Zadania powłoki: jobs, bg, fg, nohup, disown

Poziom: podstawy/średnio **Dotyczy: wszystkie dystrybucje** **Czas: ~14 min**

Zadanie (job) to pojęcie powłoki, nie jądra — dlatego odwołujesz się do niego przez `%1`, a nie przez PID. Ta strona pokazuje, jak zarządzać procesami w tle bez zrywania połączenia SSH i dlaczego samo `&` to za mało, żeby coś przeżyło Twoje wylogowanie.

Skróty użyte na tej stronie (4)

Skrót	Co znaczy
PID	Process ID: numer procesu w systemie
SID	w Oracle: System Identifier, nazwa instancji; w Windows: Security Identifier, numer konta
SIGHUP	sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Fiszka

TYLKO ODCZYT

Uruchomienie polecenia w tle: `&`

```
dlugie_zadanie.sh &
```

Uruchamia polecenie w tle bieżącej powłoki i od razu oddaje nam z powrotem prompt — nie czekamy na jego zakończenie. Powłoka przypisuje mu numer zadania i wypisuje go razem z PID-em.

Przykład:

```
[1] 14930
```

TYLKO ODCZYT

Zatrzymanie procesu pierwszoplanowego: `Ctrl+Z`

Zatrzymuje bieżący proces pierwszoplanowy sygnałem `SIGTSTP`, nie kończy go. Proces zamiera dokładnie w miejscu, w którym się znajdował, ze stanem w pamięci nietkniętym — wznawiamy go potem przez `fg` albo `bg`.

Przykład:

```
^Z
[1]+  Stopped                 sleep 300
```

TYLKO ODCZYT

Wypisanie listy zadań: `jobs`

```
jobs
```

Pokazujemy zadania bieżącej powłoki wraz ze stanem (`Running`/`Stopped`), bez numerów PID.

Przykład:

```
[1]+  Running                 sleep 300 &
```

PID = Process ID: numer procesu w systemie · **jobs** = pokaż zadania działające w tle bieżącej sesji powłoki

TYLKO ODCZYT

Wypisanie listy zadań z PID-ami: `jobs -l`

```
jobs -l
```

To samo, tylko z numerem PID przy każdym zadaniu — przydaje się, gdy chcemy odwołać się do procesu przez `kill PID` zamiast przez numer zadania.

Przykład:

```
[1]+ 14930 Running                 sleep 300 &
```

PID = Process ID: numer procesu w systemie · **jobs** = pokaż zadania działające w tle bieżącej sesji powłoki

TYLKO ODCZYT

Powrót zadania na pierwszy plan: `fg %1`

```
fg %1
```

Przenosi zadanie numer 1 z powrotem na pierwszy plan — terminal znowu należy do niego, nie do nas. Przydaje się, gdy zadanie w tle nagle potrzebuje wejścia z klawiatury.

TYLKO ODCZYT

Wznowienie zadania w tle: `bg %1`

```
bg %1
```

Wznawia zatrzymane przez `Ctrl+Z` zadanie, ale w tle — proces działa dalej, a terminal zostaje nasz.

TYLKO ODCZYT

Zignorowanie SIGHUP: `nohup`

```
nohup dlugie_zadanie.sh &
```

Uruchamia proces z zignorowanym `SIGHUP` od samego początku — przeżyje wylogowanie i zamknięcie terminala. Jeśli nie przekierujemy jego wyjścia, `nohup` samo utworzy plik `nohup.out` w bieżącym katalogu i tam skieruje standardowe wyjście i błędy.

Przykład:

```
nohup: ignoring input and appending output to 'nohup.out'
```

SIGHUP = sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej

□□ TYLKO ODCZYT

Usunięcie zadania z tabeli zadań powłoki: `disown`

```
disown %1
```

Nie zmienia zachowania samego procesu wobec `SIGHUP` — usuwa zadanie z **tabeli zadań powłoki**. Skoro powłoka przestaje o nim wiedzieć, przestaje też wysyłać mu sygnały przy własnym zamknięciu. Stosujemy to *po fakcie*, do procesu, który już działa w tle — w odróżnieniu od `nohup`, które trzeba użyć przy starcie.

