Linux

Skrypty powłoki

Spis treści.

Wprowadzenie.

Powłoka (shell) jest interpretatorem poleceń (wydanych przez użytkownika, lub pobranych z pliku) – odpowiednik command.com w systemie Windows.

Polecenia wpisywane z klawiatury zostają zamienione właśnie przez powłokę na język zrozumiały dla jądra systemu.

Linux posiada kilka powłok np. bash, sh, csh, tcsh.

Nazwa powłoka, oraz jądro systemu zaczerpnięto ze składowych orzecha … Z jęz. ang. orzech składa się z jądra (kernel) i skorupy (shell).

Skrypty powłoki to pliki ASCI składające się z poleceń samej powłoki, oraz poleceń sterujących np. if, for, repeat, while itd.

Umożliwiają pobranie danych od użytkownika, lub z pliku, następne ich przetworzenie, oraz wyświetlenie wyniku na monitorze, lub dowolnym urządzeniu, czy zapisaniu wyniku do pliku.

Aby nasz plik mógł nosić miano skryptu, musi mieć prawo wykonania !
chmod +x nasz_plik

Pierwsza linia w skryptach powinna wyglądać następująco:
#!/bin/bash lub #!/bin/sh
Zapis ten informuje, że do przetworzenia skryptu zostanie użyta powłoka bash lub sh.

Polecenie echo, znaki cytowania oraz komentarze.
Polecenie echo służy do wyświetlania tekstu, lub wartości zmiennych.
Opcje polecenia echo:
-n – nie wyświetlaj nic, tylko przejdź do następnej lini
-e – interpretuj komendy po znaku \
\a dzwonek (bell)
\b backspace (kasuje ostatni znak)
\c przechodzi do następnej linii
\n nowa linia
\r powrót karetki
\t tabulator

\\ znak \ (backslash)

Cudzysłów w skryptach występuje w trzech rodzajach:

" - podwójny cudzysłów

' - pojedynczy cudzysłów (apostrof)

` - odwrócony cudzysłów (odwrócony apostrof)

1."Podójny cudzysłów" - Cokolwiek jest zamknięte w podwójny cudzysłów pozbawiane jest znaczenia tych znaków (z wyjątkiem \ i $).
2. 'Pojedynczy cudzysłów' - Zamknięty ciąg w pojedynczy cudzysłów pozbawiane jest znaczenia tych znaków bez wyjątków.
3. `Odwrócony cudzysłów` (tam gdzie znak tyldy) – przydatny wtedy gdy w ciągu znaków przywołujemy jakieś polecenie powłoki np.: `echo "Today is `date +%m/%d/%Y`"`

Komentarz w skrypcie oznaczamy za pomocą znaku **#.**

Przykład:

vi pierwszy_skrypt .sh

Zawartość pliku:

```
#!/bin/bash
# to przykład komentarza – linia nie będzie interpretowana podczas działania skryptu
clear
echo ‘zostaniesz poinformowany w jakim katalogu aktualnie jesteś’
echo –e ”oto bierzacy katalog: \n\a `pwd`”
```

Nadajemy prawo wykonywania utworzonemu plikowi:
chmod +x nazwa_pliku , czyli chmod +x pierwszy_skrypt.sh

Operatory arytmetyczne.

Dodawanie expr 1+3
Odejmowanie expr 5-4
Dzielenie expr 6/2
Reszta z dzielenia expr 20%3
Mnożenie expr 1*10

Definiowanie zmiennych w skryptach powłoki.

```
#!/bin/bash
a="Jakiś tekst"
echo "Zmienna a przechowuje następujący tekst: $a"
```

```
#!/bin/bash
TODAY=`date +%m-%d-%Y`
echo "Today is $TODAY"
```

Jeżeli chcemy aby nasza zmienna była dostępna również po wykonaniu skryptu, dla procesów potomnych, musimy definicję zmiennej poprzedzić słowem **export**, które powoduje, że zmienna ta jest dostępna dla kolejnych procesów po zakończeniu działania naszego skryptu.

export TEST=”Zmienna TEST dostępna globalnie, po zakończeniu skryptu”;

Zmienne systemowe / środowiskowe.

