Bash - instrukcje warunkowe, pętle i funkcje

5 grudnia 2018

1 Instrukcje warunkowe

Wewnątrz skryptu może powstać potrzeba wykonania fragmentu kodu pod pewnym warunkiem. Np. chcemy wykonać inne fragmenty kodu gdy zmienna jest pusta, albo wykonywać inne instrukcje dla różnych rozszerzeń gdy działamy na plikach. Do tego celu służą instrukcje warunkowe *if* oraz *case*. W zależności od wartości logicznej wyrażenia (wyrażenie prawdziwe lub fałszywe - true/false) podejmują one decyzję które instrukcje powinny zostać wywołane. W bashu do określenia wartości wyrażenia służy program **test**, który jest wykonywany dla danego wyrażenia, zwracając jego wartość logiczną. Instrukcje warunkowe korzystają z tego programu do podejmowania decyzji. Jako analogia do codziennego życia: "Jak nie będzie padać to pójdziemy grać w piłkę, a jak będzie to zagramy w Fifę" - w momencie kiedy należy podjąć decyzję, sprawdzamy czy pada, i w zależności od odpowiedzi podejmujemy różne decyzje.

1.1 Program test

Program **test** jest prostym programem przyjmującym wyrażenie, dla którgo można określić czy jest prawdziwe, czy też fałszywe. Np. **test 5 -eq 4** sprawdza, czy 2 liczby całkowite są sobie równe (eq - equal). Ogólnie program sprowadza się do wywołaia **test EXPRESSION**, gdzie **EXPRESSION** jest wyrażeniem, dla którego określany jest status - prawda lub fałsz. Zmienne w bash mogą posiadać 3 typy - napis (STRING), liczba całkowita (INTEGER) oraz ścieżka do pliku (FILE). W zależności od użytego typu zmiennych mamy do dyspozycji różne testy, które możemy na nich wykonać (tworzyć wyrażenia). Wynik testu jest jednorazowo dostępny w specjalnej zmiennej **$?**. Przechowuje wartość 0 gdy test zwrócił prawdę (**true**) lub 1 gdy wartosc wyrażenia była fałszem **false**.

1.1.1 Testy dla zmiennych typu STRING

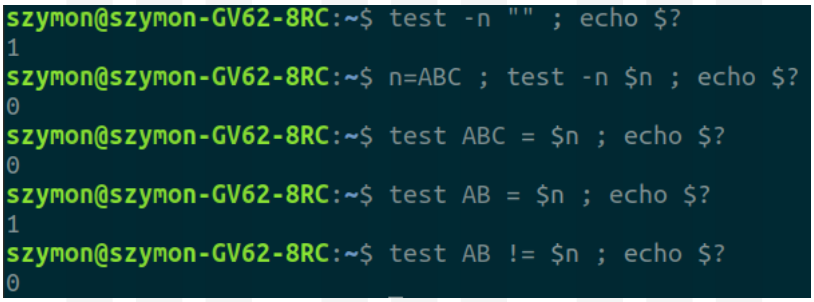
Przykłady wyrażeń które można utworzyć z napisów:

- **-n STRING** - Długość napisu jest różna od zera, jest różnoważne skróconej wersji **STRING**
- **STRING1 = STRING2** - Napisy są sobie równe (wszystkie znaki są takie same)

Rysunek 1: Domyślnie **$?** przechowuje prawdę (0). Zawsze po odwołaniu się do tej zmiennej zostaje ona automatycznie resetowana do domyślnej wartości. W przypadku gdy wyrażenie nie jest prawdziwe **$?** jest równe 1

- **STRING1 != STRING2** - Napisy są różne (Przynajmniej jeden znak jest różny)

UWAGA: spacja pomiędzy napisami jest ważna (wyrażenie typu ABC=ABC nie zadziała, musi być ABC = ABC)



Rysunek 2: “” jest pustym napisem. Wypisanie w jednej linii wielu komend oddzielonych zakiem “;” powoduje wywołanie ich po kolei

1.1.2 Testy dla zmiennych typu INTEGER

Przykłady wyrażeń które można utworzyć z liczb całkowitych:

