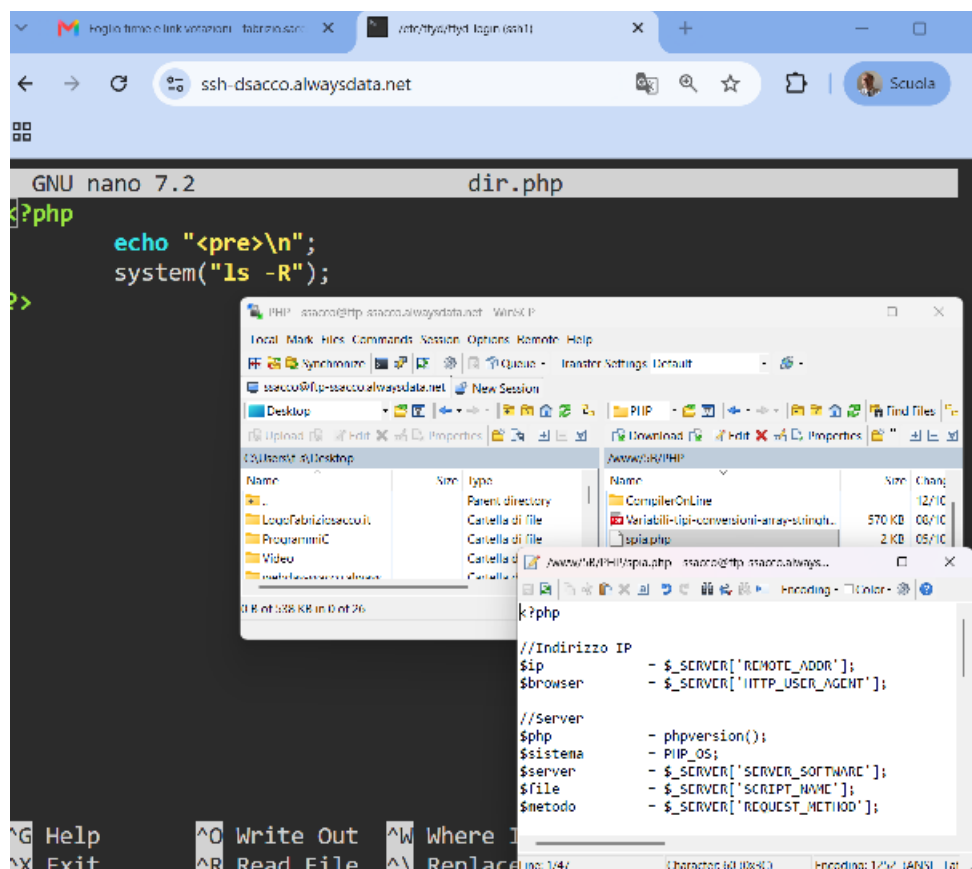


Tutorial: Comandi Linux e Software disponibile su Alwaysdata.net



Indice

1. [Comandi Linux fondamentali](#)
2. [Alwaysdata: piano gratuito e accesso SSH](#)
3. [Compilatori e interpreti disponibili via SSH](#)

Parte 1 – Comandi Linux fondamentali

Comando	Descrizione	Esempio
pwd	Mostra il percorso della directory corrente	pwd
ls	Elenca file e cartelle	ls -la
cd	Cambia directory	cd /var/www
mkdir	Crea una nuova directory	mkdir nuovo_progetto
rmdir	Rimuove una directory vuota	rmdir vecchia_cartella
cp	Copia file o directory	cp file.txt /tmp/
mv	Sposta o rinomina	mv vecchio.txt nuovo.txt
rm	Elimina file o directory	rm file.txt
cat	Mostra il contenuto di un file	cat /etc/hostname

Comando	Descrizione	Esempio
nano	Editor di testo semplice	nano ~/script.sh
chmod	Modifica i permessi di un file	chmod +x script.sh
chown	Cambia proprietario di un file	chown utente:gruppo file.txt
df	Mostra lo spazio libero sui filesystem	df -h
du	Stima lo spazio usato da file/cartelle	du -sh /var/log
ps	Elenca i processi attivi	ps aux
kill	Termina un processo tramite PID	kill -9 1234
man	Mostra il manuale di un comando	man ls
grep	Cerca testo dentro file	grep "errore" log.txt
find	Cerca file per nome, data, permessi	find /home -name "*.log"
zip/unzip	Comprime o decomprime file ZIP	zip -r backup.zip cartella/
ssh	Connessione remota sicura	ssh utente@host
wget	Scarica file da URL	wget https://esempio.com/file.zip

