



Wstęp do systemu UNIX

Cz. 4:
Interpretery poleceń

Definicja

- Interpreter poleceń to pośrednik między użytkownikiem, a systemem operacyjnym.
- Nie jest integralną częścią systemu operacyjnego. Znajomość protokołu komunikacyjnego z jądrem systemu operacyjnego wystarcza do stworzenia własnego interpretera poleceń.
- Jego konfiguracja decyduje o jakości i efektywności pracy z systemem operacyjnym.

Systematyka

- Ze względu na pełnioną funkcję:
 - logujący – dzięki któremu jesteśmy podłączeni do systemu
 - interaktywny – uruchomiony z wiersza polecenia
 - interpretujący – uruchomiony (najczęściej niejawnie) do zinterpretowania skryptu.
- Ze względu na udostępniany język programowania:
 - rodzina interpreterów sh: sh, ksh, zsh, bsh, bash
 - rodzina oparta na języku C: csh, tcsh
 - inne

Wiersz polecenia



Fazy działania

- Interpreter logujący i interaktywny działają cyklicznie, a koniec ich pracy następuje po naciśnięciu ^d, wydaniu polecenia *logout* lub *exit*.
- Cykl składa się z następujących kroków:
 1. Oczekiwanie na wprowadzenie polecenia.
 2. Po naciśnięciu klawisza enter, interpreter wyszukuje znaków specjalnych i zamienia je zgodnie ze znaczeniem. Następnie sprawdza poprawność wiersza polecenia.
 3. Jeśli wiersz jest poprawny, poszukuje polecenia do wykonania.
 4. Jeśli polecenie zostanie odnalezione uruchamia je i czeka na jego zakończenie.
 5. Przechodzi do kroku 1.

Znaki specjalne w nazwach plików

- Interpreter zakłada, że wszystko w wierszu polecenia co nie jest poleceniem lub opcją jest nazwą pliku lub znakiem specjalnym.
- W nazwach plików mogą wystąpić następujące znaki (generowania nazw plików):

Znak	Działanie
*	Dopasuj dowolny łańcuch znaków (również pusty)
?	Dopasuj pojedynczy znak alfanumeryczny.
[....]	Dopasuj pojedynczy znak z wyszczególnionych w nawiasach.
~	Zastąp znak katalogiem osobistym.
~-	Zastąp znaki poprzednim katalogiem roboczym.
~+	Zastąp znaki bieżącym katalogiem roboczym.
~login	Zastąp znaki katalogiem domowym użytkownika.

Inne znaki specjalne

- To znaki, które shell interpretuje specjalnie i na ich podstawie wykonuje określone działania.

Znak	Działanie	Przykład
;	Separator sekwencji poleceń.	cd /tmp; ls; cd
&	Wykonuj polecenie w tle.	cp -R /usr . &
>	Przekieruj standardowe wyjście (nadpisywanie).	ls -al > ls.txt
>>	Przekieruj standardowe wyjście (dopisywanie).	ls -al >> ls.txt
<	Przekieruj standardowe wejście.	./program < dane.txt
<<	Znak końca danych wejściowych (uruchamianie bez std. Wejścia).	./program << EOF 12.45466 EOF
()	Grupuj standardowe wyjścia poleceń.	(ls /tmp; ls /usr) >ls.txt

Znaki specjalne - cd.

Znak	Działanie	Przykład
	Połącz polecenia w potok.	ls -a grep `*.h`
\${zmienna}	Pobierz wartość zmiennej.	\$FILE
`polecenie`	Wykonaj polecenie i podstaw standardowe wyjście.	ls -l `which passwd`
\	Na początku wiersza polecenia – traktuj polecenie jako zewnętrzne; W środku – cytuj występujący za nim jeden znak; Na końcu – kontynuuj polecenie w linii kontynuacji.	\ls -al echo \ \$HOME ls -al \ > /usr/include
`napis`	Cytuj wszystkie znaki w napisie.	echo ` \$HOME *`
"napis"	Cytuj znaki w napisie, ale wykonaj podstawienia.	echo "kat biez `pwd`"
&	W wierszu polecenia – do oznaczenia numeru pliku	ls 2>&1 > ls.txt
&&	Przerywa wykonywanie sekwencji poleceń po pierwszym zakończonym błędem	g++ -o prog prog.c && ./prog
	Przerywa wykonywanie sekwencji poleceń po pierwszym zakończonym sukcesem	cd /tmp ls -al

