Systemy operacyjne 07

Instrukcje warunkowe, pętle, funkcje

1. Strumień danych

Każdy uruchomiony w Linuxie proces skądś pobiera dane, gdzieś wysyła wyniki swojego działania i komunikaty o błędach. Tak więc procesowi przypisane są trzy strumienie danych:

stdin (ang. standard input) czyli standardowe wejście, skąd proces pobiera dane, domyślnie jest to klawiatura

stdout (ang. standard otuput) to standardowe wyjście, gdzie wysyłany jest wynik działania procesu, domyślnie to ekran

stderr (ang. standard error) standardowe wyjście błędów, tam trafiają wszystkie komunikaty o błędach, domyślnie ekran

Linux wszystko traktuje jako plik, niezależnie od tego czy to jest plik zwykły, katalog, urządzenie blokowe (klawiatura, ekran) itd. Nie inaczej jest ze strumieniami, BASH identyfikuje je za pomocą przyporządkowanych im liczb całkowitych (od 0 do 2) tak zwanych deskryptorów plików.

I tak:

0 to plik z którego proces pobiera dane stdin

1 to plik do którego proces pisze wyniki swojego działania stdout

2 to plik do którego trafiają komunikaty o błędach stderr

Za pomocą operatorów przypisania można manipulować strumieniami, poprzez przypisanie deskryptorów: 0, 1, 2 innym plikom, niż tym reprezentującym klawiaturę i ekran.

Przełączanie standardowego wejścia

Zamiast klawiatury jako standardowe wejście można otworzyć plik:

```
< plik
```

Przykład:

Najpierw stwórzmy plik lista o następującej zawartości:

```
slackaware
redhat
debian
caldera
```

Użyjemy polecenia sort dla którego standardowym wejściem będzie nasz plik.

```
sort < lista
```

Wynikiem będzie wyświetlenie na ekranie posortowanej zawartość pliku lista:

```
caldera
debian
redhat
slackware
```

Przełączanie standardowego wyjścia

Wynik jakiegoś polecenia można wysłać do pliku, a nie na ekran, do tego celu używa się operatora:

```
> plik
```

Przykład:

```
ls -la /usr/bin > ~/wynik
```

Rezultat działania polecenia ls -la /usr/bin trafi do pliku o nazwie wynik, jeśli wcześniej nie istniał plik o takiej samej nazwie, to zostanie utworzony, jeśli istniał cała jego poprzednia zawartość zostanie nadpisana.

Jeśli chcemy aby dane wyjściowe dopisywane były na końcu pliku, bez wymazywania jego wcześniejszej zawartości, stosujemy operator:

```
>> plik
```

Przykład:

```
free -m >> ~/wynik
```

Wynik polecenia free -m (pokazuje wykorzystanie pamięci RAM i swap'a) zostanie dopisany na końcu pliku wynik, nie naruszając jego wcześniejszej zawartości.

Przełączanie standardowego wyjścia błędów

Do pliku można też kierować strumień diagnostyczny:

```
2> plik
```

Przykład:

```
#!/bin/bash
echo "Stderr jest skierowane do pliku error"
ls -y 2< ~/error    #błąd
```

W powyższym skrypcie polecenie ls jest użyte z błędną opcją -y, komunikat o błędzie trafi do pliku error.

Za pomocą operatora:

```
<< plik
```

można dopisać do tego samego pliku kilka komunikatów o błędach, dopisanie kolejnego nie spowoduje skasowania wcześniejszej zawartości pliku.

Przykład:

```
#!/bin/bash
echo "Stderr jest skierowane do pliku error"
ls -y 2> ~/error             #błąd
cat /etc/shadow 2> ~/error   #błąd2
```

Jako błąd drugi zostanie potraktowane polecenie cat /etc/shadow (zakładając, że zalogowałeś się jako użytkownik) ponieważ prawo odczytu pliku /etc/shadow ma tylko root.

2. Instrukcja warunkowa if

Sprawdza czy warunek jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub polecenia znajdujące się po słowie kluczowym then. Instrukcja kończy się słowem fi.

Składnia:

```
if warunek
then
  polecenie
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
  echo "Masz plik .bashrc"
fi
```

Najpierw sprawdzany jest warunek: czy istnieje w twoim katalogu domowym plik .bashrc, zapis ~/ oznacza to samo co /home/twój_login lub $HOME. Jeśli sprawdzany warunek jest prawdziwy to wyświetlony zostanie napis Masz plik .bashrc. W przeciwnym wypadku nic się nie stanie.

