Skrypty BASH'a

Systemy Operacyjne 2

Mateusz Hołenko

4 października 2012

Plan zajęć

- O skryptach słów kilka...
- Powłoka, wiersz poleceń
- Obsługa powłoki bash
- Składnia języka skryptowego bash'a
- Zadania

Wstęp teoretyczny

O skryptach słów kilka...

Skrypt to program napisany z użyciem *języka skryptowego*.
Języki skryptowe to jęzki programowania przenaczone do użycia w celu automatyzacji działania konkretnych aplikacji.
Języki skryptowe są często *językami interpretowanymi*.

Powłoka

- **powłoka** (ang. *shell*) stanowi interfejs użytkownika pozwalający mu na kontrolowanie systememu operacyjnego
- bash (ang. *Bourne-Again SHell*) to popularna powłoka dla systemów uniksowych
- bash posiada własny język do pisania skryptów
- bash dostarcza interfejs typu wiersz poleceń

Powłoka

- **powłoka** (ang. *shell*) stanowi interfejs użytkownika pozwalający mu na kontrolowanie systememu operacyjnego
- bash (ang. *Bourne-Again SHell*) to popularna powłoka dla systemów uniksowych
- bash posiada własny język do pisania skryptów
- bash dostarcza interfejs typu wiersz poleceń

Powłoka

- **powłoka** (ang. *shell*) stanowi interfejs użytkownika pozwalający mu na kontrolowanie systememu operacyjnego
- bash (ang. *Bourne-Again SHell*) to popularna powłoka dla systemów uniksowych
- bash posiada własny język do pisania skryptów
- bash dostarcza interfejs typu wiersz poleceń

Powłoka

- **powłoka** (ang. *shell*) stanowi interfejs użytkownika pozwalający mu na kontrolowanie systememu operacyjnego
- bash (ang. *Bourne-Again SHell*) to popularna powłoka dla systemów uniksowych
- bash posiada własny język do pisania skryptów
- bash dostarcza interfejs typu wiersz poleceń

Obsługa powłoki bash

- wydawanie poleceń
  - autouzupełnianie
  - możliwosć edycji
  - historia poleceń
  - nazwy wieloznaczne
- rozwijanie nawiasów klamrowych
- zmienne środowiskowe
- przekierowanie strumieni danych

Obsługa powłoki bash

- wydawanie poleceń
  - autouzupełnianie
  - możliwosć edycji
  - historia poleceń
  - nazwy wieloznaczne
- rozwijanie nawiasów klamrowych
- zmienne środowiskowe
- przekierowanie strumieni danych

Obsługa powłoki bash

- wydawanie poleceń
  - autouzupełnianie
  - możliwosć edycji
  - historia poleceń
  - nazwy wieloznaczne
- rozwijanie nawiasów klamrowych
- zmienne środowiskowe
- przekierowanie strumieni danych

Obsługa powłoki bash

- wydawanie poleceń
  - autouzupełnianie
  - możliwosć edycji
  - historia poleceń
  - nazwy wieloznaczne
- rozwijanie nawiasów klamrowych
- zmienne środowiskowe
- przekierowanie strumieni danych

Język skryptowy powłoki BASH

Pierwszy skrypt

```
#!/bin/bash
# To jest komentarz

echo "Witaj Świecie!"
```

Interakcja z użytkownikiem

- wyprowadzanie tekstu na ekran
  - **echo** wyświetla wiersz tekstu
    - **-n** nie wypisuje kończącego znaku nowej linii
    - **-e** włącza interpretowanie sekwencji specjalnych np. \a, \t, \\, \nnn, \xnnn
  - **printf** wypisuje sformatowany tekst

- wczytywanie danych od użytkownika
  - **read** czyta wiersz ze standardowego wejścia
    - **-s** tryb cichy — brak echa
    - **-t** oczekiwanie przez skończony czas
    - **-n** oczekuje na *n* znaków
    - **-d** określa znak końca wczytywania

