Programowanie w powłoce bourne'a

Spis treści:

Parametry z jakimi można uruchomić skrypt :
Konstrukcja set -parametr

-a	Wszystkie zmienne będą eksportowane.
-c "ciag komend"	Komendy czytane z pliku.
-e	Non-interactive mode.
-f	Zablokuj kreację pliku przez shell.
-h	Zlokalizuj i zapamiętaj zdefiniowane funkcje .
-i	Interactive mode.
-k	Argumenty w środowisku do wykonania.
-n	Czytanie komend bez ich wykonywania.
-r	Restricted mode.
-s	Komendy czytane z wejścia.
-t	Pojedyncza komenda zostanie wykonana a potem wyjście z shell'a.
-u	Nieustawione zmienne będą błędami przy podstawianiu.
-v	Verbose mode
-x	Śledzenie wykonywania programu.

Zmienne środowiskowe:

CDPATH	Ścieżka przeszukiwana komendą cd.
HOME	Katalog domowy użytkownika.
IFS	Separator pól - zwykle space, tabulator, i znak nowej lini.
MAIL	Ścieżka do specjalnego pliku (mail box), używana przez e-mail.
PATH	Ścieżka przeszukiwana przy wykonywaniu pliku.
PS1	Pierwszy znak zachęty - zwykle :$.
PS2	Drugi znak zachęty - zwykle : >.
TERM	Typ terminala .

Korzystanie z argumentów skryptu (zmienne specjalne):
Argumenty dostarczamy wykonując skrypt :
$ nazwaskryptu argument1 argument2 argument3 ...

$#	Liczba argumentów dostarczonych do programu.
$-	Flagi z jakimi uruchomiono skrypt komendą set.
$?	Status błędu ostatnio wykonanej komendy.
$$	Numer procesu aktualnego shell'a.
$!	Numer procesu aktualnego podprocesu.
$@	Wszystkie argumenty w ciągu.
$*	Wszystkie argumenty w ciągu.
$*n*	Argumenty w tablicy,gdzie 'n' jest numerem argumentu.
$0	Nazwa aktualnego shell'a.

Znaki specjalne odwzorowujące nazwy plików:

*	Jakikolwiek ciąg
?	Pojedynczy znak
[-,!]	Ranga , lista lub nie taki

Przekierunkowywanie wyjścia/wejścia plików:

>(file)	Przekierowanie wyjścia do pliku.
>>(file)	Dopisanie wyjścia do pliku.
<(file)	Przekierowanie wejścia z pliku.
;	Oddziela komendy.
|	Połączenie wyjścia 1 shell'a z wejściem 2-go.
&	Uruchomienie w tle(jeśi na końcu komendy).

"	Zastępowanie komendy - wyjście traktowane jako argumenty.

Przykłady : foo="hello there \$name"
Zmienna foo będzie miała wartość hello there $name dzięki znakowi \
foo='hello there $name'
Zmienna foo będzie miała wartość hello there $name dzięki nawiasom '.'
foo=`ls -l | fgrep Jul`
wyrażenie ls -l | fgrep Jul zostanie zinterpretowane a wynik zostanie przekazany jako nowa wartość zmiennej foo

---

Słowa kluczowe shellu borne'a:

if	for	then	while	else	until	elif	do	fi	done	case	{	esac	}

```
if instrukcja_1
then
  instrukcja_2
  instrukcja_3
fi
instrukcja_4
```

```
if instrukcja_1
then
   instrukcja_2
   instrukcja_3
else
   instrukcja_4
   instrukcja_5
fi
```

```
case value in
   pattern1)
      instrukcja
      instrukcja;;
   pattern2)
      instrukcja
      instrukcja;;
   ...
   patternn)
      instrukcja;
esac
```

Przyklad:

```
read CHOICE
case "$CHOICE" in
   1 | R ) echo "Restoring..."
```

```
        cpio -i </dev/rmt0;;
  2 | B ) echo "Archiving..."
        ls | cpio -o >/dev/rmt0;;
  3 | U ) echo "Unloading..."
        ls | cpio -o >/dev/rmt0;;
  *) echo "Sorry, $CHOICE is not a valid choice"
     exit 1
esac
```

```
while instrukcja
do
  instrukcja
  instrukcja
  ...
  instrukcja
done
```

