

## Skrypty powłoki w systemie Linux

Wykonywanie codziennych czynności w systemie operacyjnym jest męczące, gdy za każdym razem trzeba wpisywać te same zestawy komend. Znacznie wygodniej byłoby zapisać je w plikach, które możemy później uruchomić w dowolnej chwili. Pod linuxem dzięki skryptom powłoki (shella) wykonamy te zadania bez problemu.

Skrypty powłoki to ciągi instrukcji zapisane w postaci zwykłych plików tekstowych, które mogą zostać uruchomione i wykonywać za nas określone operacje.

Taki skrypt może też zawierać instrukcje warunkowe, pętle, przeprowadzić wybrane działania dla parametrów podanych z linii poleceń (po nazwie skryptu), zapamiętywać te parametry w zmiennych itp.

### **Zasady tworzenia skryptów:**

Każde zapisane w skrypcie polecenie musi być zapisane w oddzielnej linii (no... chyba że użyjemy separatora ';').

Pierwsza linia każdego skryptu musi zawierać wywołanie shella, pod którym skrypt będzie wykonywany.

Np. dla basha:  
#!/bin/bash

Aby skrypt dało się uruchomić należy zmienić mu uprawnienia (tak aby przynajmniej właściciel mógł go uruchamiać).

chmod 700 skrypt

Żeby uruchomić skrypt musimy wpisać (będąc np. w tym samym katalogu) ścieżkę dostępu do niego:

./skrypt

### **Ćwiczenie**

Utwórz za pomocą edytora vim plik tekstowy o nazwie **info** o zawartości:

```
#!/bin/bash
date
uname -a
whoami
hostname
```

Dodaj sobie uprawnienia do uruchamiania pliku skrypt i go uruchom.

## Instrukcja wyjścia echo

np.

```
echo To jest napis
```

## Zmienne w skryptach

Mogą przechowywać wartości logiczne teksty i liczby.

Deklarujemy je na bieżąco przez przypisanie wartości za pomocą znaku '='

Przykłady:

```
USR=ala  
LICZBAKONT=0  
KATALOG=/var/www
```

Stosując akcenty `` do zmiennej możemy wstawić wynik działania dowolnego polecenia

Przykład:

```
#liczba aktywnych użytkowników  
ZM=`who | wc -l`
```

```
#zapamiętanie w zmiennej nazwy bieżącego katalogu  
KATALOG=`pwd`
```

Jeśli chcemy na przykład wyświetlić zawartość zmiennej to używamy znaku \$

np.

```
echo $KATALOG
```

## Instrukcja warunkowa IF

Składnia:

```
if  
then  
else  
fi
```

Przykłady :

```
#jeśli podano pierwszy parametr  
#to wydrukuj napis "PARAMETR PODANO"  
#w przeciwnym razie wydrukuj napis "BRAK PARAMETRU"
```

```
if [ "$1" ]  
then  
    echo "PARAMETR PODANO"  
else  
    echo "BRAK PARAMETRU"  
fi
```

## Sprawdzanie warunków

Po zapoznaniu się ze składnią polecenia **if** pojawia się pytanie, jak są sprawdzane warunki. Do sprawdzania warunków służy polecenie **test**.

Składnia:

```
test wyrażenie1 operator wyrażenie2  
lub  
[ wyrażenie1 operator wyrażenie2 ]
```

W przypadku tego drugiego zapisu (z którego korzystaliśmy w przykładach dla instrukcji **if**) należy pamiętać o spacji pomiędzy nawiasami, a treścią warunku.

### Wartość zwracana

Polecenie **test** zwraca wartość 0 (*true*), jeśli warunek jest spełniony lub wartość 1 (*false*) w przeciwnym wypadku. Wartość zwracana umieszczana jest w zmiennej specjalnej \$?

Poniżej wyszczególniono kilka przykładowych operatorów polecenia **test**:

- -d - plik istnieje i jest katalogiem
- -e - plik istnieje
- = - sprawdza czy wyrażenia są równe
- != - sprawdza czy wyrażenia są różne
- -n - wyrażenie ma długość większą niż 0
- -z - wyrażenie ma zerową długość
- -lt - mniejsze niż
- -gt - większe niż
- -ge - większe lub równe
- -le - mniejsze lub równe

## Instrukcja case

Instrukcja **case** pozwala na dokonanie wyboru wielowariantowego. Wartość zmiennej porównywana jest z poszczególnymi wzorcami. Jeśli wzorzec ma taką samą wartość jak zmienna, wówczas wykonywane są polecenia przypisane do tego wzorca. Jeśli nie uda się znaleźć dopasowania wykonywane jest polecenie domyślne oznaczone symbolem \* (gwiazdka). Dlatego warto zawsze to polecenie domyślne umieszczać w instrukcji **case**, co będzie zabezpieczeniem przed błędami popełnionymi przez użytkownika.

Składnia:

```
case zmienna in  
"wzorzec1") polecenie1 ;;  
"wzorzec2") polecenie2 ;;  
"wzorzec3") polecenie3 ;;  
) polecenie_domyślne  
esac
```

Przykład:

```
#!/bin/bash  
echo "Podaj cyfry dnia tygodnia"
```

Systemy operacyjne - Skrypty w Linux - funkcje, instrukcje sterujace i obliczenia arytmetyczne 5

```
read d
case "$d" in
"1") echo "Poniedzialek" ;;
"2") echo "Wtorek" ;;
"3") echo "Sroda" ;;
"4") echo "Czwartek" ;;
"5") echo "Piatek" ;;
"6") echo "Sobota" ;;
"7") echo "Niedziela" ;;
*) echo "To nie jest dzien tygodnia"
esac
```

## Pętla FOR

Składnia:

```
for
do
done
```

Przykłady:

```
for x in jeden dwa trzy; do
echo "To jest $x"
done
```

```
#zmieniamy nazwy wszystkich plików
#w bieżącym katalogu
#dodając im rozszerzenia .old
for d in *
do
mv $d $d.old
done
```

## Pętla WHILE

Składnia:

```
while
do
done
```

Przykład:

```
#co minute wydrukuj na terminalu
#biezaca date
```

```
while [ true ]
do
date
sleep 60
done
```

# Polecenie read

Polecenie **read** umożliwia odczytanie ze standardowego wejścia pojedynczego wiersza.

Składnia:

```
read parametry nazwa_zmiennej
```

Przykład:

```
#!/bin/bash
echo -ne "Wprowadz tekst: \a"
read wpis
echo $wpis
```

Wprowadzony przez użytkownika tekst zostanie zapisany w zmiennej wpis (polecenie: read wpis). Zmienna ta zostanie następnie wypisana na ekran przy użyciu polecenia echo \$wpis. Polecenie read pozwala również na przypisanie kilku wartości kilku zmiennym. Przedstawia to przykładowy skrypt:

```
#!/bin/bash
echo "Wpisz trzy wyrazy:"
read a b c
echo $a $b $c
echo "Wartosc zmiennej a to: $a"
echo "Wartosc zmiennej b to: $b"
echo "Wartosc zmiennej c to: $c"
```

W skrypcie tym widać, iż spacje pomiędzy wyrazami są separatorami dla wartości przypisanych do zmiennych a, b oraz c.

Wybrane parametry polecenie read:

- -p - pokaże znak zachęty bez kończącego znaku nowej linii:

```
#!/bin/bash
read -p "Pisz:" odp
echo $odp
```

- -a - kolejne wartości przypisywane są do kolejnych indeksów zmiennej tablicowej

```
#!/bin/bash
echo "Podaj elementy zmiennej tablicowej:"
read -a tablica
echo ${tablica[*]}
```

- -e - jeśli nie podano żadnej nazwy zmiennej, wczytany wiersz trafia do \$REPLY

```
#!/bin/bash
echo "Wpisz cos:"
read -e
echo $REPLY
```

# Znaki

Pisząc skrypty stosujemy różne znaki, których znaczenie jest specjalne. Oto ich lista:

