Polecenie echo

Polecenie echo służy do wydrukowania na standardowym wyjściu (**stdout** - domyślnie jest to ekran) napisu.

Składnia:

```
#!/bin/bash
echo -ne "jakiś napis"
echo "jakiś napis"      #wydrukuje tekst na ekranie
```

Można też pisać do pliku. W tym wypadku **echo** wydrukuje tekst do pliku, ale zmaże całą jego wcześniejszą zawartość, jeśli plik podany na standardowym wyjściu nie istnieje, zostanie utworzony.

```
echo "jakiś napis" > plik
```

Tutaj napis zostanie dopisany na końcu pliku, nie zmaże jego wcześniejszej zawartości.

```
echo "jakiś napis" >> plik
```

Parametry:

- **-n** nie jest wysyłany znak nowej linii
- **-e** włącza inetrpretacje znaków specjalnych takich jak:
  - **\a** czyli alert, usłyszysz dzwonek
  - **\b** backspace
  - **\c** pomija znak kończący nowej linii
  - **\f** escape
  - **\n** form feed czyli wysuw strony
  - **\r** znak nowej linii
  - **\t** tabulacja pozioma
  - **\v** tabulacja pionowa
  - **** backslash
  - **\nnn** znak, którego kod ASCII ma wartość ósemkowo
  - **\xnnn** znak, którego kod ASCII ma wartość szesnastkowo

---

Słowa zastrzeżone (ang. reserved words)

Mają dla powłoki specjalne znaczenie, wtedy gdy nie są cytowane.

Lista słów zastrzeżonych:

- **!**
- **case**
- **do**
- **done**
- **elif**
- **else**
- **esac**
- **fi**
- **for**
- **function**

- **if**
- **in**
- **select**
- **then**
- **until**
- **while**
- **{**
- **}**
- **time**
- **[**
- **]**

Cytowanie

Znaki cytowania służą do usuwania interpretacji znaków specjalnych przez powłokę.

Wyróżnia się nastepujące znaki cytowania:

- **cudzysłów** (ang. *double quote*)
- " "

  Między cudzysłowami umieszcza się tekst, wartości zmiennych zawierające spacje. Cudzysłowy zachowują znaczenie specjalne trzech znaków:

  - **$** wskazuje na nazwę zmiennej, umożliwiając podstawienie jej wartości
  - **** znak maskujący
  - ` ` odwrotny apostrof, umozliwia zacytowanie polecenia

  Przykład:

  ```
  #!/bin/bash

  x=2
  echo "Wartość zmiennej x to $x"    #wydrukuje Wartość zmiennej x to 2
  echo -ne "Usłyszysz dzwonek\a"
  echo "Polecenie date pokaże: `date`"
  ```

- **apostrof** (ang. *single quote*)
- ' '

  Wszystko co ujęte w znaki apostrofu traktowane jest jak łańcuch tekstowy, apostrof wyłącza interpretowanie wszystkich znaków specjalnych, traktowane są jak zwykłe znaki.

  Przykład:

  ```
  #!/bin/bash

  echo '$USER'   #nie wypisze twojego loginu
  ```

- **odwrotny apostrof** (ang. *backquote*)
- ``

  umożliwia zacytowanie polecenia, bardzo przydatne jeśli chce się podstawić pod zmienną wynik jakiegoś polecenia np:

  Przykład:

  ```
  #!/bin/bash

  x=`ls -la $PWD`
  echo $x                    #pokaże rezultat polecenia
  ```

  Alternatywny zapis, który ma takie samo działanie wygląda tak:

  ```
  #!/bin/bash

  echo $(ls -la $PWD)
  ```

- **backslash** czyli znak maskujący
- \

  Jego działanie najlepiej wyjaśnić na przykładzie: w celu by na ekranie pojawił się napis $HOME

  Przykład:

  ```
  echo "$HOME"                #wydrukuje /home/ja
  ```

  aby wyłączyć interpretacje przez powłokę tej zmiennej, należy wpisać:

  ```
  echo \$HOME                #i jest napis $HOME
  ```

Zmienne programowe *(ang. program variables)*

To zmienne definiowane samodzielnie przez użytkownika.

nazwa_zmiennej="wartość"
Na przykład:
x="napis"
Do zmiennej odwołujemy się poprzez podanie jej nazwy poprzedzonej znakiem **$** i tak dla zmiennej **x** może to wyglądać następująco:
echo $x
Na co należy uważać? **Nie może być spacji po obu stronach!**
x = "napis"
ten zapis jest błędny

Pod zmienną możemy podstawić wynik jakiegoś polecenia, można to zrobić na dwa sposoby:

- Poprzez użycie odwrotnych apostrofów:

  `polecenie`

  Przykład:

  ```
  #!/bin/bash

  GDZIE_JESTEM=`pwd`
  echo "Jestem w katalogu $GDZIE_JESTEM"
  ```

  Wartością zmiennej **GDZIE_JESTEM** jest wynik działania polecenia **pwd**, które wypisze nazwę katalogu w jakim się w danej chwili znajdujemy.

- Za pomocą rozwijania zawartości nawiasów:

  $(polecenie)

  Przykład:

  ```
  #!/bin/bash

  GDZIE_JESTEM=$(pwd)
  echo "Jestem w katalogu $GDZIE_JESTEM"
  ```

Zmienne specjalne *(ang. special variables, special parameters)*
To najbardziej prywatne zmienne powłoki, są udostępniane użytkownikowi tylko do odczytu (są wyjątki). Kilka przykładów:

- **$0**

  nazwa bieżącego skryptu lub powłoki

  Przykład:

  ```
  #!/bin/bash

  echo "$0"
  ```

  Pokaże nazwę uruchomionego skryptu.

- **$1..$9**

  Parametry przekazywane do skryptu (wyjątek, użytkownik może modyfikować ten rodzaj $-ych specjalnych.

  ```
  #!/bin/bash

  echo "$1 $2 $3"
  ```

Jeśli wywołany zostanie skrypt z jakimiś parametrami to przypisane zostaną zmiennym: od **$1** do **$9**. Zobacz co się stanie jak podasz za małą liczbę parametrów oraz jaki będzie wynik podania za dużej liczby parametrów.

- **$@**

  Pokaże wszystkie parametry przekzywane do skryptu (też wyjątek), równoważne **$1 $2 $3**..., jeśli nie podane są żadne parametry **$@** interpretowana jest jako nic.

  Przykład:

  #!/bin/bash

  echo "Skrypt uruchomiono z parametrami: $@"

  A teraz wywołaj ten skrypt z jakimiś parametrami, mogą być brane z powietrza np.:

  ./plik -a d

  Efekt będzie wyglądał następująco:

  Skrypt uruchomiono z paramertami -a d

- **$?**

  kod powrotu ostanio wykonywanego polecenia

- **$$**

  PID procesu bieżącej powłoki

Zmienne środowiskowe *(ang.environment variables)*

Definują środowisko użytkownika, dostępne dla wszystkich procesów potomnych. Można je podzielić na:

- globalne - widoczne w każdym podshellu
- lokalne - widoczne tylko dla tego shella w którym został ustawione

  Aby bardziej uzmysłowić sobie różnicę między nimi zrób mały eksperyment: otwórz xterma (widoczny podshell) i wpisz:

  x="napis"
  echo $x
  xterm

  x="napis" zdefiniowałeś właśnie zmienną **x**, która ma wartość "napis"
  echo $x wyświetli wartość zmiennej x
  xterm wywołanie podshella

wpisz więc jeszcze raz:

echo $x nie pokaże nic, bo zmienne lokalne nie są widoczne w podshellach

Możesz teraz zainicjować zmienną globalną:

export x="napis"

Teraz zmienna **x** będzie widoczna w podshellach, jak widać wyżej służy do tego polecenie **export**, nadaje ono wskazanym zmiennym atrybut zmiennych globalnych. W celu uzyskania listy aktualnie eksportowanych zmiennych należy wpisać **export**, opcjonalnie **export -p** . Na tej liście przed nazwą każdej zmiennej znajduje się zapis:

declare-x

To wewnętrzne polecenie BASH-a, służące do definiowania zmiennych i nadawania im atrybutów, **-x** to atrybut eksportu czyli jest to, to samo co polecenie **export**. Ale tu uwaga! Polecenie **declare** występuje tylko w BASH-u, nie ma go w innych powłokach, natomiast **export** występuje w **ksh**, **ash** i innych, które korzystają z plików startowych **/etc/profile**. Dlatego też zaleca się stosowanie polecenia **export**.

export -n zmienna

spowoduje usunięcie atrybutu eksportu dla danej zmiennej

Niektóre przykłady zmiennych środowiskowych:

$HOME #ścieżka do twojego katalogu domowego
$USER #twój login
$HOSTNAME #nazwa twojego hosta
$OSTYPE #rodzaj systemu operacyjnego

itp. dostępne zmienne środowiskowe można wyświetlić za pomocą polecenia:

printenv | more

expr

Najprostszym sposobem jest użycie polecenia expr. Trzeba przy tym pamiętać, żeby osobne tokeny (tzn. liczby i operatory arytmetyczne) były podawane w osobnych argumentach. Wynika to stąd, że expr potrafi też operować na łańcuchach znakowych (czym się nie będziemy w tej chwili zajmować), więc musi jakoś te łańcuchy dostawać, a jedyną droga to przez argumenty.

Dostępnych jest pięć operatorów arytmetycznych:

- dodawanie (+),
- odejmowanie (-),
- mnożenie (*),
- dzielenie (/),
- modulo - reszta z dzielenia (%).

Ponadto możemy wykonywać porównania <, <=, =, == (synonim =), !=, >=, >. W wyniku mamy 1, gdy relacja jest spełniona i 0 w przeciwnym przypadku.

Trzeba też pamiętać by znaki specjalne poprzedzać backslashem lub brać w cudzysłowy. Przykłady:

```
bashtest@host:~$ expr 2\*3
2*3
bashtest@host:~$ expr 2 \* 3
6
bashtest@host:~$ expr '2 * 3'
2 * 3
bashtest@host:~$ expr 2 \* \(7 - 1\)
expr: argument nieliczbowy
bashtest@host:~$ expr 2 \* \( 7 - 1 \)
12
bashtest@host:~$ a=5
bashtest@host:~$ a=`expr $a + 1`
bashtest@host:~$ echo $a
6
bashtest@host:~$ expr 3 \<= 4
1
bashtest@host:~$ expr 3 '<=' 1
0
bashtest@host:~$
```

let

let jest wbudowanym poleceniem Basha i używamy go, podając mu jako argumenty wyrażenia do przetworzenia.

let wyrażenie1 wyrażenie2 ...

równoważne jest ciągowi poleceń

((wyrażenie1))
((wyrażenie2))
...

Przykład:

```
bashtest@host:~$ x=0
bashtest@host:~$ let x+=2 "x += 4"
bashtest@host:~$ echo $x
6
bashtest@host:~$
```

Trzeba pamiętać, że wyrażenie zawierające odstępy trzeba ujmować w cudzysłowy, aby formowały jeden argument.

Wyrażenia regularne

Wyrażenie regularne (ang. regular expressions) to wzorce, które opisują różne ciągi znakowe. Mają identyczne zastosowanie co standardowe symbole wieloznaczne, ale są znacznie bardziej zaawansowane i dają więcej możliwości. Popularne komendy, które mogą korzystać z wyrażeń regularnych to `grep`, `sed` i `find`.

Oficjalna dokumentacja dla wyrażeń regularnych:

```
man 7 regex
```

Wyrażenia regularne również używają odpowiednich symboli, aby można było opisać ciągi znaków. Część z tych symboli należy do kategorii rozszerzonych (E - ang. extended) i aby ich używać trzeba zazwyczaj w komendzie użyć dodatkowego parametru aby o tym komendę poinformować i aby były poprawnie zinterpretowane.

Symbole wyrażeń regularnych, ich znaczenia i przykłady:

Wzorzec/symbol	**Znaczenie**
.	dowolny znak
*	poprzedni znak/wyrażenie może wystąpić 0 lub więcej razy
+	poprzedni znak/wyrażenie może wystąpić 1 lub więcej razy (E)
?	poprzedni znak/wyrażenie może wystąpić 0 lub raz (E)
[]	dowolny znak spośród tych między nawiasami
^	początek linii
$	koniec linii
\	anuluje działanie symbolu wieloznacznego po tym znaku
()	grupuje symbole w zestawy (E)

Wzorzec/symbol	**Znaczenie**
{n}	poprzedni znak/wyrażenie musi wystąpić n razy (E)
|	rozdziela alternatywne ciągi znaków
.*	dowolny ciąg znaków
a.*j.	ciąg znaków, gdzie pierwszy znak to litera a, przedostatnia j
[0-9]:[0-9]	dwie cyfry rozdzielone dwukropkiem
(oo)+	przynajmniej jedno wystąpienie ciągu oo (E)
(root){2}	dokładnie 2 wystąpienia ciągu root (na końcu jest spacja) (E)
[[:digit:]]	cyfra
[[:blank:]]	biały znak
[[:lower:]]	mała liter
[[:upper:]]	wielka litera

m}	dokładnie m wystąpień
{m,}	co najmniej m wystąpień
{,n}	co najwyżej n wystąpień
{m,n}	od m do n wystąpień
[...]	jeden znak spośród zbioru znaków
[^...]	jeden znak spoza zbioru znaków
A | B	dopasowanie A lub B

Symbole standardowe a regex (ekwiwalenty):

standardowy	**regex**
*	.*
?	.
[agk]	[agk]
[a-e]	[a-e]

Przykłady użycia wyrażeń regularnych:

```
ls -l /etc | grep '.*conf'
ls -l /etc | grep -E '[[:digit:]]{4}'
ls -l /etc | grep -E '(se|mt).*conf$'
ls -l /etc | grep -F 'conf'
```

Komenda grep

Jedna z najpopularniejszych komend systemu linux - grep (global regular expression print) - używana najczęściej w połączeniu z innymi komendami za pomocą potoków służy do filtrowania wyświetlanego na konsoli tekstu. Dzięki komendzie można wyświetlić wybrane (wyfiltrowane) linie pliku tekstowego lub wybrane linie wyniku dowolnej komendy.

Podstawowa składnia i przykłady użycia:

grep [OPCJE] FILTR plik-tekstowy

inna-komenda | grep [OPCJE] FILTR

grep --colour tcp /etc/services

ifconfig | grep --colour inet

Komenda przyjmuje wiele parametrów, ale te decydujące o tym jak ma być interpretowany filtr to:

- -E (extended) - pozwala na używanie symboli rozszerzonych regex, ekwiwalentem dla grep -E jest komenda egrep
- -F (fixed) - użycie tylko prostych ciągów znaków, bez regexp czy podstawowych symboli wieloznacznych, ekwiwalentem dla grep -F jest komenda fgrep
- bez powyższych parametrów - używa podstawowych symboli regexp

W przypadku potrzeby użycia symbolu rozszerzonego regexp należy pamiętać o dodaniu parametru -E lub użyciu komendy egrep, na przykład:

```
ls /etc | grep -E "is{2}ue"
ls /etc | egrep "is{2}ue"
```

Inne przydatne parametry:

- -v – wyświetla te linie, dla których wzorca nie znaleziono
- -i – zignoruje wielkość liter
- --colour – koloruje wynik (grep wyświetla domyślnie całe linie ze znalezionym fragmentem pasującym do wzorca, ale ten fragment zostaje pokolorowany - przydatne w nauce regexp)
- -o – pokaż tylko pasujące do wzorca ciągi znaków (zamiast całe linie zawierające te ciągi)
- -n – wyświetl dodatkowo numer linii w wyniku
- -R lub -r – przeszukuj rekursywnie w podkatalogach (-R zagląda również w zawartość linków symbolicznych)

http://home.agh.edu.pl/~mkuta/tk/re/re.html

http://kurslinux.ovh.org/02przeszukiwanie.php

http://designconcept.webdev20.pl/notatki/linux-podstawowe-polecenia/

Pierwszy skrypt

• plik skryptu – plik tekstowy

#!/bin/bash

#komentarz.

echo Pierwszy skrypt

echo Mój katalog domowy to $HOME

• nadanie praw wykonywania

chmod u+x skrypt1.sh

• uruchomienie

./skrypt1.sh

---

chmod to polecenie do zmiany praw dostępu do pliku.

Wystarczy w wierszu poleceń wpisać komendę chmod po spacji nowe prawa do pliku i po spacji nazwę pliku, którego prawa mają zostać zmienione.

Prawa plików można zapisać na kilka sposobów. Najpopularniejszy to dziesiętne określenie praw. Dostępne sposoby: numeryczny i literowy.

