Pliki startowe powłoki: co, gdzie i kiedy się wczytuje

Poziom: podstawy **Dotyczy: RHEL/Rocky 9–10, różnice Debian/Ubuntu zaznaczone**

Czas: ~18 min

Każde uruchomienie basha czyta zestaw plików, zanim pokaże Ci znak zachęty — ale nie zawsze ten sam zestaw. Ta strona odpowiada na pytanie, które zadaje sobie każdy, kto raz zmienił coś w `.bashrc` i „to nie działa": który plik w ogóle się wczytał i dlaczego akurat ten.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
PAM	Pluggable Authentication Modules: wymienne moduły logowania w Linuksie, decydują na przykład o sile wymaganego hasła
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

Odczyt pliku `~/.bashrc`

```
cat ~/.bashrc
```

Plik czytany przy **każdym** uruchomieniu interaktywnej powłoki nielogowania — czyli w praktyce przy każdym nowym oknie terminala i każdym ręcznym wpisaniu `bash`. Tu trafiają aliasy, funkcje, `PS1`, historia, `shopt`.

Przykład:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

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

export HISTSIZE=5000
export PATH="$PATH:$HOME/.local/bin"
```

PS1 = zmienna powłoki z wyglądem znaku zachęty (promptu), nie ma nic wspólnego z PowerShellem mimo podobnej nazwy · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Odczyt pliku `~/.bash_profile`

```
cat ~/.bash_profile
```

Plik czytany przy sesji **logowania** — jednorazowo, zanim cokolwiek innego się uruchomi. Domyślna zawartość na RHEL/Rocky niemal zawsze dogrywa `.bashrc`, żeby aliasy i funkcje działały też w powłoce logowania.

Przykład:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/.local/bin:$HOME/bin
export PATH
```

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Odczyt pliku `/etc/profile`

```
cat /etc/profile
```

Ogólnosystemowy odpowiednik `.bash_profile` — czytany dla **każdego** użytkownika przy sesji logowania, przed jego prywatnym plikiem. Ustawia `PATH` bazowy, `umask`, i — co najważniejsze — pętlą dogrywa wszystko z `/etc/profile.d/`.

Przykład:

```
# /etc/profile

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL

if [ -d /etc/profile.d ]; then
    for i in /etc/profile.d/*.sh; do
        [ -r "$i" ] && . "$i"
    done
    unset i
fi
```

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Wylistowanie zawartości `/etc/profile.d/`

```
ls /etc/profile.d/
```

Katalog, w który pakiety — i my sami — wrzucają osobne pliczki `.sh` z ustawieniami dla wszystkich użytkowników. `/etc/profile` czyta je wszystkie po kolei, w porządku alfabetycznym.

Przykład:

```
bash_completion.sh
colorgrep.sh
lang.sh
less.sh
```

```
vim.sh
vte.sh
```

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Odczyt systemowego pliku rc: `/etc/bashrc` (RHEL/Rocky) / `/etc/bash.bashrc` (Debian/Ubuntu)

```
cat /etc/bashrc          # RHEL, Rocky, Fedora
cat /etc/bash.bashrc     # Debian, Ubuntu
```

Ogólnosystemowy odpowiednik `.bashrc` — ustawienia interaktywne (prompt, `umask`, aliasy) wspólne dla wszystkich. Sam z siebie nie jest czytany przez basha automatycznie; to `~/.bashrc` (patrz pierwsza fiszka) go dociąga liniami `if [ -f /etc/bashrc ]; then . /etc/bashrc; fi`. Nazwa pliku to jedyna realna różnica między dystrybucjami — treść i rola są identyczne.

Przykład:

```
# /etc/bashrc (Rocky Linux 9)

if [ -z "$PS1" ]; then
   return
fi

umask 022
```

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Ponowne wczytanie `~/.bashrc` poleceniem `source`

```
source ~/.bashrc
```

Wczytuje plik ponownie w **bieżącej** powłoce, bez otwierania nowego terminala. `source` to wbudowane polecenie basha — czyta plik i wykonuje go tak, jakby jego zawartość była wpisana ręcznie, w tej samej powłoce (żaden nowy proces nie powstaje).

□□ **source** = wykonaj plik w bieżącej powłoce zamiast w nowej, zmienne z niego zostają dostępne dalej

□□ TYLKO ODCZYT

Ponowne wczytanie `~/.bashrc` kropką (POSIX)

```
. ~/.bashrc
```

Dokładnie to samo co `source ~/.bashrc` — kropka to POSIX-owy skrót, `source` to alias bashowy dla wygody. W skryptach przenośnych (`#!/bin/sh`) działa tylko kropka, bo `source` bywa niedostępne poza bashem.

