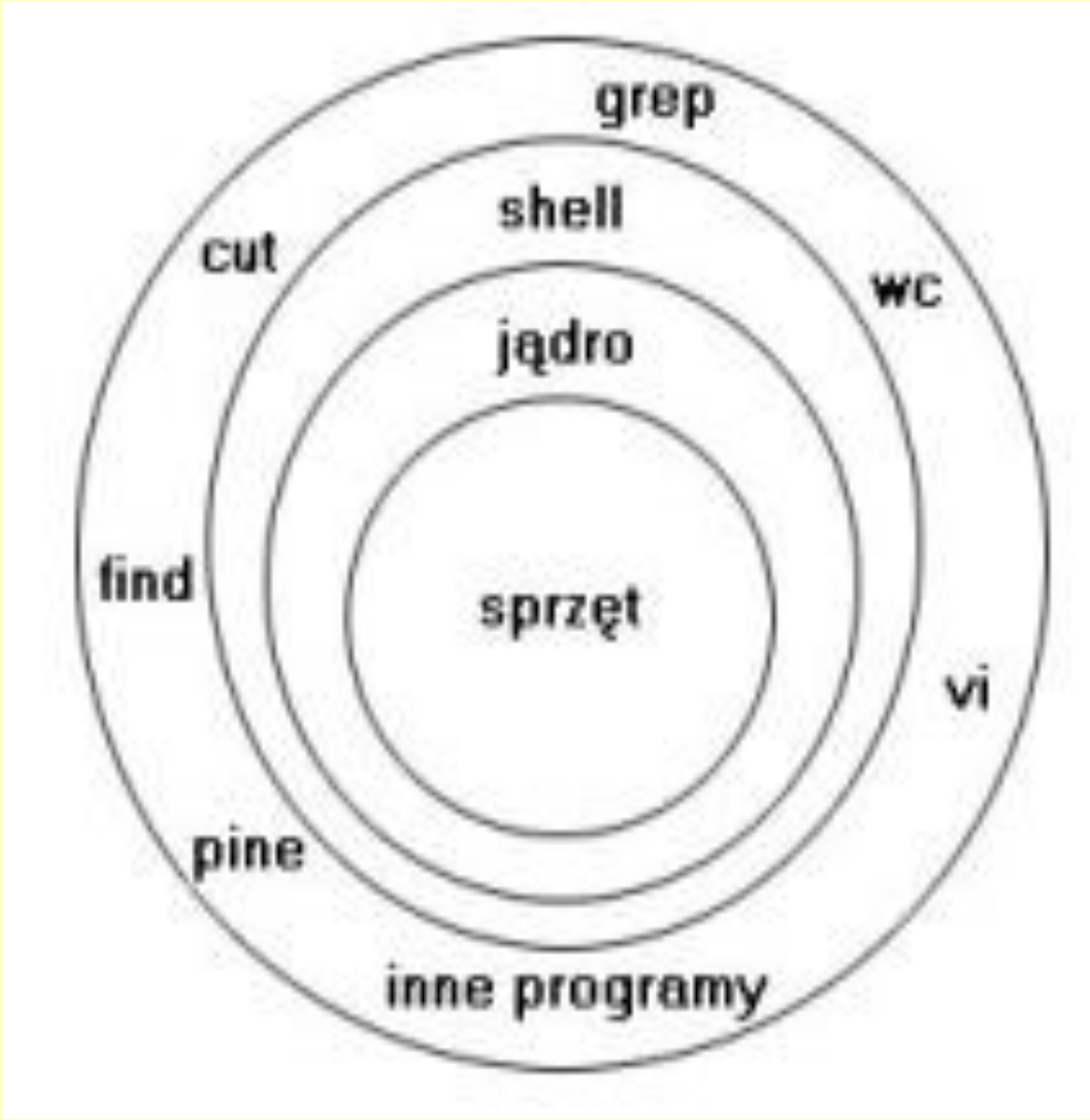
BASH


grep
shell
cut
wc
jądro
find
sprzęt
vi
pine
inne programy

Shell (powłoka) – interpreter poleceń; interfejs pomiędzy użytkownikiem, a jądrem systemu

- zapewnia użytkownikowi pewien poziom abstrakcji w odniesieniu do jądra
- akceptuje polecenia; przetwarzane przez jądro systemu.
- język programowania, pozwala na pisanie tzw. **skryptów shellowych**, które pozwalają na umieszczenie wielu komend w jednym pliku (z atrybutem wykonywania)
- udostępnia środowisko do konfigurowania systemu i programowania

sh	csh
ksh	tcsh
bsh	
bash	

BASH, PDKSH, TCSH

Dwa podstawowe shelle to

- **„Bourne shell”** (*sh:/bin/sh*)
- **„C shell”** (*csh:/bin/csh*).

