[ilość modułów: 1,5]

Temat zajęć: Tworzenie skryptów powłoki systemu operacyjnego.

Czas realizacji zajęć:	*135 min.*
Zakres materiału, jaki zostanie zrealizowany podczas zajęć:	Zmienne środowiskowe oraz ich eksportowanie, argumenty wywołania skryptów, instrukcja warunkowa, pętle, pobieranie wartości w trakcie wykonywania skryptu, uruchamianie skryptów z debugowaniem.

I. Interpreter poleceń oraz zmienne środowiskowe.

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. Dostępnych jest wiele różnych powłok, z których najpopularniejsze wydają się następujące:

- interpreter Korna – uruchamiany poleceniem **`ksh`**,
- interpreter **`tcsh`** oraz
- najpopularniejszy i najpowszechniej obecnie stosowany **`bash`** (ang. *Bourne Again Shell*).

Wszystkie dalsze przykłady będą prezentowane właśnie dla powłoki **`bash`**.

Zmienne środowiskowe to bardzo wygodny i uniwersalny sposób konfigurowania i parametryzowania powłok systemowych i – za ich pomocą – 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
```

W tym wypadku ciąg znaków **`ZMIENNA`** to nazwa zmiennej, a **`wartosc`** to jej wartość – należy zwrócić uwagę, że pomiędzy nazwą zmiennej, operatorem przypisania i wartością nie może być spacji. Odwołanie się do wartości zmiennej jest możliwe dzięki specjalnemu znakowi **`$`**; np. wyświetlenie wartości zmiennej na ekranie jest możliwe z wykorzystaniem polecenia **`echo`**, które wyświetla linię tekstu oraz wartości zmiennej pobranej znakiem **`$`**:

```
SYSTEM=Unix
echo $SYSTEM
Unix
```

Polecenie systemowe

```
set
```

pozwala wyświetlić wartości wszystkich zmiennych środowiskowych, a polecenie

```
unset
```

usuwa zmienną środowiskową, oto przykład:

```
unset SYSTEM
```

Jak wspomniano środowisko wykonania procesu jest przekazywane do procesów potomnych – jednak nie wszystkie zmienne powłoki muszą być przekazywane do uruchamianych programów. Zmienne, które są przekazywane nazywa się *zmiennymi eksportowanymi*, a zmienne, które nie są przekazywane, nazywa się *zmiennymi lokalnymi*. Z reguły nowo tworzone zmienne są początkowo zmiennymi lokalnymi i niezbędne jest jawne wskazanie, że mają być zmiennymi eksportowanymi. Nazywa się to *eksportowaniem zmiennych* i jest realizowane przez poleceniem:

`export` *lista_zmiennych*

Oto przykład tworzenia zmiennej i jej eksportowania:

```
SYSTEM=Unix
export SYSTEM
```

Powyższe zlecenia można także zrealizować jednym poleceniem:

```
export SYSTEM=Unix
```

Każdy użytkownik może łatwo (np. poleceniem **set**) zweryfikować, że w systemie jest zdefiniowanych wiele zmiennych środowiskowych, oto znaczenie podstawowych z nich:

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

II. Skrypty i ich argumenty.

Skrypty powłoki to pliki tekstowe, które zawierają ciągi poleceń dla powłoki systemowej. Współczesne powłoki pozwalają także aby skryty zawierały konstrukcje programistyczne, takie jak instrukcje warunkowe czy pętle, polecenia pozwalające na interaktywną prace skryptu, czy też przekazywanie argumentów ich wywołania. Skryty są bardzo pomocnym rozwiązaniem kiedy istnieje konieczność wykonywania złożonych poleceń i poleceń, które są powtarzane okresowo.

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: 1,2,3...9. Każda zmienna pozycyjna przechowuje tekst przekazany za pośrednictwem odpowiadającemu jej argumentu wywołania – pierwszy argument dostępny jest w zmiennej pozycyjnej 1 itd. Oto przykład pierwszego 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(1)**. Aby uruchomić skrypt należy dla pliku, w którym jest on zapisany nadać prawo wykonywania. Poniżej przedstawiono przykładowe wywołanie przedstawionego powyżej skryptu oraz jego wynik (przyjęto, że plik skryptu nazywa się **skrypt1.sh**):

