Skrypty powłoki

- Często wykonujemy powtarzający się zestaw poleceń
- Wygodniej jest zapisać je w pliku i poddać interpretacji przez powłokę systemu, niż wpisywać je za każdym razem z klawiatury
- Pliki tekstowe, napisane z pewnymi regułami składni, nazywają się skryptami powłoki (shell scripts)

Tworzenie skryptu:

- W dowolnym edytorze tworzymy plik tekstowy z treścią skryptu, np. **pico mojskrypt**
- Skrypt możemy urochomić albo przez podanie polecenia: **bash mojskrypt**
- Albo nadajemy mu atrybut wykonywalności (**chmod +x mojskrypt**) i wykonujemy jak program: **./mojskrypt**
- Dokonujemy edycji skryptu jak każdego innego pliku tekstowego

Wybór powłoki

- Skrypty najczęściej pisze się w składni Bourne shell'a (sh lub bash)
- W skrypcie można zadeklarować, która powłoka ma go interpretować: pierwsza linia skryptu musi mieć postać: **#!/bin/sh** dla Bourne Shella lub **#!/bin/bash** dla Bourne-Again-Shella

„Debugging” skryptów

- Z linii komend: **bash –opcja skrypt**
- Przy pomocy instrukcji (w tekście skryptu):

  set –opcja

gdzie opcja:

- –n - polecenia skryptu nie są wykonywane, a jedynie wyświetlane
- -v -wyświetlenie i wykonanie linii
- -x –wyświetlenie poleceń po interpretacji (wykonanie podstawień itd.)
- -e -zatrzymanie interpretera, jeśli którekolwiek polecenie zwróci błąd
- -u – zwróci błąd przy użyciu niepodstawionej zmiennej

set +opcja wyłącza daną opcję

Argumenty skryptu

bash mojskrypt ala ma kota

$1	=ala
$2	=ma
$3	=kota
$0	=mojskrypt (nazwa skryptu)
$#	=3 (liczba argumentów)

shift przesuwa parametry w lewo (pierwszy tracony)

$1	=ma
$2	=kota
$3	=-

- $* - wszystkie argumenty jako jeden ciąg znaków
- $$ - numer aktualnego procesu, czyli skryptu
- $? – status wyjścia ostatnio wykonywanej komendy

Zmienne

- Lokalne – widziane tylko w tym skrypcie

np. **a=5**

- Globalne – widziane też przez procesy potomne

np. **g=4**

export g

Odwołanie do zmiennej:

$a, $g (lepiej dać **${a}, ${g}** – jednoznaczna interpretacja)

gl=10 ; ./nprogram ..... – zmienna dziedziczona tylko przez program **nprogram**

Niektóre zmienne globalne

- PATH – ścieżka dostępu
- TERM – typ terminala
- MAIL – ścieżka do mailboxa
- PAGER – "stronicowacz", np. more
- EDITOR – defaultowo wywoływany edytor

Wyświetlanie zmiennej na ekranie:

echo $zmienna lub **echo ${zmienna}**

Wczytanie zmiennej:

read zmienna

Parametry funkcji **echo**

- **-n** nie jest wysyłany znak nowej linii (wew. echo w bash)
- **-e** jest potrzebne w LINUXie; w Solarisie /usr/bin/echo bez tego włącza interpretację znaków specjalnych takich jak:
  - **\a** czyli alert, usłyszysz dzwonek
  - **\b** backspace
  - **\c** pomija znak kończący nowej linii
  - **\f** form feed czyli wysuw strony
  - **\n** znak nowej linii
  - **\r** powrót karetki
  - **\t** tabulacja pozioma
  - **\v** tabulacja pionowa
  - **** backslash
  - **\0nnn** znak, którego kod ASCII ma wartość ósemkowo
  - **\0xnnn** znak, którego kod ASCII ma wartość szesnastkowo

Znaki specjalne

- **\z** – wzięcie znaku **z** dosłownie
- **#** - odtąd do końca linii komentarz
- ***** - dopasowuje dowolny ciąg w nazwie pliku
- **?** – dopasowuje 1 dow. znak w nazwie pliku
- **[a-z, abc]** – 1 znak spośród podanych
- **;** - sekwencyjne wykonanie poleceń
- **&** - polecenie działa w tle, startuje następne
- | - wyjście jednego jako wejście drugiego
- **< >** - przekierowanie danych
- ||, **&&** - sekwencyjne, warunkowe

Znaki specjalne, cd

- **(ttt)** – wykonanie ttt w wewnętrznym (nowym) interpreterze poleceń
- **`ttt`** - wynik działania ttt jest zmienną, np. a=`ls -al` (` - znak pod "~")
- **'ttt'** – ttt jest brane dosłownie (' apostrof)
- **"ttt"** – ttt wzięte dosłownie, ale po interpretacji **$**, **`...`** i ****

Instrukcje warunku

- **test** wyrażenie ⇔ [wyrażenie]

zwraca **0** (prawda) lub **≠0** (fałsz)

Operatory:

! – not

-a – and

-o - or

Instrukcja if

- if test .....
- if [.....]
- if inne_wyrażenie

if warunek
then
instrukcje
fi

if –cd.

if warunek
then
wyrażenie1
else
wyrażenie2
fi

if warunek
then
wyrażenie1
elif warunek
wyrażenie2
else
wyrażenie3
fi

Testowanie wyrażeń numerycznych (liczb)