W skryptach można używać także zmiennych środowiskowych. Aby wyświetlić listę wszystkich zmiennych systemowych wpisz polecenie: **env | more** lub **set | more**. Przykładowe zmienne środowiskowe: $HOME – wskazuje na katalog domowy zalogowanego użytkownika, $USER, lub $LOGNAME – login użytkownika, $UID – id użytkownika, $BASH – powłoka użytkownika, $HOSTNAME – nazwa maszyny, .

Pobieranie danych wejściowych od użytkownika.

Do pobrania danych od użytkownika posłuży nam wyrażenie **read**.

Składnia:
Read zmienna1,zmienna2, zmiennaN

Przykład:
#!/bin/bash
echo ‘Podaj Twoje imię:’
read kto
echo ”witaj $kto !”

Instrukcja if.

Składnia:

if warunek
then
polecenie/a
elif warunek 2
then
polecenie/a
else
polecenie/a
fi

Składnia warunku w instrukcji if

[wyrażenie1 operator wyrażenie2]

Między nawiasami kwadratowymi a warunkiem musi być spacja !

Operatory logiczne.

-eq –jest równe (==)
-ne –jest różne (!=)
-lt –jest mniejsze (<)
-le –jest mniejsze, bądź równe (<=)
-gt –jest większe (>)
-ge –jest większe, lub równe (>=)

Operatory po stronie lewej stosujemy dla liczb, po prawej dla napisów.

Sprawdzenie, czy zmienna tekstowa nie jest pusta ?

if [-n „$zmienna”] – pamiętaj aby analizowaną zmienną ująć w znakach cudzysłowu.

Łączniki operatorów logicznych.

Używamy do wykorzystania dwóch lub więcej warunków w jednym czasie.

Operator	Znaczenie
! wyrażenie	Logiczne NIE - negacja
wyrażenie1 -a wyrażenie2	Logiczne AND
wyrażenie1 -o wyrażenie2	Logiczne OR

Przykład skryptu z instrukcją if.

```
#!/bin/sh
clear
echo "skrypt sluzy do zamykania, restartowania systemu !"
echo "jezeli chcesz wylaczyc system podaj 1"
echo "jezeli chcesz zrestartowac system podaj 2"
echo "podaj cyfre:"
read cyfra
if [ $cyfra -eq 2 ]
then
shutdown -r now
elif [ $cyfra -eq 1 ]
then
halt
else
echo "nie wybrales zadnej dostepnej opcji !!!"
fi
```

Instrukcja case.

Przykład instrukcji case:
case $liczba in
10) echo „to jest liczba 10”;echo „przykład kolejnego polecenia oddzielonego średnikiem”;;
20) echo „to jest liczba 20”
echo „przykład kolejnego polecenia w nowym wierszu”;;
*) echo „inna liczba”
esac

Gwiazdka oznacza opcję default – wyświetli ten tekst jeśli żaden z powyższych warunków nie został spełniony.

Przykład zastosowania pętli w skryptach powłoki.

```
#!/bin/bash
for i in 1 2 3
do
echo $i
done
```

```
#!/bin/bash
for i in `find /home/pkania -name *mp3`
do
rm $i
done
```

Pętla until – instrukje wykonywane są dopóki warunek jest fałszywy. Jeżeli warunek stanie się prawdziwy pętla jest przerywana. Przykład: skrypt sprawdza co 2 sekundy, czy root jest zalogowany.

until who | grep root > /dev/null
do
sleep 2
done
echo –e \\a
echo “root się zalogował”

Zastosowanie funkcji.