Zmienne shella

- Interpreter poleceń, jak każdy program, ma swoje zmienne. Zmienne wpływają na sposób pracy interpretera.
- Zmienne mogą być tylko ustawione lub ustawione i nadana im wartość.
- Istnieją trzy główne typy zmiennych:
 1. Zmienne specjalne – mają przypisywane wartości w momencie uruchamiania shella.
 2. Zmienne środowiska – wykorzystywane do przechowywania danych pomocnych w nadzorowaniu sesji shella.
 3. Zmienne programowe – zmienne zdefiniowane przez użytkownika.

Zmienne specjalne

Zmienna	Możliwość modyfikacji	Znaczenie
\$#	Nie	Liczba argumentów, z którymi uruchomiono shell (skrypt).
\$0	Nie	Nazwa programu (skryptu).
\$1, \$2, ...	Tak	Kolejne argumenty, z którymi uruchomiono interpreter (skrypt).
\$*	Nie	Lista argumentów, z którymi uruchomiono interpreter (skrypt).
\$?	Nie	Kod powrotu (zakończenia) ostatniego polecenia.
\$\$	Nie	Numer procesu (PID) bieżącego interpretera.
\$_	Nie	Numer procesu ostatnio wykonywanego.

Zmienne środowiska

- Pełna lista dostępna w podręczniku dla danego interpretera poleceń.
- Przyjmują wartości domyślne podczas uruchamiania interpretera.
- Ich wartości mogą zostać zmienione przez użytkownika podczas pracy interpretera.
- Listę zmiennych środowiskowych bieżącego interpretera udostępnia polecenie *set*
- Listę zmiennych, które zostaną przekazane do środowiska procesu potomnego udostępnia polecenie *env*

Wybrane zmienne środowiskowe

Zmienna (bash)	Opis
HOME	Ścieżka dostępu do katalogu domowego bieżącego użytkownika.
PATH	Zawiera ścieżki dostępu do katalogów, w których interpreter poszukuje programów do wykonania.
HISTFILE	Nazwa pliku z historią poleceń.
HISTSIZE	Liczba pamiętanych poleceń.
TMOUT	Czas bezczynności w sekundach, po którym interpreter zakończy działanie.
SHELLOPTS	Opcje shella. Ustawia się je poleceniem <code>set -o</code> , np. <code>set -o noclobber</code> spowoduje, że nadpisanie pliku przez przekierowanie nie będzie możliwe.
PS1	Postać podstawowego znacznika zachęty.
PS2	Postać znacznika zachęty linii kontynuacji.
IGNOREEOF	Jej wartość mówi ile razy z rzędu należy nacisnąć <code>^d</code> aby odłączyć się od systemu.
LANG	Język sesji (np. <code>En_us</code> , <code>Pl_pl</code>).

Nadawanie wartości zmiennym (środowiska i programowym)

- Obowiązuje schemat: *zmienna=wartość*
- Nazwy zmiennych programowych powinny być rozsądnej długości, zaczynać się od litery lub znaku podkreślenia. W rodzinie sh używa się zazwyczaj dużych liter.

```
PS1='[t \u@\h \w]\$ '
MIESIAC=`date +%m`
HASLO="Ala ma zumwia"
PUSTY=
```

- Uwaga na zmienną PATH. Wymagane jest zachowanie poprzedniej wartości, gdyż jej „zgubienie” spowoduje iż polecenia zewnętrzne trzeba będzie uruchamiać z bezwzględną ścieżką dostępu!!!

```
PATH=$PATH:..
PATH=$PATH:$HOME/binarki
```

- Zmienna jest dostępna od momentu zdefiniowania do zakończenia działania procesu lub użycia polecenia *unset*

```
unset HASLO
```