Numeryczny zapis praw do pliku jest najczęściej stosowany ze względu na łatwy i zrozumiały zapis. Są trzy prawa dotyczące pliku - odczyt, zapis i wykonywanie. Dla każdego z nich przypisana jest wartość potęgi dwójki **(1 - prawo do uruchomienia, 2 - prawo do zapisu i 4 - prawo do odczytu).** Określenie praw do pliku następuje poprzez dodanie do liczby określającej prawo wagę akcji jaka może być wykonywana na pliku.

Każdy plik może mieć określone prawa dla trzech grup użytkowników - właściciel, grupa oraz pozostali użytkownicy. Dlatego więc numeryczny zapis praw do plików składa się najczęściej z trzech liczb - prawa dla właściciela, prawa dla grupy i prawa dla wszystkich.

Dlatego więc zapis chmod 777 oznacza przypisanie wszystkich możliwych praw dla wszystkich możliwych użytkowników.

Tabela wyjaśniająca znaczenie liczby

w prawach dostępu do pliku:

Liczba	Prawa	
0	Brak praw	
1	Wykonywanie	
2	Zapis	
3	Wykonywanie i zapis	
4	Odczyt	
5	Odczyt i wykonywanie	
6	Odczyt i zapis	
7	Odczyt, pisanie i wykonywanie	rwx

Literowy zapis praw do pliku

Literowy zapis praw dostępu do plików w systemach unixowych ma nieco bardziej skomplikowaną składnię, niż numeryczny zapis. Jest jednak również stosowany, ze względu na możliwość dodawania, bądź usuwania określonych uprawnień. Pozwala to na upewnienie się, że prawa pliku są określone, bez ich wcześniejszej analizy.

Literowy zapis ma następującą postać:

chmod [kto][+/-/=][co]

[kto] ma następujące możliwości: u (właściciel pliku), g (grupa użytkowników), o (pozostali użytkownicy tzw. cały świat), a (wszyscy). Przy czym można używać kilku opcji na raz, np. 'ug' będzie oznaczać nadanie praw właścicielowi pliku oraz grupie.

PRZYKŁADY I POMOCE:

Zmienne środowiskowe:

ZMIENNA=123

set

env

export ZMIENNA

set

env

echo $ZMIENNA

read - czytanie ze standardowego wejścia:

echo Podaj wartosc zmiennej

read zmienna

echo Podaj co najmniej 3 słowa

read pierwsze drugie reszta

echo Pierwsze słowo to $pierwsze

echo Drugie slowo to $drugie

echo A reszta to $reszta

• $HOME, $USER, $PATH, …

Argumenty skryptu:

• $1, $2, ..., (dla więcej niż jednocyfrowych ${10})

• $0 – nazwa pliku skryptu,

• $@ - wszystkie argumenty

• $# - liczba argumentów

• shift – „przesunięcie” argumentów tzn. wcześniejsze $2 teraz będzie $1,

wcześniejsze $3 teraz będzie $2, itd.(też np.: shift 2)

Wyświetlanie

• echo

echo -e "\tLinia nr 1 \n\tLinia nr 2"

Obliczanie wartości wyrażeń:

• $[…]

np.

SUMA=$[23+12]

echo $SUMA

SUMA=$[SUMA+3]

echo $SUMA

• polecenie expr

np.

WYNIK=`expr 6 + 3`

WYNIK=`expr $WYNIK \* 3`

echo $WYNIK

Wykorzystanie wyniku polecenia:

• `...`

np. echo Moj katalog biezacy to `pwd`

rm `find . -name "*.old"`

• echo `cat *.txt` - rozwija „*”

• IMIE=Karolina

PODPIS=`echo "Pozdrawiam, $IMIE"` - rozwija $ZMIENNA

echo $PODPIS

Dodatkowe polecenia przydatne do zadań:

• grep wzorzec plik – wyszukiwanie wzorca w pliku

np.

-i - bez rozróżniania wielkości liter

-l - tylko nazwy plików

-v - wyklucza wzorzec

Wzorce:

^wzorzec - rozpoczynające się od „wzorzec"

[abc] - "a" lub "b" lub „c”

wzorzec$ - kończące się na „wzorzec”

• cut - do wycinania fragmentu z każdej linii wejścia

np. -c (znaki)

-d (separator)

-f (pola)

• wc – zliczanie w pliku np.:

-c (bajty)

-w (słowa)

-l (linie)