Działanie na wynikach: -exec, -delete, xargs

Poziom: średni | Dotyczy: wszystkie dystrybucje | Czas: ~14 min

`find` potrafi znaleźć pliki, ale sam ich nie skasuje ani nie zmieni. Do działania na wynikach służą `-exec`, `-delete` i `xargs` — i każde z nich ma inną pułapkę na nazwy plików ze spacjami.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor

Fiszka

OSTROŻNIE · NIEODWRACALNE

Uruchomienie osobnej komendy na każdy plik: -exec ... {} \;

```
find . -name "*.tmp" -exec rm {} \;
```

Uruchamiamy nowy proces `rm` osobno dla każdego dopasowanego pliku. `{}` to placeholder podstawiany nazwą bieżącego pliku, a średnik musi być zescapowany (`\;`), bo dla powłoki nieuciszony `;` kończyłby całą komendę `find` w tym miejscu. Wolniejsze przy tysiącach plików — każdy to osobny `fork()`/`exec()`.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

OSTROŻNIE · NIEODWRACALNE

Uruchomienie jednej komendy na wszystkie pliki naraz: -exec ... {} +

```
find . -name "*.tmp" -exec rm {} +
```

`find` zbiera dopasowane ścieżki w grupy i wywołuje komendę **rzadko**, z wieloma plikami naraz — podobnie do tego, co robi `xargs`. Warunek: `{}` musi wystąpić dokładnie raz, na końcu, jako ostatni argument.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

OSTROŻNIE · NIEODWRACALNE

Potwierdzenie przed każdym uruchomieniem: -ok

```
find . -name "*.bak" -ok rm {} \;
```

Przed każdym uruchomieniem pytamy `y/n`. Nie istnieje wariant `-ok ... +` — potwierdzenie ma sens tylko dla pojedynczych wywołań, nie dla partii plików naraz.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Wbudowane kasowanie: -delete

```
find . -type f -name "*.tmp" -delete
```

Kasujemy dopasowany wpis bez uruchamiania zewnętrznego `rm` — szybciej, bez problemu z cudzysłowami czy spacjami w nazwach. Musi stać **na końcu** wyrażenia, po wszystkich testach filtrujących — inaczej skasuje wszystko, zanim testy zdążą cokolwiek odfiltrować (patrz *Częste błędy*).

```
# NIGDY tak — -delete przed testami skasuje WSZYSTKO
# find . -delete -name "*.tmp"      # <- BŁĄD, nie uruchamiaj
```

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Budowanie wywołań komendy z linii na wejściu: xargs

```
find . -name "*.log" | xargs rm
```

Podajemy wyniki `find` jako argumenty jednej (albo kilku) komendy `rm`. Rozwiązuje to inny problem niż `-exec`: przydaje się, gdy komenda w ogóle nie jest wywoływana przez `find`, albo

gdy potrzebujemy kontroli, jakiej `-exec` nie daje.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Bezpieczne rozdzielenie nazw ze spacjami: -print0 / xargs -0

```
find . -name "*.log" -print0 | xargs -0 rm
```

`find ... -print0` rozdziela wyniki bajtem NUL zamiast znakiem nowej linii, a `xargs -0` dzieli wejście po tym samym bajcie zamiast po białych znakach. Jedyna w pełni bezpieczna para dla nazw plików ze spacjami, cudzysłowami czy znakami nowej linii.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

ZMIENIA SYSTEM

Ograniczenie liczby argumentów na wywołanie: xargs -n

```
find . -name "*.jpg" -print0 | xargs -0 -n 20 convert -resize 50%
```

Wołamy komendę wielokrotnie, po maksymalnie N argumentów za każdym razem — przydatne, gdy program nie radzi sobie z tysiącami argumentów naraz.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Uruchomienie wywołań równoległych: xargs -P

```
find . -name "*.gz" -print0 | xargs -0 -P 4 -n 1 gunzip
```

Uruchamiamy do N procesów jednocześnie zamiast jednego po drugim — zauważalne przyspieszenie przy operacjach ograniczonych wydajnością CPU albo dysku, jak kompresja czy konwersja obrazów.