Seguono alcuni esempi di utilizzo dei comandi principali di linux

pwd — chiedo dove sono? Mi indica la cartella dove mi trovo

```
pwd
$ /home/utente
```

Stampa il percorso assoluto della directory corrente.

ls — elencare i file

```
ls          # elenco semplice
ls -l       # formato lungo: permessi, proprietario, dimensione, data
ls -a       # include i file nascosti (che iniziano con .)
ls -lh      # dimensioni "umane" (K, M, G)
ls -la /var/www # combinato, su un percorso specifico
```

cd — spostarsi

```
cd /var/www      # percorso assoluto
cd ~/sito         # nella cartella "sito" sotto la home
cd ..            # su di un livello
cd -             # torna alla directory precedente
cd              # torna alla home directory
```

mkdir — creare cartelle

```
mkdir progetto
mkdir -p progetto/src/js    # -p crea tutti i livelli intermedi
```

rmdir / rm — eliminare

```
rmdir vuota/                # solo cartelle vuote
rm file.txt                 # elimina un file (senza conferma!)
rm -r cartella/            # elimina ricorsivamente
rm -rf cartella/           # anche senza prompt: usare con molta cautela
```

File e testo

nano — editor semplice

```
nano index.html            # apre il file (lo crea se non esiste)
```

Scorciatoie essenziali (mostrate in basso, ^ = Ctrl):

- Ctrl+O → salva (poi Invio)
- Ctrl+X → esci
- Ctrl+W → cerca
- Ctrl+K → taglia riga

Altri esempi di comandi utili per i file

```
cp file.txt copia.txt      # copiare
cp -r dir/ backup/        # copiare una cartella
mv vecchio.txt nuovo.txt  # spostare o rinominare
mv file.txt ~/archivio/   # spostare in altra cartella
cat file.txt              # mostrare il contenuto
less file.txt             # visualizzare paginato (q per uscire)
head -n 20 file.log       # prime 20 righe
tail -f /var/log/app.log  # seguire le righe aggiunte in tempo reale
grep "errore" log.txt     # cercare testo
grep -rn "funzione" src/  # ricerca ricorsiva con numeri di riga
find . -name "*.conf"     # cercare file per nome
touch nuovo.txt           # creare file vuoto / aggiornare data
```

Permessi e chmod

Ogni file ha permessi per **proprietario**, **gruppo** e **altri**, in forma `rwXr-Xr--` (lettura, scrittura, esecuzione).

```
ls -l script.sh
# -rwxr-xr-- 1 utente gruppo 245 ott 11 10:00 script.sh
```

chmod — modificare i permessi

```
chmod +x script.sh      # rendere eseguibile
chmod 755 script.sh     # rwxr-xr-x (tipico per script/web)
chmod 644 index.html    # rw-r--r-- (tipico per file web)
chmod 600 chiave_ssh    # rw----- (obbligatorio per le chiavi private SSH!)
```

Tabella dei numeri (somma: r=4, w=2, x=1):

Numero	Permesso	Significato
7	rwx	tutto
6	rw-	lettura + scrittura
5	r-x	lettura + esecuzione
4	r--	sola lettura
0	---	nessuno

Altri comandi collegati:

```
chown utente:gruppo file # cambiare proprietario (spesso serve root)
chgrp gruppo file        # cambiare solo il gruppo
```

Spazio disco

df — spazio libero sui filesystem

```
df -h      # dimensioni umane (es. 12G, 450M)
df -h /home # solo il filesystem che contiene /home
```

du — spazio usato da file/cartelle

```
du -h file.txt      # dimensione di un file
du -sh cartella/    # totale "sintetico" di una cartella
du -h --max-depth=1 # quanto pesa ogni sottocartella del livello corrente
```

Processi e sistema

```
ps aux      # tutti i processi in esecuzione
top         # monitor interattivo (q per uscire)
kill 1234   # terminare un processo (PID)
kill -9 1234 # forzare la terminazione
free -h    # uso della memoria RAM
uptime     # carico e tempo di attività
uname -a   # versione del kernel e architettura
```

Rete e SSH

```
ssh utente@ssh-esempio.alwaysdata.net # connettersi a un server remoto
ssh -p 22 utente@host                  # specificando la porta
scp file.txt utente@host:~/            # copiare un file via SSH
scp utente@host:~/remoto.txt .         # scaricare un file remot
```

Compilatori e interpreti

I **compilatori** traducono l'intero codice sorgente in un file eseguibile binario (es. gcc, g++, javac); gli **interpreti** eseguono il codice riga per riga senza produrre un eseguibile (es. python, node, ruby, php). Su un server di hosting come alwaysdata in genere si usano gli interpreti; i compilatori servono per programmi in C, C++ o linguaggi compilati (Go, Rust hanno il loro compilatore integrato).

