Systemy operacyjne 06

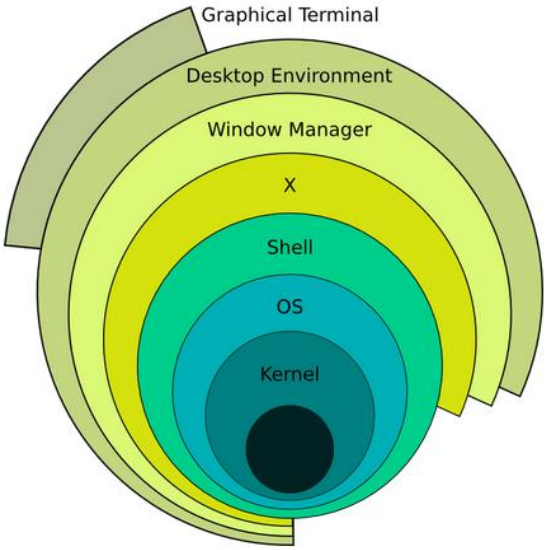
Wstęp do skryptów bashowych

Bash (Bourne-Again Shell) jest najpopularniejszą odmianą shell'a. Innymi znanymi są np. ksh (Korn Shell), csh (C Shell).

Shell możemy kojarzyć z linią poleceń, dzięki której możemy wpisywać komendy z klawiatury. Służą one na ogół do uruchamiania innych programów, poleceń lub wyświetlania informacji. Shell służy też do uruchamiania własnoręcznie napisanych skryptów.

Najprostsze skrypty są po prostu ciągiem poleceń, ale możemy też pisać bardziej skomplikowane mechanizmy, które w istocie rzeczy można nazwać wręcz programami. Składnia skryptów nie jest skomplikowana. Oprócz różnych programów użytkowych i systemowych jest zaledwie kilka reguł, których wystarczy się nauczyć, żeby swobodnie pisać skrypty składające poznane programy w celu wykonania zaplanowanych działań.

Shell jest podstawowym narzędziem pod system typu UNIX, ale nie tylko. Pod Windowsem mamy np. interpretator poleceń, a skryptami są np. pliki o rozszerzeniu bat. Niemniej możliwości i funkcjonalność takich skryptów pod Windowsem są znikome (w porównania do Basha pod Unixem) ze względu na ograniczenia składniowe, jak i dostępny zbiór poleceń użytkowych. Aby uzyskać możliwość uruchamiania skryptów napisanych w Bashu pod Windowsem możemy sobie zainstalować pakiet Cygwin



Zastosowanie

- kompilacja, budowanie aplikacji,
- przetwarzanie plików (tworzenie, usuwanie, szukanie, itp.),
- prosta obróbka (np. tekstowa) plików,
- administracja systemem (np. konfiguracja, uruchamianie demonów)

1. Pierwszy skrypt

Utwórz plik, w którym umieszczony zostanie pierwszy skrypt:

```
touch skrypt
```

Następnie za pomocą dowolnego edytora tekstowego wpisz do niego następującą zawartość:

```
#!/bin/bash

#To jest komentarz.

echo "Hello world"
```

Znak **#** (hasz) oznacza komentarz, wszystko co znajduje się za nim w tej samej linii, jest pomijane przez interpreter. Pierwsza linia skryptu zaczynająca się od znaków: **#!** ma szczególne znaczenie - wskazuje na rodzaj shella w jakim skrypt ma być wykonany, tutaj skrypt zawsze będzie wykonywany przez interpreter poleceń **/bin/bash**, niezależnie od tego jakiego rodzaju powłoki w danej chwili używamy.

```
echo "Hello World"
```

Wydrukuje na standardowym wyjściu (stdout) czyli na ekranie napis: **Hello World**.

Aby móc uruchomić skrypt należy mu jescze nadać atrybut wykonywalności za pomocą polecenia:

```
chmod +x skrypt
```

Jeśli katalog bieżący w którym znajduje się skrypt nie jest dopisany do zmiennej PATH, to nasz skrypt możemy ururchomić w ten sposób:

```
./naszskrypt
```

podobnie można podać również pełną ścieżkę:

```
/home/user/skrypt
```

lub

```
bash skrypt

source skrypt
```

2. 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

3. Cytowania

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, umożliwia zacytowanie polecenia

Przykład:

```
#!/bin/bash
x=2
echo "Wartość zmiennej x to $x" #wydrukuje Wartość zmiennej x
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
```