Przekazywanie zmiennej ze środowiskiem

- Zmienna zdefiniowana będzie dostępna w środowisku bieżącego interpretera (pokaże ją polecenie *set*, nie będzie widoczna w *env*).
- Aby została przekazana ze środowiskiem należy ją „wyeksportować” – polecenie *export*
- Raz wyeksportowana zmienna będzie przekazywana do kolejnych procesów potomnych.
- Mechanizm jest niezależny od rodziny interpretera.

```
TMOUT=900
```

```
export TMOUT
```

```
export HASLO="3 czerwone wiewiorki wyladowaly w Bagdadzie"
```

```
export PATH TMOUT HASLO
```

Zmienne tylko do odczytu

- Istnieje możliwość zabezpieczenia zmiennej przed zmianą wartości. Służy do tego polecenie *readonly*
- Jest to jedynie atrybut, który nie jest przekazywany ze środowiskiem do procesu potomnego. Stąd proces potomny może zmienić jej wartość.

```
HASLO="Ala ma kota, Jola psa"  
readonly HASLO  
export HASLO  
export readonly ODZEW="I co z tego"
```

Poszukiwanie programu do uruchomienia

- Interpreter poleceń poszukuje programu do wykonania w następujących miejscach:
 1. Jako polecenia wbudowanego, stanowiącego kod interpretera (shift, break, cd, fc, echo, set, unset, read, ...).
 2. Jeśli nie znalazł polecenia wbudowanego, to jako aliasu – skróconej nazwy polecenia zdefiniowanej przez użytkownika.
 3. Jeśli nie jest to alias, to pozostało przeglądnięcie katalogów stanowiących wartość zmiennej PATH w poszukiwaniu polecenia zewnętrznego.
- Nie znalezienie programu w trzech powyższych lokacjach skutkuje komunikatem o błędzie.

Aliasy

- Są wykorzystywane do nadawania krótkiej nazwy poleceniom często uruchamianym zazwyczaj z dużą liczbą opcji.
- Alias nie udostępnia możliwości przekazywania argumentów. W takich przypadkach definiujemy funkcje.
- Zasadniczo zarządzanie wymaga znajomości dwóch poleceń:
 1. alias – umożliwia wypisywanie listy zdefiniowanych aliasów oraz definiowanie nowych.
 2. unalias – służy usuwaniu zdefiniowanych aliasów.

Zarządzanie aliasami

- Polecenie `alias` wypisuje listę zdefiniowanych aliasów w kolejności alfabetycznej.

```
boryczko@student:~$ alias
alias d='dir'
alias dir='/bin/lis $LS_OPTIONS --format=vertical'
alias ls='/bin/lis $LS_OPTIONS'
alias mc='./usr/share/mc/bin/mc-wrapper.sh'
alias v='vdir'
boryczko@student:~$ echo $LS_OPTIONS
-F -b -T 0 --color=auto
```

- Definiowanie aliasów: *alias nazwa=definicja*
- Nazwa ma oddawać to co alias robi. Może być nazwą polecenia zewnętrznego („przykrywa” go).
- Alias może wykorzystywać polecenie wbudowane, inny alias lub polecenie zewnętrzne.

```
boryczko@student:~$ ls
-bash: ls: command not found
boryczko@student:~$ alias ls='ls | less'
boryczko@student:~$ alias rm='rm -i'
```

- Usuwanie aliasów: *unalias nazwa*

```
boryczko@student:~$ unalias rm
boryczko@student:~$ unalias ls
```

Zmienna PATH

- Zawiera ścieżki dostępu do katalogów, w których interpreter poszukuje programu do uruchomienia oddzielone znakiem ":".
- Wartość zmiennej przeglądana jest od lewej do prawej. Jeśli polecenie zostanie odnalezione, to jest uruchamiane, a przeglądanie przerywane.
- Katalog bieżący dodaje się zazwyczaj na końcu wartości. Nie jest to rozwiązanie efektywne, ale bezpieczne, gdyż polecenia z katalogu bieżącego nie „przysłonią” systemowych.