Bourn shell używa składni takiej samej, jak oryginalny shell na wczesnych wersjach systemu UNIX. Najbardziej rozpowszechnionymi powłokami są :
bash oraz **tcsh**.

bash (Bourne-Again Shell) został stworzony przez Briana Foxa i Cheta Rameya i udostępniony jest na zasadach licencji GNU. Zawiera 48 wbudowanych poleceń oraz 12 funkcji wywołania. Jest w pełni kompatybilny z powłoką *sh* oraz zawiera wiele ciekawych rozwiązań przejętych od powłok Korn (ksh i C csh).

tcsh- jest rozszerzoną wersją oryginalnego shella *csh* stworzonego przez Billy’ego Joy’a. Wiele konstrukcji z tego shella przypomina konstrukcje w języku C- stąd nazwa tej powłoki.

Zmienne:

- lokalne
- środowiskowe (sposób ich przekazywania jest analogiczny do przekazywania parametrów przez wartość – nowa powłoka dysponuje kopiami zmiennych środowiskowych)

nazwa zmiennej może się składać z dowolnych liter, cyfr oraz ze znaku ”_”, nie może jednak rozpoczynać się cyfrą

zmienne lokalne można definiować poleceniem: **zmienna=wartosc** (obok operatora przypisania nie mogą znajdować się spacje!!)

należy się do nich odwoływać poprzez symbol **$zmienna**

unset zmienna – usuwa zmienną

set wyświetla zmienne lokalne powłoki

: ${zz:=1} (warunkowe ustawienie : - polecenie puste-true)

${zm=ls} ${zm=pwd}

Wartości przypisywane zmiennym mogą zawierać znaki, które mają szczególne znaczenie:

”.... ” – maskuje znaki specjalne (spacja, *, ?, [,], ., <, >, &, |),

‘.... ’ – działają jak ”…”, dodatkowo maskują znak $,

`.....` – wymuszają wykonanie polecenia,

\ – maskuje jeden znak.

export zmienna tworzy zmienną środowiskową ze zdefiniowanej zmiennej

Linux definiuje **specjalne** zmienne powłoki, które są używane do konfigurowania powłoki. Ich nazwy są zarezerwowanymi słowami kluczowymi.

env wyświetla listę zdefiniowanych zmiennych specjalnych

Najważniejsze zmienne specjalne powłoki BASH:

HOME – ścieżka katalogu macierzystego

USERNAME – nazwa użytkownika,

SHELL – ścieżka powłoki logowania,

PATH – lista katalogów rozdzielonych dwukropkami, które są przeszukiwane w celu znalezienia polecenia do uruchomienia,

PS1 – monit podstawowy; może zawierać następujące kody:

\! – nr z listy historii,

\d – data,

\s – bieżąca powłoka,

\t – czas,

\u – nazwa użytkownika,

\w – bieżący katalog,

PS2 – monit wtórny (dla drugiego i kolejnych wierszy polecenia),

CDPATH – lista katalogów przeszukiwanych podczas wykonywania komendy cd,

TERM – nazwa terminala,

HISTSIZE – ilość zdarzeń historii

HISTFILE – nazwa pliku historii zdarzeń,

HISTFILESIZE –rozmiar pliku historii,

MAIL – ścieżka do pliku skrzynki pocztowej,

MAILCHECK – określa odstęp czasu, po jakim użytkownik jest powiadamiany o nowej poczcie,

MAILPATH – nazwy ścieżek plików skrzynek pocztowych

IFS – separator pola wejściowego

BASH używa kilku dodatkowych zmiennych określających właściwości powłoki, będącymi opcjami, które mogą być włączone lub wyłączone. Są to: **ignoreeof**, **noclobber**, **noglob**. Ustawienie zmiennej dokonuje się poleceniem:

set –o zmienna

Aby wyłączyć działanie zmiennej, należy użyć polecenia:

set +o zmienna

Znaczenie poszczególnych zmiennych:

ignoreeof – blokuje możliwość wylogowania się za pomocą [CTRL+d] (domyślnie do 10 razy). Można nadać jej dowolną wartość (np. ignoreeof=30),

noclobber – zapobiega nadpisaniu plików podczas przekierowania (>),

noglob – blokuje rozwijanie znaków specjalnych.

