Łukasz Sosna

Linux

KOMENDY I POLECENIA

Wydanie V

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/link5v
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-9061-4

Printed in Poland.

Spis treści

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. Każda nowa linia to nowe polecenie, więc nie można zapisywać polecenia w dwóch liniach.

Na przykład — aby wyświetlić listę zawartości swojego katalogu głównego, całe polecenie trzeba zapisać w jednej linii, ponieważ zapisanie go w dwóch liniach (lub większej ich liczbie) 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 pożądany 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 za pomocą polecenia `chmod` (opisywanego we wcześniejszej części książki).

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

W celu uruchomienia skryptu należy odpowiednio go wywołać. Zapiszmy skrypt w pliku `skrypt`. W takim przypadku 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 powłoce 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 powłoce 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 jej w innym miejscu skryptu cudzysłów zabezpiecza nas przed wystąpieniem błędu).

Zadeklarujmy zmienną o nazwie `zmienna` 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 zostałaby 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`

Zastosowanie tego parametru nie doprowadzi do wypisania 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` Za pomocą tego parametru można rozpoznać i interpretować 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` Zastosowanie tego parametru powoduje nieinterpretowanie znaków specjalnych we wpisywanym tekście i pominięcie ich wykonania.

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

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

`\a` Zastosowanie tego parametru powoduje pojawienie się alarmu w postaci sygnału dźwiękowego.

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

`\b` Wykorzystanie tego parametru po wypisaniu tekstu pozwala przesunąć kursor o jeden znak bliżej początku tekstu.

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

`\c` Zastosowanie tego parametru powoduje niewypisanie znaku nowego wiersza na końcu linii.

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

`\f` Zastosowanie tego parametru powoduje wysunięcie strony i zmianę miejsca kursora w tekście.

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

`\n` Zastosowanie tego parametru powoduje pojawienie się nowego wiersza po zakończeniu wypisywania tekstu.

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

`\r` Zastosowanie tego parametru powoduje powrót karetki do początku linii.

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

`\t` Zastosowanie tego parametru powoduje pojawienie się znaku tabulacji w poziomie.

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

`\v` Zastosowanie tego parametru 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\\
```

`\'` Zastosowanie tego parametru pozwala na wypisanie pojedynczego cudzysłowu.

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

`\"` Zastosowanie tego parametru pozwala na wypisanie podwójnego cudzysłowu.

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

`\nnn` Zastosowanie tego parametru pozwala na wypisanie znaku z tabeli kodów ASCII o notacji ósemkowej.

```
#!/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 $?
```

Zastosowanie tego programu powinno na końcu doprowadzić do wyświetlenia liczby określającej, czy powiodło się wyświetlenie zawartości katalogu, czy 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 wpisywania do zmiennej wartości porównania.

`-d` Za pomocą tego parametru sprawdzamy, czy plik o podanej nazwie jest katalogiem.

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

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

`-f` Za pomocą tego parametru sprawdzamy, czy plik jest zwykłym plikiem, czy też z prawami do wykonywania.

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

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

`-L` Za pomocą tego parametru sprawdzamy, czy plik jest dowiązaniem symbolicznym.

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

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

`-r` Za pomocą tego parametru sprawdzamy, czy dany plik istnieje i czy można go odczytać.

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

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

`-w` Za pomocą tego parametru sprawdzamy, czy dany plik istnieje i czy można go zapisać.

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

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

`-x` Za pomocą tego parametru sprawdzamy, czy plik o danej nazwie istnieje i czy można go uruchomić.

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

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

`-s` Za pomocą tego parametru sprawdzamy, 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` Za pomocą tego parametru sprawdzamy, czy *plik1* jest nowszy od *plik2*.

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

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

`-ot` Za pomocą tego parametru sprawdzamy, czy *plik1* jest starszy od *plik2*.

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

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

`=` Za pomocą tego parametru sprawdzamy, czy ciągi podane po jego obu stronach są identyczne.

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

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

`!=` Za pomocą tego parametru sprawdzamy, czy ciągi podane po jego obu stronach nie są identyczne.

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

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

`-z` Za pomocą tego parametru sprawdzamy, czy ciąg podany za nim ma zerową długość.

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

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

`-n` Za pomocą tego parametru sprawdzamy, czy ciąg podany za nim ma niezerową długość.

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

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

`-eq` Za pomocą tego parametru sprawdzamy, czy wartości podane po jego obu stronach są sobie równe.

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

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

`-ne` Za pomocą tego parametru sprawdzamy, czy wartości podane po jego obu stronach nie są sobie równe.

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

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

`-gt` Za pomocą tego parametru sprawdzamy, czy pierwsza wartość jest większa od drugiej.

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

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

`-ge` Za pomocą tego parametru sprawdzamy, czy pierwsza wartość jest większa od drugiej lub równa drugiej.

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

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

`-lt` Za pomocą tego parametru sprawdzamy, czy pierwsza wartość jest mniejsza od drugiej.

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

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

`-le` Za pomocą tego parametru sprawdzamy, czy pierwsza wartość jest mniejsza od drugiej 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

Zastosowanie instrukcji pozwala sprawdzić, czy dany warunek jest spełniony, i — w zależności od tego — wykonać 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, którego użycie pozwala sprawdzić, czy wartości są równe.

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

Wykorzystanie tego warunku pozwala sprawdzić, czy wartość jest spełniona, i — w zależności od wyniku sprawdzenia — wykonać 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, którego użycie pozwala sprawdzić, czy wartości są równe, czy nie.

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

Za pomocą polecenia sprawdzamy, czy warunki są spełnione. Jeśli którykolwiek z warunków zostanie spełniony, polecenie zostanie wykonane. Jeżeli zaś żaden z warunków nie zostanie spełniony, wykonana zostanie funkcja `zrób coś innego`.

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

Instrukcja case

Zastosowanie instrukcji `case` pozwala sprawdzić, czy warunek ma odpowiednią wartość, a następnie przejść do odpowiedniego fragmentu kodu w programie.

