Anatomia komendy: polecenie, opcje, argumenty

Poziom: podstawy **Dotyczy: wszystkie dystrybucje** **Czas: ~10 min**

Każda komenda ma tę samą budowę: nazwa, opcje, argumenty. Ta strona tłumaczy, skąd powłoka wie, gdzie kończą się opcje, a zaczynają dane — i dlaczego `which` czasem kłamie.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
BSD	Berkeley Software Distribution: rodzina Uniksów z uczelni Berkeley, inna gałąź niż Linux, stąd czasem inna składnia tych samych poleceń
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

TYLKO ODCZYT

Podstawowa budowa: polecenie, opcje, argumenty

```
ls -l -a /etc
```

`ls` to polecenie, `-l` i `-a` to dwie osobne opcje krótkie, `/etc` to argument.

Przykład:

```
total 1084
drwxr-xr-x.  94 root root  8192 Sep 17 08:02 .
dr-xr-xr-x.  18 root root   244 Jan  5  2026 ..
-rw-r--r--.   1 root root    16 Jul  2  2025 .updated
```

```
-rw-r--r--.   1 root root  1529 Aug 14 11:02 adjtime
drwxr-xr-x.   2 root root  4096 Aug 14 11:02 alternatives
-rw-r--r--.   1 root root   541 Jun 10  2025 bashrc
```

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Sklejenie opcji krótkich

```
ls -la /etc
```

Kilka opcji krótkich można skleić za jednym myślnikiem. Wynik identyczny jak wyżej — to dokładnie ta sama komenda, tylko zapisana krócej.

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Odpowiedniki opcji długich

```
ls --all --human-readable /etc
```

Większość opcji krótkich ma długi, opisowy odpowiednik. `--human-readable` nie robi tu nic widocznego, bo bez `-l` (formatu długiego) nie ma rozmiarów do sformatowania — widać za to efekt `--all`.

Przykład:

```
.   ..  .updated  adjtime  alternatives  bashrc  crontab  default
dhcp  dnf  environment  exports  fstab  group  hostname  hosts
```

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Opcja długa z argumentem obowiązkowym

```
grep --regexp=ERROR aplikacja.log
```

Wartość po `=` jest dla `--regexp` obowiązkowa, więc `--regexp ERROR aplikacja.log` (ze spacją zamiast `=`) da dokładnie ten sam wynik.

Przykład:

```
aplikacja.log:2026-09-17 08:14:02 ERROR polaczenie z baza przerwane
aplikacja.log:2026-09-17 08:16:47 ERROR timeout klienta
```

grep = global regular expression print: szukaj wzorca w tekście

TYLKO ODCZYT

Opcja długa z argumentem opcjonalnym — tylko `=`

```
ls --color=auto /etc/profile.d
```

Gdy wartość opcji jest opcjonalna, GNU akceptuje ją wyłącznie po znaku `=`. Zapis ze spacją zamienia wartość w osobny argument — więcej w sekcji Częste błędy.

Przykład:

```
256term.csh  colorls.sh  lang.sh  less.sh  vim.sh
```

ls = list: wypisz zawartość katalogu

OSTROŻNIE · NIEODWRACALNE

Koniec opcji: --

```
rm -- -rf
```

Wszystko po `--` jest traktowane jak argument, nawet jeśli zaczyna się od myślnika — to jedyny przenośny sposób usunięcia pliku o takiej nazwie. Powodzenie jest ciche, `rm` niczego nie wypisuje.

```
grep -- -v plik.txt
```

Bez `--` `grep` odczytałby `-v` jako opcję (odwrócenie dopasowania) zamiast szukanego tekstu.

Przykład:

```
opcja -v włącza tryb pełny
```

rm = remove: usuń (bez kosza, na zawsze) · **grep** = global regular expression print: szukaj wzorca w tekście

TYLKO ODCZYT

Sprawdzenie, czym jest dana nazwa

```
type -a echo
```

Wypisujemy **wszystkie** znaczenia nazwy, w kolejności, w jakiej powłoka je rozstrzyga: najpierw alias (jeśli istnieje), potem builtin, na końcu plik z dysku.