HISTORIA

history - wyświetla listę ponumerowanych poleceń (zdarzeń). Można się do nich odwoływać używając poleceń z wykrzyknikiem.

- **!3**–wyświetla i wykonuje polecenie historii nr 3,
- **!ab**–wyświetla i wykonuje ostatnio wydane polecenie rozpoczynające się ciągiem znaków „ab”,
- **!?ab?**–wyświetla i wykonuje ostatnio wydane polecenie zawierające ciąg znaków „ab”,
- **!-3**–wyświetla i wykonuje trzecie od tyłu wydane polecenie.
- Polecenia historii można składać. Jeśli np. polecenie o numerze 100 ma postać „**ls**”, wówczas **„!100 *.c**” jest równoważne poleceniu „**ls *.c**”.

Możliwe jest także odwołanie do parametrów ostatnio wydanego polecenia poprzez „**!!***” lub „**!***”.

ls /home/elektr-a/ea029

find !* -name szukaj

HISTSIZE, HISTFILESIZE

.bash_history,

HISTFILE

ALIASY

Aliasy, czyli utożsamienia, to dodatkowe nazwy dla najczęściej wykonywanych poleceń.

Polecenie **alias**

- wyświetla listę zdefiniowanych aliasów
- definiuje nowe aliasy

Przykłady

- **alias kto=who**
- **alias l=’ls –li’** jeśli w poleceniu występują spacje, należy użyć pojedynczych cudzysłowów,
- **alias lc=’ls –l *.c’**
- **alias cp=’cp –i ‘** nazwa polecenia również może być aliasem.
- Aby usunąć zdefiniowany alias, należy użyć polecenia **unalias**

Aliasy nie są przekazywane do powłok potomnych, istnieją jedynie w bieżącej powłoce, pamiętane są w pamięci operacyjnej .

PLIKI STARTOWE BASH

Przy logowaniu:

.profile definiowanie zmiennych

.bash_profile środowiskowych

Przy każdym uruchomieniu powłoki:

.bashrc definiowanie aliasów

Przy wylogowywaniu:

.bash_logout

SKRYPTY

1. Wykonanie skryptu pod kontrolą konkretnego shella:

   shell skrypt parametry

   gdzie: *shell* – nazwa konkretnego shella, *skrypt* – nazwa pliku z poleceniami

2. Po nadaniu skryptowi atrybutu wykonywalności (x):

   skrypt parametry

   gdzie: *skrypt* – nazwa pliku z poleceniami, parametry – wartości przekazywane do skryptu

3. W powyższych przypadkach *skrypt* jest uruchamiany w odrębnej powłoce. Można go jednak uruchomić w środowisku bieżącego shella:

   . *skrypt parametry*

   source parametry

POJĘCIA PODSTAWOWE

Metaznaki – znaki mające dla shella specjalne znaczenie. Są to:
; & ( ) | < > znak nowej linii, spacja, tabulator. (Np. ; pozwala na umieszczenie w jednej linii wielu poleceń)

Znak kontynuacji linii polecenia: \ – pozwala kontynuować długie teksty poleceń w kilku liniach.

Komentarz – część wiersza zaczynająca się znakiem # (również :). Może się zacząć w dowolnym miejscu linii. Nie może być bezpośrednio poprzedzony znakiem $, gdyż $# jest wartością parametru specjalnego.

Odwołania do wartości: zmiennych lub parametrów odbywa się przez umieszczenie przed nimi znaku $

Znaki: *, ? jeśli nie są poprzedzone znakiem $, są traktowane jako znaki dowolności we wzorcach nazw (np. plików)

PARAMETRY USTAWIANE PRZEZ SHELLA

$0	nazwa skryptu
$1..9..	parametry pozycyjne (przekazane do skryptu) lub ustalone ostatnio wykonanym poleceniem **set**. Odwołanie do parametrów większych niż 9 muszą mieć postać: **${nn}**.
$*	lista parametrów $1, $2, ...rozdzielonych separatorem ($IFS)
$@	$* bez separatorów
$?	kod (status) zakończenia ostatnio wykonanego polecenia (0 – zakończone pozytywnie)
$$	numer procesu powołanego do obsługi skryptu
$!	numer ostatnio uruchomionego procesu drugoplanowego (PID)
$#	liczba parametrów w wywołanym skrypcie
$-	pusty napis