□□ **POSIX** = Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach

□□ TYLKO ODCZYT

Wymuszenie sesji logowania: `bash -l`

```
bash -l
```

Uruchamia nową powłokę i **wymusza** tryb logowania, niezależnie od tego, jak została wywołana. Przydaje się, kiedy chcemy sprawdzić, czy problem leży w plikach logowania (`/etc/profile`, `~/.bash_profile`), nie musząc się wylogowywać i logować ponownie.

□□ **bash** = Bourne Again Shell: uruchom powłokę bash albo wskazany w niej skrypt

□□ TYLKO ODCZYT

Sprawdzenie, czy powłoka jest sesją logowania: `shopt login_shell`

```
shopt login_shell
```

Pyta basha wprost, czy bieżąca powłoka jest powłoką logowania. Najpewniejszy sposób sprawdzenia tego bez zgadywania po tym, które pliki się wczytały.

Przykład:

```
login_shell     on
```

□□ **shopt** = shell options: włącz albo wyłącz dodatkowe opcje zachowania powłoki bash

🔒 TYLKO ODCZYT

Sprawdzenie flag bieżącej powłoki: `echo $-`

```
echo $-
```

Wypisuje flagi bieżącej powłoki jednym ciągiem znaków. Litera `i` w wyniku oznacza powłokę **interaktywną** — to jedyny pewny sposób odróżnienia interaktywnej sesji od skryptu z poziomu samego basha.

Przykład:

```
himBHs
```

W tym przykładzie `i` jest obecne (interaktywna), `h` to śledzenie lokalizacji poleceń, `m` to kontrola zadań, `B` i `H` dotyczą rozwijania nawiasów klamrowych i historii, `s` oznacza czytanie ze standardowego wejścia.

🔒 TYLKO ODCZYT

Debugowanie pliku startowego: `bash -x ~/.bashrc`

```
bash -x ~/.bashrc
```

Uruchamia plik jako skrypt z włączonym śledzeniem wykonania (`-x` to skrót od `set -x`) — każda linia jest wypisywana na ekran, zanim się wykona, z prefiksem `+`. Jedyny pewny sposób, żeby zobaczyć, w którym miejscu `.bashrc` „się wysypuje" albo co dokładnie ustawia.

Przykład:

```
+ '[' -f /etc/bashrc ']'
+ . /etc/bashrc
++ '[' -z '' ']'
++ return
+ alias 'll=ls -alh'
+ alias 'grep=grep --color=auto'
+ export HISTSIZE=5000
+ HISTSIZE=5000
```

bash = Bourne Again Shell: uruchom powłokę bash albo wskazany w niej skrypt

TYLKO ODCZYT

Skrypt wykonywany przy wylogowaniu: `~/.bash_logout`

```
cat ~/.bash_logout
```

Odwrotność `.bash_profile` — wykonywany raz, przy **kończeniu** interaktywnej powłoki logowania (wylogowanie, zamknięcie terminala pracującego jako login shell). Typowe zastosowanie: czyszczenie ekranu, kasowanie plików tymczasowych.

Przykład:

```
# ~/.bash_logout


if [ "$SHLVL" = 1 ]; then
    [ -x /usr/bin/clear_console ] && /usr/bin/clear_console -q
fi
```

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Ustawienia środowiskowe niezależne od powłoki: `/etc/environment`

```
cat /etc/environment
```

To nie jest skrypt powłoki. Format to proste linie `NAZWA=wartość`, bez `export`, bez cudzysłowów interpretowanych przez basha, bez podstawiania zmiennych (`$PATH` w tym pliku to dosłowny tekst „`$PATH`", nie wartość zmiennej). Plik czyta moduł PAM (`pam_env`) na długo przed uruchomieniem jakiejkolwiek powłoki — dlatego działa też dla sesji graficznych i dla usług uruchamianych przez PAM, nie tylko dla basha.

Przykład:

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
LANG=pl_PL.UTF-8
```

PAM = Pluggable Authentication Modules: wymienne moduły logowania w Linuksie, decydują na przykład o sile wymaganego hasła · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Plik logowania na Debianie/Ubuntu: `~/.profile`

```
cat ~/.profile
```

Na Debianie i Ubuntu domyślny użytkownik nie dostaje `~/.bash_profile` — zamiast tego jego rolę pełni `~/.profile`, plik z listy `~/.bash_profile` → `~/.bash_login` → `~/.profile`, którą bash sprawdza po kolei i bierze **pierwszy istniejący**. Ma tę zaletę, że jest czytany też przez powłoki inne niż bash (np. `dash` jako `/bin/sh`), więc jego składnia powinna trzymać się POSIX , bez bashizmów.

Przykład:

```
# ~/.profile: executed by the command interpreter for login shells.