Przykład:

```
echo is a shell builtin
echo is /usr/bin/echo
```

type = pokaż, czym jest dane słowo dla powłoki: poleceniem, aliasem czy funkcją

TYLKO ODCZYT

Wypisanie samego słowa: type -t

```
type -t ls
```

Zwracamy jedno słowo: `alias`, `keyword`, `function`, `builtin` albo `file`. Wygodne w skryptach do testu warunkowego.

Przykład:

```
alias
```

type = pokaż, czym jest dane słowo dla powłoki: poleceniem, aliasem czy funkcją

TYLKO ODCZYT

Przenośny odpowiednik: command -v

```
command -v ls
```

To samo co `type -t`, ale w wersji zgodnej z POSIX — działa identycznie w `bash`, `dash` i `zsh`. Dla aliasu wypisujemy jego pełną definicję.

Przykład:

```
alias ls='ls --color=auto'
```

POSIX = Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach · **command** = uruchom polecenie z pominięciem aliasów i funkcji powłoki o tej samej nazwie

TYLKO ODCZYT

Pominięcie aliasu

```
\ls /etc
```

Ukośnik przed nazwą każe powłoce pominąć alias i sięgnąć do kolejnego znaczenia — zwykle builtinu albo pliku.

TYLKO ODCZYT

Pominięcie aliasu i funkcji

```
command ls /etc
```

`command` idzie o krok dalej: pomija też funkcje powłoki o tej samej nazwie, zostawiając builtin albo plik z dysku.

command = uruchom polecenie z pominięciem aliasów i funkcji powłoki o tej samej nazwie

TYLKO ODCZYT

Pominięcie wszystkiego — pełna ścieżka

```
/usr/bin/ls /etc
```

Pełna ścieżka omija alias, funkcję i builtin naraz — uruchamia dokładnie ten plik wykonywalny, który tam leży.

◫◫ TYLKO ODCZYT

Lista i kasowanie aliasów

```
alias
```

Bez argumentów wypisujemy wszystkie zdefiniowane aliasy bieżącej sesji.

Przykład:

```
alias ll='ls -lah'
alias ls='ls --color=auto'
alias rm='rm -i'
```

```
unalias ll
```

Kasujemy jeden alias z bieżącej sesji. Bez efektu na przyszłe sesje, jeśli alias jest zdefiniowany w pliku startowym.

◫◫ **alias** = nadaj poleceniu własny, krótszy skrót na czas sesji powłoki

◫◫ TYLKO ODCZYT

Wszystkie flagi

Zapis	Znaczenie
`-a`	opcja krótka, jedna litera
`-la`	dwie opcje krótkie sklejone (`-l -a`)
`--all`	opcja długa
`--opcja=wartość`	opcja długa z wartością — jedyny poprawny zapis, gdy wartość jest opcjonalna
`--`	koniec opcji; wszystko dalej to argumenty, nawet jeśli zaczyna się od myślnika
`[...]` / `...` w `man`	argument opcjonalny / można go powtórzyć

Uważaj

- W sklejonych opcjach krótkich ta, która bierze argument, musi być **ostatnia**: `tar -cfz` czyta `z` jako nazwę archiwum.

- `which` to osobny program i nie widzi aliasów, funkcji ani wbudowanych poleceń Twojej powłoki — używaj `type -a`.
- Alias działa tylko na **pierwsze słowo** komendy i domyślnie nie jest rozwijany w skryptach.