Przykład:

```
# squares - prints the square of integers in succession
int=1
while [ $int -lt 5 ]
do
  sq='expr $int \* $int'
  echo $sq
  int='expr $int + 1'
done
echo "Job Complete"

$ squares
1
4
9
16
Job Complete
$
```

```
until instrukcja
do
  instrukcja
  instrukcja
  ...
  instrukcja
done
```

Przykład:

```
# shifter
until [ $# -eq 0 ]
do
  echo "Argument is $1 and 'expr $# - 1' argument(s) remain"
  shift
done

$ shifter 1 2 3 4
Argument is 1 and 3 argument(s) remain
Argument is 2 and 2 argument(s) remain
Argument is 3 and 1 argument(s) remain
Argument is 4 and 0 argument(s) remain
$
```

```
for variable in arg1 arg2  ... argn
do
  instrukcja
  ...
  instrukcja
done
```

Przykład:

```
# sumints - a program to sum a series of integers
#
if [ $# -eq 0 ]
then
  echo "Usage: sumints integer list"
  exit 1
fi
sum=0
for INT in $*
do
  sum=´´expr $sum + $INT´´
done
echo $sum
```

```
while true
do
  instrukcja
  ...
  instrukcja
done
```

```
until false
do
  instrukcja
  ...
  instrukcja
done
```

Funkcje :

```
funcname ()
{
  instrukcja
  $1 - pierwszy parametr funkcji
  Pozostałe parametry jak przy parametrach skryptu
  ... _
  instrukcja;
}
```

Konstrukcje programistyczne:

case wciąg1 in ciag1) instrukcja;; ciag2) instrukcja;; *) instrukcja;; esac	wciąg1 jest porównywany z ciag1 i ciag2. Jeśli odpowiadają sobie odpowiednie instrukcje zostaną wykonane aż do dwóch średników.Jeśli wciąg1 nie odpowiada żadnej możliwości wykonane zostaną instrukcje po gwiazdce.
for zmienna1 [in list] do instrukcja done	Pętla jest wykonywana dla każdego elementu w liście list.Lista może być zmienną zawierającą kilka fraz oddzielanych spacją lub może być listą wartości wpisywanych bezpośrednio w wyrażenie. Wyrażenie [in list] jest opcjonalne. Jeśli nie jest obecne pętla jest wykonywana dla każdej zmiennej wejściowej skryptu (parametru skryptu).
if [wyrażenie1]; then instrukcje elif [wyrażenie2]; then instrukcje else instrukcje fi	Wyrażenie1 jest obliczane i jeżeli stanowi prawdę to instrukcje po then są wykonywane .Następnie wyrażenie2 jest obliczane i jeżeli stanowi prawdę to instrukcje po drugim then są wykonywane . Jeśli wyrażenie1 i wyrażenie2 nie zwróci prawdy instrukcje po else są wykonywane. Wyrażenia elif and else są opcjonalne.
(instrukcje)	Wykonuje instrukcje w subshell'u.

{ instrukcje;}	Wykonuje instrukcje w obecnym shell'u.
name () { instrukcje;}	Definicja funkcji .
while [wyrażenie] do instrukcje done	Instrukcja while powtarza wykonywanie instrukcjii między do i done dopuki ostatnia instrukcja w wyrażeniu jest prawdą, w przeciwnym przypadku pętla jest przerywana.
until [wyrażenie] do instrukcje done	powtarza wykonywanie instrukcjii między do i done dopuki ostatnia instrukcja w wyrażeniu jest fałszem, w przeciwnym przypadku pętla jest przerywana.

Wynik operacji arytmetycznych uzyskamy dzięki dyrektywnie expr

expr zmienna1 operator zmienna2

zwróci wynik działania operator na zmiennych 1 i 2.

+	Dodawanie.
-	Odejmowanie.
*	Mnożenie.
/	Dzielenie.
%	Reszta z dzielenia.

