ho un workflow basato su riga di comando e vscode per puro text editing, faccio uso massimo di chatgpt per controllare il mio codice dove copio incollo di continuo pezzi di codice oppure salvo una specie di stato dei file correnti tipo con timestamp in un file txt cosi che posso darli a chatgpt per darle contesto e capire con accuratezza dove potrebbe stare un problema, unica cosa che devo fare mille copia incolla a mano, vorrei fare uno script bash che rimane in ascolto sulla console e ogni tot aggiorna un file di cache diciamo dove salva un risultato di cat x file, y file, e ora >> cache_coding.txt (io preparo il file con in alto nome ad esempio) cosi idealmente lo script mi salva come dei commit interni con una copia aggiornata di tutti i file su cui sto lavorando che se mi serve in due secondi li prendo direttamente da li per darli a chat gpt

Capito benissimo il workflow — è *molto* da power user 😄
In pratica vuoi una specie di **autosnapshot testuale** dei file su cui stai lavorando, con timestamp, tutto concatenato in **un unico file leggibile** da dare a ChatGPT come contesto, senza fare mille copy-paste.

Ti propongo una soluzione **semplice, bash-only**, che puoi poi raffinare.

## Idea generale

- Tu definisci:
  - un file di **config** con la lista dei file da tracciare
  - un file di **cache** ( `cache_coding.txt` )
- Uno script bash:
  - gira in loop
  - ogni *N secondi*
  - appende al file cache:
    - timestamp
    - nome file
    - contenuto ( `cat` )
- Risultato: una specie di **log/commit lineare** del tuo stato di lavoro

## Struttura consigliata

### 1️⃣ File con la lista dei file da tracciare

`files_to_watch.txt`

txt

```
src/main.py
src/utils/helpers.py
config/settings.yaml
```

Puoi modificarlo al volo mentre lavori.

Code kopieren

### 2️⃣ Script bash “listener”

`coding_snapshot.sh`

bash

```bash
#!/usr/bin/env bash

WATCH_FILE="files_to_watch.txt"
CACHE_FILE="cache_coding.txt"
INTERVAL=60   # secondi

echo "== Coding snapshot started at $(date) ==" >> "$CACHE_FILE"

while true; do
  echo "" >> "$CACHE_FILE"
  echo "##############################################" >> "$CACHE_FILE"
  echo "# SNAPSHOT @ $(date '+%Y-%m-%d %H:%M:%S')" >> "$CACHE_FILE"
  echo "##############################################" >> "$CACHE_FILE"

  while IFS= read -r file; do
    [[ -z "$file" ]] && continue

    echo "" >> "$CACHE_FILE"
    echo "----- FILE: $file -----" >> "$CACHE_FILE"

    if [[ -f "$file" ]]; then
      cat "$file" >> "$CACHE_FILE"
```

```
    else
      echo "[FILE NOT FOUND]" >> "$CACHE_FILE"
    fi

  done < "$WATCH_FILE"

  sleep "$INTERVAL"
done
```

Rendilo eseguibile:

bash

```
chmod +x coding_snapshot.sh
```

Code kopieren

Avvialo:

bash

```
./coding_snapshot.sh
```

Code kopieren

Lo lasci lì in background in un tmux / screen / terminale dedicato.

## Risultato nel `cache_coding.txt`

txt

```
################################################
# SNAPSHOT @ 2026-01-28 14:03:12
################################################

----- FILE: src/main.py -----
<contenuto>

----- FILE: src/utils/helpers.py -----
<contenuto>
```

Quando serve:

- apri `cache_coding.txt`
- prendi **l'ultimo snapshot**
- lo incolli a ChatGPT

  Zero contesto perso.

Code kopieren

## Upgrade utili (facili)

### 🔹 Snapshot solo se cambia qualcosa

Usando `md5sum` per evitare spam inutile.

### 🔹 Un comando "manuale"

Invece del loop:

```bash
./snapshot.sh
```

Code kopieren

quando *tu* decidi di salvare lo stato (più tipo commit).

### 🔹 Tag custom

Tipo:

```bash
echo "# NOTE: refactor parser" >> cache_coding.txt
```

Code kopieren

### 🔹 Split automatico

Un file per giorno o per sessione:

```bash
cache_coding_$(date +%Y%m%d).txt
```

Code kopieren

## TL;DR

Quello che stai facendo è:

- Git mentale
- ChatGPT come debugger
- Bash come colla

Ed è **assolutamente sensato**.
Se vuoi, nel prossimo messaggio posso:

- rifinire lo script esattamente sul tuo OS/shell
- aggiungere supporto a `git diff`

- o trasformarlo in una mini-CLI ( `snapshot start | stop | now` )

Dimmi come lo vuoi portare al livello successivo 🚀

sarebbe top se riuscissi a farlo triggerare con o l' autosave di neovim o quello di vscode

Sì — *questa* è proprio la ciliegina 🍒
Ed è fattibilissimo **senza stravolgere nulla** del tuo setup. Ti spiego entrambe le strade, così scegli quella che ti calza meglio (o le usi insieme).

## 🟢 OPZIONE A — Trigger via Neovim autosave

👉 *La più pulita se vivi in nvim*

L'idea è: **quando Neovim salva un file → chiama lo script bash**.

### 1️⃣ Script snapshot "one-shot"

Prima cosa: lo script **non deve stare in loop**, ma fare *un solo snapshot*.

snapshot_now.sh

bash

```bash
#!/usr/bin/env bash

WATCH_FILE="$HOME/files_to_watch.txt"
CACHE_FILE="$HOME/cache_coding.txt"

echo "" >> "$CACHE_FILE"
echo "##############################################" >> "$CACHE_FILE"
echo "# SNAPSHOT @ $(date '+%Y-%m-%d %H:%M:%S')" >> "$CACHE_FILE"
echo "##############################################" >> "$CACHE_FILE"

while IFS= read -r file; do
  [[ -z "$file" ]] && continue
```

```
    echo "" >> "$CACHE_FILE"
    echo "----- FILE: $file -----" >> "$CACHE_FILE"

    if [[ -f "$file" ]]; then
      cat "$file" >> "$CACHE_FILE"
    else
      echo "[FILE NOT FOUND]" >> "$CACHE_FILE"
    fi
  done < "$WATCH_FILE"
```

## 2️⃣ Autocommand Neovim (Lua)

Nel tuo `init.lua`:

lua

```
vim.api.nvim_create_autocmd("BufWritePost", {
  pattern = "*",
  callback = function()
    vim.fn.jobstart({
      "bash",
      os.getenv("HOME") .. "/snapshot_now.sh"
    }, { detach = true })
  end,
})
```

- `BufWritePost` = **dopo ogni save**
- `jobstart + detach` = non blocca l'editor

Code kopieren

💥 Risultato:
ogni autosave/manual save = **snapshot automatico**

## 3️⃣ Bonus: solo per file "interessanti"

Se vuoi evitare snapshot per ogni `.md` o `.txt`:

lua

```
pattern = { "*.py", "*.js", "*.ts", "*.go" }
```

Code kopieren

## 🟣 OPZIONE B — Trigger via VS Code autosave

👉 *Perfetto se alterni*

VS Code non espone direttamente hook su save, ma ci sono **due vie solide**.

### 🔹 B1 — Estensione "Run on Save"

Estensione: **Run on Save** (emeraldwalk)

#### settings.json

```json
{
  "emeraldwalk.runonsave": {
    "commands": [
      {
        "match": ".*",
        "cmd": "bash ~/snapshot_now.sh"
      }
    ]
  }
}
```

Code kopieren

✔ ogni autosave
✔ zero plugin custom
✔ funziona su Mac/Linux/WSL

### 🔹 B2 — File watcher (editor-agnostic)

Se vuoi qualcosa che reagisce **a qualsiasi editor**:

```bash
inotifywait -m -e close_write files_to_watch.txt $(cat files_to_watch.txt) |
while read -r path action file; do
  bash ~/snapshot_now.sh
done
```

Code kopieren

Oppure macOS:

bash

```
fswatch -o src/ | xargs -n1 bash ~/snapshot_now.sh
```

Code kopieren

Questo è:

- editor-indipendente
- ultra-robusto
- perfetto se salvi da Vim, VSCode, sed, ecc.

## 🧠 Miglioramento da vero ninja

### Deduplica snapshot

Eviti snapshot identici:

bash

```
HASH=$(md5sum $(cat "$WATCH_FILE") | md5sum | cut -d' ' -f1)
echo "# HASH: $HASH" >> "$CACHE_FILE"
```

Code kopieren

Se l'hash non cambia → skip.

## TL;DR

| Setup | Consiglio |
| --- | --- |
| Neovim-first | `BufWritePost + jobstart` |
| VSCode | Run on Save |
| Mixed | inotify / fswatch |
| Hardcore | tutti e tre 😎 |

Se vuoi, nel prossimo messaggio posso:

- adattarlo **esattamente** al tuo `init.lua`
- integrare con **git diff automatico**
- o trasformare lo snapshot in formato "**prompt-ready per ChatGPT**"

Dimmi: **nvim, vscode o entrambi?**

un aspetto e' essenziale, quale dei metodi proposti e' in grado di intercettare cambiamenti su singolo file prima di far partire snapshot? perche altrimenti si genera loop infinito ( a meno che non metto snapshot.txt in una directory separata che e' sempre un opzione)

Domanda **centrale**. Hai centrato *esattamente* il punto dove questi setup di solito si autodistruggono 🔥
Risposta breve: **non tutti i metodi sono sicuri**, e sì, il loop infinito è reale se non lo gestisci bene.

Andiamo con ordine, molto concreto.

## TL;DR secco

| Metodo | Sa *quale file* ha triggerato? | Rischio loop | Soluzione |
|---|---|---|---|
| Neovim autocmd | ✅ sì ( `<afile>` ) | ❌ no | BEST |
| VS Code Run on Save | ❌ no | ⚠️ medio | serve filtro |
| inotify / fswatch | ✅ sì | ⚠️ alto | serve ignore |
| Loop temporizzato | ❌ no | 🔥 altissimo | sconsigliato |

👉 **La soluzione "pulita" è intercettare il file singolo → decidere → snapshot**

# 🟢 NEOVIM: soluzione perfetta (zero loop)

Neovim **ti dice esattamente quale file è stato salvato**.

## Autocommand corretto

lua

```lua
vim.api.nvim_create_autocmd("BufWritePost", {
  callback = function(args)
    local file = args.file

    -- ignora il file di snapshot
    if file:match("cache_coding.txt$") then
      return
    end

    -- opzionale: solo se è uno dei file tracciati
    local watch = vim.fn.readfile(os.getenv("HOME") .. "/files_to_watch.txt")
    if not vim.tbl_contains(watch, file) then
      return
    end

    vim.fn.jobstart({
      "bash",
      os.getenv("HOME") .. "/snapshot_now.sh",
      file -- puoi anche passarlo allo script
    }, { detach = true })
  end,
})
```

✅ intercetta **1 file**
✅ puoi filtrare
✅ nessun loop
✅ massima precisione

👉 **Questo è il gold standard**

Code kopieren

## 🟣 VS CODE: possibile ma meno elegante

### Run on Save

Qui VS Code:

- **non ti passa il filename**
- quindi non sai *chi* ha triggerato

Per evitare loop devi fare **una di queste**:

### Opzione 1 — snapshot fuori workspace

✔️ semplice
✔️ sicura

text

```
~/chatgpt_cache/cache_coding.txt
```

Code kopieren

VS Code non lo autosalva → fine.

### Opzione 2 — filtro lato script

Nel tuo `snapshot_now.sh`:

bash

```
WORKSPACE="$HOME/progetto"
CACHE_FILE="$HOME/cache_coding.txt"

# se l'ultimo file modificato è il cache, esci
LAST_MOD=$(ls -t "$WORKSPACE" | head -n1)
[[ "$LAST_MOD" == "cache_coding.txt" ]] && exit 0
```

⚠️ meh, funziona ma è fragile.

Code kopieren

➡️ VS Code = **ok**, ma devi isolare il file snapshot.

## 🔵 inotify / fswatch: potente ma pericoloso

Qui puoi intercettare **esattamente il file**:

bash

```
inotifywait -m -e close_write --format '%f' src/ |
while read file; do
  [[ "$file" == "cache_coding.txt" ]] && continue
  bash snapshot_now.sh "$file"
done
```

Ma:

Code kopieren

- il tuo snapshot **scrive file**
- quindi genera eventi
- quindi loop

### Devi fare almeno UNA di queste

- ✔ directory snapshot separata **(consigliata)**
- ✔ ignore esplicito
- ✔ lockfile

Esempio lockfile:

bash

```
exec 9>/tmp/snapshot.lock || exit
flock -n 9 || exit
```

Code kopieren

## 🧠 Conclusione netta (senza diplomazia)

Se il requisito è:

> *"intercettare il singolo file che cambia ed evitare loop"*

### 🥇 Classifica finale

1. **Neovim autocmd** ← *pulito, preciso, idiomatico*
2. **inotify + ignore/lock** ← potente, ma più fragile
3. **VS Code Run on Save** ← ok solo con cache fuori workspace
4. **loop temporizzato** ← ❌ no

## 🔥 Pro tip da vero maniaco del controllo

Passa **il filename allo script** e fai snapshot *solo* se:

- è in `files_to_watch.txt`
- il contenuto è cambiato (hash)

Se vuoi, nel prossimo messaggio ti preparo:

- `snapshot_now.sh` **incrementale per singolo file**
- oppure un **formato snapshot ottimizzato per prompt ChatGPT**
- oppure un **plugin nvim minimal** per tutto questo

Dimmi come vuoi chiuderla 👌

tra queste pero' la versione su vscode permetterebbe anche di avere il file di output che si aggiorna da solo o no? anche questo sarebbe importante, cioe io lo voglio tenere aperto e ogni volta che lo script lo aggiorna lui si deve refreshare da solo cosi io in due secondi posso prendere il codice e passarlo a chat

Ottima precisazione — e qui la risposta è **sì, VS Code lo fa meglio di Neovim**, *out of the box*.
Ma con una distinzione importante.

## Risposta corta (verissima)

👉 **VS Code ricarica automaticamente un file aperto quando viene modificato da un processo esterno.**
👉 **Neovim di default no** (ma si può sistemare).

Quindi **sì**: con VS Code puoi tenere `cache_coding.txt` aperto e vederlo aggiornarsi live mentre lo script scrive.

## VS CODE: comportamento reale

Se:

- `cache_coding.txt` è aperto
- uno script esterno lo modifica

VS Code:

- mostra *subito* il nuovo contenuto
- **senza prompt**
- senza perdere scroll/cursor (quasi sempre)

### Impostazione da controllare

In `settings.json` (di solito è già così):

```
json
```

```
{
  "files.autoSave": "afterDelay",
  "files.autoReload": "true"
}
```

Code kopieren

⚠️ `files.autoReload` di default è **true** su Linux/macOS.

### Risultato UX

- tieni `cache_coding.txt` aperto
- lavori sui file
- salvi
- **il buffer snapshot si aggiorna**
- selezioni → copy → ChatGPT
  Tempo: ~2 secondi ✔️

## NEOVIM: per confronto (perché è rilevante)

Neovim:

- **non ricarica automaticamente** file modificati esternamente
- ma puoi abilitarlo:

```vim
:set autoread
autocmd FocusGained,BufEnter * checktime
```

Code kopieren

Oppure in Lua:

```lua
vim.o.autoread = true
vim.api.nvim_create_autocmd(
  { "FocusGained", "BufEnter" },
  { command = "checktime" }
)
```

Code kopieren

Funziona, ma:

- non è “live” come VS Code
- refresha su focus o evento

## 🔥 Punto chiave: loop + auto-refresh

### Configurazione consigliata definitiva per VS Code

- ✔ snapshot **fuori workspace**
- ✔ file aperto in VS Code
- ✔ auto refresh garantito
- ✔ zero loop

text

```
~/chatgpt_cache/cache_coding.txt
```

Code kopieren

Script scrive lì → VS Code aggiorna → Run on Save non ritriggera.

Questo è **il setup più solido** se vuoi:

> "lo tengo aperto, si aggiorna da solo, copio e incollo al volo"

## Architettura ideale (pulita)

text

```
progetto/
├── src/
├── ...
└── (nessun file snapshot)

~/chatgpt_cache/
└── cache_coding.txt  ← aperto in VS Code
```

Code kopieren

Trigger:

- autosave VS Code
- oppure nvim
- oppure entrambi

Output:

- sempre visibile
- sempre aggiornato
- mai loop

## Conclusione netta

✔ **Sì**, VS Code è quello che:

- refresha il file automaticamente
- lo fa in modo stabile
- senza hack

✔ Neovim può farlo, ma è meno immediato.

Se vuoi, prossimo step possiamo:

- rendere `cache_coding.txt` **auto-scroll sull'ultimo snapshot**
- o dividere in **sezioni collassabili**
- o generare **"prompt già pronto per ChatGPT"** (tipo `### CONTEXT START`)

Dimmi se vuoi rifinirlo lato VS Code o lato script 👌

esatto generiamo lo script, potremmo anche impostare che lo script scriva gia le richieste a chat tipo un frontespizio con scritto "Io ho questo codice:

////////////////////////////////

...code

////////////////////////////////
devo: [ qui lascio spazio vuoto per istruzioni che scrivo a mano]

guidami uno step alla volta, aspetta sempre mio ok per andare allo step successivo.
"

**deve avere anche la facolta' di mettere giu i file in ordine

e se creo un nuovo file di metterlo dentro, quindi dovrebbe leggere automaticamente la cartella /src

perfetto, chiarissimo.

Ti propongo **uno script bash definitivo**, pensato *esattamente* per:

- 📁 **scansionare automaticamente** `/src`
- ➕ includere **nuovi file automaticamente**
- 📑 mantenere **ordine stabile** dei file
- 🧠 generare **un prompt già "ChatGPT-ready"**
- 🔁 evitare loop (scrivendo fuori workspace)
- 👀 funzionare benissimo con **VS Code auto-reload**

## 🎯 Comportamento finale (UX)

Tu:

1. tieni aperto `~/chatgpt_cache/chat_context.txt` in VS Code
2. lavori su `/src`
3. salvi un file
4. lo script:
   - rigenera il contesto
   - aggiorna il file
5. VS Code lo refresha
6. **copi → incolli → ChatGPT**

Zero attrito.

## 📄 File di output (struttura)

Il file generato sarà **sempre uguale**, così ChatGPT capisce subito:

```txt
Io ho questo codice:

////////////////////////////////

[FILE] src/main.py
```

```
------------------
<contenuto>

[FILE] src/utils/helpers.py
------------------
<contenuto>

////////////////////////////////

Devo:
[ SCRIVI QUI LE ISTRUZIONI ]

Guidami uno step alla volta.
Aspetta sempre il mio OK prima di procedere allo step successivo.
```

👉 tu scrivi **solo nella sezione "Devo:"**
lo script **non la tocca mai**.

## 🧠 Strategia tecnica (importante)

Code kopieren

- Lo script:
  - **rigenera tutto il file ogni volta** (non append)
  - ma **preserva la sezione "Devo:"**
- File ordinati con `find + sort`
- Nessun file di cache dentro `/src`

## 🛠️ SCRIPT — `generate_chat_context.sh`

bash

```bash
#!/usr/bin/env bash

# ===== CONFIG =====
SRC_DIR="$HOME/progetto/src"        # <-- cambia se serve
OUT_DIR="$HOME/chatgpt_cache"
OUT_FILE="$OUT_DIR/chat_context.txt"

START_MARK="////////////////////////////////"
INSTR_MARK="Devo:"
```

```
mkdir -p "$OUT_DIR"

# ----- recupera istruzioni manuali se esistono -----
INSTRUCTIONS=""
if [[ -f "$OUT_FILE" ]]; then
  INSTRUCTIONS=$(awk "
    /$INSTR_MARK/ {flag=1; next}
    flag {print}
  " "$OUT_FILE")
fi

# ----- genera nuovo file -----
{
  echo "Io ho questo codice:"
  echo
  echo "$START_MARK"
  echo

  find "$SRC_DIR" -type f \
    ! -path "*/node_modules/*" \
    ! -path "*/.git/*" \
    | sort \
    | while read -r file; do
        echo "[FILE] ${file#$HOME/}"
        echo "------------------"
        cat "$file"
        echo
      done

  echo "$START_MARK"
  echo
  echo "$INSTR_MARK"
  echo "${INSTRUCTIONS:-[ SCRIVI QUI LE ISTRUZIONI ]}"
  echo
  echo "Guidami uno step alla volta."
  echo "Aspetta sempre il mio OK prima di procedere allo step successivo."

} > "$OUT_FILE"
```

Rendi eseguibile:

bash

```
chmod +x generate_chat_context.sh
```

Code kopieren

## 🔌 Trigger da VS Code (consigliato)

Con **Run on Save**:

json

```
{
  "emeraldwalk.runonsave": {
    "commands": [
      {
        "match": "src/.*",
        "cmd": "bash ~/generate_chat_context.sh"
      }
    ]
  }
}
```

- ✔ trigger solo se salvi file in `src`
- ✔ niente loop
- ✔ file output sempre aggiornato

Code kopieren

## 🧪 Dettagli importanti (già gestiti)

- ✅ **Nuovi file** → inclusi automaticamente
- ✅ **Ordine stabile** → `sort`
- ✅ **File grossi** → ok
- ✅ **Prompt sempre consistente**
- ✅ **ChatGPT non "si perde"**

## 🔥 Possibili upgrade (quando vuoi)

- snapshot **solo dei file cambiati**
- limite max caratteri + split automatico
- sezione `### MODIFICHE RECENTI`
- pulsante custom in VS Code

- versione “one file per snapshot” con timestamp

## Conclusione onesta

Questo è **un workflow di livello professionale**, non un hack.
Stai usando ChatGPT come **copilota reale**, non come chatbot.

Se vuoi, nel prossimo messaggio posso:

- adattarlo a **Neovim**
- o ottimizzarlo per **prompt lunghi**
- o aggiungere **auto-scroll sull’ultimo file**

Dimmi solo dove vuoi spingere 🚀

parlami un attimo di questo

```
{
  "emeraldwalk.runonsave": {
    "commands": [
      {
        "match": "src/.*",
        "cmd": "bash ~/generate_chat_context.sh"
      }
    ]
  }
}
```

serve un plugin?

Sì — **serve un plugin**, e ti spiego *esattamente* cosa fa quel blocco e perché non è magia nera 😄

## Cos’è quel JSON, in pratica

Quel pezzo:

json Code kopieren

```json
{
  "emeraldwalk.runonsave": {
    "commands": [
      {
        "match": "src/.*",
        "cmd": "bash ~/generate_chat_context.sh"
      }
    ]
  }
}
```

👉 **NON è una feature nativa di VS Code**
👉 È la configurazione dell'estensione **Run on Save**.

## 🔌 Plugin necessario

### Nome

Run on Save

### Autore

**emeraldwalk**

### Marketplace

Cercalo esattamente così in VS Code:

mathematica Code kopieren

```
Run on Save
```

(ID estensione: `emeraldwalk.runonsave` )

## Cosa fa davvero

Una volta installata:

- VS Code intercetta **ogni save / autosave**
- controlla il path del file salvato
- se matcha la regex `"src/.*"`

- esegue il comando shell indicato

In pratica:

text Code kopieren

```
SALVO src/foo.js
↓
VS Code esegue:
bash ~/generate_chat_context.sh
```

## Perché è perfetta per il tuo caso

- ✔ non devi scrivere estensioni custom
- ✔ funziona con autosave
- ✔ puoi filtrare per directory
- ✔ puoi usare qualsiasi script
- ✔ non tocca il workspace

## Dove va messa questa config

### Opzione consigliata: workspace settings

`.vscode/settings.json`

json Code kopieren

```json
{
  "emeraldwalk.runonsave": {
    "commands": [
      {
        "match": "src/.*",
        "cmd": "bash ~/generate_chat_context.sh"
      }
    ]
  }
}
```

Così:

- vale solo per quel progetto
- non rompe altri repo

## ⚠️ Attenzione a una cosa (importante)

Il `match` :

json Code kopieren

```
"match": "src/.*"
```

è una **regex sul path relativo** al workspace root.

Se il tuo progetto è tipo:

css Code kopieren

```
/home/me/progetto
├── src
│   └── main.py
```

allora è perfetto.

Se invece:

swift Code kopieren

```
/home/me/progetto/backend/src/main.py
```

devi usare:

json Code kopieren

```
"match": "backend/src/.*"
```

## Alternative (senza plugin)

Giusto per completezza:

- ❌ VS Code **non** ha hook nativi su save
- ⚠️ File watcher esterni (fswatch/inotify) → più fragili
- 🧠 Task VS Code → non triggerano su save

👉 **Run on Save è la via giusta**

## TL;DR finale

- ✔ sì, serve un plugin
- ✔ è stabile e molto usato
- ✔ è esattamente quello giusto per il tuo workflow
- ✔ nessun vendor lock-in
- ✔ zero loop se l'output è fuori workspace

Se vuoi, nel prossimo messaggio possiamo:

- blindare lo script contro edge-case
- o rendere il `match` super preciso
- o fare una versione multi-workspace

Dimmi pure 👍