")
```

```
        fi
    done

    if [ ${#failed_servers[@]} -eq 0 ]; then
        log_info "All deployments successful"
        return 0
    else
        log_error "Failed servers: ${failed_servers[*]}"
        return 1
    fi
}

# Blue-green deployment
deploy_blue_green() {
    local blue_servers=("web1.example.com" "web2.example.com")
    local green_servers=("web3.example.com" "web4.example.com")

    log_info "Deploying to green servers..."

    # Deploy to green (inactive)
    for server in "${green_servers[@]}"; do
        deploy_to_server "$server" || return 1
    done

    log_info "Switching traffic to green servers..."
    # Update load balancer configuration here
    # update_load_balancer "green"

    sleep 30  # monitor green

    log_info "Deploying to blue servers..."

    # Deploy to blue
    for server in "${blue_servers[@]}"; do
        deploy_to_server "$server" || return 1
    done

    log_info "Blue-green deployment completed"
}

# Main
main() {
    local strategy=${1:-sequential}

    case $strategy in
        sequential)
            deploy_sequential
            ;;
        parallel)
            deploy_parallel
            ;;
        blue-green)
            deploy_blue_green
            ;;
        *)
            echo "Usage: $0 {sequential|parallel|blue-green}"
            exit 1
            ;;
```

```
    esac
}

main "$@"
```

11.6 Monitoring e Alerting

11.6.1 System Monitor

```
#!/bin/bash
# system_monitor.sh - Comprehensive system monitoring

set -euo pipefail

readonly LOG_FILE="/var/log/system_monitor.log"
readonly ALERT_EMAIL="admin@example.com"
readonly METRICS_FILE="/var/lib/system_monitor/metrics.json"

# Thresholds
readonly CPU_THRESHOLD=80
readonly MEM_THRESHOLD=85
readonly DISK_THRESHOLD=90
readonly LOAD_THRESHOLD=4.0

log() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"
}

# Collect CPU usage
get_cpu_usage() {
    local cpu
    cpu=$(top -bn1 | grep "Cpu(s)" | awk '{print $2}' | cut -d'%' -f1)
    echo "${cpu%.*}"  # round to integer
}

# Collect memory usage
get_memory_usage() {
    free | grep Mem | awk '{printf("%.0f", $3/$2 * 100.0)}'
}

# Collect disk usage
get_disk_usage() {
    df -h / | awk 'NR==2 {print $5}' | sed 's/%//'
}

# Collect load average
get_load_average() {
    uptime | awk -F'load average:' '{print $2}' | awk '{print $1}' | sed
        's/,//'
}

# Check thresholds
check_thresholds() {
    local alerts=()

    # CPU
    local cpu=$(get_cpu_usage)
```

```
    if [ "$cpu" -gt "$CPU_THRESHOLD" ]; then
        alerts+=("CPU: ${cpu}% (threshold: ${CPU_THRESHOLD}%)")
    fi

    # Memory
    local mem=$(get_memory_usage)
    if [ "$mem" -gt "$MEM_THRESHOLD" ]; then
        alerts+=("Memory: ${mem}% (threshold: ${MEM_THRESHOLD}%)")
    fi

    # Disk
    local disk=$(get_disk_usage)
    if [ "$disk" -gt "$DISK_THRESHOLD" ]; then
        alerts+=("Disk: ${disk}% (threshold: ${DISK_THRESHOLD}%)")
    fi

    # Load
    local load=$(get_load_average)
    if (( $(echo "$load > $LOAD_THRESHOLD" | bc -l) )); then
        alerts+=("Load: $load (threshold: $LOAD_THRESHOLD)")
    fi

    # Send alerts if any
    if [ ${#alerts[@]} -gt 0 ]; then
        local message="System Alert:\n\n"
        for alert in "${alerts[@]}"; do
            message+="- $alert\n"
            log "ALERT: $alert"
        done

        echo -e "$message" | mail -s "System Alert: $(hostname)" "
           $ALERT_EMAIL"
    fi
}

# Collect and store metrics
collect_metrics() {
    local timestamp=$(date +%s)
    local cpu=$(get_cpu_usage)
    local mem=$(get_memory_usage)
    local disk=$(get_disk_usage)
    local load=$(get_load_average)

    # Create JSON
    local metrics
    metrics=$(jq -n \
        --arg ts "$timestamp" \
        --arg cpu "$cpu" \
        --arg mem "$mem" \
        --arg disk "$disk" \
        --arg load "$load" \
        '{
            timestamp: $ts,
            cpu_usage: $cpu,
            memory_usage: $mem,
            disk_usage: $disk,
            load_average: $load
        }')
```

```
105
106     # Append to metrics file
107     mkdir -p "$(dirname "$METRICS_FILE")"
108     echo "$metrics" >> "$METRICS_FILE"
109
110     # Keep only last 1000 entries
111     tail -n 1000 "$METRICS_FILE" > "${METRICS_FILE}.tmp"
112     mv "${METRICS_FILE}.tmp" "$METRICS_FILE"
113
114     log "Metrics: CPU=$cpu% MEM=$mem% DISK=$disk% LOAD=$load"
115 }
116
117 # Generate report
118 generate_report() {
119     log "Generating system report..."
120
121     cat << EOF
122 === System Health Report ===
123 Generated: $(date)
124 Hostname: $(hostname)
125
126 CPU Usage: $(get_cpu_usage)%
127 Memory Usage: $(get_memory_usage)%
128 Disk Usage: $(get_disk_usage)%
129 Load Average: $(get_load_average)
130
131 Top 5 CPU Processes:
132 $(ps aux --sort=-%cpu | head -6)
133
134 Top 5 Memory Processes:
135 $(ps aux --sort=-%mem | head -6)
136
137 Disk Usage by Directory:
138 $(du -sh /* 2>/dev/null | sort -rh | head -10)
139
140 Active Connections:
141 $(ss -s)
142
143 Recent System Errors:
144 $(journalctl -p err --since "1 hour ago" --no-pager | tail -10)
145
146 EOF
147 }
148
149 # Main
150 main() {
151     collect_metrics
152     check_thresholds
153
154     # Generate daily report
155     local hour=$(date +%H)
156     if [ "$hour" == "00" ]; then
157         generate_report | mail -s "Daily System Report: $(hostname)" "
            $ALERT_EMAIL"
158     fi
159 }
160
161 main "$@"
```

11.7 Backup Automation

11.7.1 Comprehensive Backup Script

```
#!/bin/bash
# backup_system.sh - Comprehensive backup with rotation and verification

set -euo pipefail

# Configuration
readonly BACKUP_ROOT="/backup"
readonly RETENTION_DAYS=30
readonly RETENTION_WEEKLY=8
readonly RETENTION_MONTHLY=12

# Sources to backup
declare -A BACKUP_SOURCES=(
    [home]="/home"
    [etc]="/etc"
    [var_www]="/var/www"
    [databases]="/var/lib/mysql"
)

readonly LOG_FILE="/var/log/backup_system.log"
readonly DATE=$(date +%Y%m%d)
readonly TIMESTAMP=$(date +%Y%m%d_%H%M%S)

log() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"
}

# Create backup directory structure
init_backup_dirs() {
    mkdir -p "$BACKUP_ROOT"/{daily,weekly,monthly}
}

# Backup MySQL databases
backup_databases() {
    log "Backing up databases..."

    local backup_dir="$BACKUP_ROOT/daily/databases_$DATE"
    mkdir -p "$backup_dir"

    # Get list of databases
    local databases
    databases=$(mysql -e "SHOW DATABASES;" | grep -Ev "Database|
        information_schema|performance_schema|mysql|sys")

    for db in $databases; do
        log "Backing up database: $db"
        mysqldump --single-transaction --routines --triggers "$db" | \
            gzip > "$backup_dir/${db}_${TIMESTAMP}.sql.gz"
    done

    # Create tar archive
    tar -czf "$BACKUP_ROOT/daily/databases_${DATE}.tar.gz" -C "
        $BACKUP_ROOT/daily" "databases_$DATE"
    rm -rf "$backup_dir"
```

