Ćwiczenie 5

Skrypty powłoki – programowanie w powłoce *bash*

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie możliwości powłoki *bash* od strony programistycznej. System UNIX (Linux) pozwala tworzyć skrypty, czyli wykonywalne pliki tekstowe, zawierające polecenia powłoki oraz inne instrukcje sterujące (wzorem innych języków programowania).

2. Przygotowanie do ćwiczenia

- zapoznać się z treścią pliku *systemy-05-bash.pdf*,
- definicja powłoki systemu operacyjnego Linux (UNIX),
- typy powłok i ich właściwości,
- przetwarzanie potokowe.

3. Skrypty powłoki

Powłoka przyjmuje i interpretuje naciskane klawisze jako polecenia. W systemach UNIX dostępnych jest kilka powłok, m. in. *Bourne*, *Korn*, *C*. Powłoki te mają wiele cech wspólnych, na przykład możliwość wykonywania skryptów (działających podobnie jak pliki BAT w DOS-ie).

Dla poleceń będących zewnętrznymi programami powłoka tworzy procesy potomne, w których są one uruchamiane. Po utworzeniu taki proces potomny dziedziczy zmienne środowiska i bieżący katalog roboczy powłoki. Skrypty, czyli pliki tekstowe zawierające polecenia powłoki nie muszą mieć określonej nazwy i rozszerzenia, jednak muszą mieć nadane prawa do wykonywania.

- skrypt powłoki to plik tekstowy, rozpoczynający się sekwencją:

  ```
  #!/bin/bash
  # opis/komentarz
  polecenia
  ```

  pierwsza linia określa powłokę, w której wykonywany jest skrypt; druga to komentarz

- **polecenia:** sekwencja napisów (komend), oddzielonych białymi znakami,
- **argumenty skryptu:** pierwszy napis (**`$0`**) to nazwa polecenia lub skryptu powłoki,
- pozostałe parametry (**`$1`**...**`$n`**) przekazywane jako argumenty do polecenia,
- aby skrypt mógł być wykonany musi mieć nadane prawa do wykonywania dla użytkownika.

2. Zmienne i podstawienie w skryptach

- przypisywanie wartości: **`imie = ala`**,
- obliczanie wartości arytmetycznych: **`echo $[1+1]`** (polecenie **`echo`** wypisuje argument na standardowym wyjściu: **`echo $imie`**),

- polecenie **`katalog =‘ls‘`** przypisuje zmiennej **`katalog`** wynik działania polecenia **`ls`**,
- polecenie **`shift`** zmienia kolejność parametrów przekazywanych do skryptu: **`$n = $(n+1)`**.

zmienne powłoki	
`$HOME`	katalog domowy użytkownika
`$PATH`	lista (:) kartotek do przeszukiwania
`$USER`	identyfikator użytkownika
`$TERM`	typ terminala (**`vt100`**)
`$PS1`	znak zachęty pierwszego poziomu
`$PS2`	znak zachęty drugiego poziomu
`$SECONDS`	liczba sekund działania powłoki
`$HISTFILE`	nazwa pliku z historią
`$RANDOM`	liczba pseudolosowa (zawsze inna)

3. Instrukcje sterujące

Pętla `for`:

Pozwala powtarzać pewne czynności dla kolejnych wyrazów z listy:

```
for zm in lista
do
  <polecenia>
done
```

przykład:

```
for plik in *.txt
do
  cp $plik $plik.bak
done
```

Pętla `while`:

```
while wyrazenie
do
  <polecenia>
done
```

przykład:

```
n=1
while [ $1 ]
do
  echo $n’$’ -- $1
  n = $[ n + 1 ]; shift
done
```

Instrukcja warunkowa `if`:

```
if wyrazenie
```

```
then
<polecenia>
else
<polecenia>
fi
```

przykład:

```
if [ -f .profile ]; then echo "Jest!"; fi
```

Polecenie `test`:
Używane z poleceniami sterującymi powłoki:

```
if [ "$1" == "hej" ]
then
echo Pierwszy parametr to \"hej\"
fi
```

`t1 = t2`	równość tekstów	
`t1 != t2`	różność tekstów	
`t1`	prawdziwy, gdy **`t1`** jest zdefiniowana	*operatory tekstowe*
`-n t1`	prawdziwy, gdy tekst jest niepusty	
`-z t1`	prawdziwy, gdy tekst jest pusty	
`-d plik`	prawdziwy, gdy **`plik`** jest kartoteką	
`-f plik`	prawdziwy, gdy **`plik`** istnieje i jest regularny	
`-r plik`	prawdziwy, gdy **`plik`** może być czytany	*operatory plikowe*
`-w plik`	prawdziwy, gdy do **`pliku`** można pisać	
`-s plik`	prawdziwy, gdy **`plik`** ma długość dodatnią	
`-x plik`	prawdziwy, gdy **`plik`** jest wykonywalny	
`!w`	wyrażenie **`w`** jest fałszywe	
`w1 -a w2`	oba wyrażenia są prawdziwe	*operatory logiczne*
`w1 -o w2`	przynajmniej jedno jest prawdziwe	

