2. BASH: Podstawowy skryptowania

Wstęp

Bash jest to najprawdopodobniej najpopularniejsza powłoka systemowa oraz język skryptowy dla systemów z rodziny Linux. Pozwala na bardzo zaawansowane korzystanie z komputera w trybie tekstowym.

Skrypty

Skrypt powłoki jest to plik, zawierający kolejne komendy do zinterpretowania przez powłokę systemową. Są to pliki tekstowe. W Windows mamy najczęściej pliki .bat (typowo powłokowe) oraz pliki .vbs (VisualBasic Script), pod Linuksem i w większości systemów Unix-owych jest .sh (od shell), chociaż akurat rozszerzenie jest kwestią umowną.

Skupimy się na powłoce Linuksa, a konkretnie na bash-u.

Skrypty w systemie Linux powinny mieć atrybut "wykonywalny". Jeśli w pliku nie jest określone, z jakiego interpretera należy skorzystać, wtedy skrypt uruchamia się w domyślnej powłoce systemu.

Skrypty w systemie Linux mogą zawierać informację o tym, jaki jest program interpretujący dany skrypt, robi się to tak, że na początku (pierwsza linia) umieszcza się znak hash, wykrzyknik, a następnie nazwę programu interpretera. U nas będzie to:

```
#!/bin/bash
```

Co oznacza, że program interpretujący ten skrypt to będzie właśnie bash.

Znak hash (#) oznacza komentarz, tak więc można zrobić mały skrypt (proponuję stworzyć taki) o takiej treści:

```
#!/bin/bash
# Skrypt wykorzystuje zaawansowane funkcje
```

```
# systemu do tego, aby narysować króliczka:
echo -e "    \e[m(\\_/)"
echo -e "   (=\e[33m'\e[31m.\e[33m'\e[0m=)"
```

Ćwiczenie 1

Napisz skrypt wyświetlający na terminalu tekst "Witamy w terminalu". Skorzystaj z komendy `echo` oraz informacji podanych powyżej. Pamiętaj o atrybutach i o tym jak się wywołuje programy.

Zmienne powłoki

Powłoka systemowa udostępnia metodę na przechowywanie par "nazwa:wartość" za pomocą tak zwanych zmiennych środowiskowych i zmiennych programowych (lub lokalnych).

Stworzenie nowej zmiennej programowej (będę nazywał to po prostu zmienną, albo zmienną lokalną) robi się tak:

```
ZMIENNA=wartość
```

Działa to na poziomie wiersza poleceń i w skryptach.

- Wartość może być podana po prostu, ale wtedy musi to być jedno słowo
- albo w cudzysłowach ("), wtedy może być to kilka słów i znaków, natomiast znaki specjalne są interpretowane, a zmienne tłumaczone na wartości
- albo w apostrofach (') , wtedy można przekazać dowolny ciąg znaków nie zawierający apostrofów

Zmienne lokalne (po prostu zmienne) są widoczne tylko w danej instancji powłoki (aktualnie wykonywany skrypt, lub aktualna powłoka), natomiast zmienne środowiskowe są dostępne także w powłokach i programach wykonywanych w danej powłoce (takie dziedziczenie zmiennych). Aby zmienna była widoczna w podpowłokach wystarczy wydać komendę export, na przykład:

```
export ZMIENNA=wartość
```

albo

```
ZMIENNA=wartość
export ZMIENNA
```

Odeksportowanie zmiennej środowiskowej robi się tak:

```
export -n ZMIENNA
```

a calkowite usuniecie tak:

```
unset ZMIENNA
```

Aby odczytać zawartość zmiennej można się do niej odwołać w taki sposób:

```
$ZMIENNA
${ZMIENNA}
```

Ta druga składnia jest konieczna, kiedy zmienna ma w nazwie cyfry, albo jest to tablica. Ewentualnie kiedy chcemy wykonać konkatenację (o tym za chwilę).

Ćwiczenie 2

Zobacz taką serię poleceń:

```
X=tekst
echo $X
bash
echo $X
```

Co się stało? Czy potrafisz wyjaśnić?

Zobacz:

```
X=Tekst dłuższy
echo $X
```

Co jest nie tak? jak to naprawić? Zobacz czy pomoże zastosowanie cudzysłowów i apostrofów.

Zobacz:

```
X="Tekst 1"
Y="tekst:$X"
echo "$Y"
Y='tekst:$X'
echo "$Y"
Y=tekst:$X
echo "$Y"
```

Jaka jest różnica między " a '?

```
A=Ala
echo $A ma kota, a kot ma ${A}ę
```

Tak, tu coś będzie nie tak, zaraz to naprawimy.

Zmienne '

Domyślnie istnieje kilka predefiniowanych zmiennych środowiskowych, które przechowują pewne użyteczne informacje. Przetestujemy sobie je jako ćwicznie. Ponad to są zmienne które są związane ze sposobem uruchomienia skryptu:

- `$*`, `$@` - wszystkie argumenty skryptu. Ale obie zmienne różnią się trochę. Kiedy?
- `$0` - przeważnie mówi się, że jest to nazwa skryptu. Nie. To ścieżka i sposób uruchomienia skryptu. Ale zawsze znajduje się w tej zmiennej nazwa. To fakt.
- `$1`, `$2`, ... - kolejne argumenty skryptu
- `$#` - ilość argumentów przekazanych do skryptu
- `$?` - kod zakończenia ostatniego polecenia
- `$$` - PID procesu bieżącej powłoki
- `$!` - PID ostatnio uruchomionego procesu w tle

Ćwiczenie 3

Zobacz co przechowują zmienne i **opisz** jak rozumiesz uzyskany wynik. Do czego on jest i kiedy może się wg Ciebie przydać:

```
PATH
RANDOM
PWD
PS1
USER
HOSTNAME
OSTYPE
```

Zmienne ''

Pod wartość zmiennej można przypisać standardowe wyjście jakiejś komendy. Robi się to na dwa sposoby. Bezpieczniejszy, ale dłuższy, oraz krótszy, ale w niektórych sytuacjach nie zadziała:

```
ZM=$(ls -la)
ZM=`ls -la`
```

Pierwszy sposób jest bezpieczniejszy - jest starszy i zawsze działa, ale wymaga 3 różnych znaków, natomiast drugi jest bardziej przejrzysty. Oba w tym przykładzie dadzą taki sam efekt.

