

# Comandi Linux

## comando [opzioni] <argomenti>

Navigare nel file system → cd <dir>

- → dir corrente
- .. → dir madre
- ~ → dir home

Creare dir → mkdir ~~semplicemente~~ <dir>

1 Visualizz. contenuto dir → ls [-a, l, s, F, R] <dir>

2 Creare file → touch [-a, c, m] <file> \*

3 Modificare file → nano <file> (editor) \*

4 Visualizzare file → cat [-n, v, e] <file> \*

5 Eliminare file/dir → rm [-r, i, f] <file>...<fileN> \*

(Si possono visualizzare più file alla volta)

6 Spostare file → mv [-i, f] <file>..<fileN> <dir> \*

7 Scaricare file → wget <indirizzo> -O file

1 a → visual file nascosti  
l → info file  
s → dimens. byte  
F → aggiungere un carat. al nome file  
R → visualizza la sotto dir

2 C → non crea nuovi file  
a → imposta data/ora accesso  
m → // // modifica

4 n → conta linee  
v → visual. caratteri non stampabili  
e → \$ fine ogn. linea

5 r → cancella dir + contenuto  
i → chiede conferma del rm  
f → elim. diretta

6 i → conferma prima di sovrscrivere  
f → no conferma .. ..

j .. ..  
u .. ..  
r .. ..

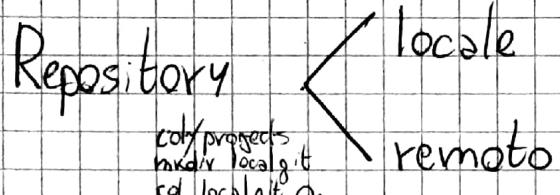
# G.t

## Sisteme a controllo di versione distribuito

### Setup utente

```
git config --global user.name/email
```

(accesso al server)



se non esiste remoto si deve prima creare il locale e collegarlo a uno remoto successivamente

**git init**

(comando creazione repository)



**git clone**

https:// ... / Homework-name.git

cd homework-name

(comando clonazione remota)



**git status**

(vediamo stato dei file)



**git add <file>**

(modifichiamo un file)