CPU = Central Processing Unit: procesor · **find** = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

ZMIENIA SYSTEM

Wstawienie argumentu w dowolnym miejscu komendy: xargs -I{}

```
find . -name "*.conf" -print0 | xargs -0 -I{} cp {} /backup/
```

Domyślnie `xargs` dokleja argumenty na **końcu** komendy. `-I{}` pozwala wstawić placeholder w dowolnym miejscu — tu jako pierwszy argument `cp`, przed stałym katalogiem docelowym.

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Pominięcie uruchomienia przy pustym wejściu: xargs -r

```
find . -name "*.orphan" -print0 | xargs -r -0 rm
```

Bez `-r` `xargs` i tak wywoła komendę raz, bez żadnych argumentów, nawet gdy `find` nic nie znalazł. `-r` (GNU xargs) pomija to wywołanie „na sucho".

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Przetwarzanie wyników bez xargs: pętla while read -d ''

```
find . -name "*.csv" -print0 | while IFS= read -r -d '' plik; do
    echo "przetwarzam: $plik"
done
```

`-d ''` mówi `read`, żeby separatorem rekordu był bajt NUL, `-r` wyłącza interpretację `\`, `IFS=` zapobiega ucinaniu białych znaków na brzegach nazwy. Wolniejsze niż `xargs` przy dużej liczbie plików, ale najbezpieczniejsze i najczytelniejsze, gdy logika przetwarzania jest złożona (warunki, zmienne pomocnicze).

IFS = Internal Field Separator: zmienna powłoki mówiąca, po jakim znaku dzielimy tekst na słowa, domyślnie spacja, tabulator i nowa linia · **find** = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Wszystko razem

Narzędzie / flaga	Co robi	Kiedy używasz
`-exec cmd {} \;`	uruchamia `cmd` osobno dla każdego wyniku	mało plików, komenda musi widzieć jeden plik naraz
`-exec cmd {} +`	zbiera wyniki w grupy i woła `cmd` rzadziej	dużo plików, komenda przyjmuje wiele argumentów
`-ok cmd {} \;`	jak `-exec`, z pytaniem o potwierdzenie	operacja nieodwracalna, chcesz kontroli ręcznej
`-delete`	kasuje dopasowane wpisy bezpośrednio z `find`	szybsze niż `-exec rm`, mniej procesów
`xargs`	buduje wywołania komendy z linii na wejściu	komenda nie ma trybu `-exec`, albo chcesz `-P` / `-n`
`-print0` / `xargs -0`	rozdziela wpisy bajtem NUL zamiast nową linią	nazwy plików ze spacjami, cudzysłowami, nowymi liniami
`xargs -n N`	maks. N argumentów na jedno wywołanie komendy	komenda nie przyjmuje tysięcy argumentów naraz
`xargs -P N`	N procesów równolegle	dużo niezależnych, powolnych operacji (kompresja, konwersja)
`xargs -I{}`	wstawia argument w dowolne miejsce komendy	argument nie jest ostatnim parametrem
`xargs -r`	nie uruchamia komendy, gdy wejście jest puste	pusty wynik `find` nie ma odpalać komendy „na sucho"

Uważaj

- `find . -delete -name "*.tmp"` skasuje **wszystko**, bo `-delete` samo w sobie jest testem, który zawsze zwraca prawdę — kolejność w wyrażeniu ma znaczenie.
- `find . -name "*.log" | xargs rm` rozjedzie się na plikach z nazwami zawierającymi spacje — `xargs` domyślnie dzieli wejście po białych znakach, nie po liniach z `find`.
- Zawsze uruchom komendę z `-print` zamiast `-delete` / `-exec rm`, zobacz listę, dopiero potem podmień na wersję niszczącą.