□□ **SIGHUP** = sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej

□□ TYLKO ODCZYT

Ochrona zadania bez usuwania z listy: `disown -h`

```
disown -h %1
```

Wariant łagodniejszy: zadanie zostaje na liście `jobs`, tylko dostaje oznaczenie odporności na `SIGHUP`. Przydaje się, gdy chcemy dalej widzieć zadanie w `jobs`, ale nie chcemy, żeby zginęło przy wylogowaniu.

□□ **SIGHUP** = sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej

□□ ZMIENIA SYSTEM

Uruchomienie w nowej sesji od startu: `setsid`

```
setsid dlugie_zadanie.sh < /dev/null > log.txt 2>&1 &
```

Uruchamia proces w zupełnie nowej sesji, odciętej od terminala od pierwszej chwili, z nowym identyfikatorem sesji równym jego własnemu PID-owi. Taki proces nigdy nie był częścią naszej sesji terminala, więc `SIGHUP` przy jej zamknięciu go nie dotyczy.

□□ **SIGHUP** = sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej

□□ TYLKO ODCZYT

Czekanie na zakończenie procesu: `wait`

```
wait $!
```

Blokuje skrypt do zakończenia wskazanego procesu (albo wszystkich zadań w tle, jeśli wywołane bez argumentu) i zwraca jego kod wyjścia jako swój własny.

□□ TYLKO ODCZYT

Odczytanie PID-u ostatniego procesu w tle: `$!`

```
polecenie &
echo $!
```

Zmienna specjalna powłoki z numerem PID ostatnio uruchomionego procesu w tle — zapisujemy ją, żeby później zrobić na niej `kill` albo `wait`.

Przykład:

```
15104
```

□□ **PID** = Process ID: numer procesu w systemie

□□ ZMIENIA SYSTEM

Przekierowanie procesu w tle

```
./generuj_raport.sh > /tmp/raport.log 2>&1 &
```

Zawsze przekierowujemy wyjście procesu odpalanego w tle. Bez tego wypisze coś na ekran w środku naszej pracy, a jeśli spróbuje coś przeczytać z klawiatury — zamrozi się w stanie `Stopped`, bo dostanie sygnał `SIGTTIN`.

□□ TYLKO ODCZYT

Wszystkie flagi

Komenda	Co robi	Kiedy używasz

`&`	uruchamia polecenie w tle bieżącej powłoki	chcesz odzyskać terminal od razu
`Ctrl+Z`	zatrzymuje proces pierwszoplanowy (`SIGTSTP`), nie kończy go	chcesz na chwilę wrócić do powłoki, nie przerywając zadania
`jobs` / `jobs -l`	lista zadań tej powłoki, opcjonalnie z PID-ami	sprawdzasz, co masz w tle i na jakim jest miejscu (`Running` / `Stopped`)
`fg %N`	przenosi zadanie na pierwszy plan	chcesz odzyskać kontrolę nad zadaniem (np. wpisać hasło)
`bg %N`	wznawia zatrzymane zadanie w tle	zatrzymałeś coś przez `Ctrl+Z` i chcesz, żeby dalej pracowało
`nohup polecenie &`	odizolowuje proces od `SIGHUP` przy wylogowaniu	najprostszy sposób na przeżycie zamknięcia terminala
`disown`	usuwa zadanie z tabeli zadań powłoki	proces już działa, dopiero teraz chcesz go odciąć od `SIGHUP`
`setsid`	uruchamia proces w zupełnie nowej sesji	potrzebujesz pełnej niezależności od terminala od pierwszej chwili
`wait`	czeka na zakończenie zadań/procesów w tle	skrypt musi poczekać, aż wszystkie odpalone w tle się skończą
`$!`	PID ostatnio uruchomionego procesu w tle	zapisujesz PID do późniejszego `kill` albo `wait`

Uważaj

- Zadanie w tle, które pisze na ekran bez przekierowania, zaśmieca Ci terminal w środku pracy — a jeśli spróbuje czytać z klawiatury, dostanie `SIGTTIN` i zamarznie w stanie `Stopped` bez żadnego komunikatu.
- Wylogowanie zamyka terminal i wysyła `SIGHUP` do wszystkich zadań powłoki — `nohup` / `disown` to łatanie objawu, nie docelowe rozwiązanie na długo działające procesy.
- `%1` działa tylko w **tej samej powłoce**, w której zadanie uruchomiono. Nowa sesja SSH albo nowa karta terminala nie widzi cudzych zadań.

