Mateusz Lach

BASH

PRAKTYCZNE SKRYPTY

Wykorzystaj Bash do swoich celów!

- Jak działa Bash, czyli do czego potrzebna Ci ta powłoka i jak ją inteligentnie wykorzystać
- Raport wydajnościowy, czyli jak szybko znaleźć pliki, które ktoś ostatnio zmodyfikował
- W małym kinie, czyli jak zaprojektować system rezerwacji miejsc

Helion

Redaktor prowadzący: Michał Mrowiec

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Fotografia na okładce została wykorzystana za zgodą Shutterstock.com

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

Kody źródłowe wybranych przykładów dostępne są pod adresem:
ftp://ftp.helion.pl/przyklady/bashps.zip

ISBN: 978-83-283-1489-4

Printed in Poland.

Spis treści

Rozdział 7.
Silnia

Od kilku rozdziałów omawiamy kolejne elementy Basha na podstawie przykładów matematycznych. Może się to wydawać trochę nudne w porównaniu do naszych pierwszych lekcji, które dotyczyły programowania na przykładzie rad dotyczących dobrego zachowania. Zapewniam jednak, że odrobina matematyki jeszcze nikomu nie zaszkodziła, a pozwala ona tworzyć dobrane pod względem dydaktycznym przykłady.

Na potrzeby tego rozdziału uczynię mały wyjątek i zaczniemy od małego wprowadzenia teoretycznego, ponieważ nie każdy Czytelnik musi wiedzieć lub pamiętać, czym jest silnia. Jest ona prostym działaniem matematycznym i zapisuje się ją w postaci ***n!***, gdzie *n* to dowolna liczba naturalna. Działanie to polega na obliczeniu iloczynu kolejnych liczb naturalnych, które są większe od 0 i nie większe on *n*, natomiast silnia z 0 z definicji wynosi 1, czyli:

$0! = 15$

$1! = 1$

$2! = 1 \cdot 2 = 2$

$3! = 1 \cdot 2 \cdot 3 = 6$

$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$

$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$

Przykład realizacji tego zadania w Bashu przedstawiam w listingu 7.1.

Listing 7.1. *Silnia — przykład realizacji w Bashu*

```
#!/usr/bin/bash

i=1 #nie-zero bo tworzy iloczyn
wynik=1 # nie-zero bo tworzy iloczyn
while (($i <= $1))
do
  wynik=$(($wynik * $i))
  i=$(($i + 1))
done
echo $wynik
```

W skrypcie z listingu 7.1 przedstawiam realizację zadania matematycznego przy użyciu jedynie znanych nam już mechanizmów i technik. W listingu 7.2 natomiast zostało wykonane to samo zadanie z zastosowaniem nieomówionej jeszcze notacji.

Listing 7.2. *Silnia — przykład realizacji z użyciem pętli for w języku Bash*

```
#!/usr/bin/bash

wynik=1 # nie-zero bo tworzy iloczyn
for ((i=1; $i <= $1; i=$i+1))
do
  wynik=$(($wynik * $i))
done
echo $wynikecho $wynik
```

Listing 7.2 przedstawia alternatywną wersję skryptu z listingu 7.1. Spis nowości zawiera tabela 7.1.

Tabela 7.1. *Nowe terminy z listingu 7.2*

Termin	Opis
`for (( ; ; ))`	`For` to słowo kluczowe powłoki Bash, odpowiednik `while`. Działa dokładnie tak samo, różni się jednak zapisem, gdyż jego warunek jest w tym wypadku wyrażenia matematycznego podzielony na trzy sekcje, które zostaną omówione w części teoretycznej. Możliwe jest jeszcze łączenie pętli `for` ze zbiorami. To również omówię w części teoretycznej.
`do`	Słowo kluczowe powłoki Bash. Rozpoczyna pętlę `for`, tak samo jak `while`.
`done`	Słowo kluczowe powłoki Bash. Kończy pętlę `for`, tak samo jak `while`.
`i++`	Skrócony zapis dla zwiększenia licznika zmiennej o 1. Zostanie omówiony bardziej szczegółowo w rozdziale teoretycznym.

