Generalitat de Catalunya
Departament d'Educació
Institut Thos i Codina

El mòdul 'shutil' de Python

Domini Públic (CC0)

Índex

1. Copiar fitxers **1**
Detalls que cal saber . . . 2
Copiar entre objectes fitxer: `copyfileobj` . . . 2
Copiar només permisos o metadades . . . 2

2. Copiar directoris: `copytree` **3**
Excloure fitxers: `ignore_patterns` . . . 3
Enllaços simbòlics . . . 4
Errors a mitges . . . 4

3. Moure i canviar de nom: `move` **4**
Què fa per dins i per què és útil . . . 4

4. Esborrar arbres: `rmtree` **5**
Ignorar errors . . . 5
Gestionar errors per fitxer . . . 5
Seguretat . . . 6
Esborrar només el contingut . . . 6

5. Arxius comprimits: `make_archive` i `unpack_archive` **6**
Crear un arxiu . . . 6
Llistar formats disponibles . . . 7
Descomprimir . . . 7
Seguretat en descomprimir arxius de tercers . . . 7

6. Espai en disc: `disk_usage` **8**

7. Localitzar executables: `which` **8**

8. Altres funcions útils **9**
Mida de la terminal . . . 9
Propietari i grup: `chown` . . . 9

9. Exemples complets **9**
9.1. Còpia de seguretat amb data al nom . . . 9
9.2. Organitzar una carpeta de descàrregues per tipus . . . 10
9.3. Netejar fitxers temporals antics . . . 10
9.4. Copiar un projecte sense fitxers de desenvolupament . . . 10
9.5. Escriptura segura d'un fitxer amb còpia de seguretat prèvia . . . 11

10. `shutil` i `pathlib` **11**

11. Errors habituals **11**

12. Exercicis proposats **12**

13. Resum **12**

14. Referències **12**

`shutil` (*shell utilities*) ofereix **operacions d'alt nivell sobre fitxers i directoris**: copiar, moure, esborrar arbres sencers, comprimir, descomprimir, consultar l'espai en disc i localitzar executables. Fa el que faries a la shell amb `cp`, `mv`, `rm -r`, `tar`/`zip`, `df` i `which`, però de manera portable entre Linux, macOS i Windows.

```
import shutil
```

NOTA

Diferència amb os: `os` ofereix les operacions **bàsiques** (`os.rename`, `os.remove`, `os.mkdir`...), que es corresponen gairebé una a una amb crides al sistema. `shutil` s'hi construeix a sobre i resol els casos que `os` no cobreix: copiar contingut, copiar arbres, esborrar directoris no buits, moure entre sistemes de fitxers diferents i treballar amb arxius comprimits.

1. Copiar fitxers

Hi ha diverses funcions i la diferència entre elles és important.

Funció	Copia contingut	Copia permisos	Copia dates i metadades	El destí pot ser un directori
`shutil.copyfile(src, dst)`	Sí	No	No	**No**
`shutil.copy(src, dst)`	Sí	Sí	No	Sí
`shutil.copy2(src, dst)`	Sí	Sí	Sí (dates d'accés i modificació, flags...)	Sí

```
shutil.copyfile("origen.txt", "copia.txt")        # dst ha de ser un
 ↪  nom de fitxer
shutil.copy("origen.txt", "copia.txt")            # contingut +
 ↪  permisos
shutil.copy("origen.txt", "/tmp/")                # dins el directori
 ↪  /tmp, amb el mateix nom
shutil.copy2("origen.txt", "copia_exacta.txt")    # també conserva la
 ↪  data de modificació
```

Totes retornen el camí del fitxer creat.

Detalls que cal saber

- **`copyfile` no accepta un directori com a destí.** Si `dst` és un directori, llança `IsADirectoryError`. Cal indicar el nom complet del fitxer de destinació.
- **`copy2` conserva la data de modificació**, cosa útil per a còpies de seguretat i sincronització. Tanmateix, no pot copiar-ho absolutament tot: per exemple, el propietari del fitxer no es copia, i la data de creació no es conserva a Linux.
- Si origen i destí són **el mateix fitxer**, es llança `shutil.SameFileError`.
- Si el destí ja existeix, **se sobreescriu sense avisar**.
- Per defecte, si l'origen és un enllaç simbòlic, es copia el **fitxer apuntat**, no l'enllaç. Amb `follow_symlinks=False` es crea un enllaç simbòlic nou al destí.
- Des de Python 3.8, en Linux, macOS i Windows s'usen crides del sistema optimitzades (com `sendfile` o `copy_file_range`), de manera que copiar fitxers grans és ràpid i no carrega tot el fitxer a memòria.