GCC (C) e G++ (C++)

```
# Verificare le versioni installate
gcc --version
g++ --version

# Compilare un programma C
gcc sorgente.c -o programma          # produce l'eseguibile "programma"
./programma                          # eseguirlo (serve il ./ per il percorso corrente)

# Compilare un programma C++
g++ sorgente.cpp -o programma
./programma

# Con warnings e standard specifici (consigliato)
gcc -Wall -Wextra sorgente.c -o programma
g++ -Wall -std=c++17 sorgente.cpp -o programma

# Compilazione separata + link (progetti multi-file)
g++ -c modulo1.cpp -o modulo1.o
g++ -c modulo2.cpp -o modulo2.o
g++ modulo1.o modulo2.o -o programma
```

Altri strumenti collegati:

```
make          # automatizzare la compilazione (Makefile)
make clean    # pulire gli artefatti di build
```

Nota: un eseguibile compilato funziona solo se ha il permesso x (chmod +x) e se è compilato per l'architettura giusta (uname -m, tipicamente x86_64).

Interpreti comuni

```
python3 script.py          # eseguire uno script Python
python3 -m venv venv        # creare un ambiente virtuale
node app.js                 # eseguire JavaScript con Node.js
ruby script.rb              # eseguire Ruby
php script.php              # eseguire PHP da riga di comando
php -S localhost:8000        # server di sviluppo PHP
perl script.pl              # eseguire Perl
bash script.sh              # eseguire uno script shell
```

Compilatori per altri linguaggi

```
go build main.go           # Go: compila (produce un binario)
cargo build                 # Rust: compila tramite Cargo
cargo run                   # compila ed esegue
javac Main.java             # Java: produce il bytecode Main.class
java Main                   # esegue il bytecode sulla JVM
```

Combinare i comandi

```
comando1 | comando2      # pipe: l'output del primo diventa input del secondo
comando1 > out.txt        # redirectione su file (sovrascrive)
comando1 >> out.txt       # redirectione in coda (append)
ls *.log | wc -l          # contare quanti file .log ci sono
history | grep ssh        # ritrovare i comandi ssh digitati
man ls                    # manuale di un comando (q per uscire)
```

Parte 2 – Alwaysdata.net: piano gratuito e SSH

Alwaysdata è un provider di hosting francese che offre un piano gratuito permanente, senza pubblicità e senza carta di credito. L'ambiente è **managed**: non hai privilegi di root e **non puoi installare software tramite apt o simili**. Puoi solo usare il software già installato e, per alcuni linguaggi, installare librerie nello spazio utente tramite i rispettivi package manager (pip, npm, gem, ecc.). Comunque se hai i sorgenti di un software, ad esempio, ffmpeg, puoi via make ricostruire l'eseguibile e a quel punto è come aver installato il pacchetto).

Caratteristiche del piano gratuito (Cloud Public – Free)

Risorsa	Valore
Spazio disco	1 GB SSD (include file, email e database)
RAM	256 MB
CPU	1/4 di core
Backup	3 giorni rolling
Siti web	Illimitati, ma limitati all'indirizzo *.alwaysdata.net
Database	MySQL/MariaDB, PostgreSQL, RabbitMQ (illimitati come numero)
Email	Caselle illimitate, supporto dominio personalizzato
SSL	Certificati Let's Encrypt gratuiti
Accesso remoto	SSH, SFTP, FTP, WebDAV

Limitazioni: non può essere usato per scopi di lucro; la creazione di siti web è limitata all'indirizzo *.alwaysdata.net del proprio account.

Accesso SSH

L'accesso SSH è **disabilitato di default** e va attivato dalla dashboard:

1. Accedi a admin.alwaysdata.com
2. Vai su **Remote access** → **SSH/SFTP**
3. Abilita l'utente SSH (autenticazione con password o chiave)

Connessione:

ssh [account]@ssh-[account].alwaysdata.net

- **Host:** ssh-[account].alwaysdata.net
 - **Porta:** 22
 - **Utente:** il tuo nome account Alwaysdata
-

Parte 3 – Compilatori e interpreti disponibili via SSH

Nella tabella seguente sono elencati i linguaggi supportati ufficialmente da Alwaysdata, con i comandi principali, le versioni disponibili e una breve descrizione.

