

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2010

Linux. Komendy i polecenia. Wydanie III

Autor: Łukasz Sosna
ISBN: 978-83-246-2602-1
Format: 115×170, stron: 144



Odkryj wielką siłę drzemiącą w Linuksie!

- Jak zainstalować Linux i zarządzać zasobami komputera z tym systemem?
- Co trzeba wiedzieć o administrowaniu systemem i tworzeniu skryptów powłoki?
- Jakie dodatkowe komendy i polecenia warto poznać w pierwszej kolejności?

Linux, jeden z najbardziej znanych i wydajnych systemów operacyjnych, wśród wielu użytkowników komputerów powoli staje się realną alternatywą dla Windows. Pomijając już nawet jego niesamowitą elastyczność, małe wymagania i świetnie działającą społeczność, otwartą na potrzeby nowych członków, atutem Linuksa jest możliwość sterowania nim z poziomu wiersza poleceń, co wydatnie skraca czas operacji i pozwala na pełną kontrolę pracy. Taka komunikacja z systemem jest najefektywniejszym rozwiązaniem i wbrew pozorom wcale nie wymaga od użytkownika znajomości żadnej czarnej magii.

Dziś do Twoich rąk trafia trzecie już, uzupełnione o nowy rozdział wydanie popularnej książki „Linux. Komendy i polecenia”, niezwykle przydatnej początkującym adeptom tego systemu. Znajdziesz w niej wszystko, co trzeba wiedzieć podczas obsługi Linuksa z poziomu linii poleceń — od kwestii instalacji, przez zasady zarządzania zawartością komputera, aż po szczegółowe zagadnienia z zakresu administrowania systemem. Dowiesz się także, co to są skrypty powłoki i nauczysz się używać zaawansowanych poleceń oszczędzających Twój czas.

W książce omówiono następujące tematy:

- Dostępne dystrybucje
- Instalacja systemu
- Środowisko pracy i logowanie się do systemu
- Dyski i partycje
- Operacje na plikach i katalogach
- Prawa dostępu, zmiana hasła i zmiana powłoki
- Informacje o sprzęcie i użytkownikach
- Poziom uruchomienia systemu
- Demony usług
- Użytkownicy i grupy
- Tworzenie skryptów powłoki
- Dodatkowe informacje o plikach i katalogach

Linux – naucz się praktycznej i efektywnej obsługi systemu!

Spis treści

Wprowadzenie do systemu Linux	7
Czym jest Linux?	8
Dostępne dystrybucje — jak wybrać odpowiednią dla siebie?	9
Instalacja systemu	9
 1. Korzystanie z komputera pracującego pod kontrolą systemu Linux	 14
Środowisko pracy	14
Logowanie się do systemu	16
Bezpieczne wyłączanie i restart komputera	17
Użytkownicy systemu Linux	18
Co znajduje się w poszczególnych katalogach systemu?	19
Dyski i partycje w systemie	21
Pomoc na stronach MAN	22
 2. Zarządzanie zasobami komputera	 23
Pliki i katalogi w systemie	23
Wyświetlanie zawartości katalogu	24
Przechodzenie pomiędzy katalogami	34
Tworzenie katalogów	35
Usuwanie katalogów	36
Tworzenie plików	38
Usuwanie plików	38
Wyświetlenie zawartości pliku	39
Zmiana dat modyfikacji plików i dostępu do nich	40

Kopiowanie plików i katalogów	43
Przenoszenie plików i katalogów oraz zmiana ich nazwy	46
Nadawanie praw dostępu do plików i katalogów	48
Zmiana hasła	52
Zmiana powłoki	53
Uzyskiwanie informacji o typie pliku	54
Zmiana właściciela i grupy pliku	55
Wyszukiwanie plików i katalogów	56
Wypisywanie ilości bajtów, słów i linii	62
Porównywanie plików lub zakresów bajtów	63
Uzyskiwanie informacji o ilości wolnego miejsca na partycjach	64
Ustalanie, ile miejsca zajmuje plik lub katalog	65
Polecenia more i less	67
Montowanie i odmontowywanie systemów plików	68
Aktualna ścieżka, pod którą pracujemy	70
Przełączanie się na konto innego użytkownika	70
Uzyskiwanie informacji o sprzęcie	71
Przeglądanie kalendarza	75
Aktualizacja daty i czasu	77
Kontrolowanie wysyłania wiadomości	82
Wysyłanie wiadomości do innego użytkownika	82
Wysyłanie wiadomości z pliku tekstowego	83
Wysyłanie komunikatów do wszystkich sieci z pliku tekstowego	83
Pokazywanie ostatnio zalogowanych użytkowników	83
Sprawdzanie, kto jest aktualnie zalogowany na naszym komputerze	86
Informacja o tym, kto jest zalogowany do systemu	86
Sprawdzanie swojej nazwy użytkownika	86
Pokazywanie lub ustawianie nazwy hosta systemowego	87