W sytuacji gdy test warunku zakończy się wynikiem negatywnym można wykonać inny zestaw poleceń, które umieszczamy po słowie kluczowym else:

Składnia:

```
if warunek
then
  polecenie1
else
  polecenie2
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
  echo "Masz plik.bashrc"
else
  echo "Nie masz pliku .bashrc"
fi
```

Jeśli warunek jest fałszywy skrypt poinformuje Cię o tym.

Można też testować dowolną ilość warunków, jeśli pierwszy warunek nie będzie prawdziwy, sprawdzony zostanie następny, kolejne testy warunków umieszczamy po słowie kluczowym elif.

Składnia:

```
if warunek
then
  polecenie1
elif warunek
then
  polecenie2
fi
```

Przykład:

```
#!/bin/bash
if [ -x /opt/kde/bin/startkde ]; then
   echo "Masz KDE w katalogu /opt"
elif [ -x /usr/bin/startkde ]; then
  echo "Masz KDE w katalogu /usr"
elif [ -x /usr/local/bin/startkde ]; then
  echo "Masz KDE w katalogu /usr/local"
else
  echo "Nie wiem gdzie masz KDE"
fi
```

Ten skrypt sprawdza gdzie masz zainstalowane KDE, sprawdzane są trzy warunki, najpierw czy plik wykonywalny startkde znajduje się w katalogu /opt/kde/bin jeśli go tam nie ma, szukany jest w /usr/bin, gdy i tu nie występuje sprawdzany jest katalog /usr/local/bin.

Polecenie echo służy do wydrukowania na standardowym wyjściu (**stdout** - domyślnie jest to ekran) napisu.

3. Warunki

Służy do tego polecenie test. (Uwaga! Nie można skryptom nadawać nazwy test! Nie będą działać.)

Składnia:

```
test wyrażenie1 operator wyrażenie2
```

lub może być zapisane w postaci nawiasów kwadratowych:

```
[ wyrażenie1 operator wyrażenie2 ]
```

Uwaga! Między nawiasami a treścią warunku muszą być spacje, tak jak powyżej.

w przypadku sprawdzania warunków arytmetycznych:

```
(( wyrażenie1 operator wyrażenie2 ))
```

Polecenie test zwraca wartość 0 (true) jeśli warunek jest spełniony i wartość 1 (false) jeśli warunek nie jest spełniony. A gdzie jest umieszczana ta wartość? W zmiennej specjalnej $?.

A to kilka przykładów operatorów polecenia test:

-a operator and

-o operator or

-b plik istnieje i jest blokowym plikiem specjalnym

- plik istnieje i jest plikiem znakowym

-e plik istnieje

-h plik istnieje i jest linkiem symbolicznym

= sprawdza czy wyrażenia są równe

!= sprawdza czy wyrażenia są różne

-n wyrażenie ma długość większą niż 0

-d wyrażenie istnieje i jest katalogiem

-z wyrażenie ma zerową długość

-r można czytać plik

-w można zapisywać do pliku

-x można plik wykonać

-f plik istnieje i jest plikiem zwykłym

-p plik jest łączem nazwanym

-N plik istnieje i był zmieniany od czasu jego ostatniego odczytu

plik1 -nt plik2 plik1 jest nowszy od pliku2

plik1 -ot plik2 plik1 jest starszy od pliku2

-lt mniejsze niż

-gt większe niż

-ge większe lub równe

-le mniejsze lub równe

Więcej przykładów operatorów w: man bash.

4. Polecenie read

Czyta ze standardowego wejścia pojedynczy wiersz.

Składnia:

```
read -opcje nazwa_zmiennej
```

Przykład:

```
#!/bin/bash
echo -n "Wpisz coś:\a"
read wpis
echo "$wpis"
```

To co zostało wpisane trafi do zmiennej wpis, której to wartość czyta polecenie read wpis, zmienna nie musi być wcześniej tworzona, jeśli istniała wcześniej, jej zawartość zostanie zastąpiona tym co wpisaliśmy.

Przykład:

```
#!/bin/bash
echo "Wpisz coś:"
answer="napis"
read
echo "$answer"
```

Wcześniejsza wartość zmiennej answer została zastąpiona.