Używanie zmiennych:

$zmienna	Wartość zmiennej; nic jeśli zmienna jest nie zdefiniowana
${zmienna}	to samo; Używa się takiej konstrukcji gdy wartość zmiennej ma być przed jakimś ciągiem
${zmienna-obiekt}	Wartość zmiennej jeśli zdefiniowana; w przeciwnym wypadku obiekt; zmienna pozostaje niezmieniona.
${zmienna=obiekt}	Wartość zmiennej jeśli zdefiniowana; w przeciwnym wypadku obiekt. Jeśli nie jest zdefiniowana, $zmienna jest usawiana na obiekt
${zmienna?komunikat}	Wartość zmiennej jeśli zdefiniowana; w przeciwnym wypadku wyświetlkomunikat i wyjdź z shell'a. Jeśli komunikat pusty, wyświetli: zmienna: parameter not set
$zmienna{zmienna+obiekt}	obiekt jeśli $zmienna zdefiniowana, w przeciwnym wypadku nic

---

Specjalne znaki wyświetlane przez konsolę:

\b	Backspace

\c	Linia bez znaku nowej lini
\f	Form Feed: nowa strona lub nowy ekran
\n	Nowa linia
\r	Powrót karetki
\t	Tab
\v	Vertical Tab
\\	Backslash
\0nnn	Jedno ,dwu lub trzy cyfrowa ósemkowa reprezentacja znaku ASCII

Porównywanie ciągów

str1 = str2	Prawda jeśli str1jest takiej samej długości i posiada te same znaki jak str2
str1 != str2	Prawda str1 nie jest taki sam jak str2
-n str1	Prawda jeśli długość str1 jest większa niż 0 (nie jest pustym ciągiem)
-z str1	Prawda jeśli str1 jest pusty (ma długość 0)
str1	Prawda jeśli *str1* nie jest pusty

Porównywanie zmiennych liczbowych:

int1 -eq int2	Prawda jeśli *int1* jest równe (equal) *int2*
int1 -ne int2	Prawda jeśli *int1* nie jest równe (not equal) *int2*
int1 -gt int2	Prawda jeśli *int1* jest większe (greater than) od*int2*
int1 -ge int2	Prawda jeśli *int1* jest większe lub równe (greater or equal) *int2*
int1 -lt int2	Prawda jeśli *int1* jest mniejsze (less than) niż *int2*
int1 -le int2	Prawda jeśli int1 jest mniejsze lub równe (less or equal) int2

Testowanie plików:
Konstrukcja: if [! operator(-r,-w,-d) filenm]

-r filenm	Prawda jeśli użytkownik ma prawo do czytania pliku
-w filenm	Prawda jeśli użytkownik ma prawo do pisania do pliku
-x filenm	Prawda jeśli użytkownik ma prawo do wykonywania pliku
-f filenm	Prawda jeśli filenm jest zwykłym plikiem
-d filenm	Prawda jeśli filenm jest katalogiem
-c filenm	Prawda jeśli filenm jest specjalnym plikiem znakowym
-b filenm	Prawda jeśli filenm jest specjalnym plikiem blokowym
-s *filenm*	Prawda jeśli rozmiar filenm jest różny od zera
-t fnumb	Prawda jeśli użądzenia związane z plikiem fnumb (1 orginalnie) jest terminalem

---

Warunkowe uruchomienie poleceń:
instrukcja && instrukcja - wykona 2 komende tylko jeśli 1-sza zostanie poprawnie wykonana.
komenda1 || komenda2 - wykona 2 komende tylko jeśli 1-sza **nie** zostanie poprawnie wykonana.

Przykłady skryptów:
Pytanie o kontynuację:

```
echo "Do you want to continue: Y or N \c"
read ANSWER
if [ $ANSWER = N -o $ANSWER = n ]
```

Wyświetlenie argumentów z którymi uruchomiono skrypt:

```
until [ $# -eq 0 ]do
  echo "Argument is $1 and 'expr $# - 1' argument(s) remain"
  shift
done
```

Sprawdzenie ,czy jest podłączony użytkownik:

```
if tty -s; then
echo Enter text end with \^D
fi
```

Manipulacja z ciagami :

```
   TIME=`date | cut -c12-19`

   TIME=`date | sed 's/.* .* .* \(.*\) .* .*/\1/'`

   TIME=`date | awk '{print $4}'`

   TIME=`set \`date\`; echo $4`

   TIME=`date | (read u v w x y z; echo $x)`
```