Copiar entre objectes fitxer: `copyfileobj`

Copia **dades** entre dos objectes oberts (fitxers, sockets, fluxos en memòria…):

```
with open("origen.bin", "rb") as src, open("desti.bin", "wb") as dst:
    shutil.copyfileobj(src, dst)
```

Llegeix per blocs (tercer paràmetre, mida del buffer), així que no cal que el fitxer càpiga a memòria. És útil per escriure descàrregues HTTP directament a disc:

```
import urllib.request

with urllib.request.urlopen("https://example.com/fitxer.zip") as
 ↪  resposta, \
     open("fitxer.zip", "wb") as sortida:
    shutil.copyfileobj(resposta, sortida)
```

Copiar només permisos o metadades

```
shutil.copymode("origen.sh", "desti.sh")   # només els permisos (rwx)
shutil.copystat("origen.sh", "desti.sh")   # permisos + dates + flags
```

2. Copiar directoris: `copytree`

```
shutil.copytree("projecte", "projecte_backup")
```

Copia **recursivament** tot l'arbre. Per defecte, el directori de destí **no ha d'existir** (si existeix, llança `FileExistsError`). Des de Python 3.8 es pot permetre:

```
shutil.copytree("projecte", "projecte_backup", dirs_exist_ok=True)
```

Amb `dirs_exist_ok=True`, els fitxers existents al destí **se sobreescriuen** i els que només hi són al destí es **mantenen** (no és una sincronització, és una fusió).

Excloure fitxers: `ignore_patterns`

```
shutil.copytree(
    "projecte",
    "projecte_backup",
    ignore=shutil.ignore_patterns("*.pyc", "__pycache__", ".git",
    ↪  "*.tmp"),
)
```

`ignore_patterns` accepta patrons estil shell (`*`, `?`, `[abc]`). Si necessites una lògica pròpia, `ignore` rep una funció `f(directori, llista_de_noms)` que retorna el conjunt de noms a ignorar:

```
def ignora_grans(directori, noms):
    import os
    exclosos = []
    for nom in noms:
        cami = os.path.join(directori, nom)
        if os.path.isfile(cami) and os.path.getsize(cami) > 10 * 1024
        ↪  * 1024:
            exclosos.append(nom)
    return exclosos

shutil.copytree("dades", "dades_lleugeres", ignore=ignora_grans)
```

Enllaços simbòlics

```
shutil.copytree("origen", "desti", symlinks=True)
```

- `symlinks=False` (per defecte): s'hi copia el **contingut** dels enllaços simbòlics.
- `symlinks=True`: es **recreen els enllaços** simbòlics al destí.

Errors a mitges

Si algun fitxer no es pot copiar (permisos, nom no vàlid…), `copytree` continua amb la resta i al final llança `shutil.Error` amb la llista de totes les falles. Cal capturar-lo si vols que el programa continuï:

```
try:
    shutil.copytree("origen", "desti")
except shutil.Error as errors:
    for src, dst, motiu in errors.args[0]:
        print(f"No s'ha pogut copiar {src}: {motiu}")
```

Si vols canviar com es copia cada fitxer (per defecte `copy2`), usa `copy_function=shutil.copy` o una funció pròpia.

3. Moure i canviar de nom: `move`

```
shutil.move("informe.pdf", "arxiu/informe.pdf")
shutil.move("carpeta_vella", "carpeta_nova")
shutil.move("fitxer.txt", "/tmp/")        # si el destí és un
 ↪  directori, hi mou el fitxer dins
```

Què fa per dins i per què és útil

`os.rename` només funciona **dins del mateix sistema de fitxers** (la mateixa partició o muntatge). Si l'origen i el destí són a particions diferents, `os.rename` falla amb `OSError: Invalid cross-device link`. `shutil.move` resol el problema:

1. Intenta primer `os.rename` (ràpid, atòmic).
2. Si falla perquè són sistemes de fitxers diferents, **copia** l'origen al destí i **després esborra** l'origen.

Conseqüències:

- Dins del mateix sistema de fitxers, moure és instantani. Entre sistemes diferents, triga el que trigui la còpia i **no és atòmic**: si el procés s'interromp, pots tenir un fitxer duplicat o a mitges.
- Si el destí és un directori que ja conté un element amb el mateix nom, es llança `shutil.Error`.

- Si el destí és un fitxer existent, **depèn de la plataforma** (a POSIX se sobreescriu). Si vols un comportament previsible, comprova-ho abans o usa `os.replace`.

4. Esborrar arbres: `rmtree`

```
shutil.rmtree("carpeta_temporal")
```

Esborra el directori i **tot el que conté**, recursivament. És l'equivalent de `rm -rf`.

PRECAUCIÓ

És irreversible. No passa per la paperera. Comprova sempre el camí abans (sobretot si el construeixes a partir de variables o d'entrada d'usuari) i, si cal, mostra'l o demana confirmació.

Ignorar errors

```
shutil.rmtree("carpeta_temporal", ignore_errors=True)   # no falla si
 ↪  no existeix
```

`ignore_errors=True` fa que el programa no s'aturi si la carpeta no existeix o hi ha fitxers que no es poden esborrar. Té l'inconvenient que **amaga problemes reals**. Una alternativa més precisa és capturar el cas concret:

```
try:
    shutil.rmtree("carpeta_temporal")
except FileNotFoundError:
    pass
```

Gestionar errors per fitxer

Des de Python 3.12 el paràmetre es diu `onexc` (el vell `onerror` ha quedat obsolet). Rep una funció que es crida per cada error:

```
import os, stat

def força_esborrat(funcio, cami, excepcio):
    # Típic a Windows: fitxers de només lectura (p. ex. .git)
    os.chmod(cami, stat.S_IWRITE)
    funcio(cami)

shutil.rmtree("repositori", onexc=força_esborrat)     # Python 3.12+
```

Per a versions anteriors, equival a `onerror=` i la funció rep `(funcio, cami, exc_info)`.

Seguretat

`rmtree` no segueix els enllaços simbòlics a directoris: si troba un enllaç, **esborra l'enllaç, no el seu destí**. A més, usa un mecanisme que evita atacs de condició de carrera (*symlink attacks*) en plataformes que ho suporten. Si l'argument que li passes és ell mateix un enllaç simbòlic, llança `OSError`.

Esborrar només el contingut

`rmtree` esborra el directori mateix. Per buidar-lo i conservar-lo:

```
import os, shutil

def buida_directori(cami):
    for nom in os.listdir(cami):
        ruta = os.path.join(cami, nom)
        if os.path.isdir(ruta) and not os.path.islink(ruta):
            shutil.rmtree(ruta)
        else:
            os.remove(ruta)
```

5. Arxius comprimits: `make_archive` i `unpack_archive`

Crear un arxiu

```
shutil.make_archive("copia_seguretat", "zip", root_dir="projecte")
```

Retorna el camí de l'arxiu creat (`copia_seguretat.zip`). Els paràmetres són:

Paràmetre	Significat
`base_name`	Nom de l'arxiu **sense extensió** (es pot incloure un camí)
`format`	`"zip"`, `"tar"`, `"gztar"`, `"bztar"` o `"xztar"`
`root_dir`	Directori a partir del qual es comprimeix (l'arrel de l'arxiu)
`base_dir`	Directori, relatiu a `root_dir`, des del qual comença el contingut

```
shutil.make_archive("backup", "gztar",
 ↪  root_dir="/home/ramon/projecte")   # backup.tar.gz
shutil.make_archive("backup", "xztar",
 ↪  root_dir="/home/ramon/projecte")   # backup.tar.xz
```

PRECAUCIÓ

Error habitual: l'estructura de directoris dins de l'arxiu. Si vols que l'arxiu contingui una carpeta `projecte/` al nivell superior (i no directament els seus fitxers), cal separar el pare i el nom:

```
shutil.make_archive(
    "backup", "zip",
    root_dir="/home/ramon",    # directori pare
    base_dir="projecte",       # carpeta que s'inclou
)
```

Amb `root_dir="/home/ramon/projecte"` sol, en descomprimir obtindràs els fitxers escampats directament al directori actual, sense carpeta contenidora.

Llistar formats disponibles

```
print(shutil.get_archive_formats())
# [('bztar', "bzip2'ed tar-file"), ('gztar', "gzip'ed tar-file"),
 ↪  ('tar', ...), ('xztar', ...), ('zip', 'ZIP file')]
```

Descomprimir

```
shutil.unpack_archive("backup.zip", "restauracio")          # extreu
 ↪  a la carpeta 'restauracio'
shutil.unpack_archive("backup.tar.gz", extract_dir="dades")
```

El format es dedueix de l'extensió. Es pot forçar amb `format="gztar"`.

Seguretat en descomprimir arxius de tercers

Un arxiu `tar` maliciós pot contenir noms com `../../etc/passwd` o enllaços que escapin del directori de destí (*path traversal*). Per a arxius `tar`, des de Python 3.12 (i amb *backports* a versions anteriors recents) existeix el paràmetre `filter`:

```
shutil.unpack_archive("descarregat.tar.gz", "sortida", filter="data")
```

El filtre `"data"` rebutja camins absoluts, sortides del directori de destí, fitxers especials i alguns permisos perillosos. **Si extreus arxius que no has generat tu, fes servir `filter="data"`.** Per a `zip`, el mòdul `zipfile` ja neteja els camins absoluts i els `..`, però igualment és recomanable no extreure mai arxius no fiables com a root.

6. Espai en disc: `disk_usage`

```
total, usat, lliure = shutil.disk_usage("/")

print(f"Total:  {total / 1024**3:.1f} GiB")
print(f"Usat:   {usat / 1024**3:.1f} GiB")
print(f"Lliure: {lliure / 1024**3:.1f} GiB")
print(f"Ocupació: {usat / total * 100:.1f} %")
```

Retorna una *named tuple* `(total, used, free)` en **bytes**. El camí pot ser qualsevol fitxer o directori de la partició que es vol consultar (a Windows, una unitat com `"C:\\"`).

Útil per comprovar l'espai abans d'una operació gran:

```
mida_necessaria = 5 * 1024**3   # 5 GiB
if shutil.disk_usage("/tmp").free < mida_necessaria:
    raise RuntimeError("No hi ha prou espai lliure")
```

7. Localitzar executables: `which`

```
print(shutil.which("python3"))   # '/usr/bin/python3' o None
print(shutil.which("git"))
print(shutil.which("noexisteix"))   # None
```

Cerca l'ordre als directoris de la variable `PATH`, igual que faria la shell, i retorna el camí complet o `None`. Té en compte les extensions executables a Windows (`PATHEXT`).

Per verificar dependències abans d'executar un programa:

```
requerits = ["git", "rsync", "ffmpeg"]
absents = [c for c in requerits if shutil.which(c) is None]
if absents:
    raise SystemExit(f"Falten aquestes ordres: {', '.join(absents)}")
```

És més fiable que executar l'ordre i capturar l'error, i evita dependre de `which` com a programa extern (que no existeix a Windows).

8. Altres funcions útils

Mida de la terminal

```
columnes, files = shutil.get_terminal_size()
print(f"{columnes}x{files}")
print("-" * columnes)
```

Si no es pot determinar (per exemple, quan la sortida va a un pipe), retorna una mida per defecte (`(80, 24)`) o la que indiquin les variables d'entorn `COLUMNS` i `LINES`.

Propietari i grup: chown

```
shutil.chown("fitxer.txt", user="www-data", group="www-data")   # cal
 ↪  tenir permisos (normalment root)
```

A diferència d'`os.chown`, accepta **noms** en lloc d'identificadors numèrics. Només disponible en sistemes Unix.

9. Exemples complets

9.1. Còpia de seguretat amb data al nom

```
import datetime as dt
import shutil
from pathlib import Path

origen = Path("/home/ramon/projecte")
destí_base = Path("/mnt/backups")

marca = dt.datetime.now().strftime("%Y%m%d_%H%M%S")
nom = destí_base / f"projecte_{marca}"

arxiu = shutil.make_archive(
    str(nom), "gztar",
    root_dir=origen.parent,
    base_dir=origen.name,
)
print("Backup creat:", arxiu)
```

9.2. Organitzar una carpeta de descàrregues per tipus

```
import shutil
from pathlib import Path

carpeta = Path.home() / "Downloads"
categories = {
    ".pdf": "Documents",
    ".jpg": "Imatges", ".png": "Imatges",
    ".zip": "Comprimits", ".gz": "Comprimits",
    ".mp3": "Musica",
}

for fitxer in carpeta.iterdir():
    if not fitxer.is_file():
        continue
    categoria = categories.get(fitxer.suffix.lower())
    if categoria:
        desti = carpeta / categoria
        desti.mkdir(exist_ok=True)
        shutil.move(str(fitxer), str(desti / fitxer.name))
```

9.3. Netejar fitxers temporals antics

```
import shutil, time
from pathlib import Path

temp = Path("/tmp/meva_app")
limit = time.time() - 7 * 24 * 3600     # 7 dies

for entrada in temp.iterdir():
    if entrada.stat().st_mtime < limit:
        if entrada.is_dir() and not entrada.is_symlink():
            shutil.rmtree(entrada)
        else:
            entrada.unlink()
```

9.4. Copiar un projecte sense fitxers de desenvolupament

```
import shutil

shutil.copytree(
    "web",
    "web_publicacio",
    ignore=shutil.ignore_patterns(".git", "node_modules", "*.log",
    ↪  "*.md", "__pycache__"),
    dirs_exist_ok=True,
```

```
)
```

9.5. Escriptura segura d'un fitxer amb còpia de seguretat prèvia

```
import os, shutil

def desa_amb_backup(cami, contingut):
    if os.path.exists(cami):
        shutil.copy2(cami, cami + ".bak")      # còpia del fitxer
        ↪  anterior
    tmp = cami + ".tmp"
    with open(tmp, "w", encoding="utf-8") as f:
        f.write(contingut)
    os.replace(tmp, cami)                      # substitució atòmica
```

10. `shutil` i `pathlib`

Les funcions de `shutil` accepten objectes `pathlib.Path` (protocol `os.PathLike`) a partir de Python 3.6/3.7 en la majoria de casos. `pathlib` cobreix part de les operacions senzilles (`Path.rename`, `Path.unlink`, `Path.mkdir`…) però **no té** equivalents per a copiar fitxers o arbres, ni per esborrar directoris no buits. Per això es combinen: `pathlib` per a camins i consultes, `shutil` per a còpies, moviments i arxius.

11. Errors habituals

1. **`copyfile` amb un directori com a destí** → `IsADirectoryError`. Usa `copy` o dona el nom complet.
2. **`copytree` a un destí que ja existeix** → `FileExistsError`. Usa `dirs_exist_ok=True` (Python 3.8+).
3. **`rmtree` sobre un camí equivocat**: no hi ha manera de desfer-ho. Imprimeix el camí abans, o prova primer amb una llista (`os.walk`).
4. **`shutil.move` entre particions** no és atòmic: si s'interromp, pot quedar a mitges.
5. **Arxius amb l'estructura de carpetes incorrecta** per confondre `root_dir` i `base_dir` a `make_archive`.
6. **Extreure arxius `tar` no fiables sense `filter="data"`**: risc de *path traversal*.
7. **Esperar que `copy` conservi les dates**: només `copy2` ho fa.
8. **Usar `ignore_errors=True` per defecte a `rmtree`**: amaga problemes de permisos que probablement vols conèixer.

12. Exercicis proposats

1. Escriu un script que copiï un directori a una destinació amb la data actual al nom, excloent `*.pyc` i `__pycache__`.
2. Fes una funció `mou_segur(origen, desti)` que comprovi l'espai lliure al destí amb `disk_usage` abans de moure.
3. Crea un script que, donada una carpeta, comprimeixi cada subcarpeta en un `.zip` independent dins d'un directori de sortida.
4. Comprova la diferència de comportament entre `copy` i `copy2` mirant `os.stat(...).st_mtime` en l'origen i la còpia.
5. Programa un netejador que esborri els fitxers de més de 30 dies d'un directori donat i imprimeixi quants MiB ha alliberat.
6. Escriu una funció que verifiqui que un conjunt d'eines (`git`, `python3`, `curl`) és al PATH i mostri on és cadascuna amb `shutil.which`.
7. Descomprimeix un `.tar.gz` fent servir `shutil.unpack_archive` amb `filter="data"` i captura els errors de seguretat que es puguin produir.

13. Resum

Vull…	Eina
Copiar un fitxer (només contingut)	`shutil.copyfile(src, dst)`
Copiar un fitxer (+ permisos)	`shutil.copy(src, dst)`
Copiar un fitxer (+ dates i metadades)	`shutil.copy2(src, dst)`
Copiar entre objectes fitxer	`shutil.copyfileobj(fsrc, fdst)`
Copiar un directori sencer	`shutil.copytree(src, dst, dirs_exist_ok=True)`
Excloure fitxers en copiar	`ignore=shutil.ignore_patterns("*.tmp")`
Moure o reanomenar (també entre particions)	`shutil.move(src, dst)`
Esborrar un directori amb contingut	`shutil.rmtree(cami)`
Crear un `.zip` / `.tar.gz`	`shutil.make_archive(nom, "zip", root_dir)`
Descomprimir	`shutil.unpack_archive(arxiu, destí)`
Espai total / usat / lliure	`shutil.disk_usage(cami)`
Localitzar un executable	`shutil.which("nom")`
Mida de la terminal	`shutil.get_terminal_size()`

14. Referències

- Documentació oficial de `shutil`: https://docs.python.org/es/3.15/library/shutil.html
- `pathlib`: https://docs.python.org/es/3.15/library/pathlib.html
- `zipfile`: https://docs.python.org/es/3.15/library/zipfile.html
- `tarfile` i filtres d'extracció: https://docs.python.org/es/3.15/library/tarfile.html#extraction-filters

Versions d'aquest document

- HTML
- PDF
- ODT
- MD

Domini Públic (CC0)