## # (hashmark)

Hashmark jest znakiem komentarza. Wszystkie znaki po znaku hashmark, aż do znaku nowej linii są ignorowane. Pewien komentarz ma specjalne znaczenie. Jeśli pierwszymi znakami pliku są znaki #! to komentarz ten oznacza jaki program ma interpretować skrypt.

```
#Zwyczajny komentarz, w którym
#zwykle będziemy opisywali przeznaczenie skryptu
```

```
#!/bin/bash
#powyższa linijka oznacza, że skrypt ma być interpretowany
#przez shell bash
```

## \* (gwiazdka)

Znak gwiazdka jest rozwijany w listę zawierającą nazwy wszystkich plików z bieżącego (lub wskazanego) katalogu

```
echo *

echo /bin/*
```

## \ (backslash)

Backslash jest znakiem cytującym. Usuwa on specjalne znaczenie następującego po nim znaku.

#tekst „ala ma kota” zostanie potraktowany jak komentarz i nie będzie wypisany

```
echo #ala ma kota
```

#znak '#' zwykle oznaczający początek komentarza będzie potraktowany jako zwykły znak i wypisany na ekran

```
echo \# ala ma kota
```

#wypisana zostanie '\*', w przypadku braku backslasha wypisana zostanie cała zawartość bieżącego katalogu

```
echo \*
```

```
#usuwamy specjalne znaczenie spacji
echo Ala\ \ \ \ \ Ma \ \ \ kota
```

Stosując znak \ możemy długie polecenia zapisać w kilku liniach:

```
#usuwamy specjalne znaczenie entera
# UWAGA: po znaku \ nie może być znaku spacji!
cat \
/etc/passwd | \
wc \
-1
```

## ; (średnik)

Znak średnika jest separatorem poleceń. Stosując znak średnik możemy umieścić kilka poleceń w jednej linii.

```
whoami;date;pwd;hostname
```

Jest to szczególnie przydatne przy pisaniu instrukcji **if** czy **while**

```
if [ "$1" ]; then
    echo Podano parametr $1
fi
```

## ` (akcent)

Znaki akcentu umożliwiają wykonanie polecenia i wstawienie wyników jego działania. Znak akcentu znajduje się na klawiaturze na lewo od znaku '!'. Polecenie

```
cat /etc/passwd>`hostname`.txt
```

spowoduje najpierw wykonanie polecenia hostname (ujętego w akcenty), zastąpienie napisu ujętego w akcenty wynikiem działania polecenia i wykonaniem polecenia

```
cat /etc/passwd>stargate.astrum.lan.txt
```

## ' (apostrof)

Znaki apostrof służą do **dosłownego** cytowania tekstu. Cytowanie napisu zawierającego wiele spacji przy użyciu znaku BACKSLASH jest niewygodne. Stosując apostrofy możemy pisać

```
echo 'Ala     ma     kota'
```

Wszystkie znaki cytowane w apostrofach pozostają niezmiennione

```
#porównaj poniższe instrukcje echo
KAT=`pwd`
echo '$KAT'
echo $KAT
```

## " (cudzysłów)

Znaki cudzysłowia podobnie jak apostrofy służą do cytowania tekstu. Różnica w stosunku do apostrofów polega na tym, że w cytowanym napisie nazwy zmiennych zostaną zamienione wartościami

```
echo "Ala      ma      kota"  
#zadziałała identycznie jak  
echo 'Ala      ma      kota'
```

```
DZIS=`date`  
echo "Dzisiaj jest:   $DZIS"  
echo 'Dzisiaj jest:   $DZIS'
```

Znaki cudzysłowia mają istotne znaczenie, gdy piszemy instrukcje warunkowe lub iteracje.  
Instrukcja

```
if [ $1 ];then  
    echo podano parametr  
fi
```

jest błędna. Jeśli parametr \$1 nie został podany, wówczas interpretator zgłosi błąd składniowy, gdyż (po podstawieniu pustej zmiennej) będzie próbował wykonać:

```
if [ ];then  
    echo podano parametr  
fi
```

Stosując cudzysłów zabezpieczamy się przed takim błędem, gdyż nawet jeśli parametr nie został podany, to instrukcja

```
if [ "$1" ];then  
    echo podano parametr  
fi
```

po podstawieniu wartości przyjmie postać:

```
if [ "" ];then  
    echo podano parametr  
fi
```