if [ -n "$BASH_VERSION" ]; then
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

POSIX = Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach · **cat** = concatenate: sklej i wypisz pliki

ZMIENIA SYSTEM

Dodanie własnego pliku: `/etc/profile.d/NazwaPliku.sh`

```
sudo tee /etc/profile.d/firma-env.sh > /dev/null <<'EOF'
export EDITOR=vim
export HISTTIMEFORMAT="%F %T "
EOF
sudo chmod 644 /etc/profile.d/firma-env.sh
```

Sposób na dodanie zmiennej **dla wszystkich użytkowników**, bez ryzyka, że aktualizacja pakietu `bash` nadpisze naszą zmianę. Plik musi kończyć się na `.sh` (pętla w `/etc/profile` filtruje właśnie po tym rozszerzeniu) i być czytelny — nie musi być wykonywalny, bo `/etc/profile` czyta go przez `.`, a nie uruchamia jak program.

tee = zapisz strumień wejścia do pliku i jednocześnie wypisz go na ekran · **export** = ustaw zmienną środowiskową widoczną dla uruchamianych programów · **chmod** = change mode: zmień prawa dostępu

TYLKO ODCZYT

Wszystkie pliki

Plik	Kto czyta	Kiedy
`/etc/profile`	wszyscy użytkownicy	sesja logowania
`/etc/profile.d/*.sh`	wszyscy użytkownicy	dociągane przez `/etc/profile`
`~/.bash_profile`	dany użytkownik	sesja logowania (bash)
`~/.bash_login`	dany użytkownik	logowanie, jeśli brak `.bash_profile`
`~/.profile`	dany użytkownik	logowanie, jeśli brak obu powyższych; jedyny na Debianie/Ubuntu
`~/.bashrc`	dany użytkownik	każda interaktywna powłoka nielogowania
`/etc/bashrc` (RHEL) / `/etc/bash.bashrc` (Debian)	wszyscy użytkownicy	dociągane przez `~/.bashrc`
`~/.bash_logout`	dany użytkownik	koniec powłoki logowania
`/etc/environment`	wszystkie sesje (PAM)	logowanie do systemu, niezależnie od powłoki

Uważaj

- `/etc/environment` nie przyjmuje żadnej składni powłoki — `export`, `$ZMIENNA`, cudzysłowy z podstawianiem nic tu nie zdziałają.
- Nazwa pliku bez `.sh` w `/etc/profile.d/` zostanie po cichu zignorowana przez pętlę w `/etc/profile`.
- Zmiana w `~/.bashrc` nie dotrze do zadania crona ani do usługi systemd — obie uruchamiają powłokę nielogowania i nieinteraktywną (patrz *Częste błędy*).

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **PAM** = Pluggable Authentication Modules: wymienne moduły logowania w Linuksie, decydują na przykład o sile wymaganego hasła

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> Wtorek, 9:00: Tomek dopisuje `EDITOR=vim` prosto do `/etc/profile` na app01, bo tak było najszybciej. Miesiąc później `dnf update bash` podmienia ten plik na domyślny z pakietu i zmienna znika bez śladu, a nikt nie pamięta, że w ogóle była ustawiona ręcznie. Dopiero porównanie z kopią sprzed aktualizacji pokazuje, że winny jest sam plik, nie coś w środowisku Tomka.

Po co to istnieje

> **Pojęcie: plik startowy.** Skrypt powłoki, który bash wczytuje automatycznie — bez Twojego udziału — w konkretnym momencie cyklu życia sesji: przy logowaniu, przy starcie nowej interaktywnej powłoki albo przy wylogowaniu. Nie każdy plik startowy jest czytany w każdej sytuacji; właśnie to rozróżnienie jest tematem tej strony.

Problem, który te pliki rozwiązują, jest prosty: chcesz, żeby Twój `PATH`, Twoje aliasy, Twój kolorowy prompt i Twoje zmienne środowiskowe były gotowe, zanim zaczniesz pracować — bez ręcznego wpisywania tych samych czterdziestu linii po każdym zalogowaniu. Bash daje Ci na to kilka „haczyków" w swoim cyklu startowym, a administrator systemu dostaje dodatkowe haczyki ogólnosystemowe, żeby ustawić rozsądne domyślne wartości dla wszystkich kont naraz.

Kłopot zaczyna się, gdy ktoś zakłada, że wszystkie te haczyki działają zawsze — i dziwi się, że zmienna ustawiona w `.bashrc` jest niewidoczna w skrypcie cyklicznym albo że kolorowy prompt zniknął po `su` bez myślnika. To nie jest błąd basha; to konsekwencja tego, że bash od dekad rozróżnia **cztery różne sytuacje startowe**, z których każda czyta inny zestaw plików.