Historia wiersza polecenia

- Przechowywana w pliku o nazwie \$HISTFILE w ilości \$HISTSIZE.
- Listę wypisuje polecenie *history*
- Przywoływanie:
 - Klawisze strzałkowe.
 - !! – wykonaj ostatnio wydane polecenie.
 - !n – wykonaj polecenie o numerze n.
 - !-n – wykonaj polecenie n numerów wstecz względem bieżącego.
 - !napis – wykonaj ostatnie polecenie zaczynające się od napis.
 - !napis:s/stary/nowy – wykonaj ostatnie polecenie zaczynające się od napis po uprzedniej zamianie napisów w poleceniu.

Pliki konfiguracyjne (bash)

- Wszystkie wprowadzone zmiany będą obowiązywały w bieżącej sesji.
- Interpreter posiada nadane wartości domyślne zmiennych środowiskowych. Pozostałe czyta z plików konfiguracyjnych.
- Ogólny, dla wszystkich użytkowników: `/etc/profile`
- Indywidualny użytkownika, interpreter logujący: `$HOME/.profile`,
`$HOME/.bash_profile`
- Indywidualny użytkownika, interpreter interaktywny: `$HOME/.bashrc`

Programowanie w języku interpreterów rodziny sh



Stworzenie skryptu

- W najprostszym przypadku bezpośrednio w wierszu poleceń:

```
boryczko@student:~$ while [ 1 ]; do  
> LPROC=`ps aux | wc -l`  
> LPROC=`expr $LPROC - 1`  
> echo $LPROC  
> sleep 2  
> done  
399  
400  
400  
398  
^C  
boryczko@student:~$
```

- Wadą jest to, że jest dostępny jedynie w pliku historii wiersza poleceń. Stąd znacznie bardziej praktycznie jest zapisać go w pliku.

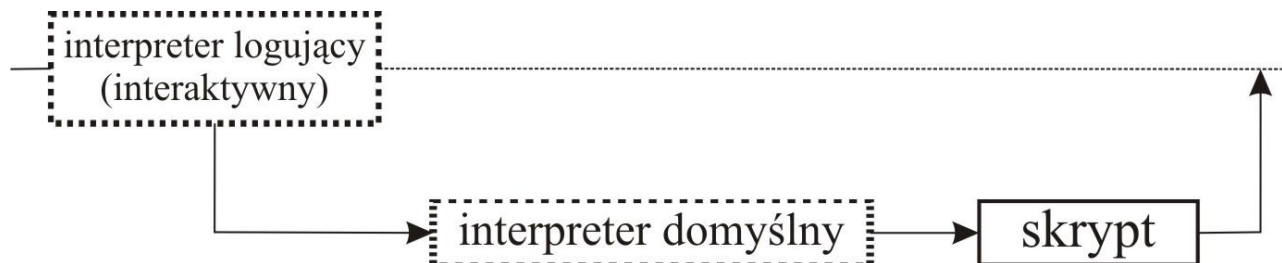
Uruchamianie skryptu (1)

- Skrypt w postaci:

```
if [ $# -eq 0 ]; then  
    echo -n "Podaj UID: "  
    read NUMER  
else  
    NUMER=$1  
fi
```

- Uruchomienie:

```
[wacek@student ~]$ ./skrypt
```



- Wymagane prawo do odczytu i wykonania.

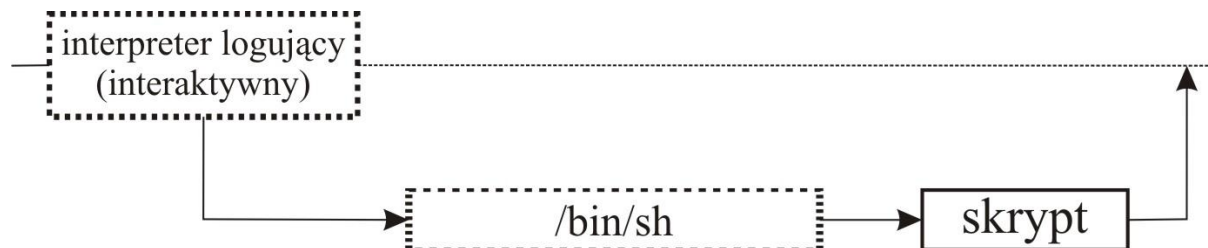
Uruchamianie skryptu (2)

- Skrypt w postaci:

```
#!/bin/sh
if [ $# -eq 0 ]; then
    echo -n "Podaj UID: "
    read NUMER
else
    NUMER=$1
fi
```

- Uruchomienie:

```
[wacek@student ~]$ ./skrypt
```



- Wymagane prawo do odczytu i wykonania.