```
53
54      log "Database backup completed"
55  }
56
57  # Backup filesystem
58  backup_filesystem() {
59      local name=$1
60      local source=$2
61
62      log "Backing up $name from $source..."
63
64      local backup_file="$BACKUP_ROOT/daily/${name}_${DATE}.tar.gz"
65
66      tar -czf "$backup_file" \
67          --exclude='*.log' \
68          --exclude='*.tmp' \
69          --exclude='cache/*' \
70          -C "$(dirname "$source")" \
71          "$(basename "$source")"
72
73      log "Filesystem backup completed: $backup_file"
74  }
75
76  # Verify backup
77  verify_backup() {
78      local backup_file=$1
79
80      log "Verifying backup: $backup_file"
81
82      if [ ! -f "$backup_file" ]; then
83          log "ERROR: Backup file not found: $backup_file"
84          return 1
85      fi
86
87      # Check file integrity
88      if tar -tzf "$backup_file" > /dev/null 2>&1; then
89          local size
90          size=$(du -sh "$backup_file" | awk '{print $1}')
91          log "Backup verified OK: $backup_file ($size)"
92          return 0
93      else
94          log "ERROR: Backup verification failed: $backup_file"
95          return 1
96      fi
97  }
98
99  # Rotate daily backups to weekly
100 rotate_to_weekly() {
101     local day_of_week=$(date +%u)  # 1=Monday, 7=Sunday
102
103     if [ "$day_of_week" -eq 7 ]; then  # Sunday
104         log "Rotating daily backup to weekly..."
105
106         for file in "$BACKUP_ROOT/daily"/*_"$DATE".tar.gz; do
107             if [ -f "$file" ]; then
108                 cp "$file" "$BACKUP_ROOT/weekly/"
109             fi
110         done
```

```
    fi
}

# Rotate weekly backups to monthly
rotate_to_monthly() {
    local day_of_month=$(date +%d)

    if [ "$day_of_month" -eq "01" ]; then  # First day of month
        log "Rotating weekly backup to monthly..."

        local last_week=$(date -d "7 days ago" +%Y%m%d)

        for file in "$BACKUP_ROOT/weekly"/*_"$last_week".tar.gz; do
            if [ -f "$file" ]; then
                cp "$file" "$BACKUP_ROOT/monthly/"
            fi
        done
    fi
}

# Cleanup old backups
cleanup_old_backups() {
    log "Cleaning up old backups..."

    # Daily backups
    find "$BACKUP_ROOT/daily" -type f -mtime +$RETENTION_DAYS -delete

    # Weekly backups (keep 8 weeks)
    local count
    count=$(find "$BACKUP_ROOT/weekly" -type f | wc -l)
    if [ "$count" -gt "$RETENTION_WEEKLY" ]; then
        find "$BACKUP_ROOT/weekly" -type f | \
            sort | \
            head -n -"$RETENTION_WEEKLY" | \
            xargs rm -f
    fi

    # Monthly backups (keep 12 months)
    count=$(find "$BACKUP_ROOT/monthly" -type f | wc -l)
    if [ "$count" -gt "$RETENTION_MONTHLY" ]; then
        find "$BACKUP_ROOT/monthly" -type f | \
            sort | \
            head -n -"$RETENTION_MONTHLY" | \
            xargs rm -f
    fi

    log "Cleanup completed"
}

# Send backup report
send_report() {
    local status=$1

    local subject
    if [ "$status" -eq 0 ]; then
        subject="Backup SUCCESS: $(hostname)"
    else
        subject="Backup FAILED: $(hostname)"
```

```
    fi

    local report
    report=$(cat << EOF
Backup Report for $(hostname)
Date: $(date)
Status: $([ "$status" -eq 0 ] && echo "SUCCESS" || echo "FAILED")

Backup Summary:
$(ls -lh "$BACKUP_ROOT/daily"/*_"$DATE".tar.gz 2>/dev/null || echo "No backups found")

Disk Usage:
$(df -h "$BACKUP_ROOT")

Recent Log:
$(tail -20 "$LOG_FILE")
EOF
)

    echo "$report" | mail -s "$subject" admin@example.com
}

# Main
main() {
    log "=== Starting backup process ==="

    local exit_code=0

    # Initialize
    init_backup_dirs

    # Backup databases
    if ! backup_databases; then
        exit_code=1
    fi

    # Backup filesystems
    for name in "${!BACKUP_SOURCES[@]}"; do
        if ! backup_filesystem "$name" "${BACKUP_SOURCES[$name]}"; then
            exit_code=1
        fi
    done

    # Verify backups
    for file in "$BACKUP_ROOT/daily"/*_"$DATE".tar.gz; do
        if [ -f "$file" ]; then
            verify_backup "$file" || exit_code=1
        fi
    done

    # Rotate backups
    rotate_to_weekly
    rotate_to_monthly

    # Cleanup
    cleanup_old_backups

```

```
226     # Report
227     send_report $exit_code
228
229     log "=== Backup process completed (exit code: $exit_code) ==="
230
231     exit $exit_code
232 }
233
234 main "$@"
```

11.8 Best Practice Automation

Scripting Best Practice

1. **Strict mode**: sempre usare `set -euo pipefail`
2. **Shebang**: sempre iniziare con `!/bin/bash`
3. **Documentation**: header con description, usage, examples
4. **Error handling**: trap errors e cleanup
5. **Logging**: structured logging con timestamp
6. **Idempotency**: script rieseguibile senza effetti collaterali
7. **Dry-run mode**: opzione per simulare senza modifiche
8. **Validation**: validare input e precondizioni
9. **Constants**: usare `readonly` per costanti
10. **Quotes**: sempre quote variabili (`"$var"`)
11. **Functions**: modularizzare codice in funzioni
12. **Testing**: testare script in ambiente safe prima produzione

11.9 Esercizi Pratici

1. Creare script deployment completo con pre-checks, backup, rollback.
2. Implementare system monitor con threshold alerts via email.
3. Creare backup automation con daily/weekly/monthly rotation.
4. Scrivere script parsing log Apache/Nginx per statistiche traffico.
5. Implementare health check multi-servizio con retry logic.
6. Creare script provisioning server che installa stack completo.
7. Implementare log aggregation da server multipli.
8. Scrivere script database backup con encryption.
9. Creare monitoring dashboard che genera report HTML.
10. Implementare CI/CD pipeline script con test e deployment.

11.10 Riepilogo

Hai imparato:

- Tecniche avanzate Bash scripting
- Pattern processing con awk, jq, xmllint
- Automation deployment e configuration management
- System monitoring e alerting
- Backup automation con rotation
- Best practice scripting e error handling

11.11 Riferimenti

- https://www.gnu.org/software/bash/manual/
- https://stedolan.github.io/jq/manual/
- https://www.shellcheck.net/ - Shell script linter
- Google Shell Style Guide
- https://explainshell.com/ - Explain shell commands

Appendice A

Cheat Sheet Comandi

A.1 Introduzione

Questa appendice fornisce una reference rapida dei comandi Linux più utilizzati, organizzati per categoria. Ogni sezione include sintassi, opzioni comuni ed esempi pratici.

A.2 Navigazione File System

pwd - Print Working Directory

```
# Mostra directory corrente
pwd

# Mostra path fisico (segue symlink)
pwd -P
```

cd - Change Directory

```
cd /path/to/directory     # Vai a directory assoluta
cd relative/path          # Vai a directory relativa
cd ~                      # Vai a home directory
cd                        # Vai a home (equivalente)
cd -                      # Torna a directory precedente
cd ..                     # Vai a directory parent
cd ../..                  # Sali due livelli
```

ls - List Directory Contents

```
ls                        # Lista file directory corrente
ls -l                     # Long format (permessi, owner, size)
ls -a                     # Mostra file nascosti (iniziano con .)
ls -lh                    # Long format con dimensioni human-
    readable
ls -la                    # Long format + file nascosti
ls -lt                    # Ordina per tempo modifica (recenti prima
    )
ls -ltr                   # Ordina per tempo (vecchi prima)
ls -lS                    # Ordina per dimensione
ls -R                     # Ricorsivo
```

```
ls -d */                    # Solo directory
ls -1                       # Un file per riga
```

A.3 Operazioni su File e Directory

Creazione

```
touch file.txt              # Crea file vuoto
mkdir directory             # Crea directory
mkdir -p path/to/dir        # Crea path completo (parent directories)
mkdir dir1 dir2 dir3        # Crea multiple directory
```

Copia