- **INTEGER1 -eq INTEGER2** - Liczby **INTEGER1** oraz **INTEGER2** są sobie równe (eq - equal)
- **INTEGER1 -ne INTEGER2** - Liczby **INTEGER1** oraz **INTEGER2** są różne (ne - not equal)
- **INTEGER1 -qe INTEGER2** - Liczba **INTEGER1** jest większa lub równa liczbie **INTEGER2** (ge - greater or equal)

- **INTEGER1 -qt INTEGER2** - Liczba **INTEGER1** jest większa od liczby **INTEGER2** (gt - greater than)
- **INTEGER1 -le INTEGER2** - Liczba **INTEGER1** jest mniejsza lub równa liczbie **INTEGER2** (le - less or equal)
- **INTEGER1 -lt INTEGER2** - Liczba **INTEGER1** jest mniejsza od liczby **INTEGER2** (lt - less than)

```
szymon@szymon-GV62-8RC:~$ test 5 -le 5 ; echo $?
0
szymon@szymon-GV62-8RC:~$ test 5 -lt 5 ; echo $?
1
```

1.1.3 Testy dla zmiennych typu FILE

Przykłady wyrażeń które można utworzyć z liczb całkowitych:

- **-d FILE** - **FILE** istnieje i jest folderem (d - directory)
- **-f FILE** - **FILE** istnieje i jest plikiem (f - file)
- **-e FILE** - **FILE** istnieje (e - exists)
- **-s FILE** - **FILE** istnieje i rozmiar jest większy od 0 (s - size)

```
szymon@szymon-GV62-8RC:~/Kat1$ ls
Kat  plikcp.txt
szymon@szymon-GV62-8RC:~/Kat1$ test -d Kat ; echo $?
0
szymon@szymon-GV62-8RC:~/Kat1$ test -f Kat ; echo $?
1
szymon@szymon-GV62-8RC:~/Kat1$ test -f plikcp.txt ; echo $?
0
```

Dodatkowo możemy dodać symbol **!** przed wyrażeniem (**! EXPRESSION**) aby zanegować wynik wyrażenia (prawda staje się fałszem i na odwrót). Np. **test ! 5 -eq 4**.

```
szymon@szymon-GV62-8RC:~$ test 5 -eq 5 ; echo $?
0
szymon@szymon-GV62-8RC:~$ test ! 5 -eq 5 ; echo $?
1
```

1.2 Instrukcja warunkowa *if*

Składnia instrukcji warunkowej wewnątrz skryptu jest następująca:

```
if [ EXPRESSION ]
then
        # instrukcje gdy prawda
else
        # instrukcje gdy falsz
fi
```

Instrukcja *if* sprawdza wartość podanego wyrażenia wywołując program **test** z argumentami podanymi wewnątrz [ ], czyli wywołuje komendę **test EXPRESSION**. Jeżeli **test** zwraca *true*, wykonywany jest fragment kodu od *then* do *else*, w przeciwnym przypadku wykonane zostaną instrukcje wewnatrz *else* do *fi*. Instrukcja *else* jest opcjonalna. W przypadku krótkich fragmentow kodu można wykorzystać inną, krótszą formę: **[EXPRESSION] && CMDgdyTrue || CMDgdyFalse** . Instrukcje warunkowe można zagnieżdżać, to znaczy wewnątrz bloku *then ... else* może wystąpić kolejna instrukcja warunkowa. UWAGA: Spacje pomiędzy [(]) oraz EXPRESSION są obowiązkowe. W przypadku braku spacji (np. if [5 -eq 4]) [5 zostanie odczytane jako całość co spowoduje błąd w składni (syntax error).

iftest.sh

```
#!/bin/bash
if [ $1 -eq 2 ]
then
        echo "Dwa"
else
        echo "Nie dwa"
fi
```

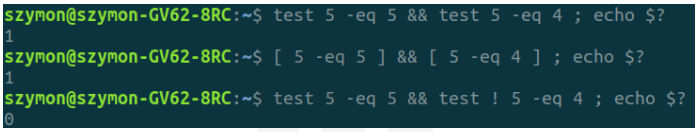
```
szymon@szymon-GV62-8RC:~$ ./iftest.sh 2
Dwa
szymon@szymon-GV62-8RC:~$ ./iftest.sh 3
Nie dwa
```