```
./skrypt1.sh abc xyz 12345
 argument nr 1: abc
 argument nr 2: xyz
 argument nr 3: 12345
```

III. Instrukcja warunkowa.

Obecnie 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

Należy zaznaczyć, że słowa kluczowe instrukcji warunkowej muszą być zapisane w osobnych liniach, dokładnie tak jak w powyższym przykładzie.

Warunek może być dowolnym poleceniem, jednak sprawdzanie warunków najczęściej odbywa się przy użyciu programu **test(1)**. 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. Możliwe do wykonania testy z zastosowaniem programu **test** prezentuje Tabela 1.

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 **liczba2**
-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

Tabela 1: Wybrane rodzaje testów polecenia test

IV. Pętle.

Skrypty powłoki mogą także zawierać pętle – podstawowe dwie z nich to pętla *for* oraz *while. 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* można podawać wzorce uogólniające powłoki. Poniższy przykład skryptu prezentuje zastosowanie pętli *for* do usunięcia wszystkich plików z rozszerzeniem **.tmp** z katalogu bieżącego:

```
#!/bin/bash
for FILE in *.tmp
do
      rm -v $FILE
done
```

Ilość iteracji powyższej pętli bezie zatem determinowana ilością plików z rozszerzeniem ***.tmp**, które utworzą listę wartości dla zmiennej **FILE**.

Realizacja pętli numerycznej z zastosowaniem pętli for jest możliwa z użyciem programu **seq(1)**, 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
```

Warunek może być dowolnym poleceniem i najczęściej jest konstruowany – tak jak w przypadku instrukcji warunkowej – z zastosowaniem programu **test**.

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(1)**, 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** (drugi argument stanie się pierwszym, trzeci drugim itd.). Pętla zakończy się jeśli zostaną wyświetlone i przesunięte wszystkie argumenty (zmienna pozycyjna **$1** będzie miała wówczas zerową długość).

Obie zaprezentowane pętle mogą zostać przerwane poleceniem **break(1)** – oto przykład zastosowania przerywania pętli:

```
#!/bin/bash
for FILE in *.tmp
do
      if [ ! -f $FILE ]
      then
            echo "$FILE nie jest plikiem!"
            break
      fi
      rm -v $FILE
done
```

Jak widać pętla zostanie przerwana, jeśli pobrana nazwa z rozszerzeniem ***.tmp** nie będzie wskazywała na plik zwykły.

V. Pobieranie wartości do skryptów oraz ich debugowanie.

Jeśli skrypt wymaga iteracji 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 odczytana 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. Polecenie to można przetestować wykonując następujące polecenia:

```
read X Y
uzytkownik adam
echo $X
uzytkownik
echo $Y
adam
```

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 drugi 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
```

VI. Zadania do samodzielnego wykonania.

1) Zdefiniuj zmienną IMIE i przypisz jej swoje imię. Wyświetl zawartość tej zmiennej. Wyeksportuj tą zmienną i sprawdź, czy jest dostępna w nowym (potomnym) interpreterze.
2) Wyświetl listę zmiennych eksportowanych.

3) Zmień własny znak zachęty, modyfikując zmienną PS1.
4) Napisz skrypt, który dla każdego z plików podanych jako argumenty wywołania posortuje jego zawartość.
5) Napisz skrypt, który dla każdego z plików podanych jako argumenty wywołania wyświetli w kolejnych liniach 3 najczęściej powtarzające się w nim słowa.
6) Napisz skrypt, który będzie kopiował plik podany jako pierwszy argument do wszystkich katalogów podanych jako kolejne argumenty wywołania.

VII.Literatura.

[Bac95] Bach M. J., *Budowa Systemu Operacyjnego UNIX*, WNT, 1995, ISBN 83-204-2015-6.

[Sob01] Sobaniec C., *Linux – Przewodnik Użytkownika.*, Wydawnictwo NAKOM, 2001, ISBN 83-86969-53-9.