Z życia wzięte

Z ŻYCIA WZIĘTE

> Środa, 15:15. Tomek na `k3s01` czyści katalog z dziesiątkami tysięcy plików tymczasowych z laboratorium Kubernetesa przez `find . -name "*.tmp" -exec rm {} \;` i widzi, że komenda mieli minutami zamiast sekund. Ola pokazuje mu `-exec rm {} +`, które zbiera nazwy w duże partie zamiast uruchamiać `rm` osobno na każdym pliku. Sprzątanie kończy się w kilka sekund, a Tomek uczy się różnicy między `\;` a `+`.

Po co to istnieje

`find` samo w sobie tylko znajduje i wypisuje. Żeby coś z wynikami zrobić — skasować, spakować, zmienić uprawnienia, przekonwertować — potrzebujesz sposobu na przekazanie tej listy do innej komendy. Są na to trzy drogi: wbudowana akcja `-exec` (albo ostrożniejszy wariant `-ok`), wbudowana akcja `-delete` dla najczęstszego przypadku, oraz zewnętrzne narzędzie `xargs`, które buduje wywołania dowolnej komendy z tego, co dostanie na wejściu.

To rozdzielenie nie jest przypadkowe. `find` przechodzi drzewo i filtruje — jedno zadanie, robione dobrze. Co zrobić z wynikiem, to inne zadanie, a Unix od dekad trzyma się zasady jednego narzędzia do jednej rzeczy. `xargs` istnieje dlatego, że mnóstwo komend — `rm`, `chmod`, `grep`, `cp` — nie wie nic o `find` i nie musi: wystarczy, że przyjmują listę nazw plików jako argumenty.

Analogia

Wyobraź sobie **brygadę sprzątającą z listą adresów** w ręku — listę, którą przed chwilą sporządził ktoś inny (to `find` z poprzedniej strony).

Masz dwa sposoby na zorganizowanie pracy. Pierwszy: dla każdego adresu z osobna wysyłasz ekipę, ona jedzie, sprząta, wraca, bierzesz następny adres. To jest `-exec cmd {} \;` — jedno wywołanie komendy na każdy wynik. Wolne, ale bezpieczne: jeśli pod jednym adresem coś pójdzie nie tak, reszta listy jedzie dalej.

Drugi sposób: dajesz jednej ekipie całą listę adresów naraz, w ramach jednego wyjazdu. To `-exec cmd {} +` albo `xargs` bez limitu — mniej wyjazdów, szybciej, ale lista musi zmieścić się w jednej furgonetce. Przy dziesiątkach tysięcy adresów trzeba ją podzielić na mniejsze kursy — to robi `xargs -n`.

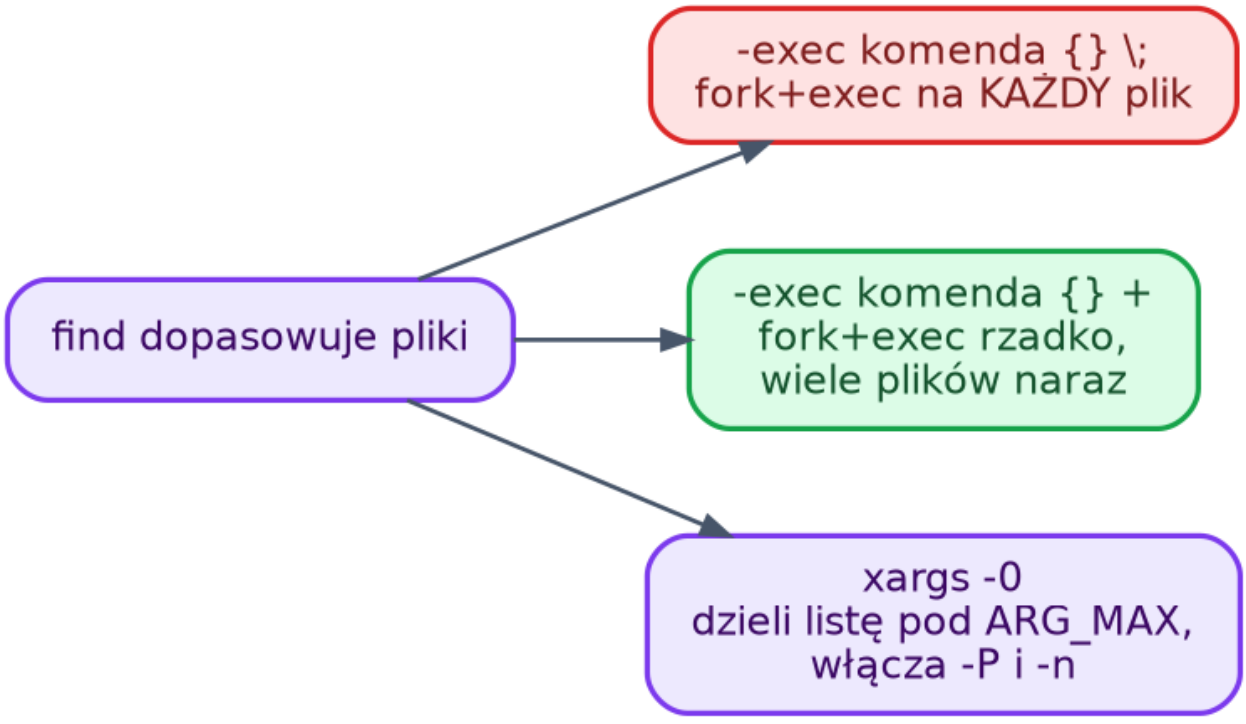
Ograniczenie, które analogia świetnie oddaje: **adres musi być czytelny**. Adres z przecinkiem w nazwie ulicy, rozdzielany po przecinkach, wyśle ekipę pod dwa błędne adresy zamiast jednego właściwego. Dokładnie to dzieje się z `xargs`, gdy nazwa pliku zawiera spację, a Ty rozdzielasz wejście spacjami.