Ćwiczenie 4

Wykonaj komendę `ls -l` w podpowłoce i przypisz ją do zmiennej `X`.

Wyświetl zawartość tej zmiennej.

Spraw, aby nie było różnicy (co najwyżej kolorki) między zwykłym wykonaniem komendy:

```
ls -l
```

a wypisaniem zawartości zmiennej `X`. Nie będzie dobrego rezultatu gdy zrobimy tak:

```
echo $X
```

Zastanów się dlaczego.

Operacje na ciągach znaków

Podczas odwoływania się do zmiennych, możemy wykonywać proste operacje związane z ciągami znaków, oraz można wykonywać operacje arytmetyczne. Zobacz:

```
echo $((2+2))
X=5
echo $((2*X))
```

Operacje na ciągach znaków (zobacz też http://www.thegeekstuff.com/2010/07/bash-string-manipulation/):

```
${#TEKST} - długość ciągu znaków
${TEKST:p} - podciąg rozpoczynający się od znaku o indeksie p
${TEKST:p:l} - podciąg rozpoczynający się od znaku o indeksie p, a
długości l
${TEKST/w/z} - zamienia tekst pasujący do "w" na tekst "z"
```

Ćwiczenie 5a

Pamiętasz jedno z zadań poprzednich?

```
A=Ala
echo $A ma kota, a kot ma ${A}ę
```

Przerób je tak, aby tekst wypisywał się poprawnie.

Napisz skrypt, który będzie wypisywał taki tekst także dla imion męskich. Pomijamy zdrobnienia i imiona które się dziwnie odmieniają (na przykład Marek). Niech skrypt ten przyjmuje jeden argument - imię. Przykładowe wywołanie:

```
$ ./pszetżkole Ala
Ala ma kota, a kot ma Alę
$ ./pszetżkole Justyna
Justyna ma kota, a kot ma Justynę
$ ./pszetżkole Stefan
Stefan ma kota, a kot ma Stefana
```

Czy jest to możliwe w czystym bash-u?

Ćwiczenie 5b

Napisz skrypt, który wczyta jako argument jakiś tekst. Skrypt ten wypisze kolejno:

1. Pierwszy znak z argumentu
2. Ostatni znak z argumentu
3. Zamieni w argumencie każdy tekst SC ⓘ ı tekst poniżej (zastosuj echo z przełącznikiem -e):

   ```
   \e[32mSOP\e[0m
   ```

Przykładowy tekst testowy:

SOP! SOP w imieniu prawa! SOP w imieniu prawa!

Wynikowo ma to wyglądać tak:

SOP! SOP w imieniu prawa! SOP w imieniu prawa!

Wyrażenia regularne (regex/regexp - regular expressions)

Bash jako taki obsługuje w niektórych aspektach wyrażenia regularne, ale niestety nie jest to tak wygodne, jakby się czasami marzyło. Na szczęście jest program sed.

Program sed jest określany przez twórców jako edytor tekstowy bez interfejsu użytkownika. Coś w tym jest. My nie będziemy się zgłębiać w całą składnie polecenia sed, natomiast skupimy się na małym wycinku - przetwarzaniu wyrażeń regularnych.

Sed najczęściej przetwarza ciąg znaków który dociera do niego przez standardowe wejście i zwraca wynik na standardowe wyjście. Składnia wyrażeń regularnych jest w większości zgodna z innymi programami. Podstawowe elementy składni wyrażeń wyglądają tak:

- `\[...\]` - w miejsce ... wchodzi lista znaków które mogą wystąpić na danym miejscu
- `()` - zbiór opcji
- `^` - początek ciągu znaków (jeśli nie występuje wewnątrz nawiasów kwadratowych)
- `[^]` - wewnątrz nawiasów kwadratowych to negacja tego co w nawiasie
- `$` - koniec ciągu znaków
- `*` - wzorzec tuż przed tym znakiem ma się powtarzać dowolną liczbę razy (od 0 do nieskończoności)
- `+` - wzorzec tuż przed tym znakiem ma się powtarzać co najmniej raz
- `.` - dowolny jeden znak
- `...-...` - zakres znaków np. [a-z]
- `\` - znak modyfikacji (escape character)

```
echo Alicja | sed s/a$/ę/g | sed 's/\([^ę]\)$/\1a/g'
```

Narzędzia, tutoriale i materiały

- https://regex101.com/ - webowy edytor/tester
- https://regexr.com/ - kolejny webowy edytor/tester
- https://www.regular-expressions.info/ - opis składni, możliwości, dokumentacja

Drobna informacja na przyszłe zajęcia:

W bash-u można także dopasowywać wyrażenia regularne w samej składni

```
[[ tekst_sprawdzany =~ wyrazenie_regularne ]] && echo pasuje
```

Ćwiczenie 6

Teraz postaraj się zrobić to zadanie z haczykiem i podmianą fragmentu zmiennej (Ala ma ...)

Zadanie domowe

Przygotuj skrypt, który odczyta plik (jego nazwa ma być argumentem skryptu) i wypisze go na terminalu tak, aby wszystkie wystąpienia Twojego imienia były podświetlone na czerwono.