Interakcja z użytkownikiem

- wyprowadzanie tekstu na ekran

  echo wyświetla wiersz tekstu

  - **-n** nie wypisuje kończącego znaku nowej linii
  - **-e** włącza interpretowanie sekwencji specjalnych np. \a, \t, \\, \nnn, \xnnn

  printf wypisuje sformatowany tekst

- wczytywanie danych od użytkownika

  read czyta wiersz ze standardowego wejścia

  - **-s** tryb cichy — brak echa
  - **-t** oczekiwanie przez skończony czas
  - **-n** oczekuje na *n* znaków
  - **-d** określa znak końca wczytywania

Zmienne

Zmienna to obszar pamięci służący do przechowywania danych podczas działania programu.

```
liczba=12
tekst="Wartością zmiennej \$liczba jest $liczba"
tekst2='Wartością zmiennej \$liczba jest $liczba'
polecenie=`pwd`
polecenie2=$(pwd)

echo $liczba $tekst $tekst2 $polecenie $polecenie2
```

Uwaga: wokół znaku przypisania nie może być spacji!

Zmienne

Zmienna to obszar pamięci służący do przechowywania danych podczas działania programu.

```
liczba=12
tekst="Wartością zmiennej \$liczba jest $liczba"
tekst2='Wartością zmiennej \$liczba jest $liczba'
polecenie=`pwd`
polecenie2=$(pwd)

echo $liczba $tekst $tekst2 $polecenie $polecenie2
```

Uwaga: wokół znaku przypisania nie może być spacji!

Zmienne specjalne

$0 nazwa skryptu

$1 pierwszy parametr przekazany do skryptu

...

$9 dziewiąty parametr przekazany do skryptu

$@ lista wszystkich parametrów przekazanych do skryptu

$# ilość parametrów przekazanych do skryptu

$? kod zakończenia ostatniego polecenia

Zmienne środowiskowe

Zmienne zdefiniowane w sposób pokazany wcześniej mają zakres **lokalny** — dostępne są jedynie w powłoce, w której zostały zdefiniowane.

Definicja **zmiennej środowiskowej** (widocznej również w powłokach potomnych):

```
export glob=3

printenv # wypisuje zmienne środowiskowe
```

Zmienne środowiskowe

Zmienne zdefiniowane w sposób pokazany wcześniej mają zakres **lokalny** — dostępne są jedynie w powłoce, w której zostały zdefiniowane.

Definicja **zmiennej środowiskowej**
(widocznej również w powłokach potomnych):

```
export glob=3

printenv # wypisuje zmienne środowiskowe
```

Tablice

Tablice pozwalają reprezentować kolekcje elementów (zmiennych).

```
tablica=(wartosc1 wartosc2 wartosc3)

echo ${tablica[0]} # pierwszy element tablicy
echo ${tablica[1]} # drugi element tablicy
echo ${tablica[2]} # trzeci element tablicy
echo ${tablica[*]} # lub: echo ${tablica[@]};
   wszystkie elementy tablicy

tablica[3]="wartosc4" # rozszerzanie tablicy
unset tablica[3] # skracanie tablicy

echo ${#tablica[*]} # długość tablicy
```

Tablice

Tablice pozwalają reprezentować kolekcje elementów (zmiennych).

```
tablica=(wartosc1 wartosc2 wartosc3)

echo ${tablica[0]} # pierwszy element tablicy
echo ${tablica[1]} # drugi element tablicy
echo ${tablica[2]} # trzeci element tablicy
echo ${tablica[*]} # lub: echo ${tablica[@]};
    wszystkie elementy tablicy

tablica[3]="wartosc4" # rozszerzanie tablicy
unset tablica[3] # skracanie tablicy

echo ${#tablica[*]} # długość tablicy
```

