Laboratorium 3 – Skrypty powłoki systemu operacyjnego

Interpreter poleceń nazywany inaczej także powłoką systemową pośredniczy pomiędzy użytkownikiem a funkcjami systemu operacyjnego. Powłoka systemowa pobiera dane i polecenia od użytkownika i przekazuje je do wykonania do jądra systemu operacyjnego.

Zmienne środowiskowe to bardzo wygodny i uniwersalny sposób konfigurowania i parametryzowania powłok systemowych a także innych programów. Dostępne zmienne środowiskowe tworzą tzw. środowisko wykonania procesu – środowisko to jest kopiowane do wszystkich nowych procesów, a więc modyfikacje zmiennych wykonane w powłoce są widoczne we wszystkich programach uruchomionych przy użyciu tej powłoki. Każdy użytkownik może definiować dowolną ilość własnych zmiennych oraz przypisywać im dowolne wartości. Aby zdefiniować zmienną środowiskową należy zastosować operator przypisania (znak „=") w następujący sposób: `ZMIENNA=wartosc` **WAŻNE – pisać bez odstępów**.

Odwołanie się do wartości zmiennej jest możliwe dzięki znakowi specjalnemu **$** np.: **`echo $SYSTEM`**. Polecenie **`set`** wyświetla wartości wszystkich zmiennych środowiskowych, a polecenie **`unset`** usuwa zmienną środowiskową.

Zmienne które są przekazywane do procesów potomnych nazywa się zmiennymi eksportowanymi, a zmienne, które nie są przekazywane, nazywa się zmiennymi lokalnymi. Nowo tworzone zmienne są zmiennymi lokalnymi i niezbędne jest jawne wskazanie, że mają być zmiennymi eksportowanymi, służy do tego polecenie: **`export`** `lista_zmiennych`.

Każdy użytkownik może sprawdzić, że w systemie jest zdefiniowanych wiele zmiennych środowiskowych, oto znaczenie podstawowych z nich:

- `HOME` – ścieżka i nazwa katalogu domowego użytkownika;
- `USER` – nazwa zalogowanego użytkownika;
- `PS1` – postać znaku zachęty użytkownika;
- `SHELL` – pełna ścieżka do domyślnego interpretatora poleceń użytkownika.

Skrypty powłoki to pliki tekstowe, które zawierają ciągi poleceń dla powłoki systemowej. Skrypty mogą być parametryzowane argumentami ich wywołania – oznacza to, że podczas uruchamiania skryptu można przekazać do niego dowolną ilość dowolnych danych. Do argumentów wywołania można się odwoływać w skryptach za pomocą tzw. zmiennych pozycyjnych, oznaczanych kolejnymi numerami: 1,2,3...9. Każda zmienna pozycyjna przechowuje tekst przekazany za pośrednictwem odpowiadającego jej argumentu wywołania. Przykład skryptu, który wyświetla wartości pierwszych trzech argumentów jego wywołania:

```
#!/bin/bash
# pierwszy skrypt
echo "argument nr 1: $1"
echo "argument nr 2: $2"
echo "argument nr 3: $3"
```

Takie polecenia należy zapisać do dowolnego pliku (zaleca się, aby pliki skryptów miały rozszerzenie .sh). Pierwsza linia pozwala wskazać jaki interpreter poleceń ma zostać wykorzystany do jego wykonania – w tym przypadku jest to powłoka bash. Druga linia prezentuje sposób umieszczania komentarzy; każda linia rozpoczynająca się od znaku #, jest traktowana jako komentarz. Kolejne trzy linie wyświetlają wartości pierwszych trzech argumentów wywołania skryptu z wykorzystaniem polecenia echo. Aby uruchomić skrypt należy dla pliku, w którym jest on zapisany nadać prawo wykonywania.

Powłoki systemowe pozwalają na to, aby skrypty zawierały konstrukcje sterujące ich wykonaniem – podstawową instrukcją jest w tym przypadku instrukcja warunkowa. Składnia tej instrukcji jest następująca:

```
if warunek
then
  instrukcje wykonywane, jeśli warunek zostanie spełniony
else
  instrukcje wykonywane, jeśli warunek nie zostanie spełniony
fi
```

Słowa kluczowe instrukcji warunkowej muszą być zapisane w osobnych liniach! Warunek może być dowolnym poleceniem, jednak sprawdzanie warunków najczęściej odbywa się przy użyciu programu test. Polecenie to jest powszechnie wykorzystywane w skryptach z zastosowaniem notacji używającej znaków “[” i “]” do realizacji testów warunków, co prezentuje kolejny przykład:

```
if [ $1 = xyz ]
then
 echo "arg nr 1 = xyz"
else
 echo "arg nr 1 <> xyz"
fi
```

Po znaku “[” i przed znakiem “]” konieczne jest wprowadzenie znaku spacji!

