Wprowadzenie do pracy w środowisku Linux

Część 2: Automatyzacja i Zarządzanie Systemem

Opracowanie dla studentów matematyki

13 stycznia 2026

Spis treści

1 Skrypty Bash – Wprowadzenie do automatyzacji

Skrypt powłoki (shell script)

Jest to plik tekstowy zawierający sekwencję poleceń, które są wykonywane przez powłokę linia po linii. Skrypty pozwalają zautomatyzować złożone lub powtarzalne zadania, takie jak przygotowywanie danych do analizy, uruchamianie symulacji numerycznych, tworzenie kopii zapasowych czy generowanie cyklicznych raportów.

Zamiast ręcznie wpisywać dziesięciu poleceń, aby pobrać dane, przetworzyć je i wygenerować wykres, możemy zapisać je w skrypcie i uruchomić za pomocą jednej komendy.

1.1 Tworzenie i edycja skryptu: edytor `nano`

Do tworzenia i edycji plików tekstowych w terminalu służą edytory tekstu. Jednym z najprostszych jest `nano`.

Terminal – Bash

```
# Otwórz (lub utwórz) plik o nazwie `analiza.sh` w edytorze nano
$ nano analiza.sh
```

Po otwarciu edytora, na dole ekranu widoczne są najważniejsze skróty klawiszowe (znak ^ oznacza klawisz `Ctrl`).

- ^O (Ctrl+O): Zapisz plik (*Write Out*).
- ^X (Ctrl+X): Wyjdź z edytora (*Exit*).

Wpiszmy w edytorze `nano` treść naszego pierwszego skryptu:

```
#!/bin/bash
# Prosty skrypt analityczny
# 1. Tworzy katalog na wyniki
# 2. Zapisuje w nim log z datą rozpoczęcia
# 3. Symuluje długotrwałe obliczenia

echo "Rozpoczynam analizę..."
mkdir -p wyniki_analizy
date > wyniki_analizy/log.txt
echo "Przeprowadzam symulację..."
sleep 5 # Czeka 5 sekund
echo "Analiza zakończona." >> wyniki_analizy/log.txt
date >> wyniki_analizy/log.txt
```

2 System uprawnień plików

Zanim uruchomimy nasz skrypt, musimy nadać mu prawo do wykonania. W Linuksie każdy plik i katalog ma precyzyjnie określone uprawnienia, które decydują o tym, kto i w jaki sposób może z nim wchodzić w interakcję.

Podstawą systemu uprawnień są trzy fundamentalne prawa, reprezentowane przez litery:

- **r (read)** – Prawo do **odczytu**.
  - Dla pliku: pozwala na przeglądanie jego zawartości (np. poleceniem `cat`).
  - Dla katalogu: pozwala na wylistowanie jego zawartości (np. poleceniem `ls`).
- **w (write)** – Prawo do **zapisu**.
  - Dla pliku: pozwala na modyfikowanie jego zawartości (edycję, nadpisywanie).
  - Dla katalogu: pozwala na tworzenie, usuwanie i zmianę nazw plików wewnątrz tego katalogu.
- **x (execute)** – Prawo do **wykonania**.
  - Dla pliku: pozwala na uruchomienie go jako programu lub skryptu.
  - Dla katalogu: pozwala na "wejście"do niego (np. poleceniem `cd`).

2.1 Odczyt uprawnień: Anatomia `ls -l`

Wynik polecenia `ls -l` zawiera szczegółowe informacje, w tym 10-znakowy ciąg opisujący uprawnienia:

Terminal – Bash

```
$ ls -l analiza.sh
-rw-r--r-- 1 student users 215 paź 26 10:30 analiza.sh
```

Analiza uprawnień: `-rw-r-r-`

Ten 10-znakowy ciąg dzielimy na cztery części:

Znak 1	Znaki 2-4	Znaki 5-7	Znaki 8-10
`-`	`rw-`	`r-`	`r-`
Typ pliku	Właściciel	Grupa	Inni

- **Typ pliku**: `-` oznacza zwykły plik, `d` to katalog (directory).
- **Właściciel (user)**: Uprawnienia dla właściciela pliku. Tutaj: odczyt (`r`) i zapis (`w`).
- **Grupa (group)**: Uprawnienia dla grupy. Tutaj: tylko odczyt (`r`).
- **Inni (others)**: Uprawnienia dla pozostałych. Tutaj: tylko odczyt (`r`).
- Myślnik `-` w miejscu uprawnienia oznacza jego brak.