**git commit -m "messaggio"**

confermiamo le modifiche nel repository locale

**git log**

elenco dei file committati

**git diff / diff--staged**

visualizzo modifiche apportate / in staging area

**git checkout "namefile"**

riporto un file in working area

**git checkout HEAD**

riporta il file all'ultimo stato prima della modifica del commit

**git revert**

~ prima di una modifica

● **Sync repository**

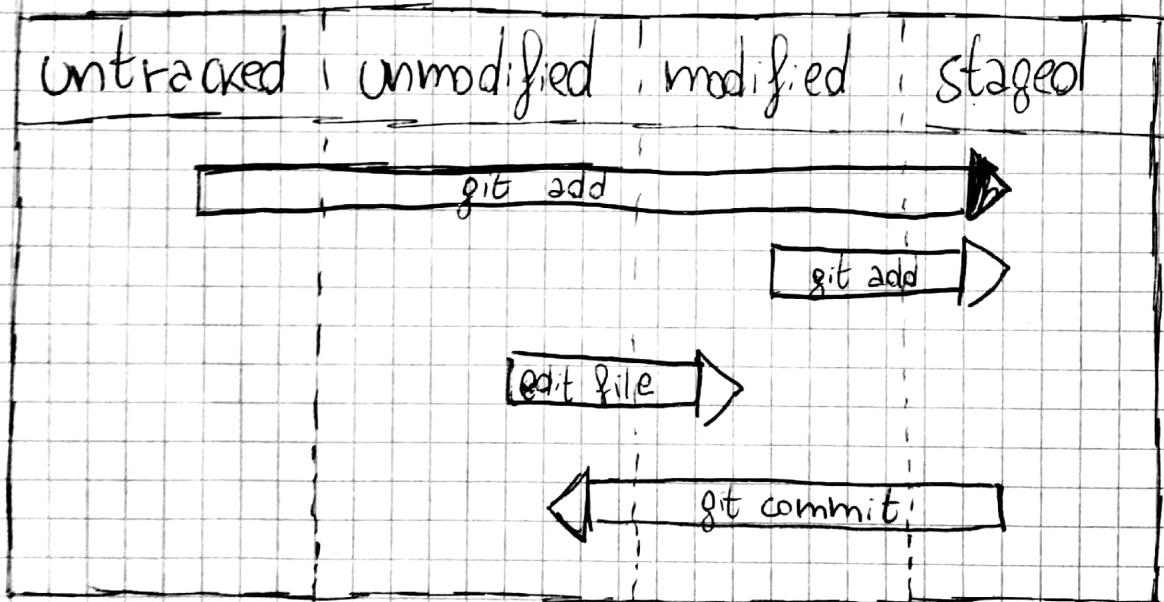
- \* **git push**
- \* **git pull**

allineare repository remoto a locale  
.. .. locale a remoto

\* → importanti

**git tag** etichetta un commit

## Ciclo vita



ARPANET

Primo prototipo sperimentale

Anni 60'

i militari vogliono una rete a prova di attacchi nucleari  
(rete distribuita)

Anni 70'

Nasce protocollo TCP/IP  
(lingua di Internet)

Rete → Sistema che permette la condivisione di info, dat. e risorse.

Tipi di Reti

LAN

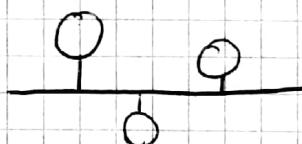
Rete locale

MAN

Unione di più reti locali

WAN

Unione di più reti locali + grande distanza  
LAN + MAN



Internet → rete di reti

Provider → fornitore accesso a internet

Gerarchia di Reti

Host

router

router

Host

App →→ protocoli app →→ App

Trasporto →→ TCP: Transmission Control program →→ Trasporto

Intraneo

Net work

Internet

Network

} IP: Internet protocol

Due tipi di reti

Commutazione a circuito (rete telefonica)

Commutazione a pacchetto (Internet)

Protocolli di rete  
(convenzione per comunicare  
tra macchine diverse)

Web; Http  
trasmissione  
visualizzazione  
file

SMTP

posta  
elettronica

FTP

trasmissione  
file

SSH  
collegamenti  
remoti

torrent  
condisone  
file

Gerarchici → instradamento e trasmissione separati e indipendenti  
dall'applicazione

End-to-End → funz. appl. svolte ai nodi della rete

best-effort → trasmissione a pacchetti nel miglior modo possibile.

TCP/IP

IP = Internet protocol

indirizzo a 32 bit  
per identificare dispositivi  
nel globo

es 174.168.29.32 /IPv4

Vengono assegnati da  
programmi chiamati  
DHCP (Dynamic Host configuration  
protocol) che possono  
fornire IP statici o dinamici

Si passerà a IPv6  
con 128 bit perché  
la capacità dell'IPv4  
sta per fermare  
(4 Bilioni di IP)

Repetizioni

DNS → Associare un IP a un nome per semplificare le ricerche (DOMAIN NAME SYSTEM)

es www.facebook.com ha un IP mascherato ~~www.facebook.com~~ dell'URL

ISP → Internet service provider

Uniform Resource Locator

Richiesta di pagina web tramite HTML → Hyper text markup Language

SSL → Secure Sockets Layer

TLS → Transport Layer Security

} Sistemi di sicurezza  
per la comunicazione

es https://... .com  
secure

Proprieta' algoritmo : 1- input specifico

2- Output specificato

3- Correttezza

4- Efficacia

5- terminazione

Debug → Attività di ricerca errori che portano il blocco o il malfunzionamento del programma.

- Tipi di errori:
- Sintassi (programma scritto sbagliato)
  - Logica (algoritmo sbagliato)
  - Runtime (errore durante l'esecuzione del program.)

## Prevenzione

Stile di  
programmazione  
adeguato

Utilizzo tecniche  
di ingegneria del  
software tipo  
design pattern

Metodologie agili

UML

tecniche  
testing codice

Interpretazione  
errori programma

Variabile → contenitore di dati (nome → pos memoria)  
possono essere di diverso tipo

Costante → come variabile ma non modificabile

locale  
variabili contenute  
in una routine

Variabile

variabile condivisa  
all'interno di tutto  
il programma

di sistema  
Variabili di default  
definite  
es mouseX, mouseY  
in processing

globale

# Costrutti condizionali

↓  
strutture di controllo

specificano se/quando e  
in che sequenza devono essere  
eseguite istruzioni

NB: in python esiste solo if else  
no switch case

if else / elif

esempio

if  $a > b$ : Se  $a > b$  allora  
 $b += 1$  Incremento b  
elif  $a == b$ : Altrimenti  $a = b$   
 $a += 1$  Incremento  
 $b += 1$  entrambi

else: Altrimenti  
 $a += 1$  Incremento a

# Costrutti iterativi

↓  
Eseguire stessa istruzione  
più volte (ripetizioni, cicli, loop)

break → uscita diretta dal ciclo

continue → passa all'iterazione  
successiva.