> **Pojęcie: sesja logowania kontra sesja nielogowania.** Sesja **logowania** (*login shell*) to powłoka, która symuluje pełne zalogowanie się do systemu — konsola tekstowa po wpisaniu loginu i hasła, `ssh użytkownik@host` bez podanej komendy, `su - użytkownik` (z myślnikiem), albo jawne żądanie flagą `bash -l` czy `bash --login`. Jej zadaniem jest przygotować **całe środowisko od zera**: `PATH`, zmienne, umask, komunikat dnia. Sesja **nielogowania** (*non-login shell*) to każda kolejna powłoka uruchomiona już wewnątrz istniejącej sesji — nowe okno terminala, wpisanie `bash` w istniejącej powłoce, `su użytkownik` (bez myślnika). Zakłada ona, że środowisko systemowe już istnieje, i dogrywa tylko rzeczy specyficzne dla interaktywnej pracy: aliasy, funkcje, prompt.

Analogia

Wyobraź sobie **biurowiec z recepcją i długim korytarzem pełnym drzwi wewnętrznych**.

Kiedy wchodzisz z ulicy przez główne wejście, przechodzisz przez recepcję: pokazujesz przepustkę, dostajesz identyfikator gościa, ochroniarz informuje Cię o zasadach budynku obowiązujących dziś (remont windy, zmiana godzin pracy stołówki). To dzieje się **raz** — przy wejściu. Recepcja to `/etc/profile` i `~/.bash_profile`: pełne przygotowanie środowiska, wykonywane jednorazowo na starcie sesji logowania.

Będąc już w środku, przechodzisz między pokojami przez drzwi wewnętrzne — nie wracasz do recepcji za każdym razem, kiedy zmieniasz salę konferencyjną. Ale przy każdych takich drzwiach wisi mała tabliczka z bieżącymi ogłoszeniami działu: „drukarka na tym piętrze nie działa", „spotkanie przesunięte na 15:00". Czytasz ją za każdym razem, kiedy otwierasz nowe drzwi. To jest `~/.bashrc` — czytany przy **każdej** nowej interaktywnej powłoce nielogowania, choć budynek (system) już dawno działa.

A teraz wyobraź sobie automatycznego robota kurierskiego, który w nocy rozwozi paczki między działami według z góry zaprogramowanej trasy. Nie wchodzi przez recepcję — ma własne, techniczne wejście dla zaopatrzenia. Nie czyta tabliczek na drzwiach wewnętrznych — jego trasa jest zapisana w innym systemie i nikt jej nie aktualizuje ogłoszeniami z korytarza. Dokładnie tak zachowują się cron i systemd: uruchamiają powłokę, która nie jest ani sesją logowania, ani sesją interaktywną, więc żadna tabliczka na korytarzu (`~/.bashrc`) ani rozmowa z ochroniarzem (`/etc/profile`) do niej nie dotrze.