Jak to działa

Rozwiń: jak to działa pod spodem

`-exec komenda {} \;` uruchamia nowy proces `komenda` dla każdego dopasowanego pliku z osobna. `{}` to placeholder, który `find` podstawia nazwą bieżącego pliku. Średnik na końcu musi być zescapowany (`\;` albo w cudzysłowie `';'`), bo dla powłoki nieuciszony `;` kończyłby całą komendę `find` w tym miejscu. Każdy plik to osobny `fork()` i `exec()` — przy tysiącach plików to zauważalny narzut czasu.

`-exec komenda {} +` działa inaczej: `find` zbiera dopasowane ścieżki w grupy (tak duże, na ile pozwala limit długości listy argumentów) i wywołuje `komenda` **rzadko**, z wieloma plikami naraz — mechanizm zbliżony do tego, co robi `xargs`. Warunek: `{}` musi wystąpić dokładnie raz, na końcu, jako ostatni argument komendy.



Trzy sposoby przekazania wyników find dalej: -exec na każdy plik, -exec + w partiach, xargs -0 (kliknięcie otwiera pełny rozmiar)

`-ok` to `-exec` z pytaniem `y/n` przed każdym uruchomieniem. Nie ma wariantu `-ok ... +` — potwierdzenie ma sens tylko dla pojedynczych wywołań.

`-delete` to wbudowana akcja `find`, która kasuje dopasowany wpis bez uruchamiania zewnętrznego `rm`. Jest szybsza (brak `fork/exec` na każdy plik) i nie ma problemu z cudzysłowami czy spacjami, bo `find` operuje na deskryptorach, nie na tekście przez powłokę. Ma za to własną, poważną pułapkę: **implikuje** `-depth` (zawartość katalogu przed samym katalogiem) i jest samo w sobie testem zwracającym zawsze prawdę — pozycja w wyrażeniu decyduje, co faktycznie zostanie skasowane.

`xargs` rozwiązuje inny problem: co zrobić, gdy komenda, której chcesz użyć, w ogóle nie jest wywoływana przez `find` (bo np. budujesz listę plików inaczej niż przez `find`), albo gdy potrzebujesz kontroli, jakiej `-exec` nie daje — równoległości (`-P`), limitu argumentów na wywołanie (`-n`), wstawienia argumentu w dowolnym miejscu (`-I{}`).