While "condizione":

Se l'istruzione è vera  
si continua a ciclare  
altrimenti si termina

for "condizione":

è ogni iterazione  
si aggiorna la cond.  
fino a quando non  
diventa falsa.

# Stringhe, tuple, liste\*

Sequenza di caratteri;  
(posso accedere a uno  
specifico carattere  
con un [indice])

Slicing [inizio: fine]  
esempio

a = "Andreati"

a[0:2] = "An"

a[2:5] = "dre"

ultimo carattere definito  
viene spesso

Iterare un "array"  
si utilizzano for/while  
per analizzare tutti gli  
elementi con un indice  
che incrementa a ogni  
ciclo

# Dizionari

Sono simili alle liste con  
elementi caratterizzate  
da chiave: valore

Esempio:  
d = {"dec": 10}

Sequenze immutabili  
di oggetti  
Si definisce una tupla  
con le ( )  
es. t = ("abc", 3, "ciao")

len, max, min sono funzioni  
utilizzabili, NB non si possono  
aggiungere o modificare  
elementi

# \*Liste,

Sequenza mutabile di  
oggetti, quindi si possono  
aggiungere/modificare  
elementi nella lista, con vari  
metodi.

es. l = [1, 2, 3]

l[0] = 4 diventa

[4, 2, 3]

# for (each)

Questo itera elementi per elemento

differenze dal classico for:

es. for player in players:  
print(player)

- più "robusto"
- non c'è indice
- non si possono confrontare array in parallelo
- itera solo in avanti
- non si confronta elemento precedente / successivo

## Nested array

array a n-dimensioni  
(come matrici)

es for i in range(10):  
    for j in range(10):

matrice 10x10

## Procedurale

funkzioni elementi  
organizzativo principio

(return) → funzioni → ritorna un valore  
(return) → procedure → esegue una procedura

- Classi      - Metodi

- Istanze      - Ereditarietà, classi e sottoclassi

- Attributi

## Programmazione

object-oriented  
programming

{ Oggetti (OOP)

gli oggetti svolgono  
la funzione di racchiudere  
un'unica unità organizzativa  
e i suoi comportamenti

Classi → per definire caratteristiche di un oggetto,  
• i suoi attributi e i suoi metodi  
("astratte" quando non si riferiscono  
a un oggetto specifico ma rappresentano  
solo un modello usato per creare istanze)

Istanze → oggetti creati a partire da una classe  
Una classe può essere usata per  
creare ~~una~~ istanze diverse dello  
stesso tipo con attributi diversi.

Attributi → Sono i valori che vengono assegnati  
all'istanza

es Rectangle(3, 5)  
istanza      base      altezza  
                              attributi

Metodi → descrivono il comportamento di un oggetto  
simili a funzioni e procedure

Ereditarietà, Classe e Sottoclassi → l'ereditarietà permette

di creare una nuova classe da una già esistente.  
(modificabile/estendibile)

es. Square eredita da Rectangle

Rectangle è superclasse mentre Square sottoclasse (o subclass)  
(da Rectangle io posso creare uno Square) ~~ma non~~  
(ma non viceversa)