CYTOWANIA

nnn, *znak*	znak o kodzie *nnn* lub dany znak np. * oznacza ‘*’
”*tekst*”	podstawienie tekstu. Odwołania do zmiennych lub parametrów ($1, $zmienna) oraz znaki wyróżnione (np. \t, \n) są interpretowane
’*tekst*’	podstawienie tekstu bez rozwijania zawartych w nim odwołań do zmiennych, parametrów i znaków wyróżnionych
`*polecenie*`	podstawienie wyników wyprowadzanych przez *polecenie*. Jest równoważne z zapisem $(*polecenie*).
$[*wyrażenie*]	podstawienie wartości wyrażenia arytmetycznego
$((*wyrażenie*))	podstawienie wartości wyrażenia arytmetycznego

ZMIENNE

Nazwy zmiennych muszą zaczynać się od litery lub znaku podkreślenia. Rozróżniane są duże i małe litery. Do tworzenia nazw zmiennych nie powinno się używać znaków: ? * itp.

Przypisanie wartości zmiennej odbywa się za pomocą znaku równości. (Np. ILE=5). Przy znaku = nie może być spacji.

Aby zdefiniować puste zmienne (NULL) należy użyć poleceń:
NAZWA= lub NAZWA=””

Aby dana zmienna stała się zmienną globalną (dostępną dla procesów potomnych) należy użyć polecenia:
export *nazwa1* [*nazwa2*] [...] [*nazwa-n*]

POTOKI I LISTY POLECEŃ

Potok to polecenie proste lub ciąg kilku poleceń prostych oddzielonych znakiem |

Lista poleceń to sekwencja jednego lub kilku potoków oddzielonych jednym z symboli: ; & && || i opcjonalnie zakończonych znakiem ; lub &. Symbole: ; i & mają taki sam priorytet, niższy od priorytetów symboli: && i || (równorzędnych).

; Separator sąsiednich potoków.Oznacza ich sekwencyjne wykonanie. Może go zastąpić dowolna ilość znaków nowej linii.

& Potok poprzedzony tym znakiem jest wykonywany w tle. Z wykonaniem następnego potoku shell nie czeka na jego zakończenie.

POTOKI I LISTY POLECEŃ

&& Oznacza, że występująca za tym symbolem lista poleceń zostanie wykonana tylko wtedy, gdy ostatnie polecenie poprzedzającego go potoku zostanie zakończone powodzeniem.

|| Oznacza, że występująca za tym symbolem lista poleceń zostanie wykonana tylko wtedy, gdy ostatnie polecenie poprzedzającego go potoku zostanie zakończone niepowodzeniem.

POLECENIA ZŁOŻONE - *if*

Konstrukcja prosta instrukcji warunkowej if: *if...else...fi*

if *warunek*
then
komenda1 jeżeli warunek jest prawdziwy lub jeżeli status wyjścia warunku był równy zero 0 (zero)
else
komenda2 jeżeli warunek był fałszywy lub status wyjścia warunku był >0 (niezerowy)
fi

if *warunek;* ***then***
komenda1
fi

POLECENIA ZŁOŻONE - *if*

Konstrukcja wielopoziomowa instrukcji warunkowej if: *if...elif...else...fi*

if *warunek*
then
warunek jest równy zero (true - 0) wykonuje wszystkie polecenia do wyrażenia elif
elif *warunek2*
then
warunek2 jest równy zero (true - 0) wykonuje wszystkie polecenia do wyrażenia elif
elif *warunek3;* ***then***
warunek3 jest równy zero (true - 0) wykonuje wszystkie polecenia do wyrażenia else
else
żaden z powyższych warunków nie jest spełniony (wszystkie miały wartość różną od 0) wykonuje wszystkie polecenia do fi
fi

POLECENIA ZŁOŻONE - *case*

Konstrukcja instrukcji warunkowej case: *case...esac*

case *zmienna* ***in***
wart1_zmiennej)
potok instrukcji 1 ***;;***
wart2_zmiennej)
potok instrukcji 2 ***;;***
wart3_zmiennej)
potok instrukcji 3 ***;;***
wart4_zmiennej)
potok instrukcji 4 ***;;***
**)* *„zmienna ma inną wartość niż wyżej wymienione”*
potok instrukcji dla przypadków różnych niż powyższe ***;;***
esac

Należy zwrócić uwagę na podwójny znak średnika kończący każdy potok instrukcji. Jeśli by go zabrakło shell wykonałby kolejne instrukcje.