```
cp source dest              # Copia file
cp -r source/ dest/         # Copia directory ricorsivamente
cp -p file dest             # Preserva permessi, timestamps
cp -i file dest             # Interactive (chiedi conferma)
cp -u file dest             # Copia solo se source più recente
cp file{,.bak}              # Crea backup (file.bak)
cp file1 file2 dir/         # Copia multipli file in directory
```

Spostamento e Rinomina

```
mv oldname newname          # Rinomina file/directory
mv source dest/             # Sposta in altra directory
mv -i source dest           # Interactive (chiedi conferma)
mv -u source dest           # Sposta solo se più recente
mv *.txt dir/               # Sposta tutti .txt in directory
```

Eliminazione

```
rm file.txt                 # Elimina file
rm -i file.txt              # Interactive (chiedi conferma)
rm -f file.txt              # Force (no conferma, ignora errori)
rm -r directory/            # Elimina directory ricorsivamente
rm -rf directory/           # Force recursive (PERICOLOSO!)
rmdir directory             # Elimina directory vuota
rm *.log                    # Elimina tutti .log
```

A.4 Visualizzazione Contenuti File

cat - Concatenate and Print

```
cat file.txt                # Mostra intero file
cat file1 file2             # Concatena e mostra file multipli
cat -n file.txt             # Mostra con numeri riga
cat -b file.txt             # Numera solo righe non vuote
```

```
cat > file.txt            # Crea file (Ctrl+D per terminare)
cat >> file.txt           # Append a file
```

less/more - Paginazione

```
less file.txt             # Visualizza con paginazione
# Comandi in less:
# Spazio: pagina avanti
# b: pagina indietro
# /pattern: cerca
# n: prossima occorrenza
# q: esci

more file.txt             # Paginazione semplice (solo avanti)
```

head/tail - Inizio/Fine File

```
head file.txt             # Prime 10 righe
head -n 20 file.txt       # Prime 20 righe
head -n -5 file.txt       # Tutte tranne ultime 5

tail file.txt             # Ultime 10 righe
tail -n 20 file.txt       # Ultime 20 righe
tail -n +10 file.txt      # Dalla riga 10 in poi
tail -f file.log          # Follow (aggiornamenti real-time)
tail -f -n 50 file.log    # Follow ultime 50 righe
```

A.5 Ricerca File

find - Cerca File

```
# Per nome
find . -name "*.txt"                    # File .txt
find . -iname "*.TXT"                   # Case-insensitive
find /home -name "file.txt"             # In path specifico

# Per tipo
find . -type f                          # Solo file
find . -type d                          # Solo directory
find . -type l                          # Solo symlink

# Per dimensione
find . -size +100M                      # Più grandi di 100MB
find . -size -1k                        # Più piccoli di 1KB
find . -size 50M                        # Esattamente 50MB

# Per tempo
find . -mtime -7                        # Modificati ultimi 7
    giorni
find . -mtime +30                       # Modificati più di 30
    giorni fa
find . -mmin -60                        # Modificati ultima ora
```

```
find . -atime -1                          # Acceduti ieri

# Per permessi
find . -perm 644                          # Permessi esatti
find . -perm -644                         # Almeno questi permessi
find . -perm /u+w,g+w                     # Owner O group writable

# Azioni
find . -name "*.log" -delete              # Elimina trovati
find . -name "*.txt" -exec cat {} \;      # Esegui comando
find . -type f -exec chmod 644 {} \;      # Cambia permessi
find . -name "*.tmp" -ok rm {} \;         # Interactive delete

# Combinazioni
find . -name "*.txt" -size +1M -mtime -7
find /var/log -name "*.log" -mtime +30 -delete
```

grep - Cerca Pattern nei File

```
grep "pattern" file.txt                   # Cerca pattern
grep -i "pattern" file.txt                # Case-insensitive
grep -r "pattern" directory/              # Ricorsivo
grep -n "pattern" file.txt                # Mostra numeri riga
grep -v "pattern" file.txt                # Inverted (righe NON
    matching)
grep -c "pattern" file.txt                # Conta occorrenze
grep -l "pattern" *.txt                   # Solo nomi file
grep -w "word" file.txt                   # Match parola intera
grep -A 3 "pattern" file.txt              # 3 righe After match
grep -B 3 "pattern" file.txt              # 3 righe Before match
grep -C 3 "pattern" file.txt              # 3 righe Context (before+
    after)

# Regex
grep "^start" file.txt                    # Inizia con "start"
grep "end$" file.txt                      # Finisce con "end"
grep "err.*fatal" file.txt                # err seguito da fatal
grep -E "error|warn" file.txt             # Extended regex (OR)
```

A.6 Permessi e Ownership

chmod - Cambia Permessi

```
# Notazione simbolica
chmod u+x file.sh                         # User: aggiungi execute
chmod g-w file.txt                        # Group: rimuovi write
chmod o+r file.txt                        # Others: aggiungi read
chmod a+x file.sh                         # All: aggiungi execute
chmod u+rwx,g+rx,o+r file                # Combinato

# Notazione numerica
chmod 644 file.txt                        # rw-r--r--
chmod 755 script.sh                       # rwxr-xr-x
```

```
chmod 700 private.key                    # rwx------
chmod 600 config.conf                    # rw-------
chmod 777 shared/                        # rwxrwxrwx (sconsigliato)

# Ricorsivo
chmod -R 755 directory/

# Permessi speciali
chmod u+s executable                     # SUID
chmod g+s directory                      # SGID
chmod +t directory                       # Sticky bit
chmod 4755 file                          # SUID + 755
```

chown - Cambia Owner

```
chown user file.txt                      # Cambia owner
chown user:group file.txt                # Cambia owner e group
chown :group file.txt                    # Solo group
chown -R user:group directory/           # Ricorsivo
chown --reference=ref.txt file.txt       # Copia ownership da altro
    file
```

A.7 Gestione Processi

ps - Process Status

```
ps                                       # Processi sessione
    corrente
ps aux                                   # Tutti processi (BSD style
    )
ps -ef                                   # Tutti processi (Unix
    style)
ps -u username                           # Processi di user
ps -p 1234                               # Processo specifico PID
ps aux | grep nginx                      # Filtra processi
ps aux --sort=-%cpu                      # Ordina per CPU
ps aux --sort=-%mem                      # Ordina per memoria
```

top/htop - Monitor Processi

```
top                                      # Monitor interattivo
# Comandi in top:
# k: kill processo
# r: renice (cambia priorità)
# M: ordina per memoria
# P: ordina per CPU
# q: quit

htop                                     # top migliorato (se
    installato)
```

kill - Termina Processi

```
kill PID                                # SIGTERM (graceful shutdown)
kill -9 PID                             # SIGKILL (force kill)
kill -15 PID                            # SIGTERM (default)
kill -HUP PID                           # SIGHUP (reload config)
killall process_name                    # Kill per nome
pkill pattern                           # Kill per pattern
pgrep pattern                           # Trova PID per pattern
```

Background e Jobs

```
command &                               # Esegui in background
jobs                                    # Lista jobs correnti
fg                                      # Porta job in foreground
fg %1                                   # Porta job 1 in foreground
bg                                      # Continua job in background
Ctrl+Z                                  # Sospendi processo corrente
nohup command &                         # Esegui immune a hangup
```

A.8 Compressione e Archivi

tar - Tape Archive

```
# Creare archivio
tar -cf archive.tar files/              # Crea tar
tar -czf archive.tar.gz files/          # Crea tar.gz (gzip)
tar -cjf archive.tar.bz2 files/         # Crea tar.bz2 (bzip2)
tar -cJf archive.tar.xz files/          # Crea tar.xz (xz)

# Estrarre
tar -xf archive.tar                     # Estrai tar
tar -xzf archive.tar.gz                 # Estrai tar.gz
tar -xjf archive.tar.bz2                # Estrai tar.bz2
tar -xf archive.tar -C /path/           # Estrai in path specifico

# Visualizzare contenuti
tar -tf archive.tar                     # Lista file
tar -tzf archive.tar.gz                 # Lista file in tar.gz

# Append
tar -rf archive.tar newfile             # Aggiungi file

# Opzioni comuni
tar -cvzf archive.tar.gz files/         # v=verbose
tar -xvzf archive.tar.gz                # verbose extract
```

gzip/gunzip - Compressione