# Creazione di una classe (esempio C-018 drive)

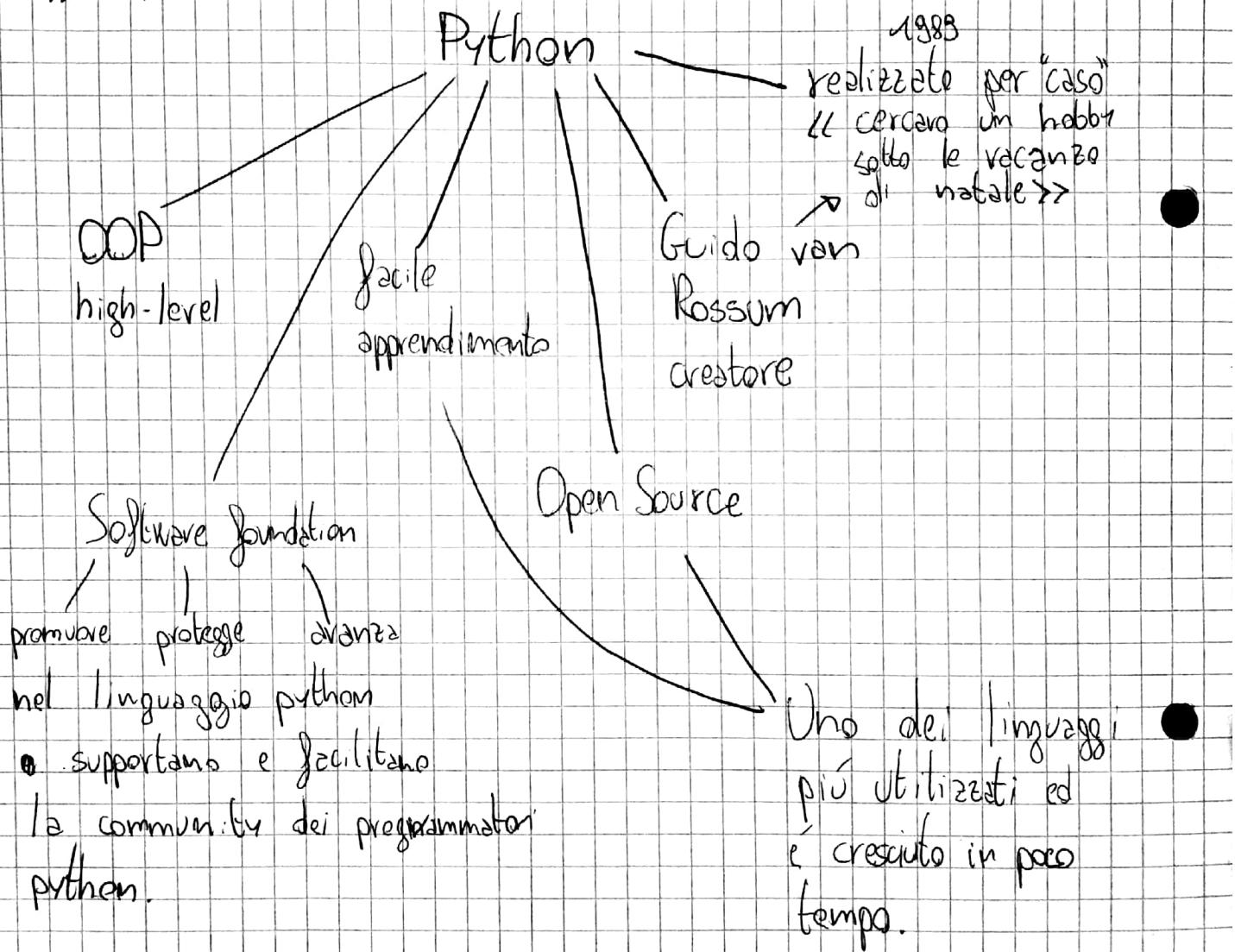
~~Persona~~

Name: Persona

Campi / Attributi : nome, cognome, altezza, ecc...

Metodi : mangia(), dormi(), studia(), ecc..

~~Scrivere~~



# Lavorare con i file

"r" → read mode

"w" → write mode

"a" → append mode

```
with open("filename.txt", "r, w, a") as file  
    file.write("ciao")
```

## Exception

Le eccezioni aiutano a gestire gli errori, dato che gli errori bloccano il programma il try, except ti permette di continuare l'esecuzione e prevenire es. il crash del programma.

try:

```
num = int(num)
```

```
except ValueError: ← Valuta questo errore  
    print("errore...")
```

~~se si accetta un errore specifico~~

Può essere utilizzato anche l'else dopo il try-except, il try deve solo contenere il codice che causa l'errore, se non causa errore e quindi il try viene eseguito successivamente viene eseguito l'else.

Si può anche accettare l'errore "silenziosamente" utilizzandolo il pass nel blocco except, così il programma continua

N.B. Solo except considera tutti gli errori, sconsigliato, bisognerebbe specificare che tipo di errore si ha.