Polecenie read pozwala na przypisanie kilku wartości kilku zmiennym.

Przykład:

```
#!/bin/bash
echo "Wpisz cztery wartości:"
read a b c
echo "Wartość zmiennej a to: $a"
echo "Wartość zmiennej b to: $b"
echo "Wartość zmiennej c to: $c"
```

Nie przypadkiem w powyższym przykładzie pojawiło się polecenie wpisania czterech wartości, pierwsza wartość trafi do zmiennej a, druga do zmiennej b, natomiast trzecia i czwarta oraz rozdzielające je znaki separacji przypisane zostaną zmiennej c.

Wybrane opcje:

-p - Pokaże znak zachęty bez kończącego znaku nowej linii.

```
#!/bin/bash
read -p "Pisz:" odp
echo "$odp"
```

-a - Kolejne wartości przypisywane są do kolejnych indeksów zmiennej tablicowej.

Przykład:

```
#!/bin/bash
echo "Podaj elementy zmiennej tablicowej:"
read tablica
echo "${tablica[*]}"
```

-e - Jeśli nie podano żadnej nazwy zmienej, wiersz trafia do $REPLY.

Przykład:

```
#!/bin/bash
echo "Wpisz coś:"
read -e
echo "$REPLY"
```

-t timeout czas wygaśnięcia w sekundach

-s nie wyświetlaj znaków wpisanych przez użytkownika.

```
#!/bin/bash
#Hasło wpisywane bez echa, max. przez 30 sekund
read -p "Password: " -s  -t 30 password
echo $password
```

5. Instrukcja case

Pozwala na dokonanie wyboru spośród kilku wzorców. Najpierw sprawdzana jest wartość zmiennej po słowie kluczowym case i porównywana ze wszystkimi wariantami po kolei. Oczywiście musi być taka sama jak wzorzec do którego chcemy się odwołać. Jeśli dopasowanie zakończy się sukcesem wykonane zostanie polecenie lub polecenia przypisane do danego wzorca. W przeciwnym wypadku użyte zostanie polecenie domyślne oznaczone symbolem gwiazdki: *) polecenie_domyślne. Co jest dobrym zabezpieczeniem na wypadek błędów popełnionych przez użytkownika naszego skryptu.

Składnia:

```
case zmienna in
  "wzorzec1") polecenie1 ;;
  "wzorzec2") polecenie2 ;;
  "wzorzec3") polecenie3 ;;
  *) polecenie_domyślne
esac
```

Przykład:

```
#!/bin/bash
echo "Podaj cyfrę dnia tygodnia"
```

```
read d
case "$d" in
  "1") echo "Poniedziałek" ;;
  "2") echo "Wtorek" ;;
  "3") echo "Środa" ;;
  "4") echo "Czwartek" ;;
  "5") echo "Piątek" ;;
  "6") echo "Sobota" ;;
  "7") echo "Niedziela" ;;
  *) echo "Nic nie wybrałeś"
esac
```

Przy zastosowaniu ;; po przypasowaniu do warunku i wykonaniu instrukcji wychodzi z konstrukcji case. Przy zastosowaniu ;& wykonuje instrukcje także kolejnego warunku.

Jak widać mamy w skrypcie wzorce od 1 do 7 odpowiadające liczbie dni tygodnia, każdemu przypisane jest jakieś polecenie, tutaj ma wydrukować na ekranie nazwę dnia tygodnia. Jeśli podamy 1 polecenie read czytające dane ze standardowego wejścia przypisze zmiennej d wartość 1 i zostanie wykonany skok do wzorca 1, na ekranie zostanie wyświetlony napis Poniedziałek. W przypadku gdy podamy cyfrę o liczbie większej niż 7 lub wpiszemy inny znak na przykład literę to wykonany zostanie wariant defaultowy oznaczony gwiazdką:

```
*) echo "Nic nie wybrałeś".
```

6. Pętla for

Wykonuje polecenia zawarte wewnątrz pętli, na każdym składniku listy (iteracja).