Linguaggio	Comando/i	Versioni disponibili	Descrizione
Python	python	2.4 – 3.14	Interprete Python. Include <code>pip</code> per la gestione dei pacchetti. Framework come Django, Flask e Sanic sono autoinstallabili. Usa sempre <code>python</code> (non <code>python3/python2</code>).
PHP	php	4.4 – 8.5	Interprete PHP, eseguito in FastCGI dietro Apache. Puoi personalizzare <code>php.ini</code> e gestire estensioni dalla dashboard.
Node.js	node, npm	6 – 24	Runtime JavaScript lato server. <code>npm</code> è il gestore di pacchetti.
Ruby	ruby, gem	1.8 – 4.0	Interprete Ruby. <code>gem</code> è il gestore di pacchetti. Supporta framework come Ruby on Rails e Sinatra.
Perl	perl	Perl 5 (5.10+)	Interprete Perl, supportato per script CGI e applicazioni web.
Lua	lua	5.1 – 5.4	Interprete Lua. Versioni multiple disponibili.
Elixir	elixir	1.13 – 1.18	Linguaggio funzionale basato sulla BEAM VM. Supporta il framework Phoenix.
Deno	deno	(runtime supportato)	Runtime JavaScript/TypeScript moderno, alternativo a

Linguaggio	Comando/i	Versioni disponibili	Descrizione
			Node.js.
Java	javac, java	8, 11, 17, 21, 25	JDK completo. javac compila il codice Java in bytecode; java esegue il bytecode. Supporta framework JVM come Scala, Spring, Struts, Hibernate.
Go	go	1.22	Toolchain Go: compila ed esegue programmi Go.
Rust	rustc, cargo	1.78	rustc compila Rust; cargo è il build tool e package manager.
.NET	dotnet	5.0, 6.0, 8.0, 10.0	CLI .NET per compilare ed eseguire applicazioni C#, F#, VB.NET.
C / C++	gcc, g++	<i>non ufficialmente elencati</i>	Non sono elencati tra i linguaggi supportati ufficialmente. L'ambiente è basato su Debian, quindi potrebbero essere presenti: verifica con <code>gcc --version / g++ --version</code> . In alternativa, compila localmente e carica i binari.

Dettaglio dei singoli comandi

- **python** – Avvia l'interprete Python. Per eseguire uno script: `python script.py`. Per installare un pacchetto nello spazio utente: `python -m pip install --user <pacchetto>`.
- **php** – Esegue uno script PHP da riga di comando: `php script.php`.
- **node** – Esegue JavaScript lato server: `node app.js`. **npm** gestisce i pacchetti: `npm install <pacchetto>`.
- **ruby** – Esegue script Ruby: `ruby script.rb`. **gem** gestisce le gemme: `gem install --user-install <gemma>`.
- **perl** – Esegue script Perl: `perl script.pl`.
- **lua** – Esegue script Lua: `lua script.lua`.
- **elixir** – Esegue script Elixir: `elixir script.exs`. Per i progetti: `mix`.
- **deno** – Esegue script TypeScript/JavaScript: `deno run script.ts`.
- **javac** – Compila sorgenti Java: `javac Main.java` produce `Main.class`. **java** esegue la classe compilata: `java Main`.

- **go** – Compila ed esegue: `go build` genera un binario, `go run main.go` esegue direttamente.
 - **rustc** – Compila un file Rust: `rustc main.rs` produce un eseguibile. **cargo** gestisce progetti: `cargo build`, `cargo run`.
 - **dotnet** – Compila ed esegue progetti .NET: `dotnet build`, `dotnet run`.
 - **gcc** – Compilatore C: `gcc -o programma main.c`. **g++** – Compilatore C++: `g++ -o programma main.cpp`.
-

Piano Free (0 €/mese, senza limiti di tempo)

- **1 GB** di spazio disco
- **256 MB** di RAM
- **¼ di CPU**
- Accesso **SSH, FTP, SFTP, WebDAV, HTTPS**
- Siti con indirizzo **alwaysdata.net**; email con proprio dominio
- Database **PostgreSQL** (e MariaDB (MySQL)) senza limiti di numero
- Certificati SSL gratuiti (Let's Encrypt), fino a 10 processi attivi
- Nessuna carta di credito richiesta; uso **personale**

Conclusione

Questo documento integra i comandi Linux di base con l'elenco completo dei compilatori e interpreti disponibili su Alwaysdata (piano gratuito). L'ambiente è managed: non è possibile installare software di sistema, ma puoi sfruttare i linguaggi preinstallati e i loro package manager per le librerie utente.

Le versioni e la disponibilità possono variare; consulta la documentazione ufficiale di Alwaysdata per informazioni aggiornate.