Wyświetlanie i ustalanie parametrów interfejsu sieciowego	89
Wyszukiwanie nazwy lub adresu IP zdalnego komputera	90
Sprawdzanie, czy dana domena jest już zarejestrowana	91
Sprawdzenie dostępności hosta	91
Czas, jaki upłynął od uruchomienia systemu	92
3. Administrowanie systemem	93
Poziom uruchomienia systemu	93
Demony usług	94
Użytkownicy	96
Grupy	99
Szukanie łańcuchów w bazie whatis	99
4. Tworzenie skryptów powłoki	100
Zmienne	102
Wypisywanie tekstu na ekranie użytkownika	103
Wartości logiczne	106
Polecenie test	107
Instrukcja if	112
Instrukcja case	114
Pętla while	115
Pętla until	115
Pętla for	115
Break	116
Continue	117
Argumenty pobierane z wiersza powłoki	117
5. Polecenia dodatkowe	118
SSH	118
Historia poleceń użytych w powłoce	123
Wypisywanie pierwszych wierszy pliku	124

Wypisywanie ostatnich linii pliku	125
Uzyskiwanie informacji o trybie tworzenia nowych plików i katalogów	126
Wyświetlanie atrybutów plików i katalogów	127
Dodatkowe prawa dostępu do plików	128
Sprawdzanie dodatkowych uprawnień do plików	129
Wyszukiwanie danych w plikach	129
Skorowidz	139

Rozdział 4. Tworzenie skryptów powłoki

Pisanie skryptów powłoki to programowanie odpowiednich instrukcji — programów, które usprawniają wykonywanie wielu czynności. Skrypty powłoki obsługują zmienne, instrukcje warunkowe, pętle i wiele innych przydatnych elementów.

Skrypty powłoki to po prostu zgrupowane polecenia zapisane w jednym pliku. Podobnie jak przy wpisywaniu poleceń w okienku terminala, powinieneś pamiętać o tym, jak będą one wpisywane do pliku. Pamiętaj o tym, że każda nowa linia to nowe polecenie, więc nie można zapisywać poleceń w dwóch liniach.

Na przykład, aby wyświetlić listę zawartości swojego katalogu głównego, całe polecenie trzeba zapisać w jednej linii, ponieważ zapisanie go w dwóch lub więcej spowoduje błąd.

```
#!/bin/bash
vdir
/home/lukasz

[lukasz@localhost ~]$ ./skrypt
razem 44
drwx----- 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 32 cze 10 13:41 skrypt
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:41 skrypt~
drwx----- 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
./skrypt: line 3: /home/lukasz: is a directory
```

Poprawnie zapisany skrypt będzie wyglądał następująco:

```
#!/bin/bash
vdir /home/lukasz
```

Wykonanie skryptu da pożądaný efekt — listę zawartości katalogu głównego.

```
[lukasz@localhost ~]$ ./skrypt
razem 44
drwx----- 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:41 skrypt
-rwxrwxrwx 1 lukasz lukasz 29 cze 10 13:40 skrypt~
drwx----- 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
```

Skrypty powłoki muszą zostać poprzedzone odpowiednią instrukcją odwołującą się do interpretera powłoki, której używamy.

```
#!/bin/bash
```

Dodatkowo plik taki musi mieć prawa do wykonywania, które należy nadać mu poleceniem `chmod` (opisywanym we wcześniejszej części książki).

```
[lukasz@localhost ~]$ chmod 777 skrypt
```