```
gzip file.txt                          # Comprimi (crea file.txt.gz)
gzip -k file.txt                       # Comprimi, mantieni originale
gzip -9 file.txt                       # Max compressione
gunzip file.txt.gz                     # Decomprimi
gzip -d file.txt.gz                    # Decomprimi (alternativa)
gzip -l file.txt.gz                    # Info file compresso
zcat file.txt.gz                       # Visualizza senza decomprimere
```

zip/unzip

```
zip archive.zip file1 file2            # Crea zip
zip -r archive.zip directory/          # Zip ricorsivo
zip -e secure.zip file.txt             # Zip con password
unzip archive.zip                      # Estrai zip
unzip archive.zip -d /path/            # Estrai in directory
unzip -l archive.zip                   # Lista contenuti
```

A.9 Networking

Configurazione Interfacce

```
# ip (moderno)
ip addr show                           # Mostra interfacce
ip link show                           # Mostra link status
ip route show                          # Mostra routing table
ip neigh show                          # Mostra ARP

# ifconfig (legacy)
ifconfig                               # Mostra interfacce
ifconfig eth0                          # Interfaccia specifica
```

Test Connettività

```
ping google.com                        # Ping hostname
ping -c 4 8.8.8.8                      # 4 ping
ping -i 0.2 192.168.1.1                # Interval 0.2s

traceroute google.com                  # Traccia route
traceroute -n 8.8.8.8                  # No DNS resolution

nc -zv host 80                         # Test porta TCP
nc -zvu host 53                        # Test porta UDP
```

Connessioni e Porte

```
# ss (moderno)
ss -tulpn                                   # TCP/UDP listening ports
ss -t                                       # TCP connections
ss -ta                                      # All TCP sockets
ss -s                                       # Statistiche

# netstat (legacy)
netstat -tulpn                              # Listening ports
netstat -an                                 # All connections
```

DNS

```
nslookup google.com                         # Query DNS
nslookup google.com 8.8.8.8                 # Query server specifico

dig google.com                              # DNS lookup dettagliato
dig google.com +short                       # Solo risposta
dig google.com MX                           # Record MX
dig -x 8.8.8.8                              # Reverse lookup

host google.com                             # Semplice lookup
```

Trasferimento File

```
# scp
scp file.txt user@host:/path/               # Upload file
scp user@host:/path/file.txt ./             # Download file
scp -r dir/ user@host:/path/                # Upload directory
scp -P 2222 file.txt user@host:/path/       # Porta custom

# rsync
rsync -av source/ dest/                     # Sync directories
rsync -avz source/ user@host:/dest/         # Sync remoto
rsync -avz --delete source/ dest/           # Sync + delete
rsync -avzP source/ dest/                   # Con progress
```

Download

```
# wget
wget https://example.com/file.zip           # Download file
wget -O output.zip URL                      # Nome custom
wget -c URL                                 # Resume download
wget -r -np URL                             # Download ricorsivo

# curl
curl https://api.example.com/data           # GET request
curl -o file.zip URL                        # Salva file
curl -O URL                                 # Usa nome originale
curl -L URL                                 # Segui redirect
curl -X POST -d "data" URL                  # POST request
curl -H "Header: value" URL                 # Custom header
```

A.10 Gestione Utenti

User Management

```
# Creazione/modifica utenti
sudo useradd username                    # Crea utente
sudo useradd -m -s /bin/bash user        # Con home e shell
sudo passwd username                     # Imposta password
sudo usermod -aG group username          # Aggiungi a gruppo
sudo userdel username                    # Elimina utente
sudo userdel -r username                 # Elimina + home

# Info utenti
id                                       # Info utente corrente
id username                              # Info utente specifico
whoami                                   # Username corrente
who                                      # Utenti loggati
w                                        # Utenti loggati + attività
last                                     # Storico login
```

Group Management

```
sudo groupadd groupname                  # Crea gruppo
sudo groupdel groupname                  # Elimina gruppo
groups username                          # Gruppi di utente
getent group groupname                   # Info gruppo
```

A.11 Systemd e Servizi

Service Management

```
sudo systemctl start service             # Avvia servizio
sudo systemctl stop service              # Ferma servizio
sudo systemctl restart service           # Riavvia servizio
sudo systemctl reload service            # Ricarica config
sudo systemctl status service            # Status servizio

sudo systemctl enable service            # Abilita avvio automatico
sudo systemctl disable service           # Disabilita avvio
sudo systemctl enable --now service      # Enable + start

systemctl is-active service              # Check se attivo
systemctl is-enabled service             # Check se enabled

systemctl list-units --type=service      # Lista servizi
systemctl --failed                       # Servizi falliti
```

Logs con journalctl

```
journalctl                               # Tutti i log
journalctl -u service                    # Log di servizio
journalctl -f                            # Follow (real-time)
```

```
journalctl -u nginx -f                    # Follow servizio
journalctl -n 50                          # Ultime 50 righe
journalctl --since today                  # Da oggi
journalctl --since "1 hour ago"           # Ultima ora
journalctl --since "2025-01-01"           # Da data
journalctl -p err                         # Solo errori
journalctl -b                             # Boot corrente
journalctl -k                             # Kernel messages
```

A.12 Disk Usage

df - Disk Free

```
df                                        # Spazio disco
df -h                                     # Human-readable
df -i                                     # Inode usage
df -T                                     # Mostra filesystem type
df /path                                  # Spazio per path specifico
```

du - Disk Usage

```
du -sh directory/                         # Dimensione directory
du -sh *                                  # Dimensione file/dir
    correnti
du -sh /* | sort -rh | head -10           # Top 10 directory
du -ah directory/                         # All files (human-readable
    )
du -d 1 directory/                        # Depth 1
```

A.13 Text Processing

awk

```
awk '{print $1}' file.txt                 # Prima colonna
awk '{print $1, $3}' file.txt             # Colonne 1 e 3
awk -F: '{print $1}' /etc/passwd          # Delimiter custom
awk 'NR==5' file.txt                      # Riga 5
awk '/pattern/' file.txt                  # Righe con pattern
awk '$3 > 100' file.txt                   # Condizione numerica
awk '{sum+=$1} END {print sum}' file      # Somma colonna
```

sed - Stream Editor

```
sed 's/old/new/' file.txt                 # Sostituisci (prima
    occorrenza)
sed 's/old/new/g' file.txt                # Sostituisci (tutte)
sed -i 's/old/new/g' file.txt             # Modifica file in-place
sed -n '5,10p' file.txt                   # Stampa righe 5-10
sed '/pattern/d' file.txt                 # Elimina righe con pattern
```

```
sed 's/^/prefix /' file.txt                    # Aggiungi prefix
```

cut

```
cut -d: -f1 /etc/passwd                        # Campo 1 (delimiter :)
cut -c1-10 file.txt                            # Caratteri 1-10
cut -d, -f1,3 file.csv                         # Campi 1 e 3 (CSV)
```

sort

```
sort file.txt                                  # Ordina alfabeticamente
sort -r file.txt                               # Reverse
sort -n file.txt                               # Ordine numerico
sort -u file.txt                               # Unique (rimuovi duplicati)
sort -k2 file.txt                              # Ordina per colonna 2
sort -t: -k3 -n /etc/passwd                    # Ordina per campo 3 numerico
```

uniq

```
uniq file.txt                                  # Rimuovi righe duplicate consecutive
uniq -c file.txt                               # Conta occorrenze
uniq -d file.txt                               # Solo duplicati
uniq -u file.txt                               # Solo unique

# Tipico uso con sort
sort file.txt | uniq                           # Rimuovi tutti duplicati
sort file.txt | uniq -c | sort -rn             # Conta + ordina
```

A.14 Variabili e Environment

Environment Variables

```
# Visualizzare
echo $HOME                                     # Variabile specifica
echo $PATH
env                                            # Tutte le variabili
printenv                                       # Alternativa a env
printenv HOME                                  # Variabile specifica

# Impostare
export VAR=value                               # Export variabile
VAR=value                                      # Solo sessione corrente
export PATH=$PATH:/new/path                    # Aggiungi a PATH

# File configurazione
~/.bashrc                                      # Bash configuration
~/.bash_profile                                # Login shell
~/.profile                                     # Generic shell
```