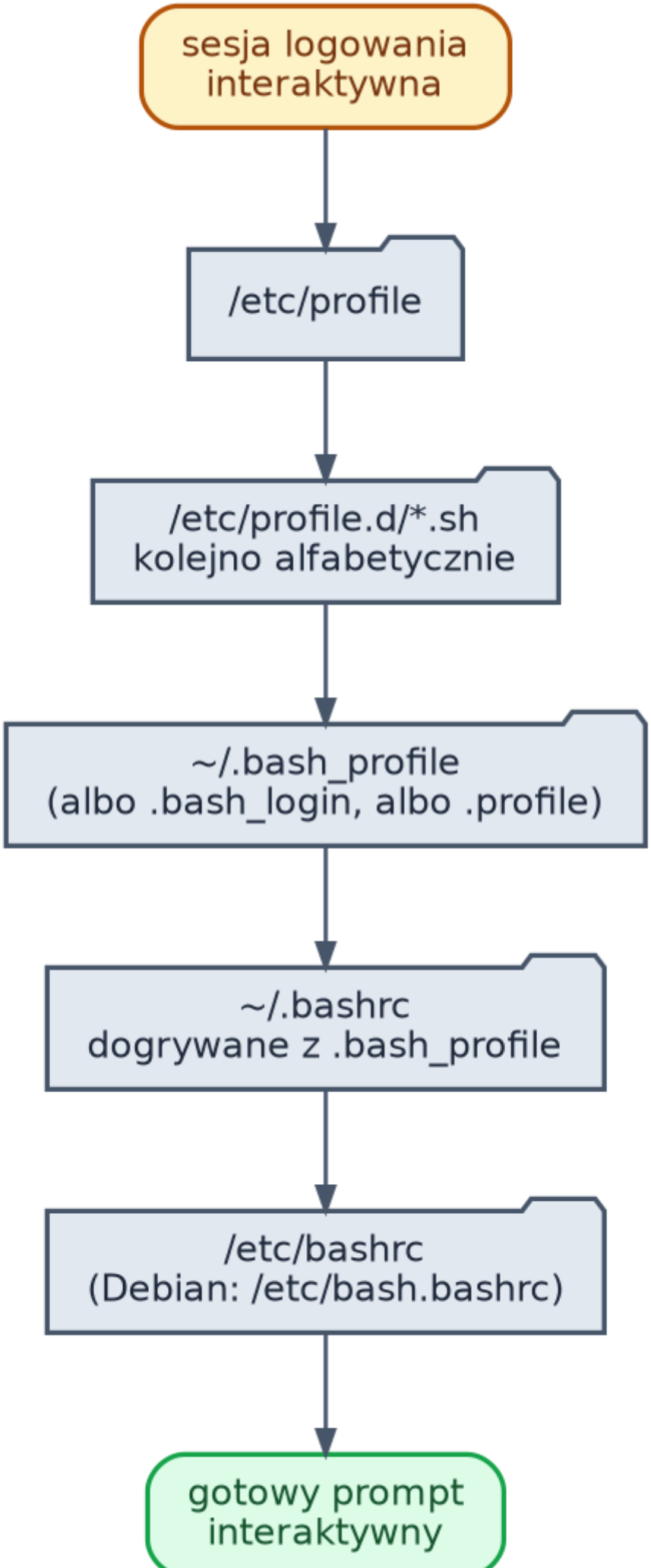
Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: powłoka interaktywna kontra nieinteraktywna.** Powłoka **interaktywna** to taka, która czeka na Twoje wpisywanie poleceń i pokazuje znak zachęty (`$` albo `#`) — terminal, w którym pracujesz na co dzień. Powłoka **nieinteraktywna** wykonuje z góry zapisany ciąg poleceń bez pytania nikogo o zdanie — skrypt uruchomiony jako `./skrypt.sh`, polecenie w crontabie, `ExecStart=` w jednostce systemd. Bash pozna, którą jest, patrząc, czy jego standardowe wejście i wyjście są podłączone do terminala (i po obecności flagi `-i`) — a Ty poznasz to samo poleceniem `echo $-` z Fiszki: litera `i` w wyniku znaczy „interaktywna".

> **Pojęcie: plik profilu kontra plik rc. Plik profilu** (`/etc/profile`, `~/.bash_profile`, `~/.profile`) jest czytany **tylko raz**, na starcie sesji logowania — jego zadaniem jest zbudować środowisko: `PATH`, zmienne eksportowane, umask, komunikaty powitalne. **Plik rc** (od *run commands* — `~/.bashrc`, `/etc/bashrc`) jest czytany przy **każdym** starcie interaktywnej powłoki, logowania czy nie — jego zadaniem są rzeczy specyficzne dla interaktywnej pracy w danym oknie: aliasy, funkcje, prompt, historia. Zmienne eksportowane (`export ZMIENNA=wartość`) dziedziczą procesy potomne niezależnie od tego, w którym pliku je ustawisz — różnica dotyczy tylko *kiedy* dany plik się wykona, nie tego, jak działa `export` (pełny mechanizm dziedziczenia zmiennych opisuje strona o zmiennych środowiskowych i `PATH`).

Te dwa rozróżnienia — logowanie/nielogowania i interaktywna/nieinteraktywna — są od siebie niezależne, co daje **cztery kombinacje**. To jest sedno tej strony:


sesja logowania
interaktywna
/etc/profile
/etc/profile.d/*.sh
kolejno alfabetycznie
~/.bash_profile
(albo .bash_login, albo .profile)
~/.bashrc
dogrywane z .bash_profile
/etc/bashrc
(Debian: /etc/bash.bashrc)
gotowy prompt
interaktywny

Kolejność wczytywania plików startowych dla sesji logowania i interaktywnej, od /etc/profile do gotowego promptu (kliknięcie otwiera pełny rozmiar)

Kombinacja	Typowy przykład	Pliki wczytywane, w tej kolejności
Logowanie + interaktywna	konsola tekstowa, `ssh user@host` bez komendy, `su - user`, `bash -l`	`/etc/profile` (który pętlą dogrywa `/etc/profile.d/*.sh`) → pierwszy istniejący z: `~/.bash_profile`, `~/.bash_login`, `~/.profile`
Logowanie + nieinteraktywna	`bash --login -c "polecenie"`, `bash --login skrypt.sh`	dokładnie to samo co wyżej — o wczytaniu plików logowania decyduje **flaga** `--login`, nie interaktywność; ta kombinacja jest rzadka i zwykle celowa
Nielogowania + interaktywna	nowe okno terminala w środowisku graficznym, ręczne wpisanie `bash` w istniejącej sesji	tylko `~/.bashrc` — który zwykle sam dogrywa `/etc/bashrc` (RHEL/Rocky) albo `/etc/bash.bashrc` (Debian/Ubuntu)
Nielogowania + nieinteraktywna	`./skrypt.sh`, polecenie w crontabie, `ExecStart=` w jednostce systemd	**żaden** z powyższych plików — jedyny wyjątek to zmienna `BASH_ENV` : jeśli jest ustawiona, bash wczytuje wskazany przez nią plik nawet tutaj