1.3 Operacje logiczne (Boolean operations)

Wyrażenia możemy łączyć w przypadku bardziej skompilikowanych warunków, np. *Wykonaj mnożenie jeżeli liczba jest mniejsza od 10 i większa od 0* lub *Wykonaj dodawanie jeżeli liczba jest mniejsza od 0 lub większa od 10.* W takich przypadkach nie wystarczy nam jedno wyrażenie, musimy użyć dwóch. Do tego celu służą 2 operatory:

1. **&&** - logiczne ”i” (and) - wartość wyrażenia **[EXPR1] && [EXPR2]** jest prawdą tylko gdy **oba** podwyrażenia **są prawdziwe**

2. || - logiczne ”lub” (or) - wartość wyrażenia [**EXPR1**] **&&** [**EXPR2**] jest prawdą gdy **przynajmniej jedno** z podwyrażeń **jest prawdziwe**



Rysunek 3: Wywołanie **test expr** jest równoważne wywołaniu [**expr**]

2 Pętle

Pętle służą do wykonywania podobnych (lub tych samych) instrukcji wielokrotnie. Np. Liczenie sumy 10 liczb jest wykonaniem dodawania 10 (lub 9) razy, możemy też chcieć wykonać jakieś instrukcje dla wszystkich plików z listy, czy też wypisać wszystkie elementy z tablicy spałniające jakiś warunek.

2.1 Pętla while

Składnia pętli while

```
while [ EXPRESSION ]
do
        # instrukcje dla petli
done
```

Instrukcje wewnątrz bloku **do ... done** są wykonywane tak długo, jak warunek zwraca prawdę. Przed każdym wykonaniem bloku **do ... done** sprawdzany jest warunek. Ważnym jest, aby **EXPRESSION** zwrócił w którymś momencie wartosc **false**, w przeciwnym wypadku będzie to pętla nieskończona (inifinite loop).

whileloop.sh

```
#!/bin/bash
i=1
while [ $i -lt 10 ]
do
        echo Inside loop i = $i
        ((i++))
done
echo Outside i = $i
```

```
szymon@szymon-GV62-8RC:~$ ./whileloop.sh
Inside loop i = 1
Inside loop i = 2
Inside loop i = 3
Inside loop i = 4
Inside loop i = 5
Inside loop i = 6
Inside loop i = 7
Inside loop i = 8
Inside loop i = 9
Outside i = 10
```

Rysunek 4: Wywołanie skryptu whileloop.sh

2.2 Pętla for

W języku bash pętla for jest bardziej skomplikowana

```
for VAR in LIST
do
        # instrukcje dla petli
done
```

W typ przypadku zmienna VAR (może to być dowolna nazwa, np. name, i, file) będzie przyjmować po kolei wartości z listy. Listę definiujemy jako napis oddzielony spacjami (np. list='1 2 3 4'). Przykład: **for a in 1 2 3 4** *a* będzie przyjmować kolejno wartości 1, 2, 3 oraz 4 w kolejnych przebiegach pętli. Możemy także użyć zakresu, tzn. **for a in {1..4}**. W takiej postaci otrzymujemy liczby od 1 do 4 z krokiem równym 1. Aby zmienić krok wykorzystujemy **for a in {1..10..2}** - wtedy od 1 do 10 z krokiem 2 - dostaniemy listę '1 3 5 7 9'. Innym narzędziem do zakresów jest program **seq**. Możemy go wykorzystać w następujący sposób **for a in $(seq 1 2 10)** aby uzyskać ten sam efekt. Zaletą seq jest możliwość wykorzystania zmiennej do ustalenia wartości początkowej, końcowej oraz kroku zapisu.