W celu uruchomienia skryptu należy odpowiednio go wywołać. Zapiszmy skrypt w pliku `skrypt`. Wówczas mamy go w katalogu głównym użytkownika i aby go wywołać, nie wystarczy wpisać jego nazwy, gdyż powłoka będzie wyszukiwała polecenia o takiej nazwie w ścieżkach wyszukiwania. Przed skryptem należy wpisać pełną ścieżkę dostępu do niego, zaczynając od znaku `/`, a gdy jesteśmy w katalogu, w którym jest umieszczony skrypt, wystarczy wpisać `./` (aktualny katalog, w którym znajduje się skrypt). W takim wypadku będziemy mieli pewność, że skrypt się uruchomi.

```
[lukasz@localhost ~]$ ./skrypt
```

Drugim sposobem uruchamiania skryptu jest użycie powłoki i przekazanie do niej skryptu w formie argumentu.

```
[lukasz@localhost ~]$ bash skrypt
razem 44
drwx----- 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:42 skrypt
-rwxrwxrwx 1 lukasz lukasz 32 cze 10 13:41 skrypt~
drwx----- 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
```

Trzecim sposobem uruchomienia skryptu w aktualnej powłocie jest użycie znaku specjalnego. Dokonujemy tego za pomocą znaku ..

```
[lukasz@localhost ~]$ . skrypt
razem 44
drwx----- 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 31 cze 10 13:42 skrypt
-rwxrwxrwx 1 lukasz lukasz 32 cze 10 13:41 skrypt~
drwx----- 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
```

Zmienne

Zmienne to elementy, które mogą przechowywać wartości. W powłocie istnieją zmienne mogące przechowywać wartości logiczne, tekst i liczby. Nie trzeba deklarować typu zmiennej na samym początku skryptu — wystarczy podać dla niej wartość podczas wpisywania skryptu.

Zmienną definiuje przypisywana do niej wartość. Wartość do zmiennej najlepiej wpisywać w cudzysłowach (przy późniejszych manipulacjach jej wartością lub próbach użycia w innym miejscu skryptu cudzysłów zabezpiecza nas przed wystąpieniem błędu).

Zadeklarujmy zmienną nazywającą się `zmienna` i zawierającą słowo `tekst`.

```
#!/bin/bash
zmienna="tekst"
```

Jak widać, `zmienna` jest zwykłym tekstem. Przy jej deklarowaniu nie trzeba dodawać żadnych znaków specjalnych przed czy za nią.

```
#!/bin/bash
zmienna="tekst"
echo zmienna
```

Przy wyświetlaniu wartości zapisanej w zmiennej należy poprzedzić ją znakiem dolara „\$”, aby wyświetlanie zadziałało, to znaczy aby wyświetliła się jej wartość, a nie nazwa zmiennej.

```
#!/bin/bash
zmienna="tekst"
echo $zmienna
```

Gdybyśmy nie dodali znaku dolara przed nazwą zmiennej w instrukcji `echo`, po wywołaniu tego skryptu zostanie wyświetlona na ekranie wartość `tekst` zamiast wartości `zmienna`.

```
[lukasz@localhost ~]$ ./skrypt
zmienna
Skrypt bez dodania znaku dolara przed nazwą zmiennej

[lukasz@localhost ~]$ ./skrypt
tekst
Skrypt ze znakiem dolara przed nazwą zmiennej
```

Wypisywanie tekstu na ekranie użytkownika

Do wypisywania tekstu używamy kilku poleceń, spośród których najpopularniejszym jest `echo`.

W celu wypisania tekstu na ekranie użytkownika po poleceniu `echo` deklarujemy tekst, który zostanie wyświetlony po wywołaniu skryptu.

```
#!/bin/bash
echo To jest tekst
```

Po wywołaniu tego skryptu otrzymamy rezultat:

```
[lukasz@localhost ~]$ ./skrypt
To jest tekst
```

-n

Parametr ten nie wypisze na końcu linii znaku nowej linii, dzięki czemu wszystkie informacje zostaną wypisane w jednym wierszu.

```
#!/bin/bash
echo -n To jest tekst
echo To jest tekst
```

```
[lukasz@localhost ~]$ ./skrypt
To jest tekstTo jest tekst
```