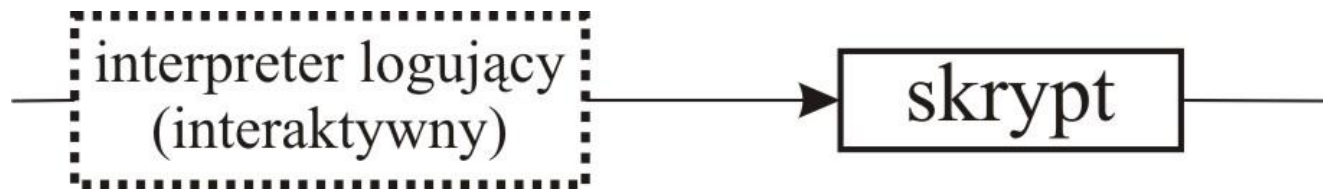
Uruchamianie skryptu (3)

- Skrypt w postaci:

```
if [ $# -eq 0 ]; then
    echo -n "Podaj UID: "
    read NUMER
else
    NUMER=$1
fi
```

- Uruchomienie:

```
[wacek@student ~]$ . skrypt
```



- Wymagane prawo do odczytu i wykonania.
- Zachowane bieżące środowisko!!!

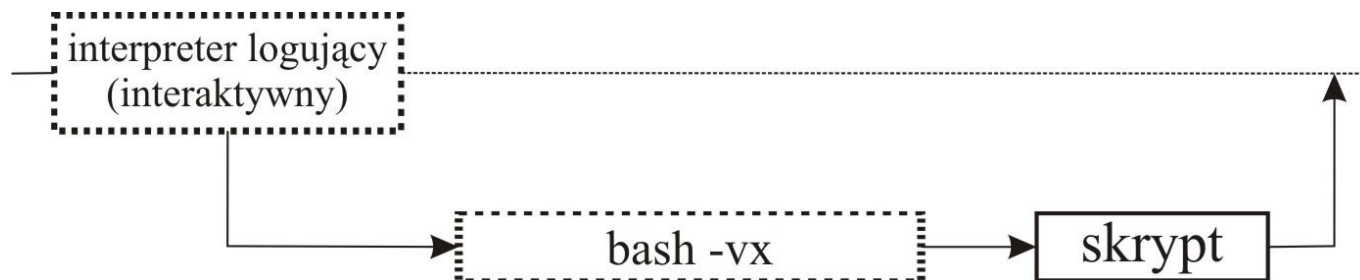
Uruchamianie skryptu (4)

- Skrypt w postaci:

```
if [ $# -eq 0 ]; then
    echo -n "Podaj UID: "
    read NUMER
else
    NUMER=$1
fi
```

- Uruchomienie:

```
[wacek@student ~]$ bash -vx skrypt
```



- Wymagane prawo do odczytu.
- Opcje vx wykorzystywane do debuggowania skryptu.

Polecenie test

(podstawowe warunki testujące)

Warunek	Zwraca
-r <i>plik</i>	Prawdę, jeśli plik istnieje i użytkownik ma prawo do odczytu.
-w <i>plik</i>	Prawdę, jeśli plik istnieje i użytkownik ma prawo do zapisu.
-x <i>plik</i>	Prawdę, jeśli plik istnieje i użytkownik ma prawo do wykonywania.
-f <i>plik</i>	Prawdę, jeśli plik istnieje i jest plikiem zwykłym.
-d <i>plik</i>	Prawdę, jeśli plik istnieje i jest katalogiem.
-p <i>plik</i>	Prawdę, jeśli plik istnieje i jest potokiem (fifo).
-s <i>plik</i>	Prawdę, jeśli plik istnieje i ma rozmiar > 0
-z <i>napis</i>	Prawdę, jeśli długość napisu <i>napis</i> wynosi 0 (napis pusty).
-n <i>napis</i>	Prawdę, jeśli długość napisu <i>napis</i> jest różna od 0 (niepusty).
<i>napis1</i> = <i>napis2</i>	Prawdę, jeśli napisy <i>napis1</i> i <i>napis2</i> są identyczne.
<i>napis1</i> != <i>napis2</i>	Prawdę, jeśli napisy <i>napis1</i> i <i>napis2</i> nie są identyczne.
<i>napis</i>	Prawdę, jeśli długość napisu <i>napis</i> jest > 0 (napis niepusty).
<i>l1</i> -eq <i>l2</i>	Prawdę, jeśli liczby <i>l1</i> oraz <i>l2</i> są równe (-le, -lt, -ge, -gt).