SIGHUP = sygnał wysyłany, gdy zamykamy terminal, w którym proces wystartował, dawniej oznaczał rozłączenie linii telefonicznej · **PID** = Process ID: numer procesu w systemie · **SSH** = Secure Shell: szyfrowane zdalne logowanie do serwera

Z życia wzięte

Z ŻYCIA WZIĘTE

O 14:00 Tomek odpala import danych na docker01 zwykłym `&`, bez `nohup`, i zamyka sesję SSH żeby zająć się czymś innym. Import ginie w połowie, bo `SIGHUP` dociera do procesu razem z rozłączeniem terminala, a nic go nie chroniło. Przy kolejnym uruchomieniu Tomek dopisuje `nohup` z przodu i pamięta o tym już zawsze.

Po co to istnieje

Łączysz się przez SSH z `srv01`, odpalasz skrypt, który ma pracować 20 minut, i chcesz w międzyczasie zrobić coś innego w tym samym terminalu — sprawdzić log, wydać drugą komendę — bez otwierania drugiej sesji. Powłoka daje na to gotowy mechanizm: **job control**, czyli zarządzanie zadaniami. Zadanie to nie to samo co proces — to opakowanie, które konkretna powłoka (bash, zsh) nakłada na jeden lub więcej procesów, żeby dało się nimi sterować z klawiatury: zatrzymywać, wznawiać, przełączać między pierwszym planem a tłem.

> **Pojęcie: terminal sterujący.** Proces uruchomiony interaktywnie — w sesji SSH, w oknie terminala — jest podłączony do jednego, konkretnego terminala, który jądro nazywa jego terminalem sterującym. To ten terminal dostarcza mu wejście z klawiatury i przyjmuje jego wyjście, i to jego rozłączenie (zamknięcie okna, zerwanie SSH) jest tym, co uruchamia wysyłkę `SIGHUP` do wszystkich procesów danej sesji. Proces uruchomiony przez `setsid` nie ma terminala sterującego wcale — stąd jest całkowicie odporny na sygnały, które przez ten terminal by przyszły.

Dlatego zadanie ma swój numer (`%1`, `%2`) zamiast PID-u. PID jest własnością jądra i istnieje niezależnie od tego, która powłoka go odpaliła. Numer zadania jest własnością tej jednej, konkretnej powłoki — inna sesja terminala, inny bash, w ogóle go nie widzi, bo dla niej to zadanie nigdy nie istniało.

Analogia

Wyobraź sobie **biurko z szufladą** w otwartej przestrzeni biurowej.