-e

Parametr ten rozpoznaje i interpretuje wszystkie znaki specjalne wpisywane przez nas do skryptu. Znaki specjalne deklaruje się przez poprzedzenie ich znakiem backslasha.

```
#!/bin/bash
echo -e To jest tekst\a
```

```
[lukasz@localhost ~]$ ./skrypt
To jest tekst
```

-E

Parametr ten powoduje nieinterpretowanie znaków specjalnych we wpisywanym tekście i pomija ich wykonanie.

```
#!/bin/bash
echo -E To jest tekst\a
```

```
[lukasz@localhost ~]$ ./skrypt
To jest teksta
```

\a

Parametr powoduje pojawienie się alarmu w postaci sygnału dźwiękowego.

```
#!/bin/bash
echo -e To jest tekst\a
```

\b

Ten parametr po wypisaniu tekstu przesuwa kursor o jeden znak bliżej początku tekstu.

```
#!/bin/bash
echo -e To jest tekst\b
```

\c

Parametr powoduje niewypisanie znaku nowego wiersza na końcu linii.

```
#!/bin/bash
echo -e To jest tekst\c
```

\f

Ten parametr powoduje wysunięcie strony i zmianę miejsca kursora w tekście.

```
#!/bin/bash
echo -e To jest tekst\f
```

\n

Parametr powoduje pojawienie się nowego wiersza po zakończeniu wypisywania tekstu.

```
#!/bin/bash
echo -e To jest tekst\n
```

\r

Parametr powoduje powrót karetki do początku linii.

```
#!/bin/bash
echo -e To jest tekst\r
```

\t

Parametr powoduje pojawienie się znaku tabulacji w poziomie.

```
#!/bin/bash
echo -e To jest tekst\t
```

\v

Parametr powoduje pojawienie się tabulacji w pionie.

```
#!/bin/bash
echo -e To jest tekst\v
```

\\

Parametr ten służy do wypisania znaku backslasha.

```
#!/bin/bash
echo -e To jest tekst\\
```

\'

Parametr ten pozwala na wypisanie pojedynczego cudzo-
słowa.

```
#!/bin/bash
echo -e To jest tekst\'
```

\"

Parametr pozwala na wypisanie podwójnego cudzo-
słowa.

```
#!/bin/bash
echo -e To jest tekst\"
```

\nnn

Parametr ten pozwala na wypisanie znaku z tabeli kodów
ASCII o ósemkowej notacji.

```
#!/bin/bash
echo -e To jest tekst\nnn
```

Wartości logiczne

W powłoce — tak jak w każdym innym języku programowania — występują wartości logiczne, czyli wartości `TRUE` lub `FALSE`. W systemie wartość `0` zawsze oznacza prawdę, czyli `TRUE`, a jakakolwiek inna wartość oznacza fałsz, czyli wartość `FALSE`.

Wszystkie programy działające w powłoce zwracają informację o tym, czy udało im się poprawnie zakończyć działanie. Wartość ta jest umieszczana w specjalnej zmiennej \$?.

```
#!/bin/bash
vdir /home/lukasz
echo $?
```

Program ten powinien na końcu wyświetlić liczbę określającą, czy udało mu się wyświetlić katalog, czy też nie.

```
[lukasz@localhost ~]$ ./skrypt
razem 44
drwx----- 5 lukasz lukasz 4096 gru 29 19:50 Desktop
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Dokumenty
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Muzyka
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Obrazy
-rw-r--r-- 1 lukasz lukasz 0 maj 24 13:04 pik.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik2.txt
-rw-r--r-- 1 lukasz lukasz 7 maj 24 17:30 plik.txt
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Pobieranie
-rwxrwxrwx 1 lukasz lukasz 39 cze 11 18:30 skrypt
-rwxrwxrwx 1 lukasz lukasz 34 cze 11 18:20 skrypt~
drwx----- 6 lukasz lukasz 4096 maj 24 13:07 tmp
drwxrwxr-x 2 lukasz lukasz 4096 lis 24 2005 Wideo
0
```

Jak widać powyżej, katalog został wyświetlony i dlatego program zwrócił wartość TRUE, czyli liczbę 0 na końcu kodu. W przypadku niepowodzenia zwróciłby wartość 1, tak jak poniżej.

```
[lukasz@localhost ~]$ ./skrypt
vdir: /home/lukasz2: Nie ma takiego pliku ani katalogu
1
```

Polecenie test

Polecenie test służy do porównywania liczb lub ciągów znaków i wypisywania do zmiennej wartości porównania.