Składnia:

```
for zmienna in lista
do
  polecenie
done
```

Przykład:

```
for x in jeden dwa trzy
do
  echo "To jest $x"
done
```

Zmiennej x przypisana jest lista, która składa się z trzech elementów: jeden, dwa, trzy. Wartością zmiennej x staje się po kolei każdy element listy, na wszystkich wykonywane jest polecenie: echo "To jest $x". Pętla for jest bardzo przydatna w sytuacjach, gdy chcemy wykonać jakąś operację na wszystkich plikach w danym katalogu. Na przykład chcemy uzyskać listę wszystkich plików o danym rozszerzeniu znajdujących się w jakimś katalogu, robimy to tak:

```
#!/bin/bash
for x in *html
do
  echo "To jest plik $x"
done
```

lub jeśli chcemy zmienić nazwy plików pisane DUŻYMI literami na nazwy pisane małymi literami:

```
#!/bin/bash
for nazwa in *
do
  mv $nazwa `echo $nazwa | tr '[A-Z]' '[a-z]'`
done
```

Za zmianę DUŻYCH liter na małe (i na odwrót) odpowiedzialne jest polecenie tr

7. Pętla select

Wygeneruje z listy słów po in proste ponumerowane menu, każdej pozycji odpowiada kolejna liczba od 1 wzwyż. Poniżej menu znajduje się znak zachęty PS3 gdzie wpisujemy cyfrę odpowiadająca wybranej przez nas pozycji w menu. Jeśli nic nie wpiszemy i wciśniemy ENTER, menu będzie wyświetlone ponownie. To co wpisaliśmy zachowywane jest w zmiennej REPLY. Gdy odczytane zostaje EOF (ang. End Of File) czyli znak końca pliku (CTRL+D) to select kończy pracę. Pętla działa dotąd dopóki nie wykonane zostaje polecenie break lub return.

Składnia:

```
select zmienna in lista
do
  polecenie
done
```

Praktyczny przykład:

```
#!/bin/bash
echo "Co wybierasz?"
select y in X Y Z Quit
do
  case $y in
    "X") echo "Wybrałeś X" ;;
    "Y") echo "Wybrałeś Y" ;;
    "Z") echo "Wybrałeś Z" ;;
    "Quit") exit ;;
    *) echo "Nic nie wybrałeś"
  esac
break
done
```

Najpierw zobaczymy proste ponumerowane menu, składające się z czterech elementów: X, Y, Z i Quit, teraz wystarczy tylko wpisać numer inetersującej nas opcji, a resztę zrobi instrukcja case. Polecenie break, które znajduje się w przedostatniej linii skryptu, kończy pracę pętli. Słowo kluczowe continue - przerywa wykonywanie instrukcji w ciele pętli i przechodzi do następnej iteracji.

8. Pętla while

Najpierw sprawdza warunek czy jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub lista poleceń zawartych wewnątrz pętli, gdy warunek stanie się fałszywy pętla zostanie zakończona.

Składnia:

```
while warunek
do
polecenie
done
```

Przykład:

```
#!/bin/bash
x=1;
while [ $x -le 10 ]; do
echo "Napis pojawił się po raz: $x"
x=$[x + 1]
done
```

Sprawdzany jest warunek czy zmienna x o wartości początkowej 1 jest mniejsza lub równa 10, warunek jest prawdziwy w związku z czym wykonywane są polecenia zawarte wewnątrz pętli: echo "Napis pojawił się po raz: $x" oraz x=$[x + 1], które zwiększa wartość zmiennej x o 1. Gdy wartość x przekroczy 10, wykonanie pętli zostanie przerwane.

9. Pętla until

Sprawdza czy warunek jest prawdziwy, gdy jest fałszywy wykonywane jest polecenie lub lista poleceń zawartych wewnątrz pętli, miedzy słowami kluczowymi do a done. Pętla until kończy swoje działanie w momencie gdy warunek stanie się prawdziwy.

Składnia:

```
until warunek
do
polecenie
done
```

Przykład:

```
#!/bin/bash
x=1;
until [ $x -ge 10 ]; do
echo "Napis pojawił się po raz: $x"
x=$[x + 1]
done
```

Mamy zmienną x, która przyjmuje wartość 1, następnie sprawdzany jest warunek czy wartość zmiennej x jest większa lub równa 10, jeśli nie to wykonywane są polecenia zawarte wewnątrz pętli. W momencie gdy zmienna x osiągnie wartość, 10 pętla zostanie zakończona.