Schemat blokowy dla skryptu z listingu 7.2 przedstawiam na rysunku 7.1.

Pętla for

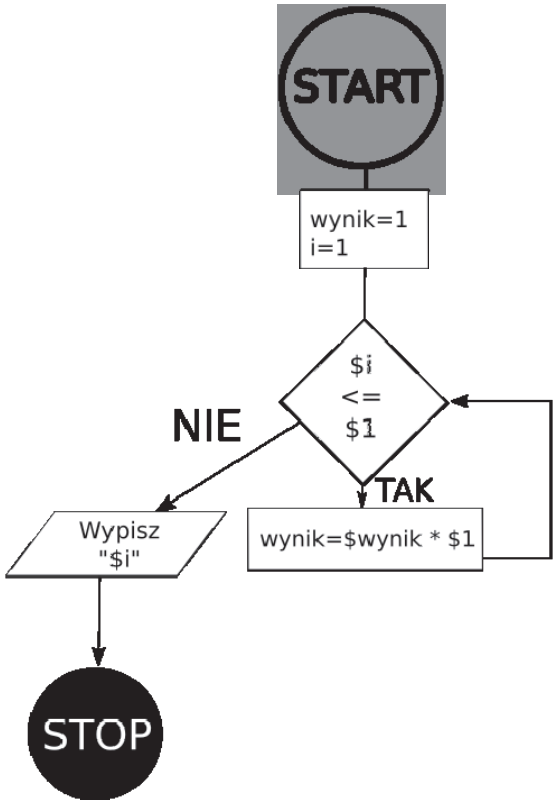
W Bashu istnieje kilka rodzajów instrukcji sterujących, które oferują wielokrotne wykonanie bloku kodu (tzw. pętle). Jedną z nich jest pętla `for`, która poza skróconym zapisem nie różni się niczym w porównaniu do pętli `while`.

Zapis arytmetyczny

Pętla `for` obsługuje kilka możliwości działania. Jedną z nich jest notacja wykorzystująca wyrażenia arytmetyczne. Ogólny zapis pętli w postaci pseudokodu zamieszczam poniżej:

```
dla (( instrukcje startowe; warunek wykonania pętli; instrukcje po każdym przebiegu))
wykonaj
    # blok pętli, który powinien zostać wykonany, jeśli warunek wykonania jest spełniony
koniec
```

Rysunek 7.1. *Schemat blokowy dla skryptu z listingu 7.2*

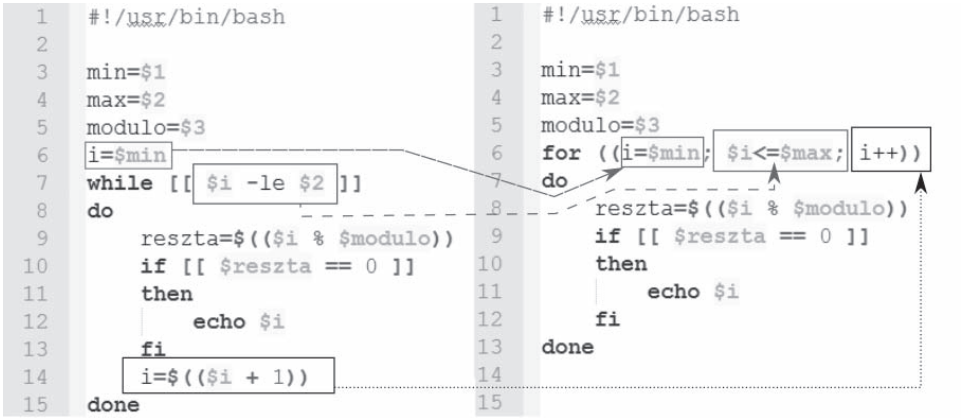


Na rysunku 7.2. znajduje się analiza porównawcza pętli `while` i `for`, której dokonałem na przykładzie skryptu z listingu 6.1 i stanowiącego jego alternatywną wersję skryptu z listingu 7.3.