-d

Parametr sprawdza, czy plik o podanej nazwie jest katalogiem.

```
#!/bin/bash
test -d plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

-f

Parametr sprawdza, czy plik jest zwykłym plikiem, czy też z prawami do wykonywania.

```
#!/bin/bash
test -f plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

-L

Parametr sprawdza, czy plik jest dowiązaniem symbolicznym.

```
#!/bin/bash
test -L plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

-r

Parametr sprawdza, czy dany plik istnieje i czy można go odczytać.

```
#!/bin/bash
test -r plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

-w

Parametr sprawdza, czy dany plik istnieje i czy można go zapisać.

```
#!/bin/bash
test -w plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

-x

Parametr sprawdza, czy plik o danej nazwie istnieje i czy można go uruchomić.

```
#!/bin/bash
test -x plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

-s

Parametr sprawdza, czy dany plik został zapisany na dysku i czy jego wartość (długość) nie jest zerowa.

```
#!/bin/bash
test -s plik.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

-nt

Parametr sprawdza, czy *plik1* jest nowszy od *plik2*.

```
#!/bin/bash
test plik.txt -nt plik2.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

-ot

Parametr sprawdza, czy *plik1* jest starszy od *plik2*.

```
#!/bin/bash
test plik.txt -ot plik2.txt
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

=

Parametr sprawdza, czy ciągi podane po jego obu stronach są identyczne.

```
#!/bin/bash
test "abc"="abc"
echo $?

[lukasz@localhost ~]$ ./skrypt
0
```

!=

Parametr sprawdza, czy ciągi podane po jego obu stronach nie są identyczne.

```
#!/bin/bash
test "abc"!="abc"
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

-z

Parametr sprawdza, czy ciąg podany za nim ma zerową długość.

```
#!/bin/bash
test -z abc
echo $?

[lukasz@localhost ~]$ ./skrypt
1
```

-n

Parametr sprawdza, czy ciąg podany za nim ma niezerową długość.

```
#!/bin/bash
test -n abc
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
0
```

-eq

Parametr sprawdza, czy wartości podane po jego obu stronach są sobie równe.

```
#!/bin/bash
test 1 -eq 1
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
0
```

-ne

Parametr sprawdza, czy wartości podane po jego obu stronach nie są sobie równe.

```
#!/bin/bash
test 1 -ne 1
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
1
```

-gt

Parametr sprawdza, czy pierwsza wartość jest większa od drugiej.

```
#!/bin/bash
test 2 -gt 1
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
0
```

-ge

Parametr sprawdza, czy pierwsza wartość jest większa lub równa drugiej.

```
#!/bin/bash
test 2 -ge 1
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
0
```

-lt

Parametr sprawdza, czy pierwsza wartość jest mniejsza od drugiej.

```
#!/bin/bash
test 2 -lt 1
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
1
```

-le

Parametr sprawdza, czy pierwsza wartość jest mniejsza lub równa drugiej.

```
#!/bin/bash
test 2 -le 1
echo $?
```

```
[lukasz@localhost ~]$ ./skrypt
1
!
```

Parametr ten służy do negowania testu, bardzo często stosuje się go w instrukcjach warunkowych i w pętlach.

Instrukcja if

Instrukcja sprawdza, czy dany warunek jest spełniony, i w zależności od tego wykonuje odpowiednie czynności.

```
if wartość
then
zrób coś
fi
Najprostszy wariant polecenia if
```

```
#!/bin/bash
if [ 1 = 1 ]
then
echo Wartosci sa rowne
fi
```

Przykład skryptu sprawdzającego, czy wartości są równe.

```
[lukasz@localhost ~]$ ./skrypt
Wartosci sa rowne
```

Ten warunek sprawdza, czy wartość jest spełniona, i w zależności od wyniku sprawdzenia wykonuje określoną operację (wartość spełniona — zrób coś, w przeciwnym wypadku — zrób coś innego).

```
if wartość
then
zrób coś
else
zrób coś innego
fi

#!/bin/bash
if [ 1 = 2 ]
then
echo Wartosci sa rowne
else
echo Wartosci sa rozne
fi
```

Przykład skryptu sprawdzającego, czy wartości są równe, czy też nie.

```
[lukasz@localhost ~]$ ./skrypt
Wartosci sa rozne
```

Polecenie sprawdza, czy warunki są spełnione. W przypadku, gdyby którykolwiek z warunków został spełniony, wykona polecenie.

