Aliasy i funkcje powłoki

Poziom: podstawy **Dotyczy: wszystkie dystrybucje** **Czas: ~16 min**

Alias to skrót na jedną komendę, funkcja powłoki to skrót na cały mały skrypt. Ta strona pokazuje, gdzie kończą się możliwości aliasu i zaczyna się potrzeba funkcji — i dlaczego oba działają inaczej, niż większość ludzi zakłada.

Fiszka

TYLKO ODCZYT

Sprawdzenie definicji aliasu: `alias NazwaAliasu`

```
alias ll
```

Bez argumentu po nazwie `alias` wypisuje, na co dokładnie jest rozwijany dany skrót — przydatne, gdy nie pamiętamy, czy `ll` u nas znaczy `ls -lah`, czy coś innego, bo skopiowaliśmy czyjś `.bashrc`.

Przykład:

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

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

TYLKO ODCZYT

Utworzenie aliasu: `alias ll='ls -lah'`

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

Definiuje nowy skrót w bieżącej powłoce. Od tej chwili wpisanie `ll` jest dla basha dokładnie tym samym, co wpisanie `ls -lah` — alias działa jako **podmiana tekstu**, wykonywana zanim cokolwiek się uruchomi. Obowiązuje tylko w tej sesji; żeby przetrwał do następnego logowania, musi trafić do pliku startowego (patrz dalej).

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

□□ TYLKO ODCZYT

Usunięcie aliasu: `unalias NazwaAliasu`

```
unalias ll
```

Kasuje alias z bieżącej powłoki. `ll` znowu będzie szukane jako zwykłe polecenie — jeśli nic takiego nie ma w systemie, dostajemy `command not found`.

Przykład:

```
bash: ll: command not found
```

□□ TYLKO ODCZYT

Wylistowanie wszystkich aliasów: `alias`

```
alias
```

Wypisuje wszystkie aliasy zdefiniowane w bieżącej powłoce, w formie gotowej do ponownego wklejenia. Pierwsze polecenie do wpisania, kiedy logujemy się na cudze konto i chcemy wiedzieć, czego się spodziewać po znajomych komendach.

Przykład:

```
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -alh --color=auto'
alias ls='ls --color=auto'
alias rm='rm -i'
```

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

□□ OSTROŻNIE · NIEODWRACALNE

Jednorazowe ominięcie aliasu: `\komenda`

```
\rm plik.txt
```

Ukośnik przed nazwą mówi bashowi „nie rozwijaj tego jako alias, wykonaj dosłownie". Jeśli ktoś na serwerze zaaliasował `rm` na `rm -i` (patrz *Częste błędy*), a akurat chcemy usunąć sto plików bez potwierdzania każdego z osobna, `\rm` jest najszybszym wyjściem.

TYLKO ODCZYT

Ominięcie aliasu i funkcji: `command komenda`

```
command ls
```

Robi to samo co `\komenda`, ale jawniej i szerzej: `command` każe bashowi pominąć zarówno aliasy, jak i funkcje powłoki o tej nazwie, i wykonać polecenie wbudowane albo plik z `PATH`. Używany też w samych funkcjach — funkcja, która ma się nazywać tak samo jak istniejące polecenie (np. własna funkcja `ls`), musi w środku wywołać `command ls`, inaczej wywoła samą siebie w nieskończoność.

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

TYLKO ODCZYT

Sprawdzenie, czym naprawdę jest dana nazwa: `type -a nazwa`

```
type -a ll
```

Pokazuje **wszystkie** znaczenia danej nazwy, w kolejności, w jakiej bash by je rozstrzygał — jeśli coś jest jednocześnie aliasem i plikiem wykonywalnym o tej samej nazwie w `PATH`, zobaczymy oba wpisy, jeden pod drugim.

