

 [Index](#) >> [Aktykuły](#) >> **Kurs Basha** Autor: [Itros](#) 01-marca-2007r.

---

**Poziom trudności:** 

---

...: Kurs basha :...

- 0 - Wstep
- 1 - Początek
- 2 - Słowa zastrzeżone
- 3 - Zmienne oraz cytowanie
- 4 - Read
- 5 - Funkcje
- 6 - Obliczenia matematyczne
- 7 - Instrukcje warunkowe
- 8 - Pętle
- 9 - Operacje na plikach
- 10 - Polecenia tr, cut, wc, exec

## 0 - Wstep

Co to jest w ogóle bash ? Bash (Bourne Again Shell) - jest to jedna z wielu powłok występujących w systemach unixowych, która łączy w sobie zalety ksh i csh. Głównymi jej programistami byli Briana Fox i Cheta Ramey. Poza tym bash jest także językiem skryptowym.

## 1 - Początek

Na początek wpisujemy w konsoli:

```
bash-2.05b$ touch nazwa_skryptu
```

teraz jak już mamy plik w którym będziemy pisać nasz skrypt. Następnie edytujemy ten plik jednym z naszych ulubionych edytorów (scite, nedit, vi).

```
#!/bin/bash
#Jakis komentarz
echo "Hello world"
```

**#!** - wskazuje powłokę w której ma zostać wykonany nasz skrypt w naszym wypadku jest to /bin/bash

**#** - oznacza komentarz, wszystko co znajduje się po haszu jest pomijane przez interpreter basha.

**echo** - jest to polecenie wyświetlające dany napis na ekranie w naszym przypadku

```
"Hello world"
```

Polecenie echo moze takze zapisywac wyswietlony napis do pliku w nastepujacy sposob:

```
echo "Hello world" > nazwa_pliku - to polecenie czysci zawartosc podanego pliku i zapisuje w nim "Hello world" lub w przypadku nie istnienia pliku tworzy go.
```

```
echo "Hello world" >> nazwa_pliku - to polecenie dospiuje na samym koncu pliku podany przez nas ciag znakow nie kasujac zawartosci pliku.
```

lub takze wystepowac z taiki parametrami jak :

```
* -n nie jest pokazany znak na koncu wiersza
* -e wlancza mozliwosc dodawania znakow:
  \a - alert
  \b - backspace
  \c - to samo co z opcja -n
  \e - escape
  \f - line feed
  \n - nowa linia
  \r - powrot karetki (\f\r to jest to samo co \n)
  \t - tabulacja pozioma
  \v - tabulacja pionowa
  \' - apostrof
  \\ - backslash
  \nnn - znak ascii o wartosci osemkowej
  \xnn - znak ascii o wartosci szesnastkowej
```

No to juz gdy mamy napisany swoj pierwszy skrypt nalezy zapisac go oraz nadac mu odpowiedni atrybut wykonywalny. Atrybut pliku wykonywalnego nadajemy poleceniem:

```
bash-2.05b$ chmod +x nazwa_skryptu
```

No i tak oto napisalismy swoj pierwszy skrypt w bashu teraz pozostalo go nam tylko uruchomic w tym celu wpisujemy w konsoli nastepujace polecenie:

```
bash-2.05b$ ./nazwa_skryptu
```

## 2 - Słowa zastrzeżone

Słowa zastrzeżone są to takie słowa lub znaki które wykonują określone polecenia jeśli chcemy użyć ich w jakimś zdaniu należy je zacytować.

```
* !
* [
* ]
* {
* }
* case
* do
* for
* function
```

```
* if
* in
* select
* then
* done
* elif
* else
* esac
* fi
* until
* while
* time
```

### 3 - Zmienne oraz cytowanie

W cudzyslowie umieszczamy text, zmienne itp. Zmienna okreslamy podobnie jak w PHP znakiem \$ np.

```
#!/bin/bash
x=8
echo "Tu jest napisana liczba $x"
```

zmienne mozemy wykozystac takze do wykonywania konkretnych polecen np.:

```
#!/bin/bash
x=`netstat`
echo $x
```

polecenie to uruchamia nam netstat i pokazuje aktualny stan sieci.

Istnieja trzy rodzaje cudzyslowi

- "" - ciag znakow jest traktowany jako text
- " - ciag znakow jest traktowany jako text
- `` - umozliwia cytowanie polecen

Istnieja takze znaki maskujace charakteryzujace sie \ (backslash) sluzą one do wylanczania interpretacji zmiennych.

W bashu procz zmiennych programowych (to te cos sa opisane powyzej) wystepuja takze zmienne specjalne, srodowiskowe, oraz tablicowe.