POLECENIA ZŁOŻONE - *for*

Konstrukcja pętli for: *for...in...do...done*

for *zmienna* ***in*** *lista wartości*

do

potok instrukcji wykonywany jest raz dla każdego elementu na liście dopóki lista nie skończy się. (Powtarzane są wszystkie wyrażenia pomiędzy ***do*** *a* ***done****)*

done

for bez listy – równoważne for z listą argumentów wiersza poleceń

for *zmienna* ***for*** zmienna ***in*** *$@*

for((i=0;i<10;i++)); do ……..; done

POLECENIA ZŁOŻONE – *while/until*

while *warunek*
do
potok instrukcji wykonywany dopóki warunek jest spełniony
done

until *warunek*
do
potok instrukcji wykonywany dopóki warunek jest niespełniony
done

```
until who|grep "$1">/dev/null   do
        sleep 60
done
echo "$1 jest zalogowany"
```

```
z=$(who)
echo $z
```

POLECENIA ZŁOŻONE – informacje dodatkowe

Wewnątrz pętli ***for, while*** i ***until*** mogą występować pewne, dodatkowe polecenia:

break [n] zakończenie wykonywania bieżącej pętli i przejście do pierwszego polecenia po tej pętli, lub - jeżeli wyspecyfikowano *n* – do polecenia po pętli o *n* poziomów zewnętrznej

continue [n] zaprzestanie wykonywania dalszych poleceń z zakresu bieżącej pętli i przejście na jej początek, lub - jeżeli wyspecyfikowano *n* – na początek pętli o *n* poziomów zewnętrznej

(*lista* **)** wykonanie *listy* poleceń jako odrębnego procesu (w podshellu)

WYRAŻENIA WARUNKOWE

Składnia: [*wyrażenie warunkowe*] lub **test** *wyrażenie warunkowe*

Wg powyższej składni odbywa się wyliczenie wartości wyrażenia warunkowego i zwrócenie wartości 0 (true-prawda) jeśli wyrażenie jest prawdziwe. Jeśli wyrażenie zwróci wartość różną od 0 oznacza to, że wyrażenie jest nieprawdziwe (false-fałsz). Nawiasy [] są tutaj elementem składni.

Dostępna jest spora pula dostępnych wyrażeń. Najczęściej używane to:

Operatory logiczne (można ich użyć do wykorzystania dwóch lub więcej warunków w jednym czasie):

! *wyrażenie*	logiczne NIE - negacja
wyrażenie1 -a *wyrażenie2*	logiczne AND (i)
wyrażenie1 -o *wyrażenie2*	logiczne OR (lub)

WYRAŻENIA WARUNKOWE

Operatory dla porównania łańcuchów znakowych:

ciąg	Wartość 0, jeśli *ciąg* nie jest pusty
ciąg1=ciąg2	Wartość 0, jeśli *ciąg1* i *ciąg2* są identyczne
ciąg1!=*ciąg2*	Wartość 0, jeśli *ciąg1* i *ciąg2* są różne
-z *ciąg*	Wartość 0, jeśli *ciąg* istnieje i jest pusty (NULL)
-n *ciąg*	Wartość 0, jeśli *ciąg* istnieje i nie jest pusty

Wyrażenia testujące odnoszące się do plików:

-f *plik*	Wartość 0, jeśli *plik* istnieje i jest to plik zwykły
-d *plik*	Wartość 0, jeśli *plik* istnieje i jest to katalog
-b *plik*	Wartość 0, jeśli *plik* istnieje i jest to plik specjalny reprezentujący urządzenie obsługiwane blokowo
-c *plik*	Wartość 0, jeśli *plik* istnieje i jest to plik specjalny reprezentujący urządzenie obsługiwane znakowo
-r *plik*	Wartość 0, jeśli *plik* istnieje i jest plikiem do odczytu
-w *plik*	Wartość 0, jeśli *plik* istnieje i jest plikiem zapisywalnym
-s *plik*	Wartość 0, jeśli *plik* istnieje i jego rozmiar jest niezerowy