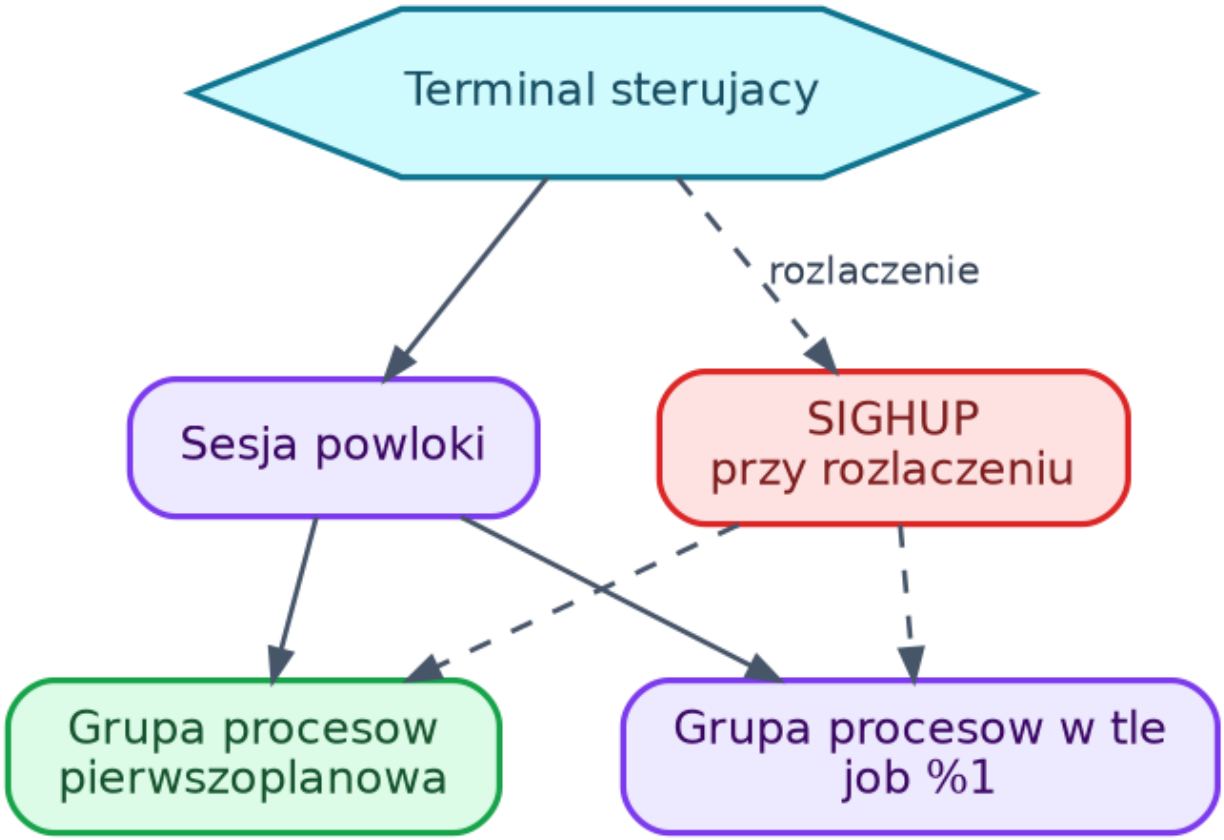
Pracujesz nad dokumentem na blacie. Dzwoni telefon — odkładasz dokument do szuflady (`Ctrl+Z`, zatrzymanie) i odbierasz. Możesz też zostawić dokument otwarty na blacie z boku i kontynuować coś innego, kątem oka widząc, że wciąż tam leży (`&`, uruchomienie w tle bez zatrzymywania). W obu przypadkach dokument jest wciąż Twój i wciąż na Twoim biurku — nikt inny w biurze go nie widzi jako swojej sprawy, tak jak inna sesja terminala nie widzi Twoich zadań.

Reguła, która robi z tej analogii coś więcej niż ładne porównanie: kiedy wychodzisz z pracy i gasisz światło w tym pomieszczeniu (wylogowanie, zamknięcie terminala), sprzątaczka wieczorem zgarnia wszystko, co zostało na blacie i w szufladzie, do kosza. Nieważne, czy dokument leżał otwarty, czy odłożony — jeśli nie przeniosłeś go wcześniej do archiwum w innym budynku (`screen`, `tmux`, `systemd`), z rana go nie będzie. Dokładnie to robi jądro: zamknięcie terminala wysyła `SIGHUP` do całej grupy procesów tej sesji, zatrzymanych i działających jednakowo.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: grupa procesów i sesja.** Kilka procesów, które powłoka uruchamia jako jeden potok (`a | b | c`) albo jedno zadanie w tle, trafia razem do jednej **grupy procesów** — dzięki temu sygnał wysłany do grupy (np. `Ctrl+Z`) trafia do wszystkich naraz, a nie tylko do jednego z nich. Wszystkie grupy procesów uruchomione z tego samego terminala logowania należą z kolei do jednej **sesji** — to na poziomie sesji jądro decyduje, komu wysłać `SIGHUP` przy rozłączeniu terminala.