Skąd wynika popularny wpis `[ -f ~/.bashrc ] && . ~/.bashrc` w `~/.bash_profile`? Wprost z pierwszego wiersza tabeli: sesja logowania **nie czyta** `~/.bashrc` sama z siebie — te dwa pliki obsługują dwie osobne kategorie zdarzeń. Bez tej jednej linii aliasy i funkcje zdefiniowane w `.bashrc` byłyby niewidoczne zaraz po zalogowaniu się na konsoli czy przez `ssh`, dopóki nie odpaliłbyś ręcznie drugiej powłoki nielogowania (np. nowego okna terminala). Dystrybucje dogrywają więc `.bashrc` z `.bash_profile`, żeby zatrzeć tę różnicę dla zwykłego użytkownika — ale mechanizm bazowy pozostaje dwuwarstwowy.

To samo rozróżnienie tłumaczy, dlaczego **zmienne ustawione w `.bashrc` nie docierają do zadań crona ani do usług systemd**. Ostatni wiersz tabeli — nielogowania i nieinteraktywna — to dokładnie środowisko, w jakim crond i systemd uruchamiają Twoje polecenia. `crond` uruchamia każdą linię crontaba przez `/bin/sh -c`, bez żadnego z plików startowych basha (pełny mechanizm i sposoby obejścia opisuje strona o cronie). Usługa systemd startuje bezpośrednio przez `execve()`, z minimalnym środowiskiem zdefiniowanym w samej jednostce (`Environment=`, `EnvironmentFile=`) — bez powłoki logowania w ogóle, więc `~/.bashrc` czy `/etc/profile` nie wchodzą tu w grę (mechanizm i właściwy sposób przekazania zmiennych do usługi opisuje strona o systemd).

Dlaczego zmiany dla wszystkich użytkowników trafiają do `/etc/profile.d/`, a nie bezpośrednio do `/etc/profile`? Bo `/etc/profile` jest częścią pakietu `bash` (albo `setup` na RHEL) — kolejna aktualizacja pakietu może nadpisać ten plik i bezpowrotnie skasować Twoją ręczną zmianę, bez ostrzeżenia. `/etc/profile.d/` istnieje właśnie po to, żeby dało się dopisywać ustawienia

systemowe bez dotykania plików należących do pakietów — ten sam wzorzec co `/etc/sudoers.d/` czy `/etc/cron.d/`: katalog na drop-iny, których menedżer pakietów nigdy nie ruszy.

Krok po kroku

1. Sprawdź, w jakim typie powłoki właśnie jesteś.

```
shopt login_shell
echo $-
```

```
login_shell    on
himBHs
```

`login_shell on` plus `i` w wyniku `$-` oznacza: sesja logowania i interaktywna jednocześnie — czyli pierwszy wiersz tabeli wyżej.

2. Porównaj z nową powłoką nielogowania.

```
bash
shopt login_shell
echo $-
exit
```

```
login_shell    off
himBHs
```

Ten sam użytkownik, ta sama maszyna — ale `login_shell` teraz `off`. `~/.bash_profile` i `/etc/profile` nie zostały wczytane dla tej powłoki, tylko `~/.bashrc`.

3. Prześledź krok po kroku, co dokładnie się wykonuje.

```
bash -x -l -c 'exit' 2>&1 | head -30
```

Wypisze każdą linię `/etc/profile`, każdego pliku z `/etc/profile.d/` i `~/.bash_profile` w kolejności wykonania, z prefiksem `+`. Nieoceniona metoda, kiedy prompt „gdzieś" traci kolor albo zmienna „gdzieś" znika — szukasz miejsca, w którym coś ją nadpisuje.

4. Dodaj zmienną dla wszystkich użytkowników bezpiecznie.

```
sudo tee /etc/profile.d/firma-editor.sh > /dev/null <<'EOF'
export EDITOR=vim
EOF
```

```
bash -l -c 'echo $EDITOR'
```

```
vim
```

5. Zrób kopię zapasową własnego `.bashrc`, zanim zaczniesz eksperymentować.

```
cp ~/.bashrc ~/.bashrc.bak.$(date +%F)
```

Jedna linia, zero ryzyka. Jeśli coś w edytowanym pliku wywali kolejną powłokę na starcie, wracasz jedną komendą: `cp ~/.bashrc.bak.2026-09-18 ~/.bashrc`.