10. Funkcje

Coś w rodzaju podprogramów. Stosuje się je gdy w naszym skrypcie powtarza się jakaś grupa poleceń, po co pisać je kilka razy, skoro można to wszystko umieścić w funkcjach. Do danej funkcji odwołujemy się podając jej nazwę, a wykonane zostanie wszystko co wpisaliśmy między nawiasy { }, skraca to znacznie długość skryptu.

Składnia:

```
function nazwa_funkcji
```

```
{
polecenie1
polecenie2
polecenie3
}
```

lub:

```
function nazwa_funkcji()
{
polecenie1
polecenie2
polecenie3
}
```

Przykład:

```
#!/bin/bash

function napis
{
echo "To jest napis"
}

napis
```

Nazwę funkcji umieszczamy po słowie kluczowym function, w powyższym przykładzie mamy funkcje o nazwie napis, odwołujemy się do niej podając jej nazwę, wykonane zostaną wtedy wszystkie polecenia, jakie jej przypiszemy.

Funkcje moga się znajdować w innym pliku, co uczyni nasz skrypt bardziej przejrzystym i wygodnym, tworzy się własne pliki nagłówkowe, wywołuje się je tak:

```
. ~/naszplik_z_funkcjami
nazwa_funkcji
```

Trzeba pamiętać o podaniu kropki + spacja przed nazwą pliku

Przykład:

```
#!/bin/bash
function nasza_funkcja
{
echo -e 'Właśnie użyłeś funkcji o nazwie "nasza_funkcja".\a'
}
```

Teraz pozostało jeszcze utworzyć skrypt w którym wywołamy funkcje: nasza_funkcja

```
#!/bin/bash
echo "Test funkcji."
. funkcja
nasza_funkcja
```

Przekazywanie parametrów

Przekazanie parametrów do funkcji następuje dokładnie tak samo jak do każdego polecenia które jest w skrypcie:

```
nazwa_funkcji parametr_1 parametr_2
```

Przykład:

```
#!/bin/bash

funkcja_z_parametrami()
{
        echo "Przekazano $# parametrów"
        echo "Parametr $1"
        echo "Parametr $2"
}

funkcja_z_parametrami "param1" "param2"
```

Zmienna specjalna $0 przechowująca nazwę skryptu nie jest dostępna!!! Choć na pierwszy rzut oka powinna przechowywać nazwę funkcji.

11. Obliczanie wyrażeń arytmetycznych

Interpretacja wyrażeń arytmetycznych.

Kiedy zachodzi potrzeba przeprowadzenia jakichś obliczeń można skorzystać z mechanizmu interpretacji wyrażeń arytmetycznych, obliczenia dokonywane są na liczbach całkowitych, nie przeprowadzana jest kontrola przepełnienia (ang. overflow).

Składnia:

```
$((wyrażenie)) lub $[wyrażenie]
```

Przykład:

```
#!/bin/bash
echo $((8/2))
wynik=$[4*5/2]
echo "$wynik"
```

W ten sposób (przykład poniżej) można ponumerować listę:

```
#!/bin/bash

for pliki_html in $(ls *.html)
do
numer=$((numer+1))
echo "$numer. "
echo $pliki_html
done
```

Wynikiem będzie ponumerowana lista wszystkich plików o rozszerzeniu .html, znajdujących się w bieżącym katalogu.

Polecenie let

Do przeprowadzenia obliczeń można też skorzystać z polecenia let.

Przykład:

```
#!/bin/bash
liczba1=5
liczba2=6
let wynik=liczba1*liczba2
echo $wynik
```

Zadania do wykonania

1. Napisać skrypt pytający się czy jest wieczór. Dla odpowiedzi 'tak' powinien wypisać 'Dobry wieczór', dla odpowiedzi 'nie' - 'Dzień dobry', dla pozostałych odpowiedzi 'Nie rozpoznana odpowiedz: ' i przytoczyć treść odpowiedzi. (Użyj instrukcji warunkowej if-elif.

2. Napisać skrypt pobierający numer dnia tygodnia i wypisujący jego nazwę lub informację "Nic nie wybrałeś". (Użyj polecenia read i instrukcji warunkowej case).

3. Wyświetlić z bieżącego katalogu nazwy wszystkich plików: *.html, *.htm, *.php, *.css, *.gif, *.jpg.(Użyj pętli for).

4. Napisać skrypt wykonujący pętlę 15 razy i wypisujący za każdym razem numer obiegu pętli. (Użyj pętli while).