N.B. Solo except prende in considerazione ogni errore, sconsigliato perché questo include interruzioni da tastiera e chiusura forzata del programma.

Meglio specificare l'errore

# Espressioni regolari (import re)

↓  
descrizione di uno schema di testo

esempio

+39 328-562 7690

tolgo il prefisso ↓

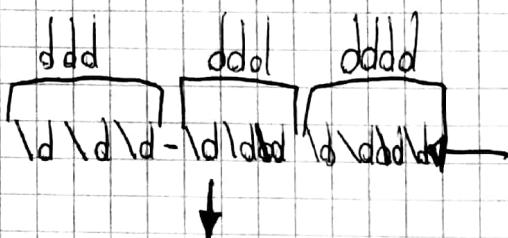
328-562 7690

↓ in corso diretta

999-999 9999

↓ metto d al posto di s

ddd-ddd dddd



\d{3}-\d{3} \d{4} (espressione regolare)

Simboli:

(\d\d\d) → gruppo di 3 cifre

\s → spazio

a|b → indica a oppure b

(a|b) → .. .. oppure nulla

\d → .. una cifra

\d{n} → .. esattamente n-cifre

[a-z] → .. un carattere da a → z (minuscolo)

[a-zA-Z] .. .. (minus. e maiuscolo)

CSV → Comma Separated Value

Separatore di valori attraverso una virgola.

È simile a un foglio excel.

Formato  
Scambio  
dati, o per

JSON → JavaScript Object Notation

Più utilizzato, ottimo per scambio dati in rete

~~è più leggero e si può usare con diversi linguaggi~~

e per inserire più elementi viene utilizzata una lista che li racchiude tutti.

XML → eXtensible Markup Language

Formato che utilizza i tag come html e inizia sempre con un elemento di root.

A ogni elemento posso fornire un attributo (come html)

JSON

Consente di scaricare le strutture dati python in file semplicemente e ricaricarne dati in un secondo avvio

Non è un formato python ma è ottimo per condividere dati e file con altri linguaggi.

Con dati memorizzati è buono saper gestire le eccezioni, assicurarsi che dei dati siano già presenti prima di sovrascrivere e tentare di caricarli prima di lavorarci.

## File

I programmi possono leggere e scrivere dati su file.

Lettura → permette il lavoro con una varietà di dati (e grazie alla scrittura un riutilizzo in un secondo momento)

Scrittura → permette di salvare dati come strutture sotto forma di liste.

Lettura → Aprire il file da programma

↓  
Utilizzo dati file

↓  
chiusura file

Scrittura → con l'argomento ~~"w"~~ "w" in open()  
possiamo scrivere su file (Python)

↓  
N.B. questo modifica tutto il file  
quindi cancella i salvataggi  
pre-esistenti.

↓  
Utilizzando "a" aggiungiamo  
dati (append) alla fine del file

Esemp. (190, 191, 192) ~~scrittura~~ ~~sovrascrittura~~ scrittura/sovrascrittura

with open("ciao.txt", "w") as f:  
...  
append/aggiunta

With open("ciao.txt", "a") as f:  
...  
append/aggiunta

N.B. con open() e il nome  
~~percorso~~ del file cerca  
direttamente nella cartella  
in cui ti trovi oppure  
de una sottocartella con  
~~percorso~~ il percorso del file.

pag 46, 47, 48 slide C024

es 193, 194 (pag 50, 51 - C024)

Moduli → file contenenti funzioni esterne  
al main (nella stessa cartella),  
utili a tenere "pulito" il main  
e a richiamare funzioni da  
questi stessi file per il  
funzionamento del programma.

## Ereditarietà delle classi

Quando una classe eredita da un'altra classe (ereditarietà)  
prende automaticamente i valori della classe  
principale, la classe "figlia" può implementare  
nuovi attributi o metodi / sovrascrivere gli  
attributi e i metodi della classe "padre"  
Per definire questa "parentela" bisogna  
includere il nome della classe "padre" quando  
ne definiamo una nuova "figlia"

Esempio

class MacchinaElettrica(Macchina):

```
    :  
    :  
def __init__(...)
```

NB Le classi vengono scritte  
in stile CamelCase (iniziali  
maiuscole)

```
super().__init__(...) ← richiamo gli attributi  
della classe padre
```

NB una classe può avere oggetti come  
attributi (per far lavorare più classi assieme)  
in situazioni complesse