W przypadku, gdyby żaden z warunków nie został spełniony, wykona ono funkcję zrób coś innego.

```
If wartość
then
zrób coś
elif wartość2
then
zrób coś 2
elif ...
...
else
zrób coś innego
fi
```

Instrukcja case

Instrukcja case sprawdza, czy warunek ma odpowiednią wartość, a następnie przechodzi do odpowiedniego fragmentu kodu w programie.

```
Case warunek in
odpowiedz1)
zrob coś 1
;;
Esac
```

Przykład skryptu sprawdzającego, jaką wartość ma liczba.

```
#!/bin/bash
wartosc=1
case "$wartosc" in
1)
echo Liczba ma wartosc 1
;;
2)
echo Liczba ma wartosc 2
;;
Esac

[lukasz@localhost ~]$ ./skrypt
Liczba ma wartosc 1
```

Pętla while

Pętla `while` powtarza wykonywanie określonych czynności dopóty, dopóki warunek w niej podany nie zostanie spełniony.

```
While polecenie
do
zrob coś
done
```

Oto skrypt z pętlą `while` wyświetlający liczbę i po każdym przejściu pętli (wykonaniu fragmentu skryptu) zwiększający ją o jeden. W przypadku, gdy liczba jest równa 2, skrypt kończy działanie.

```
#!/bin/bash
i=0
while [ $i -lt 2 ]
do
echo $i
i=`expr $i + 1`
done

[lukasz@localhost ~]$ ./skrypt
0
1
```

Pętla until

Pętla `until` powtarza polecenie w niej zapisane dopóty, dopóki warunek nie zostanie spełniony.

```
Until polecenie
do
zrob coś
done
```

Pętla for

Pętla `for` powtarza daną czynność żadaną liczbę razy (określoną w podanej do niej liczbie wykonania).

```
For zmienna in lista
do
zrob coś
done
```

Pętla `for`, która wyświetla w kolejnych liniach wszystkie wymienione w niej owoce.

```
#!/bin/bash
for owoc in Jablko Pomarancza Cytryna
do
echo $owoc
done
```

```
[lukasz@localhost ~]$ ./skrypt
Jablko
Pomarancza
Cytryna
```

Break

Polecenie `break` kończy działanie pętli, jeżeli podamy warunek do jej zakończenia.

```
#!/bin/bash
for owoc in Jablko Pomarancza Cytryna
do
echo $owoc
if [ "$owoc" = "Pomarancza" ]
then
break
fi
done
```

Ten skrypt ma zakończyć działanie (za pomocą polecenia `break`) po pojawieniu się tekstu Pomarancza.

```
[lukasz@localhost ~]$ ./skrypt
Jablko
Pomarancza
```

Continue

Polecenie `continue` wymusza przejście do kolejnej iteracji, czyli następnego kroku.

```
#!/bin/bash
for owoc in Jablko Pomarancza Cytryna
do
    echo $owoc
    if [ "$owoc" = "Pomarancza" ]
    then
        continue
    fi
    echo tekst przerywnika
done
```

W napisanym przez nas kodzie zadaniem instrukcji `continue` jest ukrycie napisu tekst przerywnika podczas przetwarzania „owocu”: Pomarancza.

```
[lukasz@localhost ~]$ ./skrypt
Jablko
tekst przerywnika
Pomarancza
Cytryna
tekst przerywnika
```

Argumenty pobierane z wiersza powłoki

Wszystkie napisane przez nas skrypty powłoki mogą przyjmować argumenty. Do argumentów wysłanych po uruchomieniu skryptu odwołujemy się za pomocą zmiennych `$1`, `$2`, `$3` ... `$n`.

```
#!/bin/bash
echo "Dzisiaj pogoda była $1"
```

Przykładowy skrypt pobierze pierwszą wartość za jego nazwą i wstawi ją w miejsce `$1`, a następnie wyświetli ją na ekranie.

```
[lukasz@localhost ~]$ ./skrypt super
Dzisiaj pogoda była super
```