Tabela 1: Wybrane rodzaje testów polecenia test

Warunek	Opis
`znaki1 = znaki2`	Weryfikacja równości dwóch łańcuchów znaków
`znaki1 != znaki2`	Weryfikacja nierówności dwóch łańcuchów znaków
`-z znaki`	Weryfikacja czy łańcuch znaków ma zerową długość
`-n znaki`	Weryfikacja czy łańcuch znaków ma niezerową długość
`liczba1 -eq liczba2`	Weryfikacja równości dwóch liczb
`liczba1 -ne liczba2`	Weryfikacja nierówności dwóch liczb
`liczba1 -qt liczba2`	Weryfikacja, czy liczba1 jest większa od liczba2
`liczba1 -lt liczba2`	Weryfikacja, czy liczba1 jest mniejsza od liczby 2
`-e nazwa`	Weryfikacja, czy podany plik istnieje
`-f nazwa`	Weryfikacja, czy podany plik jest plikiem zwykłym
`-d nazwa`	Weryfikacja, czy podany plik jest katalogiem

`-r nazwa`
`-w nazwa`
`-x nazwa`
	Weryfikacja czy użytkownik ma prawo, odpowiednio, odczytu, zapisu i wykonywania dla pliku o podanej nazwie

`warunek1 -a warunek2`	Iloczyn logiczny warunków
`warunek1 -o warunek2`	Suma logiczna warunków
`! warunek1`	Negacja warunku

Pętla **for** wykonywana jest z góry określoną ilość razy, a jej ogólna składnia jest następująca:

```
for zmienna in lista
do
  instrukcje do wykonania
done
```

Wykonanie pętli powoduje przypisywanie zmiennej *zmienna* kolejnych wartości wymienionych na liście lista; ilość iteracji, jest zatem zależna od długości podanej listy. Jako listę można pętli **for** podawać wzorce uogólniające powłoki. Realizacja pętli numerycznej z zastosowaniem pętli for jest możliwa z użyciem programu `seq`, który wypisuje kolejne liczby, np.: **for** `N` **in** `` `seq 1 10` `` spowoduje dziesięciokrotne wykonanie pętli.

Pętla **while** pozawala na realizację pętli, dla których ilość iteracji nie jest znana z góry, jej składnia dla skryptów powłoki jest następująca:

```
while warunek
do
  instrukcje do wykonania
done
```

Przykładem zastosowania pętli **while** może być skrypt wypisujący na ekranie wartości wszystkich argumentów wywołania skryptu (niezależnie od ich liczby). Pętla taka będzie wykorzystywała polecenie **shift**, które powoduje przesunięcie argumentów – oto skrypt:

```
#!/bin/bash
while [ -n "$1" ]
do
  echo $1
  shift
done
```

Warunkiem wykonania pętli jest sprawdzenie, czy pierwszy argument wywołania skryptu ma niezerową długość – jeśli skrypt został uruchomiony bez żadnych argumentów, to pętla nie zostanie wykonana. Jeśli natomiast skrypt został wykonany z argumentami, to w pierwszym wykonaniu pętli zostanie wyświetlona wartość pierwszego argumentu, a następnie nastąpi przesunięcie argumentów w lewo poleceniem **shift**. Pętla zakończy się, jeśli zostaną wyświetlone i przesunięte wszystkie argumenty. Obie zaprezentowane pętle mogą zostać przerwane poleceniem **break**.

Jeśli skrypt wymaga interakcji z użytkownikiem, to niezbędne staje się pobieranie wartości przekazywanych przez użytkownika. Służy do tego polecenie: **`read`** `argumenty`. Argumentami są nazwy zmiennych środowiskowych, które przyjmą wartość odczytaną ze standardowego wejścia (do napotkania znaku nowej linii). Jeśli jako argumenty podano kilka zmiennych, to są one inicjowane w ten sposób, że pierwsze słowo trafia do pierwszej zmiennej, drugie do drugiej itd.

Skrypty mogą być także wykonywanie w trybie debugowania, np. w celu testowania poprawności działania, warunków, pętli itp. Aby zrealizować wykonanie skryptu z wyświetlaniem informacji kontrolnych należy zastosować przełącznik **`-x`** wywołania interpretatora poleceń. Można to zrealizować na dwa sposoby: po pierwsze można dopisać tenże przełącznik w pierwszej linii skryptu: `#!/bin/bash` **`-x.`**

Po drugie można uruchomić skrypt wywołując go poprzez wskazanie interpretatora z przełącznikiem i nazwą skryptu jako argumentem; przyjmując, że skrypt posiada nazwę *skrypt.sh* uruchomienie miałoby następującą postać: `bash -x skrypt.sh`.