```
Case warunek in
odpoweidz1)
zrob coś 1
;;
Esac
```

Przykład skryptu, za pomocą którego można sprawdzić, 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`, którego wykorzystanie pozwoli wyświetlić liczbę i po każdym przejściu pętli (wykonaniu fragmentu skryptu) zwiększyć 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 żądaną liczbę razy (podaną w pętli).

```
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 potrzebny 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"
```

Zastosowanie przykładowego skryptu pozwoli pobrać pierwszą wartość za jego nazwą i wstawić ją w miejsce `$1`, a następnie wyświetlić ją na ekranie.

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

Tworzenie plików w pętli

Przy zastosowaniu na przykład pętli możemy w bardzo prosty i szybki sposób utworzyć pliki w naszym katalogu. Pętlę definiujemy jako zmienną `name` zawierającą się w przedziale od 1 do 25, po czym znajduje się kropka i rozszerzenie pliku. Zbiór danych można zastąpić dowolnym innym, na przykład literami od `a` do `z`. Wewnątrz pętli dla podanego pliku wykonujemy polecenie `touch`, które w przypadku braku pliku na dysku spowoduje jego utworzenie.

```
#!/bin/bash
for name in {1..25}.txt
do
    touch $name
done
```

Skorowidz


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Moc Linuksa w Twoich rękach!

- **Pracuj z systemem z poziomu linii poleceń**
- **Sprawdź, co zrobić w przypadku awarii interfejsu graficznego**
- **Twórz własne skrypty powłoki i ciągi instrukcji**
- **Przeprowadzaj operacje na plikach oraz katalogach za pomocą menedżera plików**

Wybór systemu operacyjnego, pod którym będzie pracował Twój komputer, w istocie sprowadza się do wyboru między standardowym produktem giganta z Redmond a niezawodnym, darmowym, fantastycznie elastycznym Linuksem. Ten ostatni system może być dla Ciebie niezastąpionym środowiskiem pracy, ale najpierw warto go trochę oswoić, a przede wszystkim nauczyć się z nim porozumiewać — nie tylko z poziomu interfejsu graficznego, lecz także z poziomu linii poleceń. Ta wiedza pozwoli Ci bez trudu wydawać nawet najbardziej skomplikowane polecenia, które Twój komputer spełni w lot. Sprawdź, a sam się przekonasz.

W tej książce znajdziesz najróżniejsze komendy i polecenia pozwalające na efektywną komunikację z komputerem. Zobaczysz, jak zainstalować Linuksa i poruszać się po tym systemie oraz tworzyć własne skrypty powłoki — na przykład w celu zautomatyzowania codziennie wykonywanych operacji. Dowiesz się także, jak używać programu Midnight Commander, optymalnie zarządzać zasobami komputera i sprawnie administrować systemem. Przejrzysty układ zapewni Ci łatwe wyszukiwanie określonych komend, a czytelne objaśnienia wraz z parametrami pomogą Ci właściwie je zastosować.

- **Wprowadzenie do systemu Linux**
- **Korzystanie z komputera pracującego pod kontrolą systemu Linux**
- **Zarządzanie zasobami komputera**
- **Administrowanie systemem**
- **Tworzenie skryptów powłoki**
- **Polecenia dodatkowe**
- **Midnight Commander**

Poczuj się w Linuksie jak ryba w wodzie!

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-9061-4

INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 37,00 zł