Zmienne specjalne:

```
$0 - nazwa naszego skryptu
$# - liczba przekazanych parametrow
$$ - numer identyfikacyjny procesu skryptu (PID)
$* - lista porozdzielanych parametrow
@$ - lista parametrow z ktorymi zostal uruchomiony skrypt
 $? - kod ostatniego wykonywalnego polecenia
$1,$2,$3,.. - poszczegolne parametry w uporzadkowanej kolejnosci
```

Zmienne srodowiskowe:

```
$HOME      - sciezka do twojego katalogu domowego
$USER      - nazwa uzytkownika
$HOSTNAME  - host
$HOSTTYPE  - zmienna architektury procesora
$OSTYPE    - rodzaj systemu operacyjnego
$LOGNAME   - login
```

by uzyskac wszystkie zmienne srodowiskowe wpisz w konsoli printenv.

Zmienne tablicowe:

Moze zaczniemy w ogole od tego co sa zmienne tablicowe. Zmienne tablicowe sa to takie zmienne ktore przechowuja liste okreslonych wartosci. W bashu stosujemy tablice jednowymiarowe, nie ma okreslonego maksymalnego rozmiaru tablic. Zmienne tablicowe indexujemy liczbami calkowitymi zaczynajac od 0. Ponizej jest pokazany prosty przyklad z zastosowaniem tablic.

```
#!/bin/bash
tablica=(cos1 cos2 cos3 cos4)
echo ${tablica[0]}
echo ${tablica[1]}
echo ${tablica[2]}
echo ${tablica[3]}
```

Ponizsza funkcja pokaze nam z ilu znakow sklada sie 1 element w tablicy w naszym wpadku bedzie to 4. jesli chcemy by pokazalo sie nam ile zmiennych posiadamy w tablicy wystarczy zamiast 0 wstawic @ lub \*

```
#!/bin/bash
tablica=(cos1 cos2 cos3 cos4)
echo ${#tablica[0]}
```

Usuwanie wartosci z tablic jest bardzo prosta sluzi nam do tego funkcja unset a o to jej zastosowanie:

```
#!/bin/bash
tablica=(cos1 cos2 cos3 cos4)
unset tablica[3]
echo ${tablica[*]}
```

powyzszy skrypt usunie nam ostania wartosc z tablicy i wyswietli pozostale elementy.

#### 4 - Read

Jesli ktos mial juz wczesniej styczność z programowaniem to wie ze polecenie read sluzi nam do wprowadzania danych. Ponizej jest podany przyklad zastosowania w bashu polecenia read wprowadza sie do niego napis ktory on nastepnie wyswietla jest to chyba najprostrzy przyklad zastosowania tego polecenia.

```
#!/bin/bash
echo -en "Wpisz cos:\n"
read wprowadzony_napis
echo "$wprowadzony_napis"
```

## 5 - Funkcje

Ogólnie mówiąc funkcja jest to jakieś rozbudowane polecenie zazwyczaj stosowane wiele razy w programie. Funkcje zapisujemy w następujący sposób:

```
function nazwa
{
instrukcja1
instrukcja2
}
```

Jeśli chcemy wywołać daną funkcję po prostu w kodzie zamieszczamy jej nazwę.

## 6 - Obliczenia matematyczne

Składnie matematyczne możemy zapisać w następujący sposób:

```
#!/bin/bash
echo $((6/2))
```

lub

```
#!/bin/bash
wynik=$((6/2))
echo "$wynik"
```

Można także zastosować polecenie `let`:

```
#!/bin/bash
liczba1=6
liczba2=2
let wynik=liczba1/liczba2
echo "$wynik"
```

Operatory arytmetyczne występujące w bashu:

```
-,+ - dodawanie odejmowanie
*,/,% - mnożenie, dzielenie całkowite, reszta z dzielenia
** - potęga
!,~ - logiczna i bitowa negacja
<<, >> - przesunięcia bitowe
^,&, | - bitowe XOR, AND, OR
||, && - logiczne OR i AND
<=,>=,<,>,! =, == - porównania
=,*=,/=,%=,+=,-=,<<=,>>=,&=,|=,^= - przypisania
```

## 7 - Instrukcje warunkowe

Instrukcje warunkowe mówią najprościej składają się następująco: "jeśli coś to wtedy coś" w bashu warunek jeżeli rozpoczynamy od if a kończymy go fi wyglądać to będzie tak:

```
#!/bin/bash
if [ -e ~/.bash_history ]
then
    echo "Plik .bash_history istnieje"
fi
```

Powyzsza procedura sprawdza nam czy plik bash\_history w naszym katalogu domowym istnieje, jeśli istnieje zwraca napis "Plik .bash\_history istnieje". Teraz rozbudujemy nasz skrypt o przeczenie czyli co się stanie gdy warunek będzie fałszywy.

```
#!/bin/bash
if [ -e ~/.bash_history ]
then
    echo "Plik .bash_history istnieje"
else
    echo "Plik .bash_history nie istnieje"
fi
```

polecenie else wykonuje się wtedy gdy warunek jest fałszywy. Można także stosować wiele warunków co ułatwia nam polecenie elif.

```
#!/bin/bash
if [ -e ~/.bash_history ]; then
    echo "Plik .bash_history istnieje"
elif [ -e ~/.bash_profile ]; then
    echo "Plik .bash_profile istnieje"
else
    echo "Plik .bash_history i bash_profile nie istnieją"
fi
```

Powyzszy program sprawdza nam czy pliki .bash\_history oraz .bash\_profile istnieją jeśli oba nie istnieją skrypt zwraca nam napis

*"Plik .bash\_history i bash\_profile nie istnieją"*

Jeśli zauważyliście do powyzszy skryptów stosowałem operatora -e sprawdza on czy dany plik istnieje. Poniżej są podane operatory stosowane w bashu.

Operatory:

- a plik  
Prawda jeśli plik istnieje.
- b plik  
Prawda jeśli plik istnieje i jest blokowym plikiem specjalnym.
- c plik  
Prawda jeśli plik istnieje i jest znakowym plikiem specjalnym.
- d plik  
Prawda jeśli plik istnieje jest katalogiem.
- e plik  
Prawda jeśli plik istnieje.
- f plik  
Prawda jeśli plik istnieje i jest plikiem zwykłym.
- g plik

Prawda jesli plik istnieje i ma ustawiony bit set-group-id.  
 -h plik  
 Prawda jesli plik istnieje i jest dowiazaniem symbolicznym.  
 -k plik  
 Prawda jesli plik istnieje i ma ustawiony bit  
 -p plik  
 Prawda jesli plik istnieje i jest potokiem nazwanym (FIFO).  
 -r plik  
 Prawda jesli plik istnieje i daje sie czytac.  
 -s plik  
 Prawda jesli plik istnieje i ma rozmiar wiekszy niz zero.  
 -t fd  
 Prawda jesli deskryptor pliku fd jest otwarty i odnosi sie do terminala.  
 -u plik  
 Prawda jesli plik istnieje i ma ustawiony bit set-user-id.  
 -w plik  
 Prawda jesli plik istnieje i daje sie don zapisac.  
 -x plik  
 Prawda jesli plik istnieje i jest wykonywalny.  
 -0 plik  
 Prawda jesli plik istnieje i jego wlascicielem jest efektywny id uzytkownika.  
 -G plik  
 Prawda jesli plik istnieje i jego wlascicielem jest efektywny id grupy.  
 -L plik  
 Prawda jesli plik istnieje i jest dowiazaniem symbolicznym.  
 -S plik  
 Prawda jesli plik istnieje i jest gniazdem.  
 -N plik  
 Prawda jesli plik istnieje i byc zmieniany od czasu ostatniego jego odczytu.  
 plik1 -nt plik2  
 Prawda jesli plik1 jest nowszy (wedlug daty modyfikacji) od pliku2.  
 plik1 -ot plik2  
 Prawda jesli plik1 jest starszy niz plik2.  
 plik1 -ef plik2  
 Prawda jesli plik1 i plik2 maja ten sam numer urzadzenia i i-wezla.  
 -o nazwa\_opcji  
 Prawda jesli opcja powloki nazwa\_opcji jest wlaczona. Zobacz zestawienie opcji w opisie opcji -o wbudowanego set, ponizej.  
 -z lancuch  
 Prawda jesli dlugosc lancucha wynosi zero.  
 -n lancuch  
 Prawda jesli lancuch ma dlugosc niezerowa.  
 lancuch1 == lancuch2  
 Prawda jesli lancuchy sa rowne. Zamiast == mozna uzyc =.  
 lancuch1 != lancuch2  
 Prawda jesli lancuchy nie sa rowne.

W instrukcjach warunkowych istenije takze dokonywanie wyborow oznacza sie ono poleceniem case oto prosty skrypt z zastosowaniem owego polecenia:

```
#!/bin/bash
echo "w ktorym miesiacu rozpoczynaja sie wakacje: (podaj liczbe)"
read miesiac
case "$miesiac" in
"6") echo "Wakacje rozpoczynaja sie w czerwcu" ;;
"7") echo "Wakacje rozpoczynaja sie w lipcu" ;;
*) echo "nie podales zadnej wartosci"
esac
```

## 8 - Petle

Petla jest to poprosu okreslone polecenie ktore ma sie wykonywac okreslana ilosc razy w bashu wyrozniamy nastepujace petle: for, while, until i select.