forloop.sh

```
#!/bin/bash
list='1 2 0 10'
for l in $list
do
        echo Inside first loop l = $l
done

for a in {1..9..3}
do
        echo Inside second loop a = $a
```

```
done

for b in $(seq $1 $2 $3)
do
        echo Inside third loop b = $b
done
```

```
szymon@szymon-GV62-8RC:~$ ./forloop.sh 1 3 9
Inside first loop l = 1
Inside first loop l = 2
Inside first loop l = 0
Inside first loop l = 10
Inside second loop a = 1
Inside second loop a = 4
Inside second loop a = 7
Inside third loop b = 1
Inside third loop b = 4
Inside third loop b = 7
szymon@szymon-GV62-8RC:~$ ./forloop.sh 1 "" 2
Inside first loop l = 1
Inside first loop l = 2
Inside first loop l = 0
Inside first loop l = 10
Inside second loop a = 1
Inside second loop a = 4
Inside second loop a = 7
Inside third loop b = 1
Inside third loop b = 2
szymon@szymon-GV62-8RC:~$
```

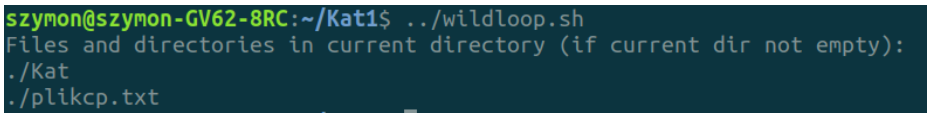
Rysunek 5: Wywołanie skryptu forloop.sh

2.2.1 Wildcards w petli for

Możemy w bardzo łatwy sposób dostać listę plików wykorzystując wildcards. Prosty przykład **for a in $1/*** - a będzie przyjmować wzgledne ścieżki do wszystkich plików wewnątrz folderu danego jako pierwszy argument wywołania skryptu.

wildloop.sh

```
#!/bin/bash
echo "Files and directories in current directory\
 (if current dir not empty):"
for a in ./*; do
        echo "$a "
done
```



Rysunek 6: Wywołanie skryptu wildloop.sh, który znajduje się w folderze nadrzędnym do Kat1

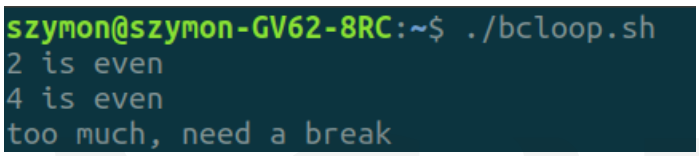
2.3 Instrukcje break i continue

Istnieją 2 specjalne instrukcje które można wywołać wewnątrz pętli:

1. **break** - przerywa działanie pętli w miejscu wystąpienia break, przechodzi do instrukcji znajdującej się za pętlą (za instrukcją **done**)
2. **continue** - przerywa aktualną iterację pętli przechodząc do następnej iteracji, tak samo jak w przypadku dotarcia do instrukcji **done**

bcloop.sh

```
#!/bin/bash
list='1 2 3 4 5 6 7 8'
for a in $list ;do
        temp=$((a % 2))
        if [ $temp -eq 0 ]
        then
                echo "$a is even"
                continue
        fi
        if [ $a -eq 5 ]
        then
                echo "too much, need a break"
                break
        fi
done
```



Rysunek 7: Wywołanie skryptu bcloop.sh

2.4 Zadania

Zadanie 1 Skrypt przyjmujący dwa argumenty a i b - dwie liczby, który porówna je ze sobą oraz wypisze na ekran jeden z komunikatów: "Equal" - gdy liczby są sobie równe, "$a < b$" gdy a jest mniejsze od b oraz "Nie wiem" w innych przypadkach.

Zadanie 2 Skrypt wypisujący liczby od 1 do 10 w jednej linii na dwa sposoby: korzystając z listy oraz zakresu.

Zadanie 3 Skrypt przyjmujący 3 zmienne - początek, koniec oraz krok zakresu, który zapisze do pliku wartości z zakresu podniesione do kwadratu.

Zadanie 4 Pierwszy skrypt tworzy plik w którym zapisane zostanie n (podane jako argument) losowych liczb od 0 do 10 (jedna liczba **$(($RANDOM % 11))**) w jednej linii, jeżeli wylosowana zostanie 10 powinna zostać zapisana do pliku, a skrypt powinien zakończyć działanie. Drugi skrypt powinien wczytać nazwę pliku, przejść po wszystkich liczbach oraz policzyć ich sumę.