Przykład:

```
ll is aliased to `ls -alh --color=auto'
```

```
type -a rm
```

Przykład:

```
rm is aliased to `rm -i'
rm is /usr/bin/rm
```

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

◫◫ ZMIENIA SYSTEM

Definicja funkcji: `nazwa() { ...; }`

```
backup() {
    cp -v "$1" "$1.bak.$(date +%F)"
}
```

Funkcja to nazwany blok poleceń, trzymany w pamięci powłoki, uruchamiany jak zwykłe polecenie. W przeciwieństwie do aliasu potrafi przyjmować argumenty w dowolnym miejscu, sprawdzać warunki, robić pętle — czyli wszystko, co potrafi skrypt basha.

```
backup /etc/hosts
```

Przykład:

```
'/etc/hosts' -> '/etc/hosts.bak.2026-09-18'
```

◫◫ **cp** = copy: skopiuj

◫◫ TYLKO ODCZYT

Wypisanie treści zdefiniowanych funkcji: `declare -f`

```
declare -f backup
```

Wypisuje pełne ciało funkcji, dokładnie tak, jakbyś czytał jej definicję w pliku źródłowym. Bez argumentu (`declare -f`) pokaże treść **wszystkich** funkcji, jakie bieżąca powłoka zna — bywa to bardzo długa lista, jeśli `.bashrc` ładuje ich dziesiątki.

Przykład:

```
backup ()
{
    cp -v "$1" "$1.bak.$(date +%F)"
}
```

◫◫ **declare** = zadeklaruj zmienną powłoki z określonym typem, na przykład jako liczbę całkowitą albo tablicę

◫◫ TYLKO ODCZYT

Wypisanie samych nazw funkcji: `declare -F`

```
declare -F
```

To samo co wyżej, ale bez treści — same nazwy, jedna na linię. Szybszy sposób na zorientowanie się, jakie funkcje w ogóle mamy zdefiniowane, zanim zaczniemy przeglądać ich kod jedna po drugiej.

Przykład:

```
declare -f backup
declare -f mkcd
declare -f wyniki
```

declare = zadeklaruj zmienną powłoki z określonym typem, na przykład jako liczbę całkowitą albo tablicę

TYLKO ODCZYT

Usunięcie funkcji: `unset -f nazwa`

```
unset -f backup
```

Kasuje funkcję z bieżącej powłoki, analogicznie do `unalias` dla aliasów. Uwaga na `unset` bez `-f`: bez tej flagi bash najpierw szuka zmiennej o tej nazwie, i dopiero gdy jej nie znajdzie, sprawdza funkcje — jeśli mamy jednocześnie zmienną i funkcję `backup`, `unset backup` skasuje zmienną, nie funkcję.

TYLKO ODCZYT

Przekazanie wszystkich argumentów dalej: `"$@"`

```
uruchom_jako_root() {
    sudo "$@"
}
```

`"$@"` w cudzysłowie rozwija się do listy wszystkich argumentów funkcji, **każdy jako osobne słowo**, z zachowaniem spacji wewnątrz argumentów. To jest właściwy sposób, żeby funkcja „przepuściła" swoje argumenty do innego polecenia, nie psując tych, które zawierają spacje.

```
uruchom_jako_root systemctl restart "moja usluga"
```

Bez cudzysłowu (`$@` albo `$*`) argument `"moja usluga"` rozpadłby się na dwa osobne słowa `moja` i `usluga`.

TYLKO ODCZYT

Udostępnienie funkcji procesom potomnym: `export -f nazwa`

```
export -f backup
```

Zwykła funkcja istnieje tylko w powłoce, w której ją zdefiniowano — podproces (np. skrypt uruchomiony jako `bash skrypt.sh`, w odróżnieniu od `source skrypt.sh`) jej nie widzi. `export -f` eksportuje funkcję przez zmienną środowiskową w specjalnym formacie, dzięki czemu bash uruchomiony jako podproces też ją odziedziczy — mechanizm działa tylko między procesami basha, nie działa dla `sh`, `dash` czy innych powłok.

export = ustaw zmienną środowiskową widoczną dla uruchamianych programów

ZMIENIA SYSTEM

Zapisanie aliasów i funkcji na stałe: `~/.bashrc`

```
cat >> ~/.bashrc <<'EOF'
alias ll='ls -alh --color=auto'
backup() { cp -v "$1" "$1.bak.$(date +%F)"; }
EOF
```

Wszystko zdefiniowane bezpośrednio w terminalu znika wraz z zamknięciem sesji. `~/.bashrc` jest czytany przy **każdej** nowej interaktywnej powłoce nielogowania (pełny mechanizm opisuje strona o plikach startowych), więc to naturalne miejsce na aliasy i funkcje, z których korzystamy codziennie.

cat = concatenate: sklej i wypisz pliki · **alias** = nadaj poleceniu własny, krótszy skrót na czas sesji powłoki