2.2 Zmiana uprawnień: polecenie `chmod`

Polecenie `chmod` (change mode) pozwala modyfikować te uprawnienia.

Terminal – Bash

```
# Zobaczmy obecne uprawnienia
$ ls -l analiza.sh
-rw-r--r-- 1 student users 215 paź 26 10:30 analiza.sh

# Nadajmy właścicielowi prawo do wykonania.
$ chmod rwx r-x r-x analiza.sh

# Sprawdźmy ponownie. Znak `x` został dodany, a nazwa pliku zmieniła
↪  kolor.
$ ls -l analiza.sh
-rwxr-xr-x 1 student users 215 paź 26 10:32 analiza.sh
```

Uprawnienia jako system ósemkowy

Każde z trzech uprawnień (`r`, `w`, `x`) można przedstawić jako bit w liczbie 3-bitowej. Jest to naturalna konsekwencja potęg dwójki:

- **r** (read) $= 2^2 = \mathbf{4}$
- **w** (write) $= 2^1 = \mathbf{2}$
- **x** (execute) $= 2^0 = \mathbf{1}$

Sumując te wartości, otrzymujemy jedną cyfrę (od 0 do 7) dla każdej kategorii użytkowników (właściciel, grupa, inni). Przykładowo, `chmod 754 plik.txt` oznacza:

- **7** dla właściciela: $4{+}2{+}1 \rightarrow$ `rwx`
- **5** dla grupy: $4{+}0{+}1 \rightarrow$ `r-x`
- **4** dla innych: $4{+}0{+}0 \rightarrow$ `r--`

Terminal – Bash

```
# Zobaczmy obecne uprawnienia
$ ls -l analiza.sh
-rw-r--r-- 1 student users 215 paź 26 10:30 analiza.sh

# Nadajmy właścicielowi prawo do wykonania.
# Chcemy: rwx r-x r-x  ->  (4+2+1)(4+0+1)(4+0+1)  ->  755
$ chmod 755 analiza.sh

# Sprawdźmy ponownie. Znak `x` został dodany, a nazwa pliku zmieniła
↪  kolor.
$ ls -l analiza.sh
-rwxr-xr-x 1 student users 215 paź 26 10:32 analiza.sh
```

Pełna tabela systemu ósemkowego uprawnień (0-7)

Liczba (Octal)	Postać binarna (rwx)	Suma	Znaczenie uprawnień
0	`---`	0+0+0	Brak jakichkolwiek uprawnień. Plik jest całkowicie niedostępny.
1	`--x`	0+0+1	Tylko wykonanie. Umożliwia wejście do katalogu lub uruchomienie pliku binarnego, ale nie pozwala na odczyt jego zawartości.
2	`-w-`	0+2+0	Tylko zapis. Rzadko używane samodzielnie, ponieważ aby zapisać plik, zazwyczaj trzeba go najpierw odczytać.
3	`-wx`	0+2+1	Zapis i wykonanie. Pozwala na modyfikację i uruchomienie pliku.
4	`r--`	4+0+0	Tylko odczyt. Typowe uprawnienie dla plików z danymi, które nie powinny być modyfikowane przez wszystkich.
5	`r-x`	4+0+1	Odczyt i wykonanie. Standardowe uprawnienie dla programów i katalogów, do których potrzebny jest dostęp.
6	`rw-`	4+2+0	Odczyt i zapis. Typowe uprawnienie dla plików, nad którymi pracujemy (np. dokumenty, kod źródłowy).
7	`rwx`	4+2+1	Pełne uprawnienia. Zapewnia pełną kontrolę nad plikiem lub katalogiem.

2.3 Uruchamianie skryptu

Gdy plik ma już prawo do wykonania, możemy go uruchomić, podając jego ścieżkę.