Częste błędy

Zmienna z `.bashrc` niewidoczna w jednorazowej komendzie przez SSH.

```
$ ssh serwer 'echo $MOJA_ZMIENNA'
```

Pusta linia zamiast wartości. Kiedy podajesz komendę bezpośrednio w wywołaniu `ssh`, zdalna powłoka jest **nieinteraktywna** (nie ma znaku zachęty, nie czeka na Twoje wpisywanie) — `~/.bashrc` w tym trybie się nie wczytuje. Naprawa: ustaw zmienną w `/etc/environment` (czytane przez PAM niezależnie od interaktywności) albo w `~/.ssh/environment` z włączonym `PermitUserEnvironment yes` po stronie serwera.

`PATH` z `.bashrc` „znika" w zadaniu crona.

```
/bin/sh: 1: moj-skrypt-w-home-local-bin: not found
```

Crontab uruchamia polecenia przez minimalną powłokę, bez wczytania `~/.bashrc` — jeśli dodałeś katalog do `PATH` tylko tam, cron go nie widzi. Naprawa: podaj pełną ścieżkę do programu w linii crontaba albo dodaj `PATH=...` bezpośrednio na górze crontaba (`crontab -e`) — szczegóły na stronie o cronie.

Edycja `/etc/profile` zamiast `/etc/profile.d/`, ofiara kolejnej aktualizacji.

Administrator dopisuje ręcznie linię `export EDITOR=vim` na końcu `/etc/profile`. Miesiąc później, po `dnf update bash`, zmienna znika bez śladu, bo pakiet nadpisał plik konfiguracyjny domyślną wersją. Naprawa: to samo ustawienie w osobnym pliku `/etc/profile.d/edytor.sh`, którego menedżer pakietów nigdy nie dotknie.

Próba użycia składni powłoki w `/etc/environment`.

```
PATH=$PATH:/opt/app/bin
```

Po zalogowaniu `echo $PATH` pokazuje dosłowny, niedziałający ciąg `$PATH:/opt/app/bin` — bo PAM nie interpretuje `$PATH` jako podstawienia zmiennej, tylko wpisuje wartość dosłownie. Naprawa: `/etc/environment` przyjmuje tylko literalne wartości; do doklejania czegoś do istniejącego `PATH` służy `/etc/profile.d/*.sh`, który *jest* skryptem powłoki i podstawienia rozumie.

Plik w `/etc/profile.d/` bez rozszerzenia `.sh` — po cichu ignorowany.

```
sudo tee /etc/profile.d/firma-editor > /dev/null <<'EOF'
export EDITOR=vim
EOF
```

Żadnego błędu, żadnego ostrzeżenia — po prostu `$EDITOR` nigdy nie zostanie ustawiony. Pętla w `/etc/profile` filtruje pliki wzorcem `*.sh`; bez tego rozszerzenia plik jest dla niej niewidoczny. Naprawa: dopisz `.sh` do nazwy pliku.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie. Punkty dotyczące `/etc/profile.d/` wymagają `sudo` — reszta nie.

Ćwiczenie 1 — cztery kombinacje na własnej skórze

```
# 1. Twoja bieżąca powłoka
echo "login_shell: $(shopt login_shell | awk '{print $2}'), flagi: $-"

# 2. Nowa powłoka nielogowania
bash -c 'echo "login_shell: $(shopt login_shell | awk "{print \$2}"), flagi: $-"'

# 3. Wymuszona powłoka logowania, nieinteraktywna
bash -l -c 'echo "login_shell: $(shopt login_shell | awk "{print \$2}"), flagi: $-"'
```

Które z trzech wywołań mają literę `i` w polu „flagi"? Które mają `login_shell: on`?

Odpowiedź

Pierwsze wywołanie zwykle ma `i` w flagach — jesteś w interaktywnej sesji (a `login_shell` zależy od tego, jak się zalogowałeś). Drugie (`bash -c`) ma `login_shell: off` i **brak** `i` — nieinteraktywna powłoka nielogowania. Trzecie (`bash -l -c`) ma `login_shell: on`, ale **brak** `i` — rzadka kombinacja „logowanie + nieinteraktywna": flaga `-l` wymusiła pliki logowania, mimo że powłoka nie jest interaktywna.