Instrukcja `case`:

```
case zmienna in
wzorzec [ | wzorzec] ... ) instrukcje ;;
wzorzec [ | wzorzec] ... ) instrukcje ;;
...
esac
```

przykład:

```
case "$poraDnia" in
"tak" | "t" | "Tak" | "TAK")
echo Dzien dobry!
;;
[nN])
echo "Dobry wieczor"
;;
```

```
*)
echo "Odpowiedz tak lub nie"
;;
esac
```

4. AWK – język do przetwarzania plików tekstowych

- wyświetlanie plików, wierszy, pól (**cut**),
- analizowanie tekstów ze względu na występowanie określonych ciągów znaków (**grep**, **egrep**),
- przygotowywanie raportów w oparciu o dane z pliku,
- filtrowanie tekstów,
- operacje arkusza kalkulacyjnego.
- podstawowymi parametrami **awk** są pliki do przetworzenia,
- pliki podzielone są na wiersze (rekordy), wiersze zaś na pola,
- domyślnym separatorem jest ciąg białych znaków,
- separatorem może być dowolny znak,
- w takim formacie przechowywana jest większość unixowych plików konfiguracyjnych,
- powyższy format mają również bardzo często dane wyjściowe poleceń (np. **ls**).

Zmienna **$1** to pierwsze pole, **$2** drugie, itd. Bieżąca linia to **$0**.
przykład: wybranie z pliku dwóch kolumn i zamiana miejscami:

```
awk '{print $3,$2 }'
```

inna możliwość:

```
awk -f program.awk dane_we1 dane_we2 > dane_wy
```

Zamiast wzorca może pojawić się warunek:

```
awk '$4 > 100 {print $1*$4}'
```

Inne warunki arytmetyczne: **==, !=, >, <, >=, <=**

Program obliczający sumę wielkości plików w kartotece:

```
ls -l | awk '/^-/{licznik = licznik+$5} END {print licznik}'
```

Program wyświetlający te linie, które są liczbami:

```
/^[0-9]+(\.[0-9]+)?$/ { print $0 }
```

zmienne wewnętrzne	
NR	liczba przeczytanych rekordów
FNR	to samo, ale w bieżącym pliku
FILENAME	nazwa bieżącego pliku
FS	separator pól
NF	liczba pól w bieżącym rekordzie
ARGC	liczba argumentów linii poleceń
ARGV	tablica argumentów linii poleceń

5. Zadania do samodzielnego wykonania

1. Napisz skrypt **con_files**, który uruchamia się z 4 parametrami – 3 pierwsze to nazwy istniejących plików tekstowych, np. **jeden.txt**, **dwa.txt** i **trzy.txt**, a ostatni – **wynik.txt**. Jeśli 3 pierwsze pliki istnieją – skrypt łączy je w jeden plik tekstowy **wynik.txt**.
2. Napisz skrypt **show**, który wyświetli wszystkie przekazane do niego argumenty, liczbę przekazanych argumentów, itp. (**$0 $1 $@ $***); następnie użyj komendy **shift** i wyświetl ponownie.
3. Napisz skrypt **interactive**, który wyświetla poniższe menu:

```
[1]  Wyswietl biezaca date"
[2]  Dopisz aktualna date do pliku jeden.txt"
[3]  Wylistuj biezacy katalog"
[q]  Koniec"
```

   Skrypt oczekuje na podanie odpowiedniego znaku (polecenie **read**) i wykonuje wskazane działanie przypisane danemu znakowi (polecenie **case**). W przypadku podania innego znaku, bądź sekwencji skrypt powinien zgłosić błąd i zakończyć działanie.
4. Zmodyfikuj powyższy skrypt, aby kończył swoje działanie **tylko** po naciśnięciu **q** (po wykonaniu bieżącego zadania powinien oczekiwać na następne polecenie z menu (pętla **while...do**).
5. Napisz skrypt **show_proc**, który jako argument przyjmuje numer procesu i wyświetla informacje o procesie z danym numerem.
6. Napisz skrypt **lss**, który działa podobnie jak **ls** i wyświetla informacje o plikach w trzech kolumnach w kolejności: nazwa pliku, rozmiar pliku i prawa (**awk**).
7. Zmodyfikuj powyższy przykład, aby dane w kolumnach: *nazwa pliku* i *prawa* były wyrównane do lewej, a dane w kolumnie *rozmiar pliku* – do prawej (**awk**, **printf**).

6. Literatura

[1] Z. Królikowski, M. Sajkowski: "*System operacyjny UNIX dla początkujących i zaawansowanych*", Wydawnictwo NAKOM, Poznań 1995,

[2] Arnold Robbins, Nelson H. F. Beebe: "*Programowanie skryptów powłoki*", Wydawnictwo HELION, Gliwice 2005,

[3] http://www.freeos.com/guides/lsst/ch01sec07.html.