- a –eq b (tzn. a = b)
- a –gt b (tzn. a > b)
- a –ge b (tzn. a ≥ b)
- a –lt b (tzn. a < b)
- a -le b (tzn. a ≤ b)
- a -ne b (tzn. a ≠ b)

Testowanie napisów

- napis (prawda, gdy tekst **napis** niezerowy)
- -z napis (prawda, gdy zerowy)
- -n napis (prawda, gdy dłuższy od zera)
- nap1 = nap2 (prawda, gdy identyczne)
- nap1 != nap2 (prawda, gdy nie identyczne)

Testowanie plików

- **-s** plik – istnieje i jest niezerowy
- **-f** plik – istnieje i jest zwykłym plikiem
- **-d** plik – istnieje i jest katalogiem
- **-r** plik – istnieje i mamy prawo czytania
- **-w** plik – istnieje i mamy prawo zapisu
- **-x** plik – istnieje i mamy prawo wykonania
- p1 **–nt** p2 – p1 nowszy niż p2
- p1 **–ot** p2 – p1 starszy niż p2

Zapis skrócony (|| na nie, && na tak)

a=3; b=4
test $a –eq $b || **echo** 'a nie jest rowne b'
◊
if [$a –eq $b] then
echo ' '
else
echo 'a nie jest rowne b'
fi

[$a –ne $b] && echo 'a nie jest rowne b'
◊
if [$a –ne $b] then
echo 'a nie jest rowne b'
fi

Instrukcja **case**

case zmienna **in**
wzorzec1)
instrukcje1
;;
wzorzec2)
instrukcje2
;;
.....
*)
instrukcje (dla spoza zakresu)
;;
esac

Pętla **for**

for zmienna **in** wart1 wart2 ...wartn

do

 #po podstawieniu zmienna = wart1 ...wartn

instrukcje

done

Przykłady

- z folderów **dane1**, **dane2** i **wyniki** skasować pliki z rozszerzeniem **bak**

```
for i in dane1 dane2 wyniki
do
rm $i/*.bak
done
```

- zmienić prawa dostępu do wszystkich plików w folderze

```
for i in `ls`
chmod go-rwx $i
done
```

Inne pętle

while warunek
do
instrukcje #wykonywane gdy warunek true
done

until warunek
do
instrukcje #wykonywane dopóki warunek false
done

Przerwanie pętli

break –przerywa działanie i wychodzi z pętli

continue – przerywa bieżący cykl i przechodzi na początek pętli

---

trap polecenie nr

wykonuje polecenie po otrzymaniu sygnału nr

(kill –nr PID)

Obliczenia arytmetyczne - **expr**

Uwaga!

```
wynik=3
wynik1=$wynik+1
echo $wynik1

wynik2 =`expr $wynik + 1`
echo $wynik2
```

```
expr 2 + 3                  5
expr 2 - 3                  -1
expr 10  /  3               3
expr 3  \*  8               24
expr 3  + 2  \* 8           19
ale:
expr 2+3                    2+3
```

Przykładowe zadania

1. Napisać skrypt usuwający wszystkie pliki ***.o** z folderów pk1, pk2 ...pk5. Przed usunięciem nazwa pliku powinna zostać wypisana na ekranie terminala i powinno pojawić się pytanie o potwierdzenie skasowania (t/n)
2. Napisać skrypt przenoszący wszystkie pliki o rozszerzeniu podanym przez **read** do podanego jako **argument1** folderu. Jeśli podany folder nie istnieje należy go najpierw stworzyć.
3. Napisać skrypt, który we wszystkich podfolderach wyszukuje plików o nazwach podanych przez argumenty skryptu. Po znalezieniu należy wypisać komunikat: Plik o nazwie ... znajduje się w folderze ....

Uwaga! Akcję powtarzać dla wszystkich argumentów

4. Napisać skrypt zbierający informacje o każdym podfolderze znajdującym się w folderze podanym jako argument, obejmujące: nazwę podfolderu i liczbę znajdujących się w nim plików (ls ... |wc –l).
5. Jeśli podana jako argument nazwa jest nazwą folderu, to wylistować zawartość folderu, podając typy plików (file), w przeciwnym wypadku podać pełną informację o pliku (ls –l) i typ pliku (instrukcja file)
6. Napisać skrypt sprawdzający, czy istnieje podany jako argument plik i wypisujący odpowiedni komunikat na ekranie. Jeśli plik istnieje powinien pojawić się napis: plik o podanej nazwie istnieje, w przeciwnym razie plik taki powinien zostać utworzony tak, by jego pierwszą linię stanowił napis: Plik utworzono: i bieżąca data/czas.

7. Skrypt działa z argumentem1 start/stop. Użyj instrukcji ***case***
i w przypadku start: uruchom program wczytany jako argument2
i w pliku log zapisz komunikat: program nnn uruchomiono: ***date***
W przypadku stop: zatrzymaj w/w program sygnałem 15 i zapisz w logu:
Program zatrzymano: date. Uwzględnij za mało lub złe argumenty.

8. Znajdź w których folderach podanych w ścieżce dostępu występuje program o nazwie podanej przez argument1. Odpowiednim komunikatem
Podaj: plik ... (nie) występuje w folderze ... Przydatna funkcja: ***tr***

9. Jako cel skryptu utwórz skrypt (listę poleceń) do skasowania ze wszystkich podfolderów bieżącego plików *.tmp starszych niż plik c.time.

10. Poczynając od swojego folderu domowego wyszukaj najstarszego pliku w drzewie. Użyj komendy ***find*** i nie tylko.