Ćwiczenie 2 — zmienna, która nie dotrze do crona

```
cd /tmp
echo 'export SEKRETNA_SCIEZKA="/tmp/z-bashrc"' >> ~/.bashrc
bash -c 'echo "Widzę: ${SEKRETNA_SCIEZKA:-BRAK}"'
```

Dlaczego druga komenda wypisuje `BRAK`, mimo że przed chwilą dopisałeś zmienną do `.bashrc`? Popraw plik `~/.bashrc` z powrotem po ćwiczeniu.

Odpowiedź

`bash -c` uruchamia nową powłokę, która jest nieinteraktywna i nielogowania — czwarty wiersz tabeli. `~/.bashrc` w tej kombinacji nie jest czytany, więc zmienna zdefiniowana tylko tam nigdy nie zostanie ustawiona. Dokładnie to samo dzieje się z każdym poleceniem w crontabie. Sprzątanie:

```
sed -i '/SEKRETNA_SCIEZKA/d' ~/.bashrc
```

Ćwiczenie 3 — debugowanie własnego `.bashrc`

```
cp ~/.bashrc ~/.bashrc.bak.$(date +%F)
bash -x ~/.bashrc 2>&1 | tail -20
```

Znajdź w wyniku linię, w której `.bashrc` dogrywa systemowy plik rc (`/etc/bashrc` albo `/etc/bash.bashrc`, zależnie od dystrybucji).

Odpowiedź

Szukana linia to zwykle coś w rodzaju `+ . /etc/bashrc` (RHEL/Rocky) albo `+ . /etc/bash.bashrc` (Debian/Ubuntu), poprzedzona testem `+ '[' -f /etc/bashrc ']'`. Jeśli plik nie istnieje na Twoim systemie, test się nie powiedzie i linia `.` w ogóle się nie wykona — bez błędu, po prostu cicho pominięta.

Pytania kontrolne

1. Otwierasz nowe okno terminala w środowisku graficznym. Które pliki startowe bash wczyta?

Odpowiedź

Tylko `~/.bashrc` (a przez niego pośrednio `/etc/bashrc` lub `/etc/bash.bashrc`). Nowe okno terminala to klasyczna powłoka nielogowania, interaktywna — drugi wiersz od dołu w tabeli czterech kombinacji. Pliki logowania (`/etc/profile`, `~/.bash_profile`) zostały wczytane raz, dawno temu, kiedy zalogowałeś się do sesji graficznej.

2. Dlaczego zmiana `PATH` w `~/.bashrc` nie wystarczy, żeby usługa systemd znalazła Twój program?

Odpowiedź

Usługa systemd startuje bezpośrednio przez jądro, bez żadnej powłoki logowania ani interaktywnej pośrodku — nie ma tu miejsca, w którym `~/.bashrc` mogłoby zostać wczytane. Środowisko procesu usługi pochodzi z jednostki `.service` (`Environment=`, `EnvironmentFile=`) i z ustawień systemowych, nigdy z prywatnych plików startowych basha konkretnego użytkownika.

3. Masz plik `/etc/environment` z linią `JAVA_HOME=$HOME/java`. Co faktycznie znajdzie się w zmiennej `JAVA_HOME` po zalogowaniu?

Odpowiedź

Dosłowny tekst `$HOME/java` — łącznie ze znakiem dolara. `/etc/environment` nie jest skryptem powłoki i nie podstawia wartości zmiennych; PAM traktuje wszystko po znaku `=` jako literalny ciąg znaków. Do podstawień i logiki warunkowej trzeba użyć pliku `/etc/profile.d/*.sh`, który jest prawdziwym skryptem basha.

4. Czym różni się rola pliku profilu od roli pliku rc, skoro oba mogą zawierać `export`?

Odpowiedź

Różnica nie leży w tym, co potrafią zapisać (oba to zwykłe skrypty basha), tylko w tym, *kiedy* są wczytywane. Plik profilu (`/etc/profile`, `~/.bash_profile`) wykonuje się raz, na starcie sesji logowania — stąd trafia tam budowa środowiska: `PATH`, zmienne systemowe, umask. Plik rc (`~/.bashrc`, `/etc/bashrc`) wykonuje się przy każdym starcie interaktywnej powłoki, także wielokrotnie w ramach jednej sesji logowania — stąd trafiają tam rzeczy specyficzne dla pojedynczego okna terminala: aliasy, funkcje, prompt, historia.

Powiązane: Zmienne środowiskowe i PATH · cron i crontab: zadania cykliczne · systemd — jednostki i usługi · Aliasy i funkcje powłoki · Prompt: PS1, kolory, kontekst

Źródła: `man 1 bash` (sekcja INVOCATION) · `man 5 pam_env` · pliki `/etc/profile`, `/etc/bashrc`, `/etc/skel/.bashrc` z RHEL/Rocky 9 i Debian 12

Revision #15
Created 2026-09-18 06:57:12 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:51 UTC by Tomasz Konieczny