Sesja terminala grupuje procesy w grupy procesów, a rozłączenie terminala wysyła SIGHUP do całej sesji, chyba że proces jest chroniony przez nohup, disown albo setsid (kliknięcie otwiera pełny rozmiar)

`&` **i grupy procesów.** Kiedy kończysz komendę znakiem `&`, powłoka uruchamia ją jako proces w tle, natychmiast oddaje Ci kontrolę nad promptem i przypisuje jej kolejny numer zadania. Proces (albo cały potok procesów, jeśli `&` kończy potok) trafia do własnej grupy procesów, ale wciąż należy do tej samej sesji terminala co Twoja powłoka.

Ctrl+Z i SIGTSTP. Naciśnięcie `Ctrl+Z` w trakcie działania procesu pierwszoplanowego wysyła do niego (a właściwie do całej grupy procesów pierwszego planu) sygnał `SIGTSTP`. To sygnał przechwytywalny, w odróżnieniu od `SIGSTOP` — ale domyślna akcja, jeśli proces go nie obsłuży sam, jest identyczna: proces przechodzi w stan `T` (zatrzymany), zamrożony dokładnie w miejscu, w którym się znajdował, bez utraty żadnego stanu w pamięci. To jest kluczowa różnica względem zakończenia: zatrzymany proces nie umiera, tylko czeka. `fg` albo `bg` wysyłają mu `SIGCONT` i wznawiają wykonanie od tego samego miejsca.

Co się dzieje przy wylogowaniu. Kiedy zamykasz sesję terminala — wylogowujesz się, zamykasz okno, zrywa się połączenie SSH — powłoka logowania kończy działanie i jądro wysyła

`SIGHUP` do wszystkich procesów, które należą do tej sesji, łącznie z zadaniami zatrzymanymi i tymi działającymi w tle. Domyślna akcja dla `SIGHUP` to zakończenie procesu, więc wszystko, co nie zostało specjalnie zabezpieczone, ginie razem z terminalem. Bash ma na to ustawienie `huponexit` (`shopt -s huponexit`), które kontroluje, czy w ogóle wysyłać `SIGHUP` do zadań w tle przy zamykaniu powłoki interaktywnej — domyślnie wyłączone w większości dystrybucji, ale włączane przez niektóre konfiguracje SSH i menedżery terminali, więc nie polegaj na tym, że „zawsze przeżywa".

`nohup`. Komenda `nohup` uruchamia proces z zignorowanym sygnałem `SIGHUP` od samego początku — proces dostanie sygnał przy wylogowaniu, ale zignoruje go i będzie działał dalej. Jeśli nie przekierujesz jego wyjścia, `nohup` samo utworzy plik `nohup.out` w bieżącym katalogu (albo w katalogu domowym, jeśli katalog bieżący nie jest zapisywalny) i tam skieruje standardowe wyjście i błędy.

`disown`. Działa inaczej niż `nohup` — nie zmienia zachowania samego procesu wobec `SIGHUP`, tylko usuwa zadanie z **tabeli zadań powłoki**. Skoro powłoka przestaje wiedzieć o zadaniu, przestaje też wysyłać mu sygnały przy własnym zamknięciu. Różnica praktyczna: `nohup` trzeba użyć *przed* uruchomieniem procesu (albo przy uruchomieniu), `disown` możesz zastosować *po fakcie*, do procesu, który już działa w tle. Flaga `disown -h` jest łagodniejsza — zostawia zadanie na liście `jobs`, tylko oznacza je jako odporne na `SIGHUP`, co przydaje się, gdy chcesz dalej widzieć je w `jobs`, ale nie chcesz, żeby zginęło przy wylogowaniu.

`setsid`. Trzecia droga, radykalniejsza od obu powyższych: uruchamia proces w zupełnie nowej sesji, odłączonej od terminala od pierwszej chwili, z nowym identyfikatorem sesji (SID) równym PID-owi nowego procesu. Taki proces nigdy nie był częścią Twojej sesji terminala, więc `SIGHUP` przy jej zamknięciu w ogóle go nie dotyczy — nie trzeba go ignorować, bo nie ma skąd przyjść.