Składnia: function function_name{
commands
commands
}

Przykład:

function utworz_katalog(){
mkdir $1
cd $1

```
echo `pwd`
}
```

Wywołanie funkcji w skrypcie:

utworz_katalog /tmp/test

Funkcje możesz zapisać w osobnym pliku. Aby się do nich odwołać w dowolnym skrypcie możesz użyć polecenia **source nazwa_pliku_z_funkcjami**. Source działa jak include.

Parametry wejściowe.

Podczas wywoływania skryptu możemy podawać tzw. parametry wejściowe. Parametry te są przechowywane w zmiennych o nazwach od **$0** do **$9**. **$0** to nazwa wywoływanego skryptu, kolejne zmienne od $1 do $9 reprezentują parametry wpisane po nazwie skryptu, które są oddzielane znakiem spacji. W ten sposób nasze skrypty stają się bardziej uniwersalne. Praktyczny przykład zastosowania parametru wejściowego znajdziesz poniżej w rozdziale **Przykładowe skrypty**.

$* - podaje wszystkie argumenty wejściowe.

$# - podaje liczbę zastosowanych parametrów.

$$ - podaje numer procesu wywołanego skryptu.

Przykład poglądowy:

Treść pliku **przyklad.sh**:

```
#!/bin/bash
echo „nazwa skryptu: $0”
echo „Liczba parametrów: $#”
echo „Nr procesu: $$”
echo „Parametr1: $1”
echo „Parametr2: $2”
echo „Parametr3: $3”
```

Wywołaj powyższego skrypt w ten sposób**: ./przyklad.sh 123 345 567**
Zaobserwuj otrzymany wynik**.**

Sprawdzenie, czy podano jakikolwiek parametr.

Jeżeli jakikolwiek parametr jest bezwzględnie wymagany, w kodzie skryptu możesz to sprawdzić w następujący sposób:

```
if [ $# -ne 0 ]
then
echo ‘podałeś parametr.’
else
echo ‘nie podałeś żadnego parametru.’
```

Powyższy kod sprawdza, czy jakikolwiek parametr został podany podczas wywołania skryptu.

Status wyjścia.

Jeżeli wykonamy jakieś polecenie zwracany jest tzw. status wyjścia. Jeżeli polecenie zakończyło się bezbłędnie zwróconą wartością będzie zero (0), jeżeli wartość zwrócona jest większa od zera (>0), polecenie nie wykonało się poprawnie. Aby zobaczyć status ostatnio wydanego polecenia wpisz: echo **$?** Wartość zmiennej **$?** możesz wykorzystać w swoich skryptach, aby sprawdzić, czy dane polecenie wykonało się prawidłowo.

Przykłady:

1) Wydaj polecenie: **pwd**, następnie **echo $?**. Zobaczysz **0** (zero) – polecenie wykonało się prawidłowo.
2) Wydaj polecenie: **rm plik_ktorego_nie_ma**, następnie **echo $?**. Zobaczysz cyfrę **1** (oznaczającą kod błędu polecenia rm) – polecenie nie zostało prawidłowo wykonane.

Zastosowanie praktyczne w instrukcji warunkowej oraz wykonanie względne (&&, ||).

if <polecenie>
then <lista_poleceń 1>
else <lista poleceń 2>
fi

Powyższe możemy zapisać również jako:

if <polecenie>; then <lista_poleceń 1>; else <lista poleceń 2>; fi

Znak ; (średnika) informuje powłokę, gdzie kończy się polecenie i zaczyna nowe.

Powłoka wykonuje dane polecenie. Jeżeli kod błędu tego polecenia = 0 to wykonywana jest lista poleceń 1, w przeciwnym razie wykonywana jest lista poleceń 2. Część else jest opcjonalna.

Praktyczny przykład:

if test –f plik.txt
then rm plik.txt

fi

To samo uzyskamy wpisując:

test –f plik.txt && rm plik.txt

&& - wykonaj drugie polecenie, jeżeli pierwsze zakończy się bezbłędnie.

|| - wykonaj drugie polecenie jeżeli pierwsze zakończy się błędnie.