Polecenie test - przykłady

```
boryczko@student:~$ test -f komendy.txt
```

```
boryczko@student:~$ echo $?
```

```
0
```

```
boryczko@student:~$ [ -d komendy.txt ]
```

```
boryczko@student:~$ echo $?
```

```
1
```

```
boryczko@student:~$ N1="Ala ma kota"; N2="Ola ma kota"
```

```
boryczko@student:~$ [ $N1 = $N2 ]
```

```
-bash: [: too many arguments
```

```
boryczko@student:~$ [ "$N1" = "$N2" ]
```

```
boryczko@student:~$ echo $?
```

```
1
```

```
boryczko@student:~$ [ "$N1" != "$N2" ]
```

```
boryczko@student:~$ echo $?
```

```
0
```

```
boryczko@student:~$ L1=35; L2=22
```

```
boryczko@student:~$ [ $L1 -eq $L2 ]
```

```
boryczko@student:~$ echo $?
```

```
1
```

```
boryczko@student:~$ [ $L1 -gt $L2 ]
```

```
boryczko@student:~$ echo $?
```

```
0
```

```
boryczko@student:~$ [ $L1 -le $L2 ]
```

```
boryczko@student:~$ echo $?
```

```
1
```

Polecenie expr

- Wykorzystywane do wykonywania podstawowych operacji arytmetycznych.
- Można również wykorzystać do porównywania (zwraca 0 – fałsz i 1 – prawda).

Operator	Opis	Zwracana wartość
*	Mnożenie całkowitoliczbowe	Liczba całkowita
/	Dzielenie całkowitoliczbowe	Liczba całkowita
%	Reszta z dzielenia całkowitoliczbowego	Liczba całkowita
+	Dodawanie całkowitoliczbowe	Liczba całkowita
-	Odejmowanie całkowitoliczbowe	Liczba całkowita
=	Równa się	Prawda/Fałsz
!=	Nie równa się	Prawda/Fałsz
>	Większa niż	Prawda/Fałsz
>=	Większa lub równa niż	Prawda/Fałsz
<	Mniejsza niż	Prawda/Fałsz
<=	Mniejsza lub równa niż	Prawda/Fałsz

Polecenie expr - przykłady

```
boryczko@student:~$ L1=35; L2=761
```

```
boryczko@student:~$ expr $L2 / $L1  
21
```

```
boryczko@student:~$ expr $L1 * $L2  
expr: syntax error
```

```
boryczko@student:~$ echo "$L1 * $L2"  
35 * 761
```

```
boryczko@student:~$ echo $L1 * $L2  
35 c2.hist komendy.txt mail testy testy.tgz 761
```

```
boryczko@student:~$ expr $L1 \* $L2  
26635
```

```
boryczko@student:~$ expr $L1 \> $L2  
0
```

```
boryczko@student:~$ expr $L1 \< $L2  
1
```

```
boryczko@student:~$ N1="Jan Nowak"
```

```
boryczko@student:~$ expr "$N1" = "Joe Kowalski"  
0
```