```
/etc/environment                    # System-wide
```

A.15 Redirection e Pipe

Redirection

```
command > file.txt                  # Redirect stdout (overwrite)
command >> file.txt                 # Redirect stdout (append)
command 2> error.log                # Redirect stderr
command &> all.log                  # Redirect stdout + stderr
command > output.txt 2>&1           # Redirect both (alternativa)
command < input.txt                 # Input da file
command > /dev/null                 # Scarta output
command 2>&1 | tee file.log         # Output + salva
```

Pipe e Combinazioni

```
command1 | command2                 # Pipe output a command2
cmd1 && cmd2                        # Esegui cmd2 se cmd1 OK
cmd1 || cmd2                        # Esegui cmd2 se cmd1 fallisce
cmd1 ; cmd2                         # Esegui sequenzialmente
cmd1 & cmd2                         # Esegui in parallelo
(cmd1; cmd2)                        # Subshell
```

A.16 Package Management

APT (Debian/Ubuntu)

```
sudo apt update                     # Aggiorna package list
sudo apt upgrade                    # Aggiorna packages
sudo apt install package            # Installa package
sudo apt remove package             # Rimuovi package
sudo apt purge package              # Rimuovi + config
sudo apt autoremove                 # Rimuovi dipendenze non usate
apt search keyword                  # Cerca package
apt show package                    # Info package
apt list --installed                # Lista packages installati
```

YUM/DNF (RedHat/CentOS/Fedora)

```
sudo yum update                     # Aggiorna packages
sudo yum install package            # Installa package
sudo yum remove package             # Rimuovi package
yum search keyword                  # Cerca package
yum info package                    # Info package
```

```
yum list installed                          # Lista installed

# DNF (newer)
sudo dnf update
sudo dnf install package
```

A.17 Shortcuts Bash

Editing Command Line

```
Ctrl+A          # Vai a inizio riga
Ctrl+E          # Vai a fine riga
Ctrl+U          # Cancella da cursore a inizio
Ctrl+K          # Cancella da cursore a fine
Ctrl+W          # Cancella parola precedente
Ctrl+Y          # Incolla ultimo cancellato
Ctrl+L          # Clear screen
Ctrl+C          # Interrompi comando
Ctrl+D          # EOF / logout
Ctrl+Z          # Sospendi processo
Ctrl+R          # Ricerca history
!!              # Ripeti ultimo comando
!$              # Ultimo argomento comando precedente
!*              # Tutti argomenti comando precedente
```

A.18 Reference Rapida Simboli

Special Characters

Simbolo	Significato
`.`	Directory corrente
`..`	Directory parent
`~`	Home directory
`/`	Root directory
`*`	Wildcard (qualsiasi caratteri)
`?`	Wildcard (un carattere)
`[]`	Character class
`\`	Escape character
`|`	Pipe
`>`	Redirect output
`»`	Append output
`<`	Input redirect
`&`	Background execution
`;`	Command separator
`&&`	AND operator
`||`	OR operator
`$`	Variable prefix
`#`	Comment

A.19 Permessi Numerici Reference

Permission Numbers

Numero	Permessi
0	—
1	–x
2	-w-
3	-wx
4	r–
5	r-x
6	rw-
7	rwx

Esempi comuni:

- `644`: rw-r–r– (file normali)
- `755`: rwxr-xr-x (script, directory)
- `700`: rwx—— (file privati)
- `600`: rw——- (chiavi SSH, config)
- `777`: rwxrwxrwx (tutto permesso - sconsigliato!)

A.20 Signal Reference

Common Signals

Signal	Numero	Azione
SIGHUP	1	Hangup (reload config)
SIGINT	2	Interrupt (Ctrl+C)
SIGQUIT	3	Quit
SIGKILL	9	Kill (non catchable)
SIGTERM	15	Terminate (graceful)
SIGSTOP	19	Stop (non catchable)
SIGTSTP	20	Stop (Ctrl+Z)
SIGCONT	18	Continue

A.21 Exit Codes

Common Exit Codes

Code	Significato
0	Success
1	General errors
2	Misuse of shell command
126	Command cannot execute
127	Command not found
128	Invalid exit argument
130	Terminated by Ctrl+C
255	Exit status out of range

Verificare exit code ultimo comando:

```
command
echo $?
```

A.22 Formati Data e Tempo

date Command Formats

```
date                                    # Data/ora corrente
date +%Y-%m-%d                          # 2025-01-15
date +%Y%m%d                            # 20250115
date +%H:%M:%S                          # 14:30:45
date +%s                                # Unix timestamp
date -d "yesterday"                     # Ieri
date -d "2 days ago"                    # 2 giorni fa
date -d "next Monday"                   # Prossimo lunedì
date -d @1234567890                     # Da timestamp

# Formati comuni
date +%Y%m%d_%H%M%S                     # 20250115_143045
date +"%Y-%m-%d %H:%M:%S"               # 2025-01-15 14:30:45
```

A.23 One-Liners Utili

Useful One-Liners

```
# Top 10 comandi più usati
history | awk '{print $2}' | sort | uniq -c | sort -rn | head -10

# File più grandi
find . -type f -exec du -h {} \; | sort -rh | head -10

# Processi che usano più CPU
ps aux --sort=-%cpu | head -10

# Processi che usano più memoria
ps aux --sort=-%mem | head -10

# IP connessi
netstat -tn 2>/dev/null | grep :80 | awk '{print $5}' | \
```

```
    cut -d: -f1 | sort | uniq -c | sort -rn

# File modificati oggi
find . -type f -mtime 0

# Backup rapido
tar -czf backup_$(date +%Y%m%d).tar.gz directory/

# Kill processi per nome
pkill -9 -f "process_name"

# Port check
for p in {1..1024}; do nc -zv localhost $p 2>&1 | grep succeeded;
    done

# Monitor file size real-time
watch -n 1 'du -sh /var/log'

# Download intero sito
wget --mirror --convert-links --page-requisites --no-parent URL
```

Appendice B

Esercizi e Soluzioni

B.1 Introduzione

Questa appendice contiene esercizi pratici organizzati per difficoltà crescente, completi di soluzioni dettagliate. Gli esercizi coprono tutti gli argomenti del corso e sono progettati per consolidare le competenze acquisite.

B.2 Esercizi Base: File System e Comandi

B.2.1 Esercizio 1: Navigazione e Operazioni Base

Obiettivo: Pratica con comandi base di navigazione e manipolazione file.

Task:

1. Creare directory `/tmp/lab` con sottodirectory `docs`, `scripts`, `backup`
2. Creare 5 file vuoti in `docs`: `file1.txt` ... `file5.txt`
3. Copiare tutti i .txt in `backup`
4. Rinominare `file1.txt` in `main.txt`
5. Eliminare `file5.txt`
6. Listare ricorsivamente tutta la struttura

Soluzione:

```
# 1. Creare directory
mkdir -p /tmp/lab/{docs,scripts,backup}

# 2. Creare file
cd /tmp/lab/docs
touch file{1..5}.txt
# Alternativa:
# for i in {1..5}; do touch "file$i.txt"; done

# 3. Copiare file
cp *.txt ../backup/
# Alternativa:
# cp file*.txt /tmp/lab/backup/

# 4. Rinominare
mv file1.txt main.txt
```

```
17
18 # 5. Eliminare
19 rm file5.txt
20
21 # 6. Listare ricorsivamente
22 cd /tmp/lab
23 ls -R
24 # Alternativa con tree (se installato):
25 # tree /tmp/lab
```

B.2.2 Esercizio 2: Ricerca File

Obiettivo: Usare `find` per ricerche avanzate.

Task:

1. Trovare tutti i file .log in /var/log modificati negli ultimi 7 giorni
2. Trovare file più grandi di 100MB in /home
3. Trovare directory vuote in /tmp
4. Trovare file con permessi 777 (pericolosi per sicurezza)
5. Eliminare tutti i file .tmp più vecchi di 30 giorni

Soluzione:

```
1  # 1. File .log recenti
2  find /var/log -name "*.log" -mtime -7
3
4  # 2. File grandi
5  find /home -type f -size +100M
6
7  # 3. Directory vuote
8  find /tmp -type d -empty
9
10 # 4. File 777 (potenzialmente insicuri)
11 find / -type f -perm 777 2>/dev/null
12
13 # 5. Eliminare .tmp vecchi
14 find /tmp -name "*.tmp" -type f -mtime +30 -delete
15 # Safer con conferma:
16 # find /tmp -name "*.tmp" -type f -mtime +30 -ok rm {} \;
```

B.2.3 Esercizio 3: Permessi

Obiettivo: Gestire permessi file e directory.

Task:

1. Creare script `backup.sh` con permessi esecuzione solo per owner
2. Creare directory `shared` accessibile a tutti ma solo owner può eliminare file
3. Creare file `config.conf` leggibile e scrivibile solo da owner
4. Trovare tutti i file SUID nel sistema

Soluzione:

```
# 1. Script eseguibile solo da owner
touch backup.sh
chmod 700 backup.sh
# Verifica:
ls -l backup.sh  # rwx------

# 2. Directory condivisa con sticky bit
mkdir shared
chmod 1777 shared
# Verifica:
ls -ld shared    # drwxrwxrwt

# 3. File config privato
touch config.conf
chmod 600 config.conf
# Verifica:
ls -l config.conf  # rw-------

# 4. Trovare file SUID
sudo find / -perm -4000 -type f -ls 2>/dev/null
# Output tipico include: /usr/bin/passwd, /usr/bin/sudo, etc.
```

B.3 Esercizi Intermedi: Text Processing

B.3.1 Esercizio 4: Log Analysis

Obiettivo: Analizzare log web server.

Dato file `access.log` con formato Apache:

```
192.168.1.1 - - [15/Jan/2025:10:30:45] "GET /index.html HTTP/1.1" 200 1234
192.168.1.2 - - [15/Jan/2025:10:31:12] "GET /about.html HTTP/1.1" 200 5678
192.168.1.1 - - [15/Jan/2025:10:32:01] "GET /api/data HTTP/1.1" 404 0
```

Task:

1. Contare quante richieste per ogni IP
2. Elencare tutti gli status code con conteggi
3. Trovare le 10 pagine più richieste
4. Calcolare traffico totale (somma byte)

Soluzione:

```
# 1. Richieste per IP
awk '{print $1}' access.log | sort | uniq -c | sort -rn
# Output:
# 2 192.168.1.1
# 1 192.168.1.2

# 2. Status code con conteggi
awk '{print $9}' access.log | sort | uniq -c | sort -rn
# Output:
# 2 200
# 1 404
```

```
12
13 # 3. Top 10 pagine richieste
14 awk '{print $7}' access.log | sort | uniq -c | sort -rn | head -10
15 # $7 è il campo URL
16
17 # 4. Traffico totale
18 awk '{sum += $10} END {print "Total bytes:", sum}' access.log
19 # $10 è il campo byte
20
21 # Bonus: Richieste per status code 404
22 awk '$9 == 404 {print $7}' access.log
23
24 # Richieste per ora
25 awk -F'[: \\[]' '{print $3":"$4}' access.log | sort | uniq -c
```

B.3.2 Esercizio 5: CSV Processing

Obiettivo: Elaborare file CSV con dati.

Dato file `sales.csv`:

```
1 Date,Product,Quantity,Price
2 2025-01-01,Laptop,2,1000
3 2025-01-01,Mouse,5,25
4 2025-01-02,Laptop,1,1000
5 2025-01-02,Keyboard,3,75
```

Task:

1. Calcolare revenue totale (Quantity × Price)
2. Trovare prodotto con vendite maggiori
3. Calcolare revenue per data
4. Estrarre solo vendite di Laptop

Soluzione:

```
1 # 1. Revenue totale
2 awk -F',' 'NR>1 {total += $3*$4} END {print "Total:", total}' sales.csv
3 # Output: Total: 3300
4
5 # 2. Prodotto con vendite maggiori
6 awk -F',' 'NR>1 {sales[$2] += $3*$4}
7            END {for (p in sales) print p, sales[p]}' sales.csv | \
8     sort -k2 -rn | head -1
9 # Output: Laptop 3000
10
11 # 3. Revenue per data
12 awk -F',' 'NR>1 {revenue[$1] += $3*$4}
13            END {for (d in revenue) print d, revenue[d]}' sales.csv |
                sort
14 # Output:
15 # 2025-01-01 2125
16 # 2025-01-02 1225
17
18 # 4. Solo vendite Laptop
19 awk -F',' '$2 == "Laptop"' sales.csv
20 # Alternativa con grep:
```

```
grep "Laptop" sales.csv

# Bonus: Media prezzo per prodotto
awk -F',' 'NR>1 {sum[$2] += $4; count[$2]++}
           END {for (p in sum) print p, sum[p]/count[p]}' sales.csv
```

B.4 Esercizi Avanzati: Scripting

B.4.1 Esercizio 6: Backup Script

Obiettivo: Creare script backup automatico.

Requisiti:

- Backup di /home/user/documents
- Formato: documents_YYYYMMDD.tar.gz
- Salva in /backup/
- Mantieni solo ultimi 7 backup
- Log operazioni in /var/log/backup.log
- Exit code appropriato

Soluzione:

```
#!/bin/bash
# backup_docs.sh - Automated document backup

set -euo pipefail

# Configuration
SOURCE="/home/user/documents"
BACKUP_DIR="/backup"
LOG_FILE="/var/log/backup.log"
DATE=$(date +%Y%m%d)
BACKUP_FILE="documents_${DATE}.tar.gz"
RETENTION=7

# Logging
log() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"
}

# Error handler
error_exit() {
    log "ERROR: $1"
    exit "${2:-1}"
}

# Main
main() {
    log "Starting backup of $SOURCE"

    # Validate source
    if [ ! -d "$SOURCE" ]; then
        error_exit "Source directory not found: $SOURCE" 1
```

```
    fi

    # Create backup directory
    mkdir -p "$BACKUP_DIR" || error_exit "Cannot create backup dir" 1

    # Perform backup
    log "Creating backup: $BACKUP_FILE"
    if tar -czf "$BACKUP_DIR/$BACKUP_FILE" -C "$(dirname "$SOURCE")" \
            "$(basename "$SOURCE")" 2>> "$LOG_FILE"; then
        log "Backup created successfully"

        # Verify backup
        if tar -tzf "$BACKUP_DIR/$BACKUP_FILE" > /dev/null 2>&1; then
            size=$(du -sh "$BACKUP_DIR/$BACKUP_FILE" | awk '{print $1}')
            log "Backup verified OK (size: $size)"
        else
            error_exit "Backup verification failed" 1
        fi
    else
        error_exit "Backup creation failed" 1
    fi

    # Cleanup old backups
    log "Cleaning up old backups (retention: $RETENTION days)"
    find "$BACKUP_DIR" -name "documents_*.tar.gz" -mtime +$RETENTION -
       delete

    log "Backup completed successfully"
    exit 0
}

main "$@"
```

Test:

```
# Test script
chmod +x backup_docs.sh
sudo ./backup_docs.sh

# Verifica backup creato
ls -lh /backup/

# Verifica log
tail /var/log/backup.log

# Aggiungi a cron per esecuzione automatica
# crontab -e
# 0 2 * * * /usr/local/bin/backup_docs.sh
```

B.4.2 Esercizio 7: System Monitor

Obiettivo: Script monitoring con alerting.

Requisiti:

- Monitora: CPU, memoria, disco
- Threshold: CPU>80%, MEM>85%, DISK>90%
- Invia alert se superato threshold

- Log metrics in JSON
- Esegui ogni 5 minuti via cron

Soluzione:

```bash
#!/bin/bash
# system_monitor.sh - System resource monitoring

set -euo pipefail

# Configuration
METRICS_FILE="/var/lib/monitor/metrics.json"
LOG_FILE="/var/log/system_monitor.log"
ALERT_EMAIL="admin@example.com"

# Thresholds
CPU_THRESHOLD=80
MEM_THRESHOLD=85
DISK_THRESHOLD=90

# Colors
RED='\033[0;31m'
YELLOW='\033[1;33m'
GREEN='\033[0;32m'
NC='\033[0m'

log() {
    echo "[$(date +'%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"
}