Instrukcja warunkowa

Instrukcja warunkowa **if** pozwala kontrolować przepływ wykonania programu.

```
if [ 1 -gt 2 ]
then
    echo '1 jest większe niż 2'
elif [ 2 -gt 1 ]
    echo '2 jest większe niż 1'
else
    echo '1 jest równe 2'
fi
```

Instrukcja warunkowa

Instrukcja warunkowa **if** pozwala kontrolować przepływ wykonania programu.

```
if [ 1 -gt 2 ]
then
    echo '1 jest większe niż 2'
elif [ 2 -gt 1 ]
    echo '2 jest większe niż 1'
else
    echo '1 jest równe 2'
fi
```

Polecenie test

Polecenie `test` wyznacza wartość prostych wyrażeń logicznych:

-d nazwa plik jest katalogiem

-f nazwa plik jest zwykłym plikiem

-L nazwa plik jest dowiązaniem symbolicznym

-r nazwa plik istnieje i można go odczytać

-w nazwa plik istnieje i można do niego pisać

-x nazwa plik istnieje i można go uruchomić

-s nazwa plik istnieje i ma niezerową wielkość

f1 -nt f2 plik f1 jest nowszy niż f2

f1 -ot f2 plik f1 jest starszy niż f2

Uwaga: Wynik pozytywny testu (tj. prawda) sygnalizowany jest za pomocą wartości zwróconej **0**, fałsz zaś jako **1**.

Polecenie test

Polecenie `test` wyznacza wartość prostych wyrażeń logicznych:

-d nazwa plik jest katalogiem

-f nazwa plik jest zwykłym plikiem

-L nazwa plik jest dowiązaniem symbolicznym

-r nazwa plik istnieje i można go odczytać

-w nazwa plik istnieje i można do niego pisać

-x nazwa plik istnieje i można go uruchomić

-s nazwa plik istnieje i ma niezerową wielkość

f1 -nt f2 plik f1 jest nowszy niż f2

f1 -ot f2 plik f1 jest starszy niż f2

Uwaga: Wynik pozytywny testu (tj. prawda) sygnalizowany jest za pomocą wartości zwróconej **0**, fałsz zaś jako **1**.

Polecenie test, cd.

s1 = s2 ciąg znaków s1 jest identyczny z ciągiem s2

s1 != s2 ciąg znaków s1 nie jest identyczny z ciągiem s2

-z s1 ciąg znaków ma zerową długość

-n nazwa ciąg znaków ma niezerową długość

a -eq b wartości całkowite a i b są sobie równe

a -ne b wartości całkowite a i b nie są sobie równe

a -gt b wartość całkowita a jest większa od b

a -lt b wartość całkowita a jest mniejsza niż b

a -ge b wartość całkowita a jest nie mniejsza niż b

a -le b wartość całkowita a jest nie większa niż b

t1 -a t2 iloczy logiczny testów

t1 -o t2 suma logiczna testów

! t1 negacja testu

\(t1 \) grupowanie testów

Instrukcja case

Instrukcja **case** wybiera akcję na podstawie pasującego wzorca.

```
#!/bin/bash
echo "Podaj cyfrę dnia tygodnia"
read d
case "$d" in
  "1") echo "Poniedziałek" ;;
  "2") echo "Wtorek" ;;
  "3") echo "Środa" ;;
  "4") echo "Czwartek" ;;
  "5") echo "Piątek" ;;
  "6") ;&
  "7") echo "Weekend" ;; # alternatywa: [67]
  *) echo "Nic nie wybrałeś"
esac
```

Instrukcja case

Instrukcja **case** wybiera akcję na podstawie pasującego wzorca.

```
#!/bin/bash
echo "Podaj cyfrę dnia tygodnia"
read d
case "$d" in
  "1") echo "Poniedziałek" ;;
  "2") echo "Wtorek" ;;
  "3") echo "Środa" ;;
  "4") echo "Czwartek" ;;
  "5") echo "Piątek" ;;
  "6") ;&
  "7") echo "Weekend" ;; # alternatywa: [67]
  *) echo "Nic nie wybrałeś"
esac
```

Pętla for

Pętla **for** służy do wykonania akcji na każdym elemencie listy.

