Skrypty powłoki 1

Skrypt - program napisany w języku skryptowym. Skrypty stosujemy najczęściej w celu automatyzacji pracy. Ideą języków skryptowych jest pośredniczenie pomiędzy użytkownikiem i wykonywanie jego “wysokpoziomowych” poleceń, co wiąże się ze zmniejszeniem nakładu pracy i zwiększeniem czytelności kodu.

W powłoce systemu korzystać będziemy z jej interpretowalnego języka.

System	Shell
Debian / Ubuntu	dash
RedHat / CentOS	bash
Alpine / OpenWrt	busybox ash
macOS	bash 3.2.57
Android	mksh
FreeBSD	FreeBSD sh（ash 系）
OpenBSD	OpenBSD ksh（pdksh 系）
Solaris 11	ksh93

Rys 1. Języki powłoki a dystrybucje systemów
Źródło: betterprogramming

Celem ćwiczenia jest zapoznanie się z wybranymi funkcjami systemowymi i podstawami programowania w języku powłoki systemu operacyjnego.

Przydatne skróty klawiszowe - przypomnienie:

Ctrl + l wyczyść terminal (lub komenda clear)
Ctrl + r szukaj polecenia w historii
Ctrl + c przerwij działanie programu
Ctrl + z zawieś działanie programu
Ctrl + u wycina wszystko od kursora w lewo
Ctrl + k wycina wszystko od kursora w prawo
Ctrl + w wycina ostatnie słowo
Ctrl + y przywraca wyciętą zawartość
alt + b przemieszcza kursor o jedno słowo wstecz
alt + f przemieszcza kursor o jedno słowo naprzód
home przemieszcza kursor na początek obecnej komendy/zestawu komend
end analogicznie, tyle, że na koniec

fg przywróć działanie programu w pierwszym planie
bg przywraca działanie programu (procesu) w tle
jobs wyświetla wstrzymane procesy
strzałki góra/dół poruszanie się po historii komend

tab uzupełnianie tekstu (komend, nazw…)
cd - cofa nas do poprzedniego katalogu (nie w hierarchii, ale w historii)

Dobrą praktyką jest umieszczenie łańcucha tekstu w znakach podwójnego cudzysłowia- "" - wtedy jest rozumiany dosłownie zważając na pewne reguły:
- \ - wprowadza znak zarezerwowany, np. \$ - znak $.
- $ - wprowadza do łańcucha zawartość zmiennej, stosowane jak zmienna
- znaki `` (odwrócony apostrof) tekst, który się tam znajduje traktowany jest jak polecenie, które jest wywoływane a jego wynik podstawiany w to miejsce.
Jeżeli nie chcemy, aby się to działo, używamy zamiast znaków " znaków ' - apostrofu.

Zmienne stosowane w skrypcie można podzielić na środowiskowe i lokalne dla danej powłoki. Czas życia takiej zmiennej lokalnej wynosi tyle, ile "żyje" powłoka. Jednak jej zasięg ograniczony jest tylko do tej konkretnej powłoki. **W tym kontekście ważne jest rozróżnianie działania source / kropki (.) oraz kropki ze slashem (./). Czas życia zmiennej środowiskowej zależy od czasu życia instancji powłoki, ale jej zasięg jest znacznie szerszy i obejmuje również powłoki potomne.**

Do wyświetlania zmiennych środowiskowych wykorzystujemy komendę **printenv**. Chcąc ustawić zmienną środowiskową należy poprzedzić jej deklarację słowem **export**, zaś czyszcząc ją należy wywołać **unset** i podać jako parametr nazwę zmiennej.

Zmienne zawsze obecne w skrypcie:
$0 – nazwa skryptu
$1, $2 … - kolejne parametry przekazane do skryptu.
$# - liczba parametrów przekazanych do skryptu
$* - wszystkie parametry jako jeden ciąg
$? - status ostatniego polecenia

Znak $:
przy ustawianiu zmiennych nie używamy spacji (nie robimy export x = 1). Powłoka próbowałaby wtedy uruchomić "x" jako polecenie z argumentami.
Chcąc odczytać wartość zmiennej używamy znaku $: np echo $x lub echo ${x}

Skrypty w formie plików tekstowych można wykonywać przy użyciu Shebangu - po #!, należy wskazać ścieżkę do interpretera. Skrypt, który chcemy wykonać musi mieć nadane stosowne uprawnienia.

Arytmetyka:
Do realizacji działań w powłoce uniksowej służy program expr. Kolejne jego argumenty to elementy działania. Zwraca on wynik działania. Parametry programu expr należy oddzielać spacjami. Przykład:

```
echo `expr 2 + 2`
```

Podstawowe operatory: +, -, *, /, %

Do arytmetyki zmiennoprzecinkowej można użyć rozszerzonego parsera równań bc. Program ten przyjmuje jednak dane z pliku, a nie jako parametry. Można to jednak obejść za pomocą przekierowań :) Program domyślnie wyświetla 0 miejsc po przecinku, stąd opcja -l.

```
echo ”2 / 4” | bc -l
```

Jeżeli korzystamy z Basha, to możemy użyć:

```
a=2
let ”a = $2 + 2”
echo $a
```

albo:

```
a=2
a=$(($a+2))
echo $a
```

Pętla for:

```
for zmienna in lista;
do
    ...
done
```

Albo:

```
for zmienna in lista
do
    ...
done
```

Listą może być np. wybór plików za pomocą wieloznaczników, zestaw ciągów znaków oddzielonych spacjami, zestaw parametrów programu, czy zestaw znaków ({1,2,3}). Spacja ma tu wyższy priorytet niż przecinek. Istnieje także program **seq**, który służy do generowania sekwencji. Przykład:

```
for i in `seq 1 3`
```

W miejsce tego wyrażenia podstawione zostaną 1 2 i 3

Zadanie1:
Napisz skrypt, którego efektem działania powinna być następująca informacja:

Hello from <nazwa skryptu>, today is <data i czas> you are in <ścieżka bezwzględna>. If you want to go to your home directory please type <odpowiednia komenda>.

Zadanie2:
Utwórz zmienną *var* i przypisz jej wartość 10. Następnie napisz skrypt, który pokaże linię “Your variable is set to <wartosc_zmiennej>”. Skrypt uruchom przy użyciu ./ i . (source). Następnie powtórz te czynności definiując *var* jako zmienną środowiskowej.

Zadanie3:
Napisz skrypt przyjmujący w parametrach ciąg liczb całkowitych, a wyprowadzający na ekran ich średnią arytmetyczną. Jako parametry podstaw kolejne cyfry swojego numeru indeksu, ale zachowaj generyczność programu tak, aby działał on dla zmiennej liczby argumentów.