# Get metrics
get_cpu_usage() {
    top -bn1 | grep "Cpu(s)" | awk '{print $2}' | cut -d'%' -f1 | cut -d
        '.' -f1
}

get_mem_usage() {
    free | grep Mem | awk '{printf("%.0f", $3/$2 * 100.0)}'
}

get_disk_usage() {
    df -h / | awk 'NR==2 {print $5}' | sed 's/%//'
}

# Check thresholds
check_threshold() {
    local metric=$1
    local value=$2
    local threshold=$3

    if [ "$value" -gt "$threshold" ]; then
        log "ALERT: $metric at ${value}% (threshold: ${threshold}%)"
        echo -e "${RED} $ {NC} $metric: ${value}% > ${threshold}%"
        return 1
    else
        echo -e "${GREEN} $ {NC} $metric: ${value}%"
        return 0
    fi
```

```
}

# Send alert
send_alert() {
    local message=$1

    log "Sending alert email"
    echo "$message" | mail -s "System Alert: $(hostname)" "$ALERT_EMAIL"
}

# Save metrics as JSON
save_metrics() {
    local cpu=$1
    local mem=$2
    local disk=$3

    mkdir -p "$(dirname "$METRICS_FILE")"

    # Create JSON
    cat > "$METRICS_FILE" << EOF
{
  "timestamp": "$(date -u +%Y-%m-%dT%H:%M:%SZ)",
  "hostname": "$(hostname)",
  "metrics": {
    "cpu_percent": $cpu,
    "memory_percent": $mem,
    "disk_percent": $disk
  }
}
EOF
}

# Main
main() {
    log "Starting system check"

    # Collect metrics
    CPU=$(get_cpu_usage)
    MEM=$(get_mem_usage)
    DISK=$(get_disk_usage)

    log "Metrics - CPU: ${CPU}%, MEM: ${MEM}%, DISK: ${DISK}%"

    # Check thresholds
    alerts=()
    check_threshold "CPU" "$CPU" "$CPU_THRESHOLD" || alerts+=("CPU: ${
        CPU}%")
    check_threshold "Memory" "$MEM" "$MEM_THRESHOLD" || alerts+=("Memory
        : ${MEM}%")
    check_threshold "Disk" "$DISK" "$DISK_THRESHOLD" || alerts+=("Disk:
        ${DISK}%")

    # Save metrics
    save_metrics "$CPU" "$MEM" "$DISK"

    # Send alert if needed
    if [ ${#alerts[@]} -gt 0 ]; then
        alert_msg="System resources exceeded thresholds:\n\n"
```

```
        for alert in "${alerts[@]}"; do
            alert_msg+="- $alert\n"
        done
        send_alert "$alert_msg"
        exit 1
    fi

    log "All checks passed"
    exit 0
}

main "$@"
```

Setup:

```
# Installa e configura
sudo mkdir -p /var/lib/monitor
sudo chmod +x system_monitor.sh
sudo cp system_monitor.sh /usr/local/bin/

# Aggiungi a cron
sudo crontab -e
# */5 * * * * /usr/local/bin/system_monitor.sh

# Test manuale
sudo /usr/local/bin/system_monitor.sh

# Verifica metrics
cat /var/lib/monitor/metrics.json | jq .
```

B.4.3 Esercizio 8: User Management Script

Obiettivo: Script automazione creazione utenti.

Requisiti:

- Legge lista utenti da CSV (username,fullname,groups)
- Crea utenti con home directory
- Aggiunge a gruppi specificati
- Genera password casuale
- Invia credenziali via email
- Log tutte le operazioni

File users.csv:

```
username,fullname,groups
alice,Alice Smith,developers,sudo
bob,Bob Jones,developers
charlie,Charlie Brown,admins
```

Soluzione:

```
#!/bin/bash
# create_users.sh - Bulk user creation from CSV

set -euo pipefail
```

```
5
6  # Configuration
7  CSV_FILE="${1:-users.csv}"
8  LOG_FILE="/var/log/user_creation.log"
9  PASSWORD_LENGTH=16
10
11 log() {
12     echo "[$(date +'%Y-%m-%d %H:%M:%S')] $*" | tee -a "$LOG_FILE"
13 }
14
15 error_exit() {
16     log "ERROR: $1"
17     exit 1
18 }
19
20 # Generate random password
21 generate_password() {
22     tr -dc 'A-Za-z0-9!@#$%^&*' < /dev/urandom | head -c "
           $PASSWORD_LENGTH"
23 }
24
25 # Create user
26 create_user() {
27     local username=$1
28     local fullname=$2
29     local groups=$3
30
31     log "Processing user: $username"
32
33     # Check if user exists
34     if id "$username" &>/dev/null; then
35         log "User $username already exists, skipping"
36         return 0
37     fi
38
39     # Create user
40     log "Creating user $username ($fullname)"
41     if ! useradd -m -c "$fullname" -s /bin/bash "$username"; then
42         log "ERROR: Failed to create user $username"
43         return 1
44     fi
45
46     # Generate and set password
47     local password
48     password=$(generate_password)
49     echo "$username:$password" | chpasswd
50
51     # Add to groups
52     if [ -n "$groups" ]; then
53         log "Adding $username to groups: $groups"
54         IFS=',' read -ra group_array <<< "$groups"
55         for group in "${group_array[@]}"; do
56             group=$(echo "$group" | xargs)  # trim whitespace
57             if ! usermod -aG "$group" "$username"; then
58                 log "WARNING: Failed to add $username to group $group"
59             fi
60         done
61     fi
```

```
62
63      # Set password expiry (force change on first login)
64      passwd -e "$username"
65
66      # Send credentials email
67      send_credentials "$username" "$fullname" "$password"
68
69      log "User $username created successfully"
70  }
71
72  # Send credentials via email
73  send_credentials() {
74      local username=$1
75      local fullname=$2
76      local password=$3
77      local email="${username}@example.com"
78
79      local message
80      message=$(cat << EOF
81  Hello $fullname,
82
83  Your account has been created.
84
85  Username: $username
86  Password: $password
87
88  Please change your password on first login.
89
90  To login:
91  ssh $username@$(hostname)
92
93  Regards,
94  System Administration
95  EOF
96  )
97
98      echo "$message" | mail -s "Account Created: $username" "$email"
99      log "Credentials sent to $email"
100 }
101
102 # Process CSV file
103 process_csv() {
104     local csv=$1
105
106     if [ ! -f "$csv" ]; then
107         error_exit "CSV file not found: $csv"
108     fi
109
110     log "Processing CSV file: $csv"
111
112     # Read CSV (skip header)
113     local line_num=0
114     while IFS=',' read -r username fullname groups; do
115         ((line_num++))
116
117         # Skip header
118         if [ $line_num -eq 1 ]; then
119             continue
```

```
        fi

        # Trim whitespace
        username=$(echo "$username" | xargs)
        fullname=$(echo "$fullname" | xargs)
        groups=$(echo "$groups" | xargs)

        # Validate
        if [ -z "$username" ] || [ -z "$fullname" ]; then
            log "WARNING: Invalid line $line_num, skipping"
            continue
        fi

        create_user "$username" "$fullname" "$groups"

    done < "$csv"

    log "CSV processing completed"
}

# Main
main() {
    # Check if running as root
    if [ "$(id -u)" -ne 0 ]; then
        error_exit "Must run as root"
    fi

    log "=== Starting user creation ==="

    process_csv "$CSV_FILE"

    log "=== User creation completed ==="
}

main "$@"
```

Usage:

```
# Preparare CSV
cat > users.csv << EOF
username,fullname,groups
alice,Alice Smith,developers,sudo
bob,Bob Jones,developers
charlie,Charlie Brown,admins
EOF

# Eseguire script
sudo chmod +x create_users.sh
sudo ./create_users.sh users.csv

# Verificare utenti creati
tail -3 /etc/passwd
grep alice /etc/group

# Verificare log
tail /var/log/user_creation.log
```

B.5 Esercizi Network e SSH

B.5.1 Esercizio 9: Network Diagnostics

Obiettivo: Script diagnostica problemi rete.

Task: Creare script che:

- Verifica interfaccia di rete up
- Test connettività gateway
- Test DNS resolution
- Test connettività internet
- Report completo con troubleshooting hints

Soluzione:

```
#!/bin/bash
# network_diagnostics.sh - Network troubleshooting

set -euo pipefail

RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

ok() { echo -e "${GREEN} $ {NC} $*"; }
fail() { echo -e "${RED} $ {NC} $*"; }
warn() { echo -e "${YELLOW}!${NC} $*"; }

echo "=== Network Diagnostics ==="
echo ""