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

Kontrola pętli

Do kontrolowania przebiegu kolejnych iteracji pętli wykorzystać można słowa kluczowe `break` oraz `continue`.

Warto wiedzieć

Zapoznaj się z poleceniem `seq`.

Pętla for

Pętla **for** służy do wykonania akcji na każdym elemencie listy.

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

Kontrola pętli

Do kontrolowania przebiegu kolejnych iteracji pętli wykorzystać można słowa kluczowe `break` oraz `continue`.

Warto wiedzieć

Zapoznaj się z poleceniem `seq`.

Pętla for

Pętla **for** służy do wykonania akcji na każdym elemencie listy.

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

Kontrola pętli

Do kontrolowania przebiegu kolejnych iteracji pętli wykorzystać można słowa kluczowe `break` oraz `continue`.

Warto wiedzieć

Zapoznaj się z poleceniem `seq`.

Pętle while oraz until

Pętle **while** oraz **until** wykonują akcje do momentu niespełnienia (*while*) / spełnienia (*until*) warunku.

```
#!/bin/bash
x=1;
while [ $x -le 10 ]; do
    echo "Napis pojawił się po raz: $x"
    x=$[x + 1]
done

x=1;
until [ $x -ge 10 ]; do
   echo "Napis pojawił się po raz: $x"
   x=$[x + 1]
done
```

Pętle while oraz until

Pętle **while** oraz **until** wykonują akcje do momentu niespełnienia (*while*) / spełnienia (*until*) warunku.

```
#!/bin/bash
x=1;
while [ $x -le 10 ]; do
    echo "Napis pojawił się po raz: $x"
    x=$[x + 1]
done

x=1;
until [ $x -ge 10 ]; do
   echo "Napis pojawił się po raz: $x"
   x=$[x + 1]
done
```

Funkcje

Kawałki kodu, które powtarzają się w skrypcie można zapisać jako **funkcje** i wywoływać wielokrotnie w dowolnych miejscach skryptu.

```
function test()
{
    echo "Wywołano mnie z $# parametrami"
}

test raz dwa trzy
```

Funkcje

Kawałki kodu, które powtarzają się w skrypcie można zapisać jako **funkcje** i wywoływać wielokrotnie w dowolnych miejscach skryptu.

```
function test()
{
    echo "Wywołano mnie z $# parametrami"
}

test raz dwa trzy
```

Obliczanie wartości arytmetycznych

Powłoka bash wspiera operacje na liczbach całkowitych.

```
#!/bin/bash

echo $((100 + 40 + 7))

wynik=$[2**10]
echo "$wynik"
```

Warto zobaczyć polecenie `let`

Obliczanie wartości arytmetycznych

Powłoka bash wspiera operacje na liczbach całkowitych.

```
#!/bin/bash

echo $((100 + 40 + 7))

wynik=$[2**10]
echo "$wynik"
```

Warto zobaczyć polecenie `let`

Tworzenie interaktywnych skryptów

- wykorzystanie wbudowanych poleceń `read`, `echo`
- polecenie `select`
- program `dialog`

Uwaga: Aplikacja `dialog` nie jest domyślnie instalowana we wszystkich dystrybucjach!

Tworzenie interaktywnych skryptów

- wykorzystanie wbudowanych poleceń `read`, `echo`
- polecenie `select`
- program `dialog`

Uwaga: Aplikacja `dialog` nie jest domyślnie instalowana we wszystkich dystrybucjach!

Tworzenie interaktywnych skryptów

- wykorzystanie wbudowanych poleceń `read`, `echo`
- polecenie `select`
- program `dialog`

Uwaga: Aplikacja `dialog` nie jest domyślnie instalowana we wszystkich dystrybucjach!

