Wywołanie skryptu

sh nazwa_skryptu
. nazwa_skryptu
./nazwa_skryptu (musi mieć atrybut wykonalności)
nazwa_skryptu (j/w, musi być w domyślnie przeszukiwanej ścieżce)

Zmienne

$0 - nazwa skryptu
$n – n-ty argument, n<10
${n} – n-ty argument skryptu, n dowolne
$* - '$1 $2 ... $n'
$@ - '$1' '$2'' ... '$n'
$# - liczba argumentów
$- - obowiązujące parametry powłoki
$? - exitcode ostatniego polecenia
$$ - PID aktualnej powłoki
$! - pid ostatnio wywołanego polecenia

program **test**

test [opcja] – zwraca exitcode 0 gdy jest spełniony warunek wynikający z opcji i exitcode 1 w pozostałych przypadkach

opcje:

-r *plik* – plik istnieje i ma atrybut czytania
-w *plik* – plik istnieje i ma atrybut pisania
-x *plik* – plik istnieje i jest wykonywalny
-f *plik* – plik istnieje i jest zwykłym plikiem
-d *plik* – plik istnieje i jest zwykłym katalogiem
-c *plik* – plik istnieje i jest plikiem specjalnym znakowym
-b *plik* – plik istnieje i jest plikiem specjalnym blokowym
-p *plik* – plik istnieje i jest buforem typu FIFO
-u *plik* – plik istnieje i ma ustawiony bit SetUId
-g *plik* – plik istnieje i ma ustawiony bit SetGId
-s *plik* – plik istnieje i nie jest pusty

-z *słowo* – długość łańcucha znaków słowo jest równa 0
-n *słowo* – długość łańcucha znaków słowo jest różna od 0
słowo1=słowo2 – łańcuchy słowo1 i słowo2 są równe
słowo1!=słowo2 – łańcuchy słowo1 i słowo2 są różne
słowo – łańcuch słowo nie jest pusty
liczba1 -op_mat liczba2 – bada relację liczb, gdzie *op_mat*:
-eq - =
-ge - >=
-gt - >
-le - <=
-lt - <
-ne - !=
wyrażenia można łączyć operatorami:
! - negacja
-a – AND
-o – OR
i łączyć nawiasami () (trzeba je maskować!)

Instrukcja wyboru

case tekst in
wzorzec1**)** lista_komend ;;
{ wzorzec2**)** lista_komend ;;
....}
esac
wzorzec – łańcuch znakowy, jeśli * to pasuje do wszystkich pozostałych wzorców
!po ostatnim wzorcu nie dajemy podwójnego średnika
np.

```
read x
case $x in
t) echo wybrales x;;
n) echo wybrałeś n;;
*) echo wybrales cos innego
esac
```

Użyteczne polecenia

read ***zm1 {... zmN}*** **–** wczytuje wiersz z stdin i przypisuje zmiennym *zmN* kolejne pola wiersza
eval *arg1 ... argN* – argumenty są łączone i wykonywane jako polecenie
$((...)) - to samo co eval, lub działanie matematyczne
`*ciąg_komend*` - *ciąg_komend* jest wykonywany a jego wynik wstawiany w miejsce polecenia `...`
$(...) - jak `...`, umożliwia zagnieżdżenia
exec [komenda] – zastępuje komendą bieżący proces (bez tworzenia nowego procesu). Jeżeli komenda jest nieobecna, to przekierowania dotyczą bieżącej powłoki

Funkcje użytkownika

definiowanie:
nazwa_funkcji **(){**
definicja
}
w definicji można odwoływać się do parametrów przez $1 ... $n
użycie:
nazwa_funkcji parametr1 ... parametr n
usuwanie:
unset *nazwa_funkcji*

np.

```
sorted_ls (){
for x in $@
do
ls -l $x| sort -rn -k5
done
}

sorted_ls /home /usr
```

Warunki logiczne

if *ciąg_komend_1*
then *ciąg_komend_2*
else *ciąg_komend_3*
fi
ciąg_komend_2 – wykonywany, jeśli *ciąg_komend_1* zakończył się z exitcode=0
ciąg_komend_3 – wykonywany w pozostałych przypadkach

lub
if *c_k_1*;**then** *c_k_2*;**else** *c_k_3*;**fi**
np.

```
if test '$x'
  then
    echo a;echo b
    echo c
  else
    exit 0
fi
if find cp aa bb;then echo ok;fi
```

ciąg_komend1 **&&**ciąg_komend1
ciąg_komend1 **||**ciąg_komend2
np.

```
test -f x &&echo jest x||echo nie ma x
```

Pętle

while *warunek*
do
...
...
done
until *warunek*
do
...
...
done
gdzie *warunek* jest spełniony dla exitcode 0 i niespełniony dla innych
for *zmienna* **in** *tekst*
do
...
...
done
for ((x=0;x<10;x++**))**
do
...
done
np.

```
x=0&&while test x -lt 10;do x=$((x+1))&&echo $x;done
x=0&&until test x -gt 10;do x=$((x+1))&&echo $x;done
for x in 0 a a13;do echo $x;done
for x in `ls`;do echo $x;done
for ((x=0;x>35;x++));do echo $x;done
```

break [n] – wyjście z [n] pętli