*Petla for:*

Na ponizszym przykadzie proste zastosowanie petli for wykazuje nam ona wszystkie pliki z rozszerzeniem mp3 ktore znajduja sie w danym katalogu

```
#!/bin/bash
for zmienna in *.mp3
do
    echo "W tym katalogu znajduje sie plik mp3 o nazwie $zmienna"
done
```

*Petla while:*

Mozna ja zastosowac do prostych hasel skryptowych czy np do sprawdzenia czy jakas operacja zostala wykonana. Zasada jej dzialania jest dosc prosta petla ta sprawdza czy warunek zostal spelniony jezeli tak to wykonuje dalsze polecenie natomiast jesli jest niespelniony powraca do pierwotnej postaci.

```
#!/bin/bash
i=1;
while [ $i ]; do
echo "Napis ten wyswietla sie poraz $i "
i=$((i+1))
done
```

powyzsza petla wyswietla dany napis nieskonczola ilosc razy za kazdym razem dodajac do niego o jeden nr wiecej.

*Petla until:*

Jest to petla bardzo podobna do petli while z ta mala roznicza ze warunek jest wykonywany do tej pory dopuki warunek jest niespelniony jesli zas warunek zostaje spelniony petla natychmiast zostaje przerwana. Ponizsza petla sprawdza co 1 sekunde czy root sie zalogowal jesli tak wyswietla nam o tym komunikat.

```
#!/bin/bash
until who | grep root > /dev/null
do
```

```
sleep 1
done
echo "root jest zalogowany"
```

*Petla select:*

W tej petli mozemy zastosowac warunek wyboru case. Ponizszy skrypt obrazuje skladnie petli select.

```
#!/bin/bash
echo "Jaki dzis dzien tygodnia ??"
select x in Poniedzialek Wtorek Sroda Czwartek \
  Piatek Sobota Niedziela Wyjście
do
case $x in
"Poniedzialek") echo "Dzis mamy Poniedzialek" ;;
"Wtorek") echo "Dzis mamy Wtorek" ;;
"Sroda") echo "Dzis mamy Srode" ;;
"Czwartek") echo "Dzis mamy Czwartek" ;;
"Piatek") echo "Dzis mamy Piatek" ;;
"Sobota") echo "WDzis mamy Sobote" ;;
"Niedziela") echo "Dzis mamy Niedziele" ;;
"Wyjście") exit ;;
*) echo "no i nie wiesz jaki dzis mamy dzien tygodnia"
esac
break
done
```

## 9 - Operacje na plikach

Pobieranie oraz zapisywanie do plikow roznnych danych w bashu jest bardzo proste sluzą nam do tego nastepujace operatory:

```
< - pobieranie danych z pliku
> - zapisuje dane do pliku kasujac jego wczesniejsza zawartosc
>> - zapisuje dane na koncu danego pliku
```

przyklad zapisywania zostal podany w rozdziale "Poczatek" wiec nie bedziemy powtarzac tego co juz bylo.

## 10 - Polecenia tr, cut, wc, exec

*tr:*

Polecenie mozemy to stosowac w usuwaniu lub zamienianiu znakow ze standardowego wejścia, polecenie tr mozemy stosowac z operatorami:

```
-c dopelnia pierwszy zbior
-d usuwa znaki zawartw w pierwszym zbiorze
-s zastepuje sekwenecje powtorzonych znakow
```

*cut:*

Polecenie to wycina okreslony fragment. Moze wystepowac z nastepujacymi operatorami:

- c pokaze nam wyliczone znaki w danym pliku
- d tabulacje
- f wypisuje wyliczone pola

jesli chcecie poznac wiecej operatorow nalezy wipasc `cut --help` lub `man cut`

*wc:*

Polecenie to wypisuje nam ilosc lini, slow oraz bajtow ktore sa zawarte w danym pliku. Moze wystepowac z operatorami:

- c pokazuje ilosc bajtow
- w pokazuje ilosc slow
- l pokazuje ilosc linii

*exec:*

polecenie to pozwala nam uruchomic dany program z roznymi parametrami.

---

Wszystkie uwagi związane z artykułem proszę kierować na [itros@linux.pl](mailto:itros@linux.pl)

Artykuł pochodzi ze strony <http://linux.slupsk.pl>