WYRAŻENIA WARUNKOWE

Operatory testujące wyrażenia arytmetyczne:

wyr1 **-eq** *wyr2*	Wartość 0, jeśli *wyr1* i *wyr2* są równe (==)
wyr1 **-ne** *wyr2*	Wartość 0, jeśli *wyr1* i *wyr2* są różne (!=)
wyr1 **-lt** *wyr2*	Wartość 0, jeśli *wyr1* jest mniejsze od *wyr2* (<)
wyr1 **-le** *wyr2*	Wartość 0, jeśli *wyr1* jest mniejsze lub równe *wyr2* (<=)
wyr1 **-gt** *wyr2*	Wartość 0, jeśli *wyr1* jest większe od *wyr2* (>)
wyr1 **-ge** *wyr2*	Wartość 0, jeśli *wyr1* jest większe lub równe *wyr2* (>=)

DEFINIOWANE FUNKCJE

W bashu także można definiować funkcje - zbiory instrukcji, które często się powtarzają w programie. Funkcję definiuje się w następujący sposób:

[function] *nazwa_funkcji()*
{
instrukcja1
instrukcja2**;**
}

Wywołanie funkcji: ***nazwa_funkcji parametry.*** Wewnątrz funkcji parametry są widoczne jako zmienne $1, $2, $*, itd. Wyjście z funkcji odbywa się po wykonaniu polecenia **return** [n] lub ostatniego polecenia w funkcji. Zwracany jest kod wyspecyfikowany w poleceniu **return** lub równy kodowi zakończenia ostatniego polecenia wewnątrz funkcji. Po zakończeniu funkcji parametry pozycyjne przyjmują poprzednie wartości.

WCZYTYWANIE DANYCH

Często potrzebne jest pobranie czegoś z klawiatury. Przydatne do tego jest polecenie **read**. Można je wykorzystać na kilka sposobów:

read	wczytuje linie ze standardowego wejścia i wysyła je do zmiennej $REPLY.
read ZMIENNA	wczytuje linie ze standardowego wejścia i wysyła je do zmiennej $ZMIENNA
read ZM1 ZM2 ZM2	wczytuje linie ze standardowego wejścia i wysyła je do kolejnych zmiennych

WYRAŻENIA ARYTMETYCZNE

Często zachodzi potrzeba wyliczenia wartości jakiegoś wyrażenia, np. 2*6. Można to zrobić na 2 sposoby: albo korzystając z mechanizmów basha albo z zewnętrznego polecenia. Mechanizm basha wygląda następująco:

- **$((wyrażenie))**
- **$[wyrażenie]**

np.: **ZM=$((2+2))**

Programem, który służy do obliczania wartości wyrażenia jest **expr**. Pobiera on jako argumenty wyrażenie do obliczenia. Każda liczba i każdy znak musi być oddzielony spacją. Przypisać wynik wyrażenia do zmiennej można tak jak przypisywanie do zmiennej wyniku działania polecenia:

- **ZMIENNA=`expr 2 \* 2` (spacje!!!)**

Kolejnym sposobem jest przypisanie za pomocą **let**:

- **let** *zmienna*=1+4

OPERATORY ARYTMETYCZNE

Shell bash używa operatorów arytmetycznych z języka C. Poniższy wykaz jest uporządkowany w kolejności wykonywania operacji – od wykonywanych jako pierwsze. Zmianę kolejności można wymusić stosując nawiasy okrągłe.

-	minus jednoargumentowy
!~	logiczna negacja; inwersja binarna (uzupełniająca jedynki)
* / %	mnożenie, dzielenie, funkcja modulo
+ -	dodawanie, odejmowanie
<< >>	przesunięcie bitowe w lewo, w prawo
== !=	równość, nierówność
<= >=	mniejsze lub równe, większe lub równe
&	bitowe AND (i)
^	bitowe wyłączne OR (lub)
|	bitowe OR (lub)
&&	logiczne AND (i)

OPERATORY ARYTMETYCZNE cd.

||	logiczne OR (lub)
=	przypisanie wartości
+= -=	przypisanie po dodaniu, po odjęciu
*= /= %=	przypisanie po mnożeniu, po dzieleniu, po funkcji modulo
&= ^= |=	przypisanie po bitowych AND, XOR, OR
<<= >>=	przypisanie po bitowym przesunięciu w lewo, w prawo

exit n – zakończenie skryptu z kodem wy **n**

domyślnie kod ostatniego polecenia skryptu

0 – sukces

1-125 –kody błędów; do wykorzystania

126 –plik nie był wykonywalny

128 i > -pojawił się sygnał

```
if [ -f .profil ]; then
exit 0
fi
exit 1
```

```
[ -f .profil ] && exit 0 || exit 1
```