Przykłady różnych skryptów.

Skrypt zamieniający duże litery na małe w nazwach plików w danym katalogu:

```
#!/bin/bash
for i in *
do
   lower=`echo $i | tr [:upper:] [:lower:]`
   mv $i $lower
done
```

Skrypt sprawdzający, czy dany plik istnieje (z zastosowaniem polecenia test, więcej informacji na temat tego przydatnego polecenia: **man test**, a poniżej znajdziesz tabelę z wybranymi opcjami tego polecenia):

```
#!/bin/bash
if test -e ~/data.sh
then
echo -e "plik data.sh istnieje"
else
echo -e "plik data.sh nie istnieje"
fi
```

Test	Znaczenie
-s plik	Nie pusty plik
-f plik	Plik istnieje lub jest to plik, a nie katalog
-d katalog	Katalog, nie plik
-w plik	czy plik jest zapisywalny
-r plik	czy plik jest tylko do odczytu
-x plik	czy plik jest wykonywalny

Zamiast definiować nazwę poszukiwanego pliku bezpośrednio w skrypcie można użyć również tzw. parametru wejściowego, który wskazuje na nazwę poszukiwanego pliku np.: **./szukaj.sh dokument.txt**. Wtedy powyższy skrypt będzie wyglądał następująco:

```
#!/bin/bash
#sprawdzam, czy parametr został podany przy wywołaniu skryptu
if [ -n "$1" ]
then
if test -e $1
then
```

```
echo -e "plik $1 istnieje"
else
echo -e "plik $1 nie istnieje"
fi
else
echo "Nie podano parametru z nazwą pliku"
fi
```

Skrypt dodający użytkownika (z pytaniem o jego katalog domowy):

```
#!/bin/bash
echo "Skrypt administracyjny - dodanie uzytkownika."
echo "Autor: Piotr Kania"
echo "wpisz login uzytkownika:"
read login

echo -e "Czy tworzyć katalog domowy ?\nWpisz TAK lub NIE:"
read d
if [ $d == "tak" -o $d == "TAK" ]
then
userdir="-m -d /home/$login"
fi

spr_login="`cut -d : -f1 /etc/passwd | grep --line-regexp $login`"

if [ $spr_login == $login ]
then
echo "Użytkownik o loginie $spr_login już istnieje."
else
echo "wykonuję polecenie useradd $login $userdir"
echo "`useradd $login $userdir`"
fi
```

Zaawansowany skrypt do tworzenia kopii zapasowych:

```
#!/bin/bash
SHELL=/bin/bash
USER=jboss
HOME=/home/jboss
PATH=/usr/bin:/bin:/srv/jboss_std:/srv/PostgreSQL/8.3/bin
LOG=/root/kopia_PROGRAMU.log
KATALOG_ARCH=/srv/archiwum
DZIS=$(date +%Y-%m-%d)
DZIS_Z=$(date +%Y-%m-%d_OLD_%H:%M:%S)
if [ -e $KATALOG_ARCH/$DZIS ]
then
zip -9 -r $KATALOG_ARCH/$DZIS_Z $KATALOG_ARCH/$DZIS
rm -r $KATALOG_ARCH/$DZIS
fi
cd /srv/AdmNarz/
./backup_create.sh -noverify -novacuum
#znajdz kopie, ktore zostaly utworzone wczesiej niz 5 dni temu i je usun
find $KATALOG_ARCH/ -mtime +5 -type d -exec rm -rf {} \;
find $KATALOG_ARCH/ -mtime +5 -type f -exec rm f {} \;

if [ -e $KATALOG_ARCH/$DZIS/PROGRAM.dump ]
then
echo -e "#####################################" >> $LOG
```

```
echo -e $DZIS >> $LOG
echo -e "kopia OK" >> $LOG
else
echo -e "####################################" >> $LOG
echo -e $DZIS >> $LOG
echo -e "kopia BAD" >> $LOG
fi
```