> **Pojęcie: ARG_MAX.** Limit jądra Linux na maksymalną łączną długość argumentów i zmiennych środowiskowych pojedynczego wywołania `execve()`, czyli pojedynczego startu programu. Sprawdzisz go komendą `getconf ARG_MAX` (typowo kilka megabajtów). Komenda w rodzaju `rm *` w katalogu ze 100 000 plików może zbudować listę argumentów przekraczającą ten limit i skończyć się błędem `Argument list too long`. `xargs` istnieje między innymi po to, żeby dzielić długą listę na porcje mieszczące się pod limitem i wywoływać komendę tyle razy, ile trzeba, automatycznie.

Domyślnie `xargs` dzieli wejście na argumenty po **białych znakach**, nie po liniach. To źródło największej pułapki tej strony: plik `raport końcowy.txt` trafi do `xargs` jako dwa osobne argumenty, `raport` i `końcowy.txt` — żaden nie istnieje jako plik, więc komenda dostanie błąd „No such file or directory".

Rozwiązanie — `-print0` i `xargs -0`. `find ... -print0` wypisuje wyniki rozdzielone bajtem NUL (`\0`) zamiast znakiem nowej linii. Bajt NUL nie może wystąpić w nazwie pliku w POSIX-owym systemie plików (to jedyny bajt, którego tam zabroniono), więc jest jedynym w pełni bezpiecznym separatorem. `xargs -0` mówi „dziel wejście po NUL, nie po białych znakach". Ta para działa poprawnie dla dowolnej nazwy pliku — ze spacjami, cudzysłowami, znakami nowej linii w nazwie (tak, to legalne w Linuksie).

`find -exec` kontra `xargs` — jeśli komenda ma wbudowaną obsługę wielu plików i nie potrzebujesz `-P`/`-n`, `-exec ... +` jest prostsze i równie bezpieczne (bez problemu ze spacjami, bo `find` przekazuje argumenty binarnie, nie przez tekst). Sięgaj po `xargs -0`, gdy potrzebujesz równoległości, kontroli nad rozmiarem partii, albo budujesz listę plików z czegoś innego niż `find` (np. `git ls-files -z`).

Alternatywa bez `xargs`: pętla `while read`. Gdy logika przetwarzania jest złożona, czytelniejsza bywa pętla:

```
find . -name "*.csv" -print0 | while IFS= read -r -d '' plik; do
    echo "przetwarzam: $plik"
done
```

`-d ''` mówi `read`, żeby separatorem rekordu był bajt NUL, `-r` wyłącza interpretację `\`, `IFS=` zapobiega ucinaniu białych znaków na brzegach nazwy. Wolniejsze przy dużej liczbie plików, ale najbezpieczniejsze i najczytelniejsze dla logiki warunkowej.

Krok po kroku

1. Sprawdź, zanim skasujesz.

```
find /tmp -type f -mtime +7 -print
```

```
/tmp/build-8821/cache.tmp
/tmp/upload-4471.part
```

Zawsze ten krok pierwszy. Dopiero gdy lista wygląda właściwie, podmieniasz `-print` na `-delete`.

2. Skasuj z potwierdzeniem, plik po pliku.

```
find /tmp -type f -mtime +7 -ok rm {} \;
```

```
< rm ... /tmp/build-8821/cache.tmp > ? y
< rm ... /tmp/upload-4471.part > ? n
```

Odpowiadasz `y` lub `n` dla każdego. Praktyczne przy małej liczbie plików i operacji, której nie chcesz automatyzować w pełni.

3. Skasuj wszystko naraz, szybciej niż `-exec rm`.

```
find /tmp -type f -mtime +7 -delete
```

Bez tworzenia procesu `rm` dla każdego pliku — `find` kasuje wpisy bezpośrednio.

4. Zmień uprawnienia masowo, osobno dla katalogów i plików.

```
find /srv/app -type d -exec chmod 755 {} +
find /srv/app -type f -exec chmod 644 {} +
```

Katalogi potrzebują bitu wykonywania (żeby dało się do nich wejść), pliki zwykle nie. Jedna wspólna komenda `chmod -R 755 /srv/app` nadałaby wykonywanie również plikom danych, co bywa niepożądane.