Listing 7.3. *Skrypt z listingu 6.1 przepisany na wersję z użyciem pętli for*

```
#!/usr/bin/bash

min=$1
max=$2
modulo=$3
i=$min
for ((i=$min; $i<=$max; i++))
do
    reszta=$(($i % $modulo))
    if [[ $reszta == 0 ]]
    then
        echo $i
    fi
done
```



Rysunek 7.2. *Analiza porównawcza skryptu z listingu 6.1 (po lewej) i skryptu z listingu 7.3 (po prawej). Dotyczy ona różnic występujących między instrukcjami sterującymi for i while*

Na rysunku 7.2 widać różnice w zapisie między obiema pętlami. W mojej ocenie pętla `while` jest dużo bardziej czytelna dla początkujących programistów i spełnia w zasadzie wszystkie wymagania, realizując te same zadania co pętla `for` w zapisie arytmetycznym, jednakże w inny sposób. Jedyną zaletą, która przemawia za pętlą `for`, jest liczba linii kodu potrzebnych do realizacji tego samego zadania. Muszę jednak przyznać, że z biegiem czasu coraz częściej korzystam z pętli `for` zamiast `while`, choć nie potrafię logicznie uzasadnić tego wyboru.

Warto jeszcze wspomnieć, że instrukcje startowe, warunkowe i końcowe przebiegu pętli `for` są instrukcjami opcjonalnymi. Oznacza to, że nie musimy ich wpisywać. Musimy jedynie pamiętać o oddzielających je średnikach. Może to posłużyć do tworzenia nieskończonych pętli, co przedstawiam w listingu 7.4.

Listing 7.4. *Nieskończona pętla z użyciem pętli for*

```
#!/usr/bin/bash

for ((;;))
do
    echo "Wykonuję się dopóty, dopóki mnie nie zamkniesz"
done
```

Z pozoru taka instrukcja jest nieprzydatna, ponieważ do momentu, kiedy sami nie zamkniemy programu, na przykład używając skrótu *CTRL+C*, program nie zakończy działania. Wbrew pierwszemu wrażeniu ma to swoje zastosowanie, ale o tym opowiem w kolejnych rozdziałach.

Cały zapis pętli `for` w wersji arytmetycznej (poza częścią będącą samym wyrażeniem) złożony jest ze słów kluczowych:

```
$ type -a for do done "((" "))"
for jest słowem kluczowym powłoki
do jest słowem kluczowym powłoki
done jest słowem kluczowym powłoki
```

```
-bash: type: ((: nie znaleziono
-bash: type: )): nie znaleziono
$
```

i++ oraz i–, czyli skrócony zapis inkrementacji/dekrementacji

W Bashu, tak jak w innych popularnych językach programowania, dostępny jest skrócony operator zwiększenia wartości zmiennej (inkrementacji; ang. *incrementation*) oraz zmniejszenia (dekrementacji; ang. *decrementation*). Ich zapis wygląda następująco:

`i++` to zwiększenie wartości zmiennej `$i` o 1, skrócony zapis dla `i=$i+1`

`i--` to zmniejszenie wartości zmiennej `$i` o 1, skrócony zapis dla `i=$i-1`

W odróżnieniu od innych języków programowania, w Bashu nie można używać tych zapisów tak po prostu. Można je stosować jedynie w notacji matematycznej, co zaprezentowałem w poniższym przykładzie:

```
$ a=4
$ a++
-bash: a++: nie znaleziono polecenia
$ declare -i b=4
$ b++ # nie działa również dla zmiennej typu integer
-bash: b++: nie znaleziono polecenia
$
```

Zapewne zastanawiasz się, gdzie się podział ulubiony znak dolara, który powinien poprzedzać odwołanie do zmiennej. O tym, że jest on w tym wypadku zbędny, świadczy ten oto przykład:

```
$ c=4
$ $c++
-bash: 4++: nie znaleziono polecenia
$
```

Bash zgłasza błąd, że polecenie `4++` nie istnieje. Dzieje się tak dlatego, że Bash nie zna tego zapisu i wpierw próbuje rozwinąć zmienną jej wartością, a następnie wykonać polecenie, które powstało z wartości zmiennej i doklejonego napisu `++`. Spróbujmy udowodnić tę teorię:

```
$ a="echo "
$ a++
-bash: a++: nie znaleziono polecenia
$ $a++ # Bash wykonał polecenie echo i wypisał dwa plusy na ekran
++
$
```

Jak widać na powyższym przykładzie, Bash rozwinął zmienną i wykonał polecenie, które powstało z wyniku połączenia obu stringów. To właśnie dlatego na ekranie pojawiły się dwa plusy.