Z życia wzięte

Z ŻYCIA WZIĘTE

Czwartek, 14:05. Tomek dodaje do skryptu wdrożeniowego na `docker01` linię `rm plik.tmp` i wystraszony widzi, że plik znika bez pytania, choć w swoim terminalu zawsze dostaje potwierdzenie. Woła Olę, a ta każe sprawdzić `type -a rm` zamiast `which rm`: w terminalu działa alias `rm='rm -i'`, którego skrypt w ogóle nie widzi. Dwie minuty później Tomek wie, że `which` i `type` odpowiadają na dwa różne pytania.

Po co to istnieje

`tar -czf backup.tar.gz /etc` wygląda jak zaklęcie. Nie jest: to nazwa programu, trzy litery opcji sklejone w jedno słowo i dwa argumenty. Kiedy zobaczysz w tym strukturę, przestaniesz kopiować komendy na ślepo i zaczniesz je czytać. Druga rzecz: powłoka rozstrzyga nazwę komendy w określonej kolejności i potrafi wykonać coś innego, niż pokazuje `which` — stąd ta sama komenda daje inny wynik w Twoim terminalu i w skrypcie z crona.

Analogia

Komenda to **formularz składany w okienku urzędu**.

Na górze jest **nazwa sprawy** — „wniosek o wydanie zaświadczenia". To polecenie. Pod spodem rubryka z **polami wyboru**: tryb pilny, odbiór osobisty, powiadomienie SMS-em. To opcje krótkie — zaznaczenie trzech to dalej jeden formularz, nie trzy. Niżej **rubryki z miejscem na treść**: „powiadomienie SMS-em na numer ____". To opcja z argumentem. Na końcu **dane sprawy**: imię, nazwisko, adres — czyli argumenty.

Analogia niesie dwa ograniczenia, które tłumaczą realne pułapki. Pierwsze: **urzędnik nie zna Twoich intencji, czyta pola po kolei**. Jeśli w rubryce „nazwisko" wpiszesz coś, co wygląda jak zaznaczone pole wyboru, potraktuje to jak pole wyboru. Dokładnie tak powłoka zachowuje się wobec pliku nazwanego `-rf` — i dlatego istnieje `--`, czyli poprzeczna kreska na formularzu z

podpisem „poniżej same dane".

Drugie: **w jednym wierszu pól wyboru tylko ostatnie ma dopisane miejsce na treść**. Jeśli zaznaczysz trzy pola i wpiszesz wartość, trafi ona do ostatniego zaznaczonego. Stąd zasada, że w sklejonych opcjach krótkich ta, która wymaga argumentu, musi stać na końcu: `tar -czf plik.tar.gz` działa, `tar -cfz plik.tar.gz` przypisuje wartość literze `f`... i literę `z` uznaje za nazwę archiwum.

Jak to działa

Rozwiń: jak to działa pod spodem

Opcje krótkie, długie i ich argumenty

Opcja krótka to myślnik plus jedna litera; kilka można skleić, więc `ls -l -a -h` to to samo co `ls -lah`. Argument opcji krótkiej może być doklejony albo oddzielony spacją (`head -n5`, `head -n 5`, `head -5`), ale taka opcja musi kończyć sklejkę.

Opcja długa to dwa myślniki i słowo: `--all`, `--human-readable`. Wartość podaje się przez `=` albo spację, i tu jest niuans wart zapamiętania:

- argument **obowiązkowy**: `--regexp=wzorzec` i `--regexp wzorzec` działają tak samo;
- argument **opcjonalny**: działa wyłącznie `--color=auto`. Zapis `--color auto` oznacza „opcja `--color` bez wartości, a `auto` to osobny argument".

GNU pozwala też skracać opcje długie, dopóki skrót jest jednoznaczny (`ls --hum` zadziała). W skryptach tego nie używaj: nowa wersja programu doda opcję i skrót przestanie być jednoznaczny.

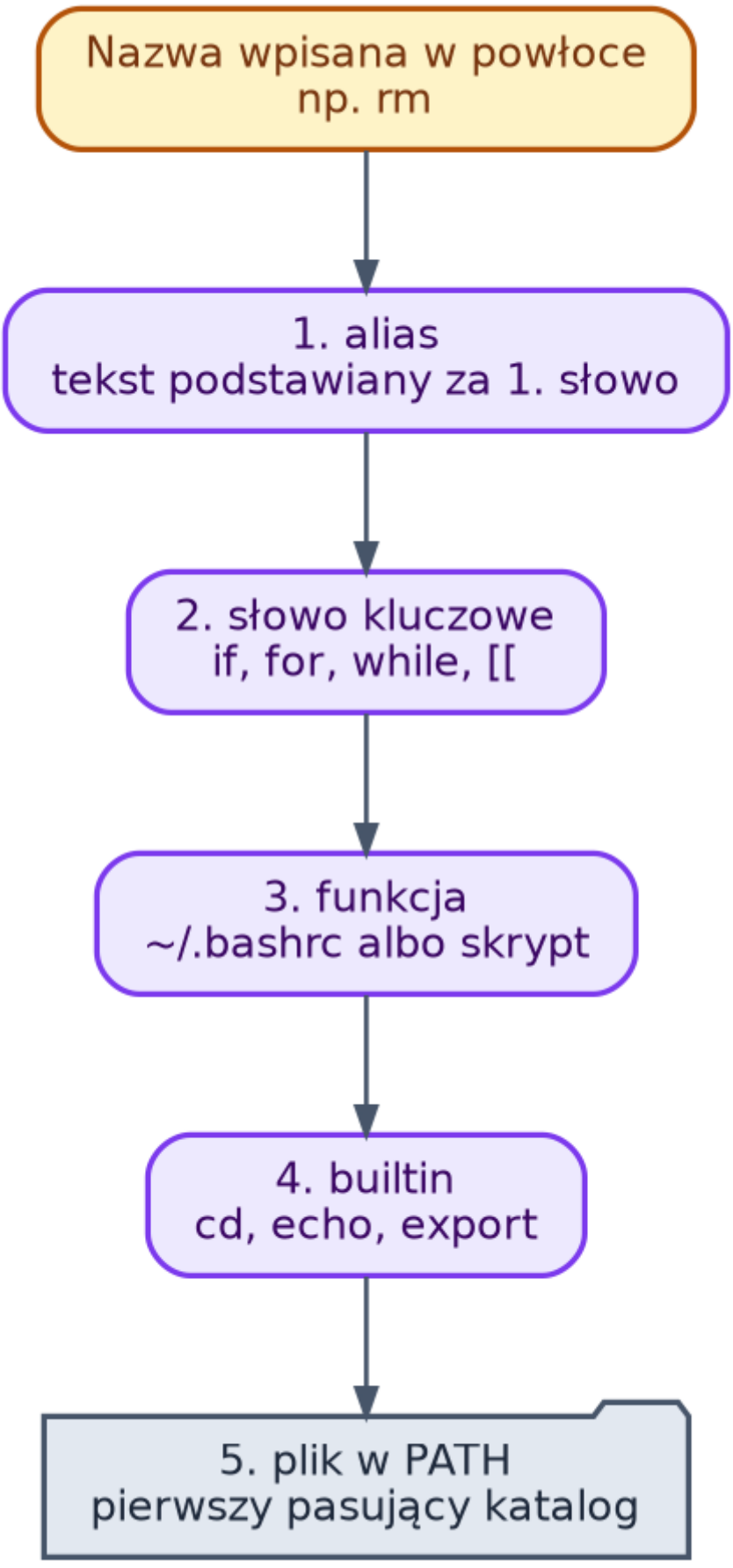
`--`, czyli koniec opcji

Wszystko po `--` jest argumentem, nawet jeśli zaczyna się od myślnika. To jedyny przenośny sposób na pracę z plikami o nieszczęśliwych nazwach i na przekazanie wzorca zaczynającego się od `-`: `rm -- -rf`, `grep -- -v plik.txt`. Drugi sposób to zapisanie ścieżki tak, by nie zaczynała się od myślnika: `rm ./-rf` wskazuje ten sam plik, ale zaczyna się od kropki, więc nikt nie pomyli go z opcją.

Builtin, funkcja, alias, program

Nie wszystko, co wpisujesz, jest plikiem na dysku. Bash rozstrzyga nazwę **w tej kolejności**:

1. **alias** — skrót tekstowy, rozwijany przy parsowaniu, tylko dla pierwszego słowa komendy;
2. **słowo kluczowe** — `if`, `for`, `while`, `[[`, `function`;
3. **funkcja** — zdefiniowana w `~/.bashrc` albo w skrypcie;
4. **polecenie wbudowane (builtin)** — `cd`, `echo`, `test`, `export`, `pwd`; kod jest w binarce basha;
5. **plik wykonywalny znaleziony w `PATH`** — pierwszy pasujący katalog wygrywa.


Nazwa wpisana w powłoce
np. rm
1. alias
tekst podstawiany za 1. słowo
2. słowo kluczowe
if, for, while, [[
3. funkcja
~/.bashrc albo skrypt
4. builtin
cd, echo, export
5. plik w PATH
pierwszy pasujący katalog

Kolejność rozstrzygania nazwy komendy przez powłokę, od aliasu do pliku w PATH (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: polecenie wbudowane (builtin).** Polecenie, którego kod jest częścią samej powłoki, a nie osobnym plikiem na dysku. Powłoka wykonuje je we własnym procesie, bez uruchamiania nowego programu, dlatego builtiny takie jak `cd` czy `export` mogą zmieniać stan samej powłoki — katalog roboczy, zmienne środowiskowe. Program zewnętrzny by tego nie potrafił: działałby w osobnym procesie i zniknąłby razem ze swoimi zmianami, gdy tylko by się zakończył.

Niektóre builtiny istnieją też jako osobne programy: `echo` jest wbudowane w bash **i** leży w `/usr/bin/echo` z coreutils. W terminalu uruchamia się wersja wbudowana, ale `find . -exec echo {} \;` uruchomi tę z dysku, bo `find` nie ma powłoki — a obie różnią się obsługą flag.

Co jest czym, sprawdza `type`:

```
type -a echo
type -t cd
```

```
echo is a shell builtin
echo is /usr/bin/echo
builtin
```

`command -v nazwa` robi to samo w wersji POSIX i jest właściwym narzędziem w skryptach do sprawdzenia „czy to w ogóle istnieje".

`which` jest inne: to **osobny program** w nowym procesie. Nie ma dostępu do aliasów, funkcji ani listy builtinów Twojej powłoki, więc przeszukuje tylko `PATH`. Dlatego `which cd` nic sensownego nie powie, a `which echo` pokaże `/usr/bin/echo` — ścieżkę do programu, którego Twoja powłoka i tak nie uruchomi. Na RHEL `which` bywa opakowane aliasem z `/etc/profile.d/which2.sh`, który dokłada aliasy i funkcje; na Debianie to prosty skrypt bez tej wiedzy. Skoro zachowanie zależy od dystrybucji: w powłoce `type -a`, w skrypcie `command -v`.

Aliasy i jak je ominąć

```
alias ll='ls -lah'
alias rm='rm -i'
```

Alias podstawia tekst za pierwsze słowo komendy, więc `ll /etc` stanie się `ls -lah /etc`. Aliasy nie przyjmują argumentów w środku (do tego służą funkcje) i domyślnie **nie są rozwijane w**

skryptach — chyba że ustawisz `shopt -s expand_aliases`. Żeby jednorazowo uruchomić oryginał:

Zapis	Pomija
`\rm` albo `'rm'`	alias
`command rm`	alias i funkcję
`/usr/bin/rm`	alias, funkcję i builtin
`builtin echo`	funkcję o tej samej nazwie

Krok po kroku

1. Rozłóż komendę na części.

```
tar -czf /var/tmp/etc-backup.tar.gz /etc
```

`tar` to polecenie; `-czf` to trzy sklejone opcje krótkie (`-c` utwórz, `-z` gzip, `-f` plik archiwum); `/var/tmp/etc-backup.tar.gz` to argument opcji `-f`; `/etc` to argument właściwy.

2. Sprawdź, czym jest nazwa, którą wpisujesz.

```
type -a ls
```

```
ls is aliased to `ls --color=auto'
ls is /usr/bin/ls
```

Pierwsza linia tłumaczy, dlaczego wynik `ls` jest kolorowy u Ciebie, a nie w skrypcie.

3. Uruchom oryginał z pominięciem aliasu: `\ls /etc` albo `command ls /etc`.

4. Przeczytaj składnię z `man`.

```
man 1 ls | head -n 8
```

```
SYNOPSIS
       ls [OPTION]... [FILE]...
```

Nawiasy kwadratowe oznaczają element opcjonalny, wielokropek — że można go powtórzyć: `ls` bez niczego jest poprawne, a opcji i plików może być wiele.

Częste błędy

`--color auto` zamiast `--color=auto`.

```
ls --color auto
```

```
ls: cannot access 'auto': No such file or directory
```

`--color` ma argument **opcjonalny**, a takie argumenty w GNU przyjmuje się wyłącznie przez `=`. Bez niego `auto` zostaje potraktowane jako nazwa pliku do wylistowania. Napraw: `ls --color=auto`.

Plik o nazwie zaczynającej się od myślnika.

Skrypt zapisał plik jako `-rf`, bo zmienna z nazwą była pusta i myślnik został sam. Próbujesz go usunąć:

```
rm -rf
```

```
rm: missing operand
Try 'rm --help' for more information.
```

`rm` odczytał `-rf` jako dwie opcje i nie dostał argumentu. Gdyby obok stał jakikolwiek inny plik, skasowałbyś go rekurencyjnie i bez pytania. Poprawnie:

```
rm -- -rf
rm ./-rf
```

Zła kolejność w sklejonych opcjach `tar`.

```
tar -cfz backup.tar.gz /etc
```

```
tar: z: Cannot open: No such file or directory
tar: Error is not recoverable: exiting now
```

`-f` bierze argument, więc wzięło pierwsze, co miało pod ręką — literę `z`. Nazwa `backup.tar.gz` została potraktowana jak kolejny plik do spakowania. Zasada: opcja z argumentem na końcu sklejki, czyli `tar -czf backup.tar.gz /etc`.

`which` pokazuje co innego, niż się wykonuje.

Masz `alias grep='grep --color=auto --exclude-dir=.git'`, a `which grep` odpowiada:

```
/usr/bin/grep
```

Wygląda niewinnie, więc wklejasz tę samą komendę do skryptu — i wyniki się różnią, bo w skrypcie aliasu nie ma. `which` jest osobnym procesem i nie widzi aliasów; prawdę mówi `type -a grep`.

`ps -aux` zamiast `ps aux`.

```
Warning: bad ps syntax, perhaps a bogus '-'?
```

`ps` obsługuje trzy składnie naraz: BSD bez myślnika, UNIX z myślnikiem, GNU z dwoma. `aux` to składnia BSD i myślnika nie chce; `-aux` znaczy formalnie „wszystkie procesy użytkownika o nazwie `x`". Ostrzeżenie dostajesz słusznie.

□□ Ćwiczenia

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

Ćwiczenie 1 — plik nie do usunięcia

```
cd /tmp
mkdir -p cwiczenie-opcje && cd cwiczenie-opcje
touch -- -rf zwykly.txt
ls -l
rm -rf
```

Usuń plik `-rf`, nie kasując `zwykly.txt`.

Odpowiedź

`rm -rf` wypisze `rm: missing operand`, bo `-rf` zostało odczytane jako opcje. Dwa poprawne sposoby:

```
rm -- -rf
rm ./-rf
```

Ten sam problem dotyczy każdego narzędzia: `cat -- -rf`, `mv -- -rf nowa-nazwa`. Sprzątanie: `cd /tmp && rm -rf cwiczenie-opcje`.

Ćwiczenie 2 — co naprawdę się wykona

```
alias echo='echo [ALIAS]'
type -a echo
echo test
\echo test
command echo test
/usr/bin/echo test
unalias echo
```

Odpowiedź

`type -a echo` wypisze trzy linie — alias, builtin i `/usr/bin/echo`, w kolejności rozstrzygania. `echo test` da `[ALIAS] test`; pozostałe trzy formy dadzą samo `test`, bo `\echo` i `command echo` pomijają alias i trafiają w builtin, a `/usr/bin/echo` uruchamia program z coreutils.

Pytania kontrolne

1. Dlaczego `rm -rf` nie usuwa pliku o nazwie `-rf` i jak to zrobić poprawnie?

Odpowiedź

Bo rozdzielaniem opcji od argumentów zajmuje się sam program (przez `getopt`), nie system plików — wszystko, co zaczyna się od myślnika, jest domyślnie opcją. `rm` widzi więc `-r` i `-f` bez żadnego argumentu i kończy na `missing operand`. Rozwiązania: `rm -- -rf` (jawny koniec opcji) albo `rm ./-rf` (ta sama ścieżka zapisana tak, by nie zaczynała się od myślnika).

2. `which python3` pokazuje `/usr/bin/python3`, ale `python3 --version` daje w Twoim terminalu inną wersję niż w skrypcie z crona. Jak to sprawdzić?

Odpowiedź

`type -a python3`. `which` przeszukuje wyłącznie `PATH` w osobnym procesie i nie wie o aliasach, funkcjach ani o tym, że Twój `PATH` w sesji interaktywnej zawiera dodatkowy katalog (venv, `~/.local/bin`), którego cron nie ma. `type -a` pokaże wszystkie znaczenia nazwy w kolejności rozstrzygania, a `echo "$PATH"` w obu środowiskach dopowie resztę.

3. `tar -cfz backup.tar.gz /etc` kończy się `tar: z: Cannot open`. Co dokładnie zrozumiał `tar`?

Odpowiedź

Że ma utworzyć archiwum (`-c`), a nazwa pliku archiwum (`-f`) to `z` — bo opcja pobierająca argument bierze go zaraz po sobie, a w sklejce `-cfz` zaraz po `f` stoi litera `z`. Tworzenie pliku `z` się nie udało, więc `tar` przerwał. Gdyby się udało, `backup.tar.gz` i `/etc` zostałyby potraktowane jako dwie rzeczy do spakowania. Poprawnie: `tar -czf backup.tar.gz /etc`.

4. Kiedy `ls --color auto` i `ls --color=auto` dają różny wynik, a `grep --regexp wzorzec` i `grep --regexp=wzorzec` ten sam?

Odpowiedź

Bo `--color` ma argument opcjonalny, a `--regexp` obowiązkowy. Przy obowiązkowym parser GNU może bezpiecznie wziąć następne słowo, skoro wartość i tak musi się pojawić. Przy opcjonalnym nie da się zgadnąć, czy następne słowo jest wartością, czy zwykłym argumentem — więc przyjmowana jest wyłącznie forma z `=`, a `auto` staje się nazwą pliku.

Powiązane: Terminal, TTY, konsola, powłoka · System plików, ścieżki, pwd, cd, ls · Zmienne środowiskowe i PATH · man, help, apropos — czytanie dokumentacji

Źródła: `man 1 bash` (sekcje *ALIASES*, *SHELL BUILTIN COMMANDS*, *COMMAND EXECUTION*) · `help type` · `man 1 which` · `man 3 getopt` · POSIX.1-2024, *Utility Syntax Guidelines*

Revision #19
Created 2026-09-17 11:56:09 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:21 UTC by Tomasz Konieczny