Tworzenie interaktywnych skryptów

- wykorzystanie wbudowanych poleceń `read`, `echo`
- polecenie `select`
- program `dialog`

Uwaga: Aplikacja `dialog` nie jest domyślnie instalowana we wszystkich dystrybucjach!

Na zakończenie

- kończenie działania skryptu — polecenie `exit`

```
exit 15
```

- włączanie zawartości innego pliku — operator `.`

```
. moj_skrypt.sh
```

Na zakończenie

- kończenie działania skryptu — polecenie `exit`

```
exit 15
```

- włączanie zawartości innego pliku — operator `.`

```
. moj_skrypt.sh
```

Zadania

Zadania praktyczne

Zadanie 1

Napisać skrypt sprawdzający czy istnieje podany jako parametr plik i wypisujący na ekranie odpowiedni komunikat.

Zadanie 2

Jak wyżej, jeśli plik istnieje powinien pojawić się komunikat, w przeciwnym razie plik taki powinien zostać utworzony tak, by jego pierwszą linię stanowił napis *To jest nowy plik*.

Zadania praktyczne

Zadanie 1

Napisać skrypt sprawdzający czy istnieje podany jako parametr plik i wypisujący na ekranie odpowiedni komunikat.

Zadanie 2

Jak wyżej, jeśli plik istnieje powinien pojawić się komunikat, w przeciwnym razie plik taki powinien zostać utworzony tak, by jego pierwszą linię stanowił napis *To jest nowy plik*.

Zadania praktyczne, cd.

Zadanie 3

Napisać skrypt zawierający informacje o każdym podkatalogu znajdującym się w katalogu podanym jako argument, obejmującą nazwę podkatalogu i liczbę znajdujących się w nim plików.

Zadanie 4

Napisać skrypt: jeśli podana jest nazwa katalogu jako argument to wylistować zawartość katalogu, w przeciwnym wypadku należy zapytać użytkownika o nazwę katalogu i wylistować podany katalog.

Zadania praktyczne, cd.

Zadanie 3

Napisać skrypt zawierający informacje o każdym podkatalogu znajdującym się w katalogu podanym jako argument, obejmującą nazwę podkatalogu i liczbę znajdujących się w nim plików.

Zadanie 4

Napisać skrypt: jeśli podana jest nazwa katalogu jako argument to wylistować zawartość katalogu, w przeciwnym wypadku należy zapytać użytkownika o nazwę katalogu i wylistować podany katalog.

Zadania praktyczne, cd.

Zadanie 5

Napisać skrypt wyświetlający w odwrotnej kolejności argumenty jego wywołania. (np. skrypt a b c d => d c b a)

Zadanie 6

Skrypt tworzący nazwę projektu, zmiennej, itp. wg określonego formatu podanego jako pierwszy parametr:

```
./chcase joined my new project # mynewproject
./chcase underline my new var # my_new_project
./chcase uppercase my new const # MY_NEW_CONST
./chcase dashes my new resource # my-new-resources
```

Obsłuż błędy.

Zadania praktyczne, cd.

Zadanie 5

Napisać skrypt wyświetlający w odwrotnej kolejności argumenty jego wywołania. (np. skrypt a b c d => d c b a)

Zadanie 6

Skrypt tworzący nazwę projektu, zmiennej, itp. wg określonego formatu podanego jako pierwszy parametr:

```
./chcase joined my new project # mynewproject
./chcase underline my new var # my_new_project
./chcase uppercase my new const # MY_NEW_CONST
./chcase dashes my new resource # my-new-resources
```

Obsłuż błędy.

Zadania praktyczne, cd.

Zadanie 7

Skrypt-predykat, który sprawdza porę dnia. Możliwe pory dnia: early, late, day, night, morning, lunchtime, evening.
Jeśli użytkownik poda porę dnia nieznaną skryptowi należy wyświetlić komunikat o błędzie.