Instrukcja warunkowa

sh, ksh, bsh, bash	csh, tcsh
<pre>if [warunek_testu] ; then lista_instrukcji fi</pre>	<pre>if (warunek_testu) then lista_instrukcji endif</pre>
<pre>if [warunek_testu]; then lista_instrukcji_1 else lista_instrukcji_2 fi</pre>	<pre>if (warunek_testu) then lista_instrukcji_1 else lista_instrukcji_2 endif</pre>
<pre>if [warunek_testu_1]; then lista_instrukcji_1 elif [warunek_testu_2] then lista_instrukcji_2 else lista_instrukcji_3 fi</pre>	<pre>if (warunek_testu_1) then lista_instrukcji_1 else if (warunek_testu_2) then lista_instrukcji_2 else lista_instrukcji_3 endif</pre>

Instrukcja warunkowa - przykład

sh, ksh, bsh, bash	csh, tcsh
<pre>if [-r ./data.txt] then cat ./data.txt else echo "Zapisujemy dane do pliku" cat > data.txt fi</pre>	<pre>if (-r ./data.txt) then cat ./data.txt else echo „Zapisujemy dane do pliku” cat > ./data.txt endif</pre>

Instrukcja wyboru

sh, ksh, bsh, bash	csch, tcsh
<pre>case \${zmienna} in wartosc_1) lista_instrukcji_1 ;; wartosc_2) lista_instrukcji_2 ;; wartosc_3 wartosc_4) lista_instrukcji_3 ;; *) lista_domyslna ;; esac</pre>	<pre>switch (\${zmienna}) case wartosc_1 : lista_instrukcji_1 breaksw case wartosc_2 : lista_instrukcji_2 breaksw case wartosc_3 : case wartosc_4 : lista_instrukcji_3 breaksw default : lista_domyslna breaksw endsw</pre>

Instrukcja wyboru - przykład

sh, ksh, bsh, bash	csch, tcsh
<pre>AKT_MIES=`date +%m` case \$AKT_MIES in 01) echo Styczen ;; 02) echo Luty ;; . . . *) echo "Problemy z data" ;; esac</pre>	<pre>set akt_mies=`date +%m` switch (\$akt_mies) case 01: echo Styczen breaksw case 02: echo Luty breaksw . . . default: echo "Problemy z data" breaksw endsw</pre>

Pętla while

sh, ksh, bsh, bash

```
while instrukcja_test
do
    lista_polecen
done
```

csh, tcsh

```
while instrukcja_test
    lista_polecen
end
```

sh, ksh, bsh, bash

```
while [ 1 ]; do
    if [ warunek_zakonczenia ]; then
        break
    else
        .
        .
    fi
done
```

csh, tcsh

```
while ( 1 )
    if ( warunek_zakonczenia )
        break
    else
        .
        .
    endif
end
```

Pętla while - przykład

sh, ksh, bsh, bash	csh, tcsh
<pre>MONTH=1 while [\$MONTH -le 12]; do echo \$MONTH MONTH = `expr \$MONTH + 1` # MONTH=\$((\$MONTH + 1)) done</pre>	<pre>set month=1 while (\$month <= 12) echo \$month @ month += 1 end</pre>

Pętla until

sh, ksh, bsh, bash	csh, tcsh
<pre>until [<i>warunek_zakonczenia</i>] do <i>lista_instrukcji</i> done</pre>	
<pre>until ["`who grep wacek`"]; do sleep 15 done # wylogowanie uzytkownika . . .</pre>	

Pętla for

sh, ksh, bsh, bash

```
for zmienna_ster in wart_1 wart_2 ...  
do  
    lista_instrukcji (zmienna_ster)  
done
```

```
for ((wart_pocz; warunek_k; zmiana ))  
do  
    lista_instrukcji (zmienna_ster)  
done
```

csh, tcsh

```
foreach zmienna_ster ( wart_1 wart_2 ... )  
    lista_instrukcji (zmienna_ster)  
end
```

Pętla for - przykłady

sh, ksh, bsh, bash	csh, tcsh
<pre>for PLIK in *.tex; do ed - \$FILE << EOF! g/shell/s//Shell/g w q EOF! done</pre>	<pre>foreach plik (*.tex) ed - \$plik << eof! g/shell/s//Shell/g w q eof! end</pre>
<pre>if [-r calosc]; then rm -f calosc fi for ((i=1; i<=20; i++)); do cat plik\$i.txt >> calosc done</pre>	