TYLKO ODCZYT

Wydzielenie aliasów do osobnego pliku: `~/.bash_aliases`

```
cat ~/.bash_aliases
```

Konwencja przyjęta w domyślnym `.bashrc` na Debianie/Ubuntu (i często kopiowana na inne dystrybucje): zamiast upychać dziesiątki aliasów w `.bashrc`, trzymamy je w osobnym pliku, a `.bashrc` dogrywa go jedną linią z testem istnienia:

Przykład:

```
# fragment ~/.bashrc

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

Zaleta: `.bashrc` zostaje czytelny, a `.bash_aliases` możemy bez wstydu skopiować między maszynami albo trzymać w repozytorium z dotfiles, niezależnie od reszty konfiguracji powłoki.

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Włączenie rozwijania aliasów w skryptach: `shopt -s expand_aliases`

```
shopt -s expand_aliases
alias hello='echo cześć'
hello
```

Domyślnie bash **nie rozwija aliasów** w powłoce nieinteraktywnej — czyli w każdym skrypcie uruchomionym jako `./skrypt.sh`. Ta opcja to włącza, ale traktuj ją jako ciekawostkę, nie jako narzędzie pracy: patrz *Częste błędy* niżej, dlaczego w skryptach zawsze lepiej sięgnąć po funkcję.

shopt = shell options: włącz albo wyłącz dodatkowe opcje zachowania powłoki bash ·**alias** = nadaj poleceniu własny, krótszy skrót na czas sesji powłoki

TYLKO ODCZYT

Wszystkie polecenia

Polecenie	Co robi
`alias NazwaAliasu='...'`	tworzy alias w bieżącej powłoce
`alias NazwaAliasu`	pokazuje definicję jednego aliasu

Polecenie	Co robi
`alias`	listuje wszystkie aliasy
`unalias NazwaAliasu`	usuwa alias
`\nazwa` / `command nazwa`	wykonuje polecenie z pominięciem aliasu (i funkcji, w przypadku `command`)
`type -a nazwa`	pokazuje, czym jest nazwa: alias, funkcja, polecenie wbudowane, plik w `PATH` — i w jakiej kolejności bash je znajdzie
`nazwa() { ...; }`	definiuje funkcję powłoki
`declare -f` / `declare -f nazwa`	pokazuje treść wszystkich funkcji / jednej funkcji
`declare -F`	listuje same nazwy funkcji, bez treści
`unset -f nazwa`	usuwa funkcję
`export -f nazwa`	udostępnia funkcję procesom potomnym (tylko bash → bash)

Uważaj

- Alias nie przyjmuje argumentów w środku — tylko na końcu, doklejone za nim.
- Aliasy nie działają w skryptach nieinteraktywnych bez `shopt -s expand_aliases`, i nawet wtedy trzeba je zdefiniować w linii **wcześniejszej** niż ich pierwsze użycie.
- `export -f` działa tylko w relacji bash-rodzic → bash-dziecko; usługa systemd czy `cron` uruchamiające `/bin/sh` tej funkcji nie zobaczą.

◫◫ OSTROŻNIE · NIEODWRACALNE

Zestaw aliasów administracyjnych do `~/.bash_aliases`

Firmowe notatki z sekcji „ALIASY UNIWERSALNE" przetrwały tylko we fragmentach — reszta poszła na tapetę przy migracji z OneNote. Poniższy zestaw odtwarza ten sam duch (uniwersalne skróty, przydatne na każdym serwerze, niezależnie od dystrybucji) na bazie poleceń faktycznie używanych w tych samych notatkach: `ps aux | grep`, `journalctl`, `firewall-cmd`, `history` bez numeracji, `sudo -l`, `crontab -e`, `netstat`/`ss`.

```
# ~/.bash_aliases


# Nawigacja i listowanie
```

```
alias ll='ls -alh --color=auto'
alias ..='cd ..'
alias ...='cd ../..'

# Bezpieczniejsze operacje na plikach
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Diagnostyka procesów i zasobów (patrz strony o ps, free, df)
alias psg='ps aux | grep -v grep | grep -i'
alias meminfo='free -h'
alias diskinfo='df -h -x tmpfs -x devtmpfs'

# Sieć (patrz strony o ss/lsof i firewalld)
alias ports='sudo ss -tunlp'
alias fwlist='sudo firewall-cmd --list-all'

# Logi i crontab
alias jctl='journalctl -xe'
alias crone='crontab -e'

# Historia bez numeracji — przydatna przy kopiowaniu poleceń do zgłoszenia
alias h='history | sed "s/^[ ]*[0-9]*[ ]*//"'

# Sprawdzenie własnych uprawnień sudo
alias mysudo='sudo -l -U "$USER"'
```

Trzynaście skrótów, żaden nie wymaga niczego ponad standardowe narzędzia opisane w innych rozdziałach tej książki. `rm`/`cp`/`mv` z `-i` warto traktować jako wygodę, nie zabezpieczenie — patrz pierwszy punkt w *Częste błędy*.

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

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 11:00: Marek dzwoni, że skrypt uruchamiany ręcznie na app01 z jego notatek działa, a ten sam skrypt uruchomiony przez `./raport.sh` zwraca `command not found` przy linii z ulubionym skrótem `log`. `log` jest aliasem zdefiniowanym w `.bashrc`, a skrypt jako powłoka nieinteraktywna w ogóle go nie rozwija. Zamiana `alias log=` na funkcję `log() { ...; }` naprawia skrypt bez dotykania niczyjego `.bashrc`.

Po co to istnieje

> **Pojęcie: alias.** Nazwany skrót, który bash podmienia na inny tekst, zanim zacznie cokolwiek interpretować — mechanizm czysto tekstowy, bez logiki, bez argumentów, bez warunków. Kiedy wpisujesz `ll -t`, bash widzi najpierw `ll`, rozpoznaje w nim alias, podmienia go na zapisany tekst (`ls -alh --color=auto`) i dopiero do **tak powstałej** linii dokleja resztę tego, co wpisałeś (`-t`) — stąd `ll -t` działa, bo `-t` po prostu doklejasz na końcu, ale nic nie da się wstawić w środku podmienianego tekstu.

Aliasy istnieją, żeby oszczędzić wpisywanie tych samych flag za każdym razem — `ls -alh --color=auto` dwadzieścia razy dziennie zamienia się w `ll`. To wygoda czysto interaktywna: nikt nie pisze skryptu administracyjnego licząc na alias, bo — jak zobaczysz w *Częste błędy* — w skryptach domyślnie ich po prostu nie ma.

> **Pojęcie: funkcja powłoki.** Nazwany blok poleceń basha, trzymany w pamięci powłoki i wywoływany jak zwykłe polecenie — ale, w odróżnieniu od aliasu, prawdziwy fragment kodu: z argumentami (`$1`, `$2`, `"$@"`), z warunkami (`if`), z pętlami (`for`), z wartością zwracaną (`return`). Tam, gdzie alias się kończy — bo potrzebujesz czegoś w środku komendy, nie tylko na końcu — zaczyna się funkcja.

Analogia

Alias to **klawisz skrótu w edytorze tekstu** — wciskasz `Ctrl+B`, edytor wstawia w tym miejscu z góry ustalony fragment. Działa świetnie, dopóki potrzebujesz zawsze dokładnie tego samego fragmentu w tym samym miejscu. Ale klawisz skrótu nie umie zapytać „jaki plik dzisiaj?" — nie ma pojęcia o parametrach, dostaje tylko sygnał „wciśnięto" i odtwarza to, co ma zapisane, bez żadnej zmiany.

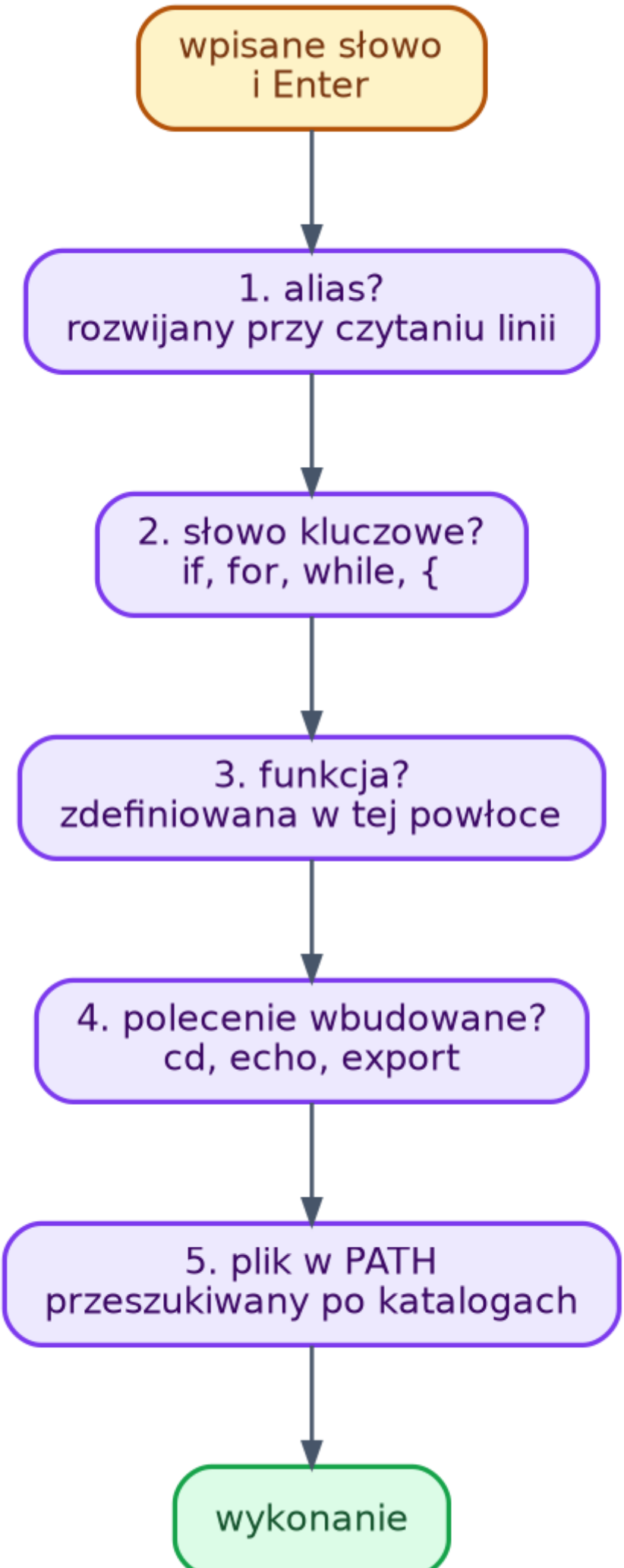
Funkcja powłoki to **makro w arkuszu kalkulacyjnym**, które przyjmuje argumenty: zaznaczasz komórkę, wpisujesz `=PRZELICZ(A1; B1)`, a makro robi z tymi dwiema wartościami dokładnie to, co zaprogramowałeś — może je dodać, pomnożyć, sprawdzić warunek i zwrócić różne wyniki zależnie od tego, co dostało na wejściu. Alias nie ma miejsca na `A1` i `B1` w środku — ma tylko gotowy tekst, który zawsze wygląda tak samo, niezależnie od tego, co dokleisz na końcu.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: kolejność rozstrzygania nazw.** Kiedy wpisujesz słowo i wciskasz Enter, bash musi zdecydować, czym to słowo jest — i sprawdza to w ustalonej kolejności, zatrzymując się na pierwszym trafieniu:

1. **Alias** — rozwijany najwcześniej, na etapie czytania linii, jeszcze zanim bash zacznie cokolwiek wykonywać (i tylko w powłoce interaktywnej, chyba że włączysz `shopt -s expand_aliases`).
2. **Słowo kluczowe** powłoki (`if`, `for`, `while`, `{` — składnia samego basha).
3. **Funkcja** — jeśli zdefiniowałeś funkcję o tej nazwie.
4. **Polecenie wbudowane** (`cd`, `echo`, `export` — wbudowane bezpośrednio w bash, bez osobnego procesu).
5. **Plik wykonywalny** znaleziony przeszukaniem katalogów z `PATH`, w kolejności, w jakiej tam występują.


wpisane słowo
i Enter
1. alias?
rozwijany przy czytaniu linii
2. słowo kluczowe?
if, for, while, {
3. funkcja?
zdefiniowana w tej powłoce
4. polecenie wbudowane?
cd, echo, export
5. plik w PATH
przeszukiwany po katalogach
wykonanie

□□ *Bash rozstrzyga nazwę w stałej kolejności: alias, słowo kluczowe, funkcja, polecenie wbudowane, plik z PATH (kliknięcie otwiera pełny rozmiar)*

`type -a` z Fiszki wypisuje wyniki dokładnie w tej kolejności — jeśli coś jest jednocześnie aliasem i plikiem w `PATH`, alias zawsze wygrywa pierwszy, chyba że jawnie go ominiesz przez `\nazwa` albo `command nazwa`.

Ta kolejność tłumaczy, dlaczego **alias podmienia tylko pierwsze słowo linii i nie przyjmuje argumentów w środku**. Bash rozwija alias w momencie, gdy widzi go jako pierwsze słowo polecenia — wszystko, co następuje po nim, zostaje po prostu doklejone za rozwiniętym tekstem, tak jak dokleiłbyś to ręcznie. Nie ma mechanizmu, który pozwoliłby wstawić argument w środek zdefiniowanego tekstu aliasu ani użyć go jako drugie czy trzecie słowo w linii (drugie słowo aliasu domyślnie w ogóle nie jest sprawdzane pod kątem aliasów, chyba że pierwszy alias kończy się spacją — szczegół, na którym opiera się m.in. trik `alias sudo='sudo '`, pozwalający aliasom działać także po `sudo`). Kiedy potrzebujesz argumentu gdziekolwiek indziej niż na samym końcu — funkcja jest jedynym narzędziem, które to potrafi.

Dlaczego aliasy nie działają w skryptach nieinteraktywnych. Rozwijanie aliasów jest domyślnie wyłączone, gdy bash nie jest interaktywny — czyli w każdym skrypcie uruchomionym jako `./skrypt.sh` albo `bash skrypt.sh`. To jest jeden z najczęstszych powodów zjawiska „działa w terminalu, nie działa w skrypcie": w terminalu masz włączone rozwijanie aliasów (bo terminal jest interaktywny), skopiowany do skryptu ten sam alias milczy albo rzuca `command not found`, bo skrypt jest nieinteraktywny. `shopt -s expand_aliases` potrafi to obejść, ale ma własną pułapkę: bash czyta cały skrypt i rozpoznaje aliasy dopiero w miarę czytania kolejnych linii, więc alias zdefiniowany w tej samej linii co jego pierwsze użycie (albo w tym samym bloku poleceń oddzielonym `;`) jeszcze nie zadziała. Funkcje nie mają żadnego z tych ograniczeń — działają identycznie w interaktywnej powłoce i w skrypcie, dlatego każda dokumentacja basha poleca funkcję jako właściwe rozwiązanie tam, gdzie alias by się nie sprawdził.

Krok po kroku

1. Zobacz, co masz już zdefiniowane.

```
alias
declare -F
```

Zanim dopiszesz coś nowego do `.bashrc`, sprawdź, czy przypadkiem dana nazwa nie jest już zajęta przez dystrybucyjny alias (np. `ls`, `grep`, `cp`, `mv`, `rm` są aliasowane domyślnie na wielu systemach).

2. Sprawdź, czym naprawdę jest podejrzana nazwa.

```
type -a cp
```

```
cp is aliased to `cp -i'
cp is /usr/bin/cp
```

Zanim zaczniesz debugować „dlaczego `cp` pyta mnie o potwierdzenie", `type -a` odpowie w jednej linii.

3. Zamień powtarzalną komendę na alias, jeśli argument zawsze ląduje na końcu.

```
alias dmesgerr='dmesg --level=err,warn --ctime'
```

4. Zamień na funkcję, jeśli argument musi wejść w środek.

```
mkcd() {
    mkdir -p "$1" && cd "$1"
}
```

`mkcd nowy_projekt` tworzy katalog i od razu do niego wchodzi — nie da się tego zrobić aliasem, bo `cd` musi dostać ten sam argument co `mkdir`, a alias nie ma jak powtórzyć argumentu dwa razy.

5. Zapisz na stałe i przeładuj.

```
cp ~/.bashrc ~/.bashrc.bak.$(date +%F)
cat >> ~/.bash_aliases <<'EOF'
alias dmesgerr='dmesg --level=err,warn --ctime'
EOF
source ~/.bashrc
```

Częste błędy

Alias `rm='rm -i'` jako fałszywe poczucie bezpieczeństwa.

```
$ rm plik.txt
rm: remove regular file 'plik.txt'?
```

Administrator przyzwyczaja się, że `rm` zawsze pyta o potwierdzenie, bo na swojej stacji roboczej ma taki alias od lat. Loguje się na świeżo postawiony serwer, wpisuje `rm -rf /var/log/stara-appka/*` z odruchu „i tak zapyta, jeśli coś pójdzie nie tak" — a tam żadnego aliasu nie ma, `rm` usuwa bez pytania. Naprawa: nie polegaj na aliasie jako mechanizmie bezpieczeństwa; jeśli chcesz nawyku

potwierdzania, ustaw go świadomie wszędzie, gdzie pracujesz, i traktuj brak potwierdzenia jako sygnał ostrzegawczy, a nie normę.

Alias skopiowany do skryptu milczy albo się wywala.

```
#!/bin/bash
alias log='echo "[$(date +%T)]"'
log "start"
```

```
skrypt.sh: line 3: log: command not found
```

Skrypt jest powłoką nieinteraktywną — rozwijanie aliasów jest w niej domyślnie wyłączone, więc druga linia nigdy nie zadziała, mimo że dokładnie ten sam alias działa bez zarzutu w terminalu. Naprawa: zamień alias na funkcję — funkcje działają identycznie wszędzie.

```
#!/bin/bash
log() { echo "[$(date +%T)] $*"; }
log "start"
```

Próba przekazania argumentu w środek aliasu.

```
alias szukaj='grep -rn'
szukaj "TODO" src/ --include='*.py'
```

Działa, bo `--include` doklejasz na końcu — ale spróbuj alias `find_pid='ps aux | grep'` i wywołanie `find_pid nginx | head`: `head` też doklei się na końcu całej linii, a nie w miejscu, w którym być może chciałeś je wstawić. Skoro dowolny argument aliasu zawsze ląduje na samym końcu wygenerowanej linii, każda próba wstawienia czegoś „w środku" po prostu nie ma się jak wydarzyć. Naprawa: funkcja z jawnym `$1`, `$2` w dowolnym miejscu ciała.

`unset` bez `-f` kasuje zmienną zamiast funkcji.

```
backup=/mnt/backup
backup() { cp -v "$1" "$1.bak"; }
unset backup
backup /etc/hosts
```

```
'/etc/hosts' -> '/etc/hosts.bak'
```

Wygląda, jakby zadziałało — ale to przypadek: `unset` bez flagi najpierw szuka zmiennej o tej nazwie i ją kasuje, funkcja `backup` przeżywa. Gdyby chodziło o skasowanie funkcji, trzeba jawnie: `unset -f backup`.

Oczekiwanie, że `export -f` uczyni funkcję widoczną dla `cron` albo `sh`.

```
export -f moja_funkcja
```

Zmienna crontaba czy jednostka systemd uruchamia polecenia przez `/bin/sh` (często `dash`, nie bash) albo przez zupełnie nową powłokę bez odziedziczonego środowiska interaktywnego — mechanizm `export -f` działa tylko między procesami basha i tylko wtedy, gdy proces potomny też jest bashem. Naprawa: dla zadań cyklicznych trzymaj logikę w osobnym skrypcie z shebangiem `#!/bin/bash`, wywoływanym z crontaba pełną ścieżką, zamiast liczyć na odziedziczone funkcje.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie. Przed edycją `.bashrc` zrób kopię: `cp ~/.bashrc ~/.bashrc.bak`.

Ćwiczenie 1 — alias kontra funkcja na tym samym przykładzie

```
alias policz='wc -l'
policz /etc/passwd

liczba_linii() {
    wc -l < "$1"
}
liczba_linii /etc/passwd
```

Obie wersje na pozorne to samo. Spróbuj teraz użyć wyniku w arytmetyce: `echo "Masz $(policz /etc/passwd) linii"` kontra `echo "Masz $(liczba_linii /etc/passwd) linii"`. Który wariant wypisze samą liczbę, a który doklei nazwę pliku?

Odpowiedź

`policz /etc/passwd` rozwija się do `wc -l /etc/passwd`, a `wc -l` z podaną nazwą pliku dopisuje ją obok liczby linii — dostaniesz coś w rodzaju `Masz 47 /etc/passwd linii`. `liczba_linii` używa przekierowania `<`, więc `wc` dostaje dane na wejściu, nie jako argument, i nie zna nazwy pliku — wynik to czysta liczba: `Masz 47 linii`. To pokazuje różnicę praktyczną, nie tylko teoretyczną:

funkcja może ukryć szczegóły implementacji (przekierowanie) w sposób, jakiego alias nigdy nie osiągnie.

Ćwiczenie 2 — pułapka aliasu w skrypcie

```
cd /tmp
cat > test_alias.sh <<'EOF'
#!/bin/bash
alias pow='echo Powiadomienie:'
pow "test"
EOF
chmod +x test_alias.sh
./test_alias.sh
```

Przewidź wynik, zanim uruchomisz. Potem popraw skrypt, zamieniając alias na funkcję, i porównaj.

Odpowiedź

```
./test_alias.sh: line 3: pow: command not found
```

Skrypt jest powłoką nieinteraktywną, więc alias `pow` nigdy się nie rozwija — bash próbuje uruchomić program o nazwie `pow`, którego nie ma. Poprawka:

```
#!/bin/bash
pow() { echo "Powiadomienie: $*"; }
pow "test"
```

Ta wersja działa bez żadnych dodatkowych `shopt`.

Pytania kontrolne

1. Masz alias `alias gs='git status'`. Czy zadziała polecenie `gs -sb`? A gdyby chodziło o wstawienie flagi zaraz po `git`, przed `status`?

Odpowiedź

`gs -sb` zadziała, bo `-sb` doklei się na końcu rozwiniętej linii: `git status -sb`. Wstawienie flagi *przed* `status` (np. `git -c core.pager=cat status`) nie jest możliwe przez ten alias — alias nie ma pojęcia o „środku" swojej treści, może być tylko rozszerzany na końcu. Do tego potrzebna byłaby funkcja z jawnym `$1`, `$2` w odpowiednim miejscu.

2. Dlaczego `type -a` jest bardziej wiarygodne niż samo przypomnienie sobie, co się kiedyś zaaliasowało?

Odpowiedź

Bo pokazuje stan faktyczny bieżącej powłoki, a nie to, co pamiętasz z któregoś `.bashrc`. Alias mógł zostać nadpisany, usunięty przez `unalias`, albo w ogóle nie zostać wczytany (np. bo jesteś w powłoce nielogowania, a definicja siedziała tylko w pliku logowania). `type -a` odpowiada na pytanie „co się stanie, jeśli teraz wpiszę tę nazwę", a nie „co powinno się stać".

3. Napisałeś funkcję `ls() { command ls -F "$@"; }`. Po co w jej wnętrzu jest `command`?

Odpowiedź

Bez `command` wywołanie `ls` wewnątrz funkcji o nazwie `ls` odnalazłoby samą siebie (funkcje mają pierwszeństwo przed plikami z `PATH`) i wpadłoby w nieskończoną rekurencję, aż bash przerwałby ją błędem przepełnienia stosu. `command ls` każe bashowi pominąć zarówno tę funkcję, jak i ewentualny alias o nazwie `ls`, i sięgnąć od razu po prawdziwy plik wykonywalny `/usr/bin/ls`.

4. Masz jednocześnie `alias grep='grep --color=auto'` i chcesz w skrypcie użyć `grep` bez kolorowania, bez usuwania aliasu. Jak to zrobić najprościej, nie dotykając `.bashrc`?

Odpowiedź

Aliasy i tak nie rozwijają się domyślnie w skryptach nieinteraktywnych, więc w praktyce zwykły `grep` w skrypcie już pomija ten alias — problem dotyczy tylko pracy w interaktywnym terminalu. Tam najszybciej: `\grep wzorzec plik` albo `command grep wzorzec plik`, oba pomijają rozwinięcie aliasu bez jego kasowania.

Powiązane: Pliki startowe powłoki: co, gdzie i kiedy się wczytuje · Prompt: PS1, kolory, kontekst · Cudzysłowy, escapowanie i ekspansja · Zmienne środowiskowe i PATH

Źródła: `man 1 bash` (sekcje ALIASES, SHELL FUNCTIONS, SHELL GRAMMAR) · `help declare` · `help unset` · notatki własne (sekcja „ALIASY UNIWERSALNE" — zachowana we fragmentach, zestaw poniżej uzupełniony o standardowe idiomy administracyjne)

Revision #15
Created 2026-09-18 06:57:14 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:53 UTC by Tomasz Konieczny