4. Zmienne programowe

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"
```

5. Zmienne specjalne

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 przekazywane 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
```

będzie wyglądał następująco:

```
Skrypt uruchomiono z paramertami -a d
```

$? - kod powrotu ostatnio wykonywanego polecenia

$$ - PID procesu bieżącej powłoki

6. Zmienne środowiskowe

Definiują 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 nano i wpisz:

```
#!/bin/bash
x="napis"
echo $x
```

`x="napis"` zdefiniowałeś właśnie zmienną x, która ma wartość "napis"

`echo $x` wyświetli wartość zmiennej x

Możesz teraz zainicjować zmienną globalną:

```
export x="napisGlobal"
```

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
```

7. Zmienne tablicowe

BASH pozwala na stosowanie zmiennych tablicowych jednowymiarowych. Czym jest tablica? To zmienna która przechowuje listę jakichś wartości (rozdzielonych spacjami), w BASH'u nie ma maksymalnego rozmiaru tablic. Kolejne wartości zmiennej tablicowej indexowane są przy pomocy liczb całkowitych, zaczynając od 0.

Składnia

```
zmienna=(wartość1 wartość2 wartość3 wartośćn)
```

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[0]}
echo ${tablica[1]}
echo ${tablica[2]}
```

Zadeklarowana została zmienna tablicowa o nazwie: tablica, zawierająca trzy wartości: element1 element2 element3. Natomiast polecenie: echo ${tablica[0]} wydrukuje na ekranie pierwszy elementu tablicy. W powyższym przykładzie w ten sposób wypisana zostanie cała zawartość tablicy. Do elementów tablicy odwołuje się za pomocą wskaźników.

Odwołanie do elementów tablicy

Składnia:

```
${nazwa_zmiennej[wskaźnik]}
```

Wskaźnikami są indexy elementów tablicy, począwszy od 0 do n oraz @, *. Gdy odwołując się do zmiennej nie poda się wskaźnika: ${nazwa_zmiennej} to nastąpi odwołanie do elementu

tablicy o indexie 0.Jeśli wskaźnikiem będą: @ lub * to zinterpretowane zostaną jako wszytskie elementy tablicy, w przypadku gdy tablica nie zawiera żadnych elemntów to zapisy: ${nazwa_zmiennej[wskaźnik]} lub ${nazwa_zmiennej[wskaźnik]} są interpretowane jako nic.

Przykład:

Poniższy skrypt robi to samo co wcześniejszy.

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[*]}
```

Można też uzyskać długość (liczba znaków) danego elementu tablicy:

```
${#nazwa_zmiennej[wskaźnik]}
```

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${#tablica[0]}
```

Polecenie echo ${#tablica[0]} wydrukuje liczbę znaków z jakich składa się pierwszy element tablicy: element1 wynik to 8. W podobny sposób można otrzymać liczbę wszystkich elementów tablicy, wystarczy jako wskaźnik podać: @ lub *.

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${#tablica[@]}
```

Co da wynik: 3.

Dodawanie elementów do tablicy

Składnia:

```
nazwa_zmiennej[wskaźnik]=wartość
```

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
tablica[3]=element4
echo ${tablica[@]}
```

Jak wyżej widać do tablicy został dodany element4 o indexie 3. Mechanizm dodawania elementów do tablicy, można wykorzystać do tworzenia tablic, gdy nie istnieje zmienna tablicowa do której dodajemy jakiś element, to BASH automatycznie ją utworzy:

```
#!/bin/bash
linux[0]=slackware
linux[1]=debian
echo ${linux[@]}
```

Utworzona została tablica linux zawierająca dwa elementy.

Usuwanie elementów tablic i całych tablic

Dany element tablicy usuwa się za pomocą polecenia unset.

Składnia:

```
unset nazwa_zmiennej[wskaźnik]
```

Przykład:

```
#!/bin/bash
tablica=(element1 element2 element3)
echo ${tablica[@]}
unset tablica[2]
echo ${tablica[*]}
```

Usunięty został ostatni element tablicy.

Aby usunąć całą tablicę wystarczy podać jako wskaźnik: @ lub *.

```
#!/bin/bash
tablica=(element1 element2 element3)
unset tablica[*]
echo ${tablica[@]}
```

Zmienna tablicowa o nazwie tablica przestała istnieć, polecenie: echo ${tablica[@]} nie wyświetli nic.

Zadania do wykonania

1. Utwórz skrypt wykorzystującego zmienne środowiskowe, zmienne programowe i odwrotne apostrofy (Użyj $USER, $PWD, $HOME.
2. Utwórz skrypt wykorzystującego zmienne specjalne. (Użyj $0, $1, $2, ..., $9, $@, $*, $?, $$.)
3. Podać przykład skryptu używającego tablice (Użyj unset, *, @, #).