Zadanie 5 Skrypt pytający użytkownika, czy na pewno wywołać skrypt, dostępne odpowiedzi to **y** oraz **n**. W przypadku opcji **y** użytkownik powinien zostac przywitany, a następnie skrypt powinien kopiować wszystkie pliki z z aktualnego katalogu, które mają rozmiar większy od zera, dodając przedrostek cpy_ do każdego skopiowanego pliku.

Zadanie 6 Skrypt zapisujący do pliku tabliczkę mnożenia od 1 do 10. Zamiast liczb parzystych powinno być wpisane 0.

2.5 Rozwiązania

Zadanie 1

```
#!/bin/bash
a=$1    # inicjalizowanie zmiennych
b=$2

if [ $a -eq $b ]        # czy a jest rowne b
then
        echo 'Equal'
else    # jezeli nie to
        if [ $a -lt $b ] # czy a jest mniejsze
        then    # gdy tak
                echo $a '<' $b
        else    # jezeli nie to
                echo 'Nie wiem'
        fi
fi
```

Zadanie 2

```
#!/bin/bash
list='1 2 3 4 5 6 7 8 9 10'     # inicjalizowanie listy
for a in $list # petla po elementach listy
do
    echo -n "$a " #wypisanie elementow
done
echo    # nowa linia

# uzycie tablic, tylko dodatkowo
#list=(1 2 3 4 5 6 7 8 9 10)
#for a in ${list[@]}
#do
#    echo -n "$a "
#done
#echo

for a in {1..10}    # zakres 1-10
do
    echo -n "$a "
done
echo

for a in $(seq 1 10)    # zakres 1-10
do
    echo -n "$a "
```

```
done
echo
```

Zadanie 3

```
#!/bin/bash
rm -f plik.dat # usuwanie pliku aby poczatkowo byl pusty
touch plik.dat # nowy pusty plik
for a in $(seq $1 $3 $2) # lista od $1 do $2 z krokiem $3
do
    a2=$(($a*$a)) # przypisanie do tymczasowej zmiennej
    echo -n "$a2 " >> plik.dat # zapis do pliku
done
```

Pierwszy

Zadanie 4

```
#!/bin/bash
n=$1
i=1
rm -f plik.dat
touch plik.dat
while [ $i -le $n ] # dopoki i <= n
do
    ((i++))     # zwieksz i o jeden
    rnd=$(($RANDOM % 11))     # wylosuj liczbe
    echo -n "$rnd " >> plik.dat #zapisz do pliku
    if [ $rnd -eq 10 ] # gdy byla to 10
    then # to
        break     # zakoncz petle
    fi
done
```

Drugi

```
#!/bin/bash
numbers=$(cat plik.dat) # przypisz zawartosc pliku
sum=0   # inicjalizowanie sumy zerem
for a in $numbers # dla wszystkich liczb
do
    let "sum=$sum+$a" #powieksz sume o ta liczbe
done
echo $sum
```

Zadanie 5

```
#!/bin/bash
read -p 'Execute script? (y/n) :' p
```

```
if [ $p = 'y' ] # gdy uzytkownik wybral y
then
    for a in * # dla wszystkich elementow w aktualnym folderze
    do
        if [ -f $a ] && [ -s $a ] # gdy a jest plikiem i rozmiar > 0
        then
            cp $a cpy_$a # skopiuj a dodajac przedrostek
        fi
    done
fi
```

Zadanie 6

```
#!/bin/bash
rm plik.dat
touch plik.dat
for a in {1..10} # dla liczb od 1 do 10
do
    for b in {1..10} # tak samo
    do
        mn=$[$a*$b] # wyznacz iloczyn
        if [ $(($mn%2)) -eq 0 ] # jezeli reszta z dzielenia 0
        then
            echo -n "0 " >> plik.dat # zapisz 0
        else
            echo -n "$mn " >> plik.dat # zapisz wartosc mnozenia
        fi
    done
    echo "" >>plik.dat
done
```