printf „format” par1 par2…

%d – l. dzies.,
%c - znak,
%s –ciąg
printf ”%s %d\t%s\n” ”wykład trwa” 90 minut

set –

```
echo data: $(date)
set $(date)
printf "%s %s\n" "dzien tygodnia to" $1
```

shift - przesuwa zmienne parametryczne o 1 pozycję

```
while [ "$1" -ne "" ]; do
echo "$1"
shift
done
exit 0
```

trap – przechwytywanie sygnałów

```
trap 'rm /tmp/tplik$$'  INT
date > /tmp/tplik$$
echo "nacisnij ctrl+c"
while [ -f /tmp/tplik$$ ]; do
echo plik istnieje przed ctrl+c
sleep 1
done
trap - INT
date > /tmp/tplik$$
echo "ctrl+c"
while [ -f /tmp/tplik$$ ]; do
echo plik istnieje po ctrl+c
sleep 1
done
```

Rozwinięcia parametryczne

```
for i in 1 2 3 4 5
do
   echo plik${i}.c
   rm -i plik${i}.c
done
```

${#par} – podaje długość par

${par%tekst} – od końca usuwa najmniejszą część par pasującą do tekst

${par%%tekst} – od końca usuwa najdłuższą część par pasującą do tekst

${par#tekst} – od początku usuwa najmniejszą część par pasującą do tekst

${par##tekst} – od początku usuwa najdłuższą część par do pasującą do tekst

${par:-wartość} – jeśli par ma wartość pustą staje się równe wartość

```
aa=/home/stud/elektr/gr1/marek/elektr/dok
echo $aa
echo ${aa%elektr*}
echo ${aa%%elektr*}
echo ${aa#*/}
echo ${aa##*/}

/home/stud/elektr/gr1/marek/elektr/dok
/home/stud/elektr/gr1/marek/
/home/stud/
home/stud/elektr/gr1/marek/elektr/dok
dok
```

Dokumenty miejscowe <<

cat << KONIEC!!!

To jest

Tekst, który

Zostanie wypisany

KONIEC!!!

DEBUGGOWANIE

sh –n skrypt	set –o noexec	jedynie szukanie
	set –n	błędów składni
sh –v	set –o verbose	powtarza
	set –v	polecenie przed wykonaniem
sh –x	set –o xtrace	powtarza
	set -x	polecenie po wykonaniu

PRZYKŁADY

```
#!/bin/bash
if [ 2 -lt 1 ]
  then
        echo OK
  else
        echo NOK
fi
```

Ten skrypt wypisze na konsoli tekst NOK (ponieważ 2 nie jest mniejsze od 1, zostanie wykonana instrukcja po **else**).

```
#!/bin/bash
i=0
while [ $i -lt 5 ]
do
 echo $i
 let i=$i+1
done
```

Ten skrypt wypisze na konsoli w kolejnych wierszach cyfry od 0 do 4.

TCSH

ZMIENNE

.

Zmienne zwykłe powłoki - polecenie **set**

set zmienna=wartosc

set zmienna = wartosc

Uwaga!

Spacje występują po obu stronach znaku „=” lub nie występują w ogóle.

usunięcie definicji zmiennej - polecenie **unset**

Podobnie jak w BASH występują zmienne o charakterze przełączników:

ignoreeof, **noclobber**, **noglob**

Dodatkowo można ustawiać następujące zmienne:

echo – sterującą wyświetlaniem (lub nie) poleceń przed ich wykonaniem,

notify – sterującą informowaniem o zakończeniu procesów tła,

verbose – sterującą wyświetlaniem polecenia po odwołaniu się do historii.

Zmienne te ustawia się poleceniem **set nazwa**.

set ignoreeof

set notify

unset verbose

Zmienne specjalne powłoki TCSH:

prompt –umożliwia definiowanie monitu

prompt2 –definiuje monit kolejnej linii polecenia

history –określa ilość pamiętanych zdarzeń historii,

savehist –określa liczbę zdarzeń, które zostaną zapisane i przechowane w pliku .history do następnego zalogowania,

path, cdpath –mają takie samo znaczenie jak PATH i CDPATH w powłoce BASH; lista katalogów objęta jest nawiasami, a poszczególne pozycje oddzielone są spacjami,

user –nazwa użytkownika,

shell –powłoka logowania,

term –terminal,

mail –tablica, której elementy oddzielone są spacjami, łącząca funkcje zmiennych MAIL, MAILCHECK i MAILPATH z powłoki BASH.

definiowane zmiennych środowiskowych - **setenv (**bez znaku równości)

Ich znaczenie i zasięg jest analogiczny jak exportowanych zmiennych powłoki BASH.