5. Przetwórz nazwy ze spacjami bezpiecznie.

```
find /home/adm_user/dokumenty -iname "*raport*" -print0 | xargs -0 -I{} cp {} /backup/
```

`-I{}` pozwala wstawić `{}` w dowolnym miejscu komendy `cp`, nie tylko na końcu — tu jako pierwszy argument, przed stałym katalogiem docelowym.

Częste błędy

`-delete` postawione przed testami.

```
find /tmp -delete -name "*.tmp"
```

`find` ocenia wyrażenie od lewej do prawej. `-delete` samo w sobie jest testem, który zawsze zwraca prawdę **i wykonuje kasowanie natychmiast**, zanim dojdzie do `-name "*.tmp"`. Efekt: ta komenda kasuje **każdy** plik i katalog w `/tmp`, bo test `-name` nigdy nie zdążył nic odfiltrować. To jedna z niewielu komend `find`, która wyrządza nieodwracalną szkodę przez samą kolejność. Naprawa — testy zawsze przed `-delete`:

```
find /tmp -name "*.tmp" -delete
```

Dodatkowo: `-delete` implikuje `-depth` (kasuje zawartość katalogu przed samym katalogiem), więc nie da się go swobodnie łączyć z ręcznie ustawionym `-maxdepth` bez przemyślenia kolejności przetwarzania.

Nazwy plików ze spacjami rozjeżdżające `xargs`.

```
find /home/adm_user -name "*.txt" | xargs rm
```

```
rm: cannot remove 'notatka': No such file or directory
rm: cannot remove 'robocza.txt': No such file or directory
```

Plik nazywał się `notatka robocza.txt`. `find` wypisał go poprawnie jako jedną linię, ale `xargs` domyślnie dzieli wejście po białych znakach, nie po liniach — dostał dwa osobne „argumenty", z których żaden nie jest istniejącym plikiem. Naprawa, zawsze ta sama para:

```
find /home/adm_user -name "*.txt" -print0 | xargs -0 rm
```

Puste wyjście `find` uruchamiające komendę „na sucho”.

```
find . -name "*.orphan" -print0 | xargs -0 rm -v
```

Jeśli `find` nic nie znalazł, `xargs` i tak wywoła `rm` **raz, bez żadnych argumentów** — kończy się to błędem albo bywa po cichu zignorowane w skrypcie, mimo że lista była pusta z niewłaściwego powodu (np. literówka w `-name`). Naprawa: flaga `-r` (GNU xargs) pomija wywołanie komendy, gdy wejście jest puste.

```
find . -name "*.orphan" -print0 | xargs -r -0 rm -v
```

Pomylenie `-exec {} \;` z `-exec {} +` przy komendzie, która nie przyjmuje wielu argumentów.

```
find . -name "*.txt" -exec echo "Plik:" {} +
```

```
Plik: a.txt b.txt c.txt
```

Zamiast trzech osobnych linii dostajesz jedną, z wszystkimi nazwami sklejonymi za jednym `{}` — `+` zbiera wszystkie dopasowania i wstawia je razem, wywołując `echo` tylko raz. Dla komend, które mają zadziałać osobno na każdym pliku, właściwa forma to `\;`:

```
find . -name "*.txt" -exec echo "Plik:" {} \;
```

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Nic nie wymaga uprawnień administratora.

Ćwiczenie 1 — pliki ze spacjami w nazwie

Przygotuj testowe pliki w `/tmp`, część z nazwami zawierającymi spacje.

```
mkdir -p /tmp/lab2
cd /tmp/lab2
touch "raport dzienny.tmp" "cache.tmp" "notatka robocza.tmp"
touch -d '10 days ago' cache.tmp
mkdir -p pusty_katalog
```

1. Wypisz (nie kasuj!) pliki `.tmp` starsze niż 5 dni.
2. Spróbuj skasować wszystkie pliki `.tmp` przez `find ... | xargs rm` bez `-print0`/`-0` i zobacz, co się nie uda.
3. Popraw to na wersję bezpieczną dla nazw ze spacjami.
4. Skasuj pusty katalog.