Terminal – Bash

```
# Kropka (.) to skrót oznaczający bieżący katalog
$ ./analiza.sh
# Wynik:
# Rozpoczynam analizę...
# Przeprowadzam symulację...
# (czekamy 5 sekund)
# Analiza zakończona.
```

3 Monitorowanie i zarządzanie procesami

Proces

To uruchomiona instancja programu. Każdy program, który działa w systemie (nawet sam terminal), ma co najmniej jeden proces. Każdy proces ma unikalny, nume-

ryczny identyfikator (**PID** - Process ID) oraz określony stan.

3.1 Stany procesów

- **Running (R)**: Proces jest aktualnie wykonywany przez procesor lub czeka w kolejce na swoją turę.
- **Sleeping (S)**: Proces czeka na zdarzenie (np. dane z dysku, odpowiedź z sieci). Większość procesów przez większość czasu jest w tym stanie.
- **Stopped (T)**: Proces został zatrzymany (np. przez użytkownika) i może być wznowiony.
- **Zombie (Z)**: Proces zakończył działanie, ale jego wpis w tablicy procesów wciąż istnieje, ponieważ proces nadrzędny nie odczytał jego statusu zakończenia.

3.2 Interaktywny monitoring: polecenie `top`

`top` to fundamentalne narzędzie diagnostyczne. Poza wyświetlaniem listy procesów, pozwala na interaktywne zarządzanie widokiem.

Interaktywne polecenia w `top`

Będąc w `top`, wciśnij klawisz, aby zmienić zachowanie:

- `M` (duże M): Sortuj procesy według użycia pamięci (%MEM).
- `P` (duże P): Sortuj procesy według użycia CPU (%CPU) – domyślne.
- `k`: "Zabij"proces. `top` zapyta o PID procesu do zabicia.
- `h`: Wyświetl pomoc.
- `q`: Wyjdź.

Ciekawostka: `htop`

`htop` to nowocześniejsza, bardziej kolorowa i interaktywna wersja `top`. Oferuje m.in. łatwiejsze przewijanie i zabijanie procesów za pomocą klawiszy funkcyjnych.

3.3 Wyszukiwanie i zabijanie procesów

Ręczne szukanie PID w `top` jest nieefektywne. Lepiej użyć dedykowanych narzędzi.

3.3.1 Wyszukiwanie procesów: `pgrep`

Polecenie `pgrep` (process grep) wyszukuje procesy po nazwie i zwraca ich PID.

Terminal – Bash

```
# Uruchommy w tle proces, który będzie długo działał
$ sleep 600 &
[1] 12345

# Znajdźmy PID procesu o nazwie `sleep`
$ pgrep sleep
12345
```

3.3.2 Zabijanie procesów: sygnały i `kill`

Polecenie `kill` nie "zabija"procesu wprost. Wysyła do niego **sygnał**, czyli komunikat systemowy. Proces może na sygnał zareagować, np. grzecznie się zamykając.

Sygnał	Numer	Opis i zastosowanie
`SIGTERM`	15	**Terminate (zakończ)**. Domyślny, "uprzejmy"sygnał. Prosi proces o zakończenie pracy, dając mu szansę na zapisanie danych i posprzątanie. To pierwsza próba zamknięcia programu.
`SIGKILL`	9	**Kill (zabij)**. Sygnał ostateczny, "brutalny". Nie może być zignorowany. Jądro systemu natychmiast usuwa proces z pamięci. Używany, gdy proces się zawiesił i nie reaguje na `SIGTERM`.
`SIGINT`	2	**Interrupt (przerwij)**. Sygnał wysyłany po naciśnięciu `Ctrl+C` w terminalu.

Terminal – Bash

```
# Znajdź PID procesu `sleep`
$ pgrep sleep
12345

# Wyślij domyślny sygnał SIGTERM (prośba o zamknięcie)
$ kill 12345

# Jeśli proces nie reaguje, użyj SIGKILL
# kill -9 <PID> LUB kill -SIGKILL <PID>
$ kill -9 12345
```

Polecenie `pkill`

`pkill` łączy w sobie funkcjonalność `pgrep` i `kill`. Znajduje procesy po nazwie i od razu wysyła do nich sygnał. Jest bardzo wygodne, ale też bardziej ryzykowne – można przypadkowo zabić wiele procesów naraz.

```
pkill -9 nazwa_procesu
```