# 1. Check interface
echo "[1] Checking network interface..."
INTERFACE=$(ip route | grep default | awk '{print $5}' | head -1)

if [ -n "$INTERFACE" ]; then
    if ip link show "$INTERFACE" | grep -q "state UP"; then
        ok "Interface $INTERFACE is UP"
        IP=$(ip addr show "$INTERFACE" | grep "inet " | awk '{print $2}'
            )
        echo "   IP: $IP"
    else
        fail "Interface $INTERFACE is DOWN"
        echo "   Try: sudo ip link set $INTERFACE up"
        exit 1
    fi
else
    fail "No default interface found"
    exit 1
fi

# 2. Check gateway
echo ""
echo "[2] Checking gateway connectivity..."
GATEWAY=$(ip route | grep default | awk '{print $3}' | head -1)
```

```
41
42 if [ -n "$GATEWAY" ]; then
43     echo "   Gateway: $GATEWAY"
44     if ping -c 2 -W 2 "$GATEWAY" > /dev/null 2>&1; then
45         ok "Gateway $GATEWAY is reachable"
46     else
47         fail "Cannot reach gateway $GATEWAY"
48         echo "   Check: ip route, physical connection"
49         exit 1
50     fi
51 else
52     fail "No default gateway configured"
53     echo "   Try: sudo ip route add default via GATEWAY_IP"
54     exit 1
55 fi
56
57 # 3. Check DNS
58 echo ""
59 echo "[3] Checking DNS resolution..."
60 if nslookup google.com > /dev/null 2>&1; then
61     ok "DNS resolution working"
62     DNS=$(cat /etc/resolv.conf | grep nameserver | head -1 | awk '{print
       $2}')
63     echo "   DNS server: $DNS"
64 else
65     fail "DNS resolution failed"
66     echo "   Check: /etc/resolv.conf"
67     echo "   Try adding: nameserver 8.8.8.8"
68     exit 1
69 fi
70
71 # 4. Check internet connectivity
72 echo ""
73 echo "[4] Checking internet connectivity..."
74 if ping -c 2 -W 3 8.8.8.8 > /dev/null 2>&1; then
75     ok "Internet connectivity OK (ping 8.8.8.8)"
76 else
77     fail "No internet connectivity"
78     echo "   Possible issues: firewall, routing, ISP"
79     exit 1
80 fi
81
82 # 5. Check HTTP/HTTPS
83 echo ""
84 echo "[5] Checking HTTP/HTTPS connectivity..."
85 if curl -s --connect-timeout 5 http://google.com > /dev/null; then
86     ok "HTTP connectivity OK"
87 else
88     warn "HTTP connectivity issues"
89 fi
90
91 if curl -s --connect-timeout 5 https://google.com > /dev/null; then
92     ok "HTTPS connectivity OK"
93 else
94     warn "HTTPS connectivity issues"
95 fi
96
97 # Summary
```

```
echo ""
echo "=== Summary ==="
ok "All network checks passed!"
echo ""
echo "Network Information:"
echo "  Interface: $INTERFACE"
echo "  IP: $IP"
echo "  Gateway: $GATEWAY"
echo "  DNS: $DNS"
```

B.5.2 Esercizio 10: SSH Key Management

Obiettivo: Script setup SSH keys su server multipli.

Requisiti:

- Genera chiave Ed25519 se non esiste
- Copia chiave su lista server
- Testa connessione senza password
- Report successi/fallimenti

Soluzione:

```
#!/bin/bash
# setup_ssh_keys.sh - Setup SSH keys on multiple servers

set -euo pipefail

# Configuration
KEY_FILE="$HOME/.ssh/id_ed25519"
SERVERS_FILE="${1:-servers.txt}"

log_ok() { echo "[OK] $*"; }
log_fail() { echo "[FAIL] $*"; }
log_info() { echo "[INFO] $*"; }

# Generate SSH key if not exists
generate_key() {
    if [ -f "$KEY_FILE" ]; then
        log_info "SSH key already exists: $KEY_FILE"
        return 0
    fi

    log_info "Generating new Ed25519 SSH key..."
    ssh-keygen -t ed25519 -f "$KEY_FILE" -N "" -C "$(whoami)@$(hostname)
        "

    if [ -f "$KEY_FILE" ]; then
        log_ok "SSH key generated: $KEY_FILE"
    else
        log_fail "Failed to generate SSH key"
        return 1
    fi
}

# Copy key to server
copy_key_to_server() {
```

```
34      local server=$1
35
36      log_info "Copying key to $server..."
37
38      if ssh-copy-id -i "$KEY_FILE" "$server" 2>/dev/null; then
39          log_ok "Key copied to $server"
40          return 0
41      else
42          log_fail "Failed to copy key to $server"
43          return 1
44      fi
45  }
46
47  # Test SSH connection
48  test_connection() {
49      local server=$1
50
51      log_info "Testing connection to $server..."
52
53      if ssh -o BatchMode=yes -o ConnectTimeout=5 "$server" "echo 'SSH OK'
        " \
54              >/dev/null 2>&1; then
55          log_ok "SSH connection successful: $server"
56          return 0
57      else
58          log_fail "SSH connection failed: $server"
59          return 1
60      fi
61  }
62
63  # Process servers file
64  process_servers() {
65      local file=$1
66
67      if [ ! -f "$file" ]; then
68          log_fail "Servers file not found: $file"
69          return 1
70      fi
71
72      local success=0
73      local failed=0
74
75      while IFS= read -r server; do
76          # Skip empty lines and comments
77          [[ -z "$server" || "$server" =~ ^# ]] && continue
78
79          echo ""
80          echo "=== Processing: $server ==="
81
82          if copy_key_to_server "$server"; then
83              if test_connection "$server"; then
84                  ((success++))
85              else
86                  ((failed++))
87              fi
88          else
89              ((failed++))
90          fi
```

```
    done < "$file"

    echo ""
    echo "=== Summary ==="
    echo "Success: $success"
    echo "Failed: $failed"

    return 0
}

# Main
main() {
    echo "=== SSH Key Setup ==="
    echo ""

    # Generate key
    generate_key || exit 1

    echo ""
    echo "Public key:"
    cat "${KEY_FILE}.pub"
    echo ""

    # Process servers
    process_servers "$SERVERS_FILE"
}

# Usage
if [ $# -lt 1 ]; then
    cat << EOF
Usage: $0 <servers_file>

servers_file format (one server per line):
  user@hostname
  user@192.168.1.100
  user@server.example.com:2222

Example:
  echo "user@server1.com" > servers.txt
  echo "admin@192.168.1.100" >> servers.txt
  $0 servers.txt
EOF
    exit 1
fi

main "$@"
```

Usage:

```
# Creare file server list
cat > servers.txt << EOF
user@web1.example.com
user@web2.example.com
admin@db.example.com
EOF

# Eseguire script
chmod +x setup_ssh_keys.sh
```

```
./setup_ssh_keys.sh servers.txt

# Testare connessioni
ssh user@web1.example.com "hostname"
```

B.6 Riepilogo

Questi esercizi coprono:

- File system operations e permessi
- Text processing e log analysis
- Bash scripting avanzato con error handling
- System administration e automation
- Network diagnostics e troubleshooting
- SSH key management

Ogni soluzione include best practices:

- Strict mode (`set -euo pipefail`)
- Proper error handling
- Logging dettagliato
- Input validation
- Exit codes appropriati
- Commenti e documentation

B.7 Esercizi Aggiuntivi (Challenges)

B.7.1 Challenge 1: Log Aggregator

Creare script che raccoglie log da server multipli via SSH, aggrega e analizza.

B.7.2 Challenge 2: Deployment Pipeline

Implementare pipeline completo: build → test → deploy → rollback se fallisce.

B.7.3 Challenge 3: Monitoring Dashboard

Generare report HTML con metriche sistema, grafici, alert history.

B.7.4 Challenge 4: Automated Backup Rotation

Sistema backup completo con daily/weekly/monthly rotation e verifica integrità.

B.7.5 Challenge 5: Security Audit Script

Script che verifica configurazione sicurezza: SSH hardening, firewall, updates, SUID files, etc.