Odpowiedź

```
# 1
find /tmp/lab2 -type f -name "*.tmp" -mtime +5 -print

# 2 — zawiedzie na plikach ze spacją w nazwie
find /tmp/lab2 -name "*.tmp" | xargs rm
# rm: cannot remove 'raport': No such file or directory
# rm: cannot remove 'dzienny.tmp': No such file or directory

# 3 — wersja bezpieczna
find /tmp/lab2 -name "*.tmp" -print0 | xargs -0 rm -v

# 4
find /tmp/lab2 -type d -empty -delete
```

Punkt 2 celowo pokazuje błąd — zobacz komunikat `No such file or directory` na własne oczy, zanim zapamiętasz, dlaczego `-print0`/`-0` jest obowiązkowe przy niepewnych nazwach plików.

Pytania kontrolne

1. Dlaczego `find . -delete -name "*.tmp"` jest dużo bardziej niebezpieczne niż zwykła literówka w nazwie flagi?

Odpowiedź

Bo `find` ocenia wyrażenie od lewej do prawej, a `-delete` jest jednocześnie testem (zawsze prawdziwym) i akcją wykonywaną natychmiast. `find` kasuje każdy napotkany plik, zanim dojdzie do sprawdzenia `-name "*.tmp"` — test filtrujący nigdy nie zdążył zadziałać. To nie błąd składni, który `find` odrzuci, tylko poprawne wyrażenie robiące coś zupełnie innego niż zamierzone, bez ostrzeżenia.

2. Kiedy `-exec komenda {} +` nie zadziała tak, jak oczekujesz, mimo że składniowo jest poprawne?

Odpowiedź

Gdy komenda ma znaczenie „per plik", a nie „na liście plików naraz" — `echo "Plik:" {} +` wypisze wszystkie nazwy w jednej linii zamiast osobno, bo `+` wstawia wszystkie dopasowania za jeden `{}` i woła komendę raz. Forma `+` ma sens dla komend przyjmujących wiele plików i robiących z nimi to samo (`chmod`, `rm`, `cat`), nie dla komend, których wynik zależy od pojedynczego argumentu.

3. Masz listę plików z `find`, w tym plik o nazwie zawierającej spację. Dlaczego `find ... | xargs komenda` zawiedzie, a `find ... -exec komenda {} +` nie?

Odpowiedź

`find ... | xargs komenda` przesyła nazwy plików jako tekst rozdzielony nowymi liniami, a `xargs` bez `-0` dzieli ten tekst dalej po białych znakach — spacja w nazwie tnie ją na dwie. `find -exec ... +` nie przechodzi przez żadną warstwę tekstową: `find` przekazuje każdą ścieżkę jako osobny, kompletny argument bezpośrednio do `execve()`, więc spacja w nazwie nie ma znaczenia. Ten sam efekt dla `xargs` daje para `-print0` i `xargs -0`.

4. Dlaczego warto dodać `-r` do `xargs`, nawet jeśli akurat jesteś pewien, że `find` coś znajdzie?

Odpowiedź

Bo skrypty żyją dłużej niż założenia, przy których je pisano. Warunki, które dziś zawsze coś znajdują, jutro mogą nie znaleźć nic — zmieni się struktura katalogów, ktoś posprząta ręcznie, zmieni się wzorzec `-name`. Bez `-r` `xargs` i tak wywoła komendę raz, bez argumentów — dla `rm` kończy się to błędem, ale dla komend interpretujących „brak argumentu" jako „operuj na wszystkim" może zrobić coś zupełnie nieoczekiwanego.

Powiązane: find: wyszukiwanie plików po atrybutach · Kod wyjścia, $?, &&, ||, średnik · Potoki i filozofia Unix · Cudzysłowy, escapowanie i ekspansja

Źródła: `man 1 find` · `man 1 xargs` · `man 1 chmod` · GNU findutils manual, sekcje „Actions" i „Delete"

Revision #18
Created 2026-09-17 15:06:51 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:50 UTC by Tomasz Konieczny