Używanie skróconego zapisu w Bashu jest możliwe jedynie w parze z wyrażeniami matematycznymi:

```
$ i=1
$ ((i++))
$ echo $i
2
$
```

Jednak również w tym przypadku nie wstawia się dolara przed nazwą zmiennej, gdyż efekt będzie taki sam jak w zapisie bez notacji arytmetycznej. Aby zrozumieć, dlaczego tak się dzieje, trzeba zrozumieć, jak działa interpreter. Najpierw podmieniane są wartości zmiennych, a następnie Bash próbuje wykonać polecenie. A oto inny dowód na tę teorię:

```
$ polecenie="echo Interpreter "
$ $polecenie"Bash"
Interpreter Bash
$
```

Innym wytłumaczeniem powodu, dla którego nie wstawiamy dolara, jest fakt, że zmieniamy wartość zmiennej numerycznej, zwiększając lub zmniejszając ją o 1. W pełnym zapisie wartość do zmiennej przypisujemy również bez znaku dolara, ponieważ ten stosujemy jedynie w celu odwołania do wartości zmiennej, np. `i=4` lub `i=$i+2`.

Ćwiczenie 7.1

Napisz rozwiązanie do listingu 6.1, używając pętli `for` zamiast `while`.

Propozycje rozwiązań znajdują się na końcu książki, w ostatnim rozdziale, zatytułowanym „Rozwiązania ćwiczeń”.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

ź

BASH

PRAKTYCZNE SKRYPTY

Powłoka Bash jest bardzo wygodnym narzędziem, pozwalającym na automatyzację wielu różnych czynności, kłopotliwych dla administratora systemu. Jest też uniwersalna: choć powstała jako powłoka dla systemów uniksowych, można używać jej także na komputerach wyposażonych w Windows. Jeśli chcesz sprawdzić, do czego warto użyć Basha, i zobaczyć, jak działa powłoka w konkretnych sytuacjach, koniecznie zajrzyj do tej książki. W przystępny, klarowny sposób omawia ona podstawowe zagadnienia związane z Bashem, a nade wszystko zawiera praktyczne, gotowe skrypty i liczne ćwiczenia.

W książce znajdziesz zestawy instrukcji pozwalających efektywnie wyszukiwać w dużej bazie określone pliki, sortować, wykonywać skomplikowane obliczenia, generować zbiory liczb, stosować funkcje rekurencyjne i sprawdzać poprawność konfiguracji sieci komputerowej. Każdy rozdział rozpoczyna się od krótkiego opisu konkretnego problemu, który można rozwiązać dzięki skryptowi Basha. Taki układ na pewno docenią wszyscy, którzy nie mają zbyt wiele czasu na poznawanie teorii, gdy natychmiast potrzebna jest im praktyka. Sprawdź, zastosuj i oszczędź swój czas!

- **Konwencje programistyczne**
- **Instalacja i konfiguracja środowiska pracy**
- **Zmienne, stringi, podstawowe polecenia**
- **Tryb interaktywny/konwersacyjny i wsadowy**
- **Instrukcje, operatory, pętle i tablice**
- **Parametry wejściowe, zmienne liczbowe i wyrażenia matematyczne**
- **Analiza zbioru plików pod kątem daty ich ostatniej modyfikacji**
- **Sortowanie liczb i funkcje**
- **Sprawdzanie poprawności konfiguracji sieci komputerowej**
- **Ciągi Fibonacciego**
- **Prosty kalkulator ze wsparciem dla arytmetyki**
- **Systemy rezerwacji miejsc**
- **Rekurencyjne wyszukiwanie dat w plikach**

Niech Twój komputer działa za Ciebie!

Helion

36826 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-1489-4

9 788328 314894

Informatyka w najlepszym wydaniu

cena: 59,00 zł