Dlaczego to wszystko to półśrodki. `nohup`, `disown` i `setsid` rozwiązują jeden konkretny problem — sygnał `SIGHUP` przy wylogowaniu — ale nie dają Ci z powrotem dostępu do tego, co proces wypisuje na ekran w czasie rzeczywistym, ani możliwości interakcji z nim (odpowiedzi na pytanie, wciśnięcia klawisza). Jeśli proces w środku działania potrzebuje Twojej uwagi, jedyne co masz, to plik z logiem, do którego zaglądasz z boku. Właściwą odpowiedzią na „ma działać niezależnie od tego, czy jestem podłączony" jest sesja **`screen`** albo **`tmux`**, którą możesz odłączyć i podłączyć ponownie, zachowując pełny interaktywny dostęp — albo, dla usług które mają działać stale i bez nadzoru z terminala, jednostka **`systemd`** (`systemctl --user` dla zadań pojedynczego użytkownika, zwykła jednostka systemowa dla usług ogólnosystemowych).

Krok po kroku

1. Odpal coś w tle i sprawdź listę zadań.

```
sleep 300 &
jobs -l
```

Liczba w nawiasach kwadratowych to numer zadania (`%1`), liczba obok to PID. `+` oznacza zadanie bieżące — to, którego dotknie `fg` / `bg` bez podania numeru.

2. Zatrzymaj proces pierwszoplanowy i wznów go w tle.

```
sleep 300
```

Naciśnij `Ctrl+Z`, a potem:

```
bg %1
```

```
[1]+ sleep 300 &
```

Proces działa dalej, ale teraz w tle — terminal jest znowu Twój.

3. Wróć do zadania w tle, żeby coś w nim wpisać.

```
fg %1
```

Zadanie wraca na pierwszy plan. Jeśli proces czekał na wejście z klawiatury, teraz może je dostać — dopóki był w tle, próba odczytu skończyłaby się dla niego sygnałem `SIGTTIN` i zatrzymaniem.

4. Uruchom proces odporny na wylogowanie, z sensownym logiem.

```
nohup ./import_danych.sh > /tmp/import.log 2>&1 &
disown
```

Przekierowanie `> plik 2>&1` jest tu obowiązkowe z dwóch powodów: nie zaśmieca terminala i nie zostawia wszystkiego w domyślnym `nohup.out`, który przy kolejnym uruchomieniu innej komendy z `nohup` zostanie po cichu dopisany do tego samego pliku. `disown` po `nohup` jest w praktyce redundantne (`nohup` już ignoruje `SIGHUP`), ale bywa dodawane z przyzwyczajenia — nie szkodzi, tylko usuwa zadanie z listy `jobs` bieżącej powłoki.

5. Odetnij proces od terminala jeszcze mocniej, przez `setsid`.

```
setsid ./serwer_testowy.sh < /dev/null > /tmp/serwer.log 2>&1 &
```

`< /dev/null` jest tu ważne — bez tego proces w nowej sesji wciąż dziedziczyłby standardowe wejście z terminala, a próba odczytania czegokolwiek z zamkniętego albo cudzego terminala kończy się błędem zamiast cichym zawieszeniem.

6. Poczekaj w skrypcie na zakończenie kilku zadań w tle.

```
#!/bin/bash
./zadanie_a.sh &
PID_A=$!
./zadanie_b.sh &
PID_B=$!

wait $PID_A
echo "zadanie A skończone, kod: $?"
wait $PID_B
echo "zadanie B skończone, kod: $?"
```

`wait PID` blokuje skrypt do zakończenia konkretnego procesu i zwraca jego kod wyjścia jako swój własny — dzięki temu `$?` po `wait` mówi, jak skończyło się zadanie w tle, a nie sam `wait`.

Częste błędy

Proces w tle bez przekierowania wyjścia.

```
$ ./generuj_raport.sh &
[1] 15022
$ ls /var/log
Raport gotowy, 4821 wierszy.
```

Skrypt w środku Twojej sesji wypisał coś na ekran w chwili, gdy wpisywałeś zupełnie inną komendę — linia wynikowa `ls` i komunikat ze skryptu wymieszały się na terminalu. Gorszy wariant tej pomyłki: skrypt w pewnym momencie próbuje coś przeczytać ze standardowego wejścia (pyta o potwierdzenie, czeka na hasło). Nie dostaje niczego, bo wejście terminala należy teraz do Twojej interaktywnej powłoki, więc jądro wysyła procesowi w tle `SIGTTIN`, a on zamiera w stanie `Stopped` bez żadnego komunikatu o błędzie:

```
$ jobs
[1]+  Stopped                 ./generuj_raport.sh
```

Naprawa na przyszłość — zawsze przekierowuj wyjście procesu odpalanego w tle:

```
./generuj_raport.sh > /tmp/raport.log 2>&1 &
```

Pełna mechanika przekierowań jest na stronie *Przekierowania: >, >>, 2>, &>, <, tee*.

Zamknięcie terminala tuż po `&`, bez `nohup`/`disown`, i zdziwienie, że proces zniknął.

```
$ ./import.sh &
[1] 15104
$ exit
```

Sesja SSH się zamyka, terminal znika, a razem z nim `SIGHUP` dociera do procesu `import.sh`, który — nie mając żadnej ochrony — kończy działanie w połowie importu. Nic tego nie zasygnalizowało w chwili wylogowania; efekt widać dopiero, gdy wracasz i sprawdzasz, że dane są zaimportowane tylko częściowo. Zanim zamkniesz terminal, sprawdź `jobs` — jeśli coś tam wisi, a miało przeżyć rozłączenie, jest już za późno na `nohup` (trzeba było go użyć przy starcie), ale `disown -h %1` jeszcze zdąży zadziałać.

Traktowanie `nohup`/`disown` jako pełnoprawnej alternatywy dla `screen`/`tmux`.

Proces przeżył wylogowanie dzięki `nohup`, ale dwa dni później trzeba do niego wrócić — sprawdzić, co dokładnie wypisuje, wcisnąć klawisz, przerwać w konkretnym miejscu. Nic z tego się nie da zrobić: `nohup` daje tylko plik z logiem, nie żywy terminal. Jeśli zadanie wymaga choćby potencjalnej interakcji albo obserwacji w czasie rzeczywistym, właściwym narzędziem od początku jest `screen -dmS nazwa komenda` albo sesja `tmux`, do której zawsze można wrócić — patrz *screen — sesje, które przeżyją rozłączenie* i strona o `tmux`.

Odwoływanie się do `%1` z innej sesji terminala.

```
# sesja A (SSH, terminal 1):
./dlugie.sh &
[1] 15200

# sesja B (SSH, terminal 2, ten sam serwer, ten sam użytkownik):
$ fg %1
bash: fg: %1: no such job
```

Numer zadania jest lokalny dla powłoki, w której zadanie powstało — druga sesja SSH ma zupełnie inną, pustą tabelę zadań, nawet jeśli logujesz się jako ten sam użytkownik na ten sam serwer. Żeby sięgnąć do procesu z innej sesji, potrzebny jest PID (`ps`, `pgrep`) albo — lepiej — żeby proces od początku żył w sesji `screen`/`tmux`, do której obie sesje terminala mogą się podłączyć.

□□ Ćwiczenia

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

Ćwiczenie 1 — trzy zadania i jedno wylogowanie

```
cd /tmp


# 1. Odpal trzy zadania w tle
sleep 100 &
sleep 200 &
sleep 300 &
jobs -l
```

Zapisz numery zadań i PID-y z wyniku.

```
# 2. Przenieś środkowe zadanie na pierwszy plan, zatrzymaj je Ctrl+Z, wznów w tle
fg %2
# naciśnij Ctrl+Z
bg %2
jobs -l
```

```
# 3. Odeślij konkretne zadanie sygnałem, bez znajomości PID-u
kill %1
jobs -l
```

```
# 4. Sprawdź, co zostaje po zamknięciu powłoki bez ochrony
bash -c 'sleep 60 & echo "PID w subshellu: $!"; exit'
```

Czego się spodziewać

Krok 2: po `Ctrl+Z` zadanie `%2` pokaże się jako `Stopped`, po `bg %2` wróci do `Running`, ale w tle. Krok 3: `kill %1` wysyła domyślny `SIGTERM` do zadania nr 1 — kolejne `jobs -l` pokaże je jako `Terminated`, a przy następnym odświeżeniu zniknie z listy. Krok 4: proces `sleep 60` uruchomiony wewnątrz podpowłoki `bash -c '...'` nie jest chroniony przed `SIGHUP` w żaden sposób — po `exit` tej podpowłoki (a to jest dokładnie to, co dzieje się przy zamykaniu terminala) proces zwykle ginie razem z nią, chyba że system ma wyłączone wysyłanie `SIGHUP` do zadań w tle. Sprawdź `pgrep sleep` chwilę później, żeby zobaczyć, czy proces przeżył.

Pytania kontrolne

1. Dlaczego `%1` działa w jednej sesji terminala, a w drugiej sesji tego samego serwera i użytkownika zwraca błąd „no such job"?

Odpowiedź

Bo numer zadania jest strukturą danych utrzymywaną przez konkretną powłokę, nie przez jądro. Każda powłoka ma własną, pustą na starcie tabelę zadań — druga sesja SSH uruchamia nową powłokę, która nigdy nie odpaliła żadnego procesu w tle, więc nie ma czego nazwać `%1`. PID natomiast jest globalny w całym systemie i widoczny z każdej sesji przez `ps` / `pgrep`.

2. Czym różni się `Ctrl+Z` od `Ctrl+C` w skutkach dla uruchomionego procesu?

Odpowiedź

`Ctrl+C` wysyła `SIGINT`, którego domyślna akcja to zakończenie procesu — proces (jeśli nie obsługuje sygnału inaczej) znika. `Ctrl+Z` wysyła `SIGTSTP`, którego domyślna akcja to zatrzymanie — proces zamiera w miejscu ze swoim stanem w pamięci nietkniętym i można go wznowić przez `fg` albo `bg`, kontynuując dokładnie tam, gdzie przerwał.

3. Uruchomiłeś proces przez `nohup komenda &`, ale nie przekierowałeś jego wyjścia. Gdzie ono trafia i dlaczego to bywa problemem?

Odpowiedź

Trafia do pliku `nohup.out` w bieżącym katalogu roboczym (albo w katalogu domowym, jeśli bieżący nie jest zapisywalny). Problem pojawia się, gdy w tym samym katalogu uruchomisz kolejne polecenie przez `nohup` bez przekierowania — nowe wyjście dopisze się do tego samego `nohup.out`, mieszając logi dwóch niepowiązanych procesów. Rozwiązanie: zawsze przekierowywać jawnie, np. `nohup komenda > /tmp/moj.log 2>&1 &`.

4. Masz proces, który ma działać tygodniami, bez nadzoru, i wymaga, żebyś czasem się do niego podłączył i zobaczył, co się dzieje. `nohup` czy `screen` / `tmux`?

Odpowiedź

`screen` / `tmux`. `nohup` chroni tylko przed sygnałem `SIGHUP` i daje Ci plik z logiem — nie pozwala wrócić do żywego, interaktywnego widoku procesu. Sesja `screen` albo `tmux` działa niezależnie od terminala i możesz się do niej podłączyć ponownie w dowolnym momencie, widząc dokładnie to, co proces wypisuje, i mogąc na niego zareagować.

Powiązane: Sygnały: kill, pkill, killall, trap · screen: sesje, które przeżyją rozłączenie · tmux · Przekierowania: >, >>, 2>, &>, <, tee · systemd i systemctl — jednostki, zależności, stany

Źródła: `man 1 bash` (sekcje JOB CONTROL, SIGNALS) · `man 1 nohup` · `man 1 setsid` · `help disown` (bash builtin) · `man 7 signal`

Revision #18
Created 2026-09-17 15:07:18 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:17 UTC by Tomasz Konieczny