Przykład

setenv PATH ($PATH /home/abc)

Powłoka zachowuje zgodność zmiennych środowiskowych: **HOME**, **USER**, **TERM**, **PATH** ze zdefiniowanymi zmiennymi zwykłymi: **home**, **user**, **term**, **path**.

set - wyświetla zdefiniowane zmienne zwykłe

setenv - pokazuje listę zdefiniowanych zmiennych środowiskowych

prompt, prompt2

%M- pełna nazwa hosta

%m- nazwa hosta do pierwszej ‘.’

%n- nazwa użytkownika

%/- nazwa bieżącego katalogu

%t- czas w formacie gg:mm

%h,%!,!- bieżący nr. polecenia w historii

%d- dzień tygodnia (w formacie Mon)

%w- nazwa bieżącego miesiąca (Jan)

%y- bieżący rok

Historia

history

!n

!polecenie

!?tekst?

!-n

!{ls} –la –zostanie powtórzone ostatnio wydane polecenie **ls** z dodaniem opcji **la**,

!{cd} / –jeśli ostatnie polecenie **cd** było bez parametrów, wykonane zostanie **cd /**

!{c} / –wykonane zostanie ostatnio wydane polecenie rozpoczynające się na **c** z parametrem **/**

!{2}1 –zostanie wykonane polecenie będące złożeniem polecenia drugiego historii i symbolu „1”,

Parametry ostatnio wydanego polecenia - **!***

historia

Wybór słowa zdarzenia

!2:0

!7:0-2

!4:^ drugie słowo

!5:$ ostatnie słowo

!3:^-$

!2:* wszystkie argumenty

! – maskowany \

aliasy

alias ‘tekst polecenia’

unalias polecenie

Przykłady:

alias dir ‘ls’

alias rm ‘rm –i’

alias sdir ‘ls –la|sort’

W tym przypadku wywołanie polecenia „**sdir /**” nie dałoby zamierzonego efektu. Aby uzyskać posortowaną listę zawartości katalogu głównego, należy alias zdefiniować w sposób następujący:

alias sdir ‘ls-l \!*|sort’

a następnie skorzystać z niego: **sdir /**

Pliki inicjacyjne

.login

.logout

.tcshrc

.login - wykonywany przy każdym logowaniu użytkownika

.logout - przy wylogowywaniu się,

.tcshrc -uruchamiany jest przy każdym wywołaniu powłoki TCSH

Skrypty TCSH

#

$#argv	Liczba argumentów pozycyjnych
$0	Nazwa programu shellowego
$argv[n]	Argumenty pozycyjne przekazane do shella
$*	Rozwijanie do „ $1$2....”
$argv[*]	Rozwijanie do „$1” „$2”....
$?	Kod powrotu ostatniego polecenia
$$	Numer procesu bieżącego shella
$!	Numer procesu ostatniego zadania wsadowego

Zmienne numeryczne @

```
set a=0
set b=0
 @ a++
 @ a+=4
@ b=$a * 4
```

Każde działanie w wyrażeniu, w którym występują inne zmienne musi być oddzielone spacją.

Polecenie *set* połączone z operacją przekierowania **$<** odczytuje dane ze standardowego wejścia

set powitanie = $<

Operatory wyrażeń

==	identyczność ciągów
!=	nieidentyczność ciągów
=~	identyczność ciągu ze wzorcem
!~	nieidentyczność ciągu ze wzorcem
&&	logiczne AND
||	logiczne OR
!	logiczne NOT
-e	plik istnieje
-r	plik jest do odczytu
-w, -x, -d, -f, -o	
-z	plik jest pusty

if (*warunek*) **then**
lista
endif

if (*warunek*) **then**
lista 1
else
lista2
endif

if (*warunek1*) **then**
lista1
else if (*warunek2*) **then**
lista2
else if
lista3
endif

foreach *zmienna (wart1 wart2...)*
czynność
end
while *(warunek)*
czynność
end

switch ($*zmienna*)
case *wartość1*
czynność1;
breaksw;

case *wartość2*
czynność2;
breaksw;

default
czynność_domyślna;
breaksw;
endsw

Skrypt listujący wszystkie pliki źródłowe języka C z bieżącego katalogu wraz z ich nazwami

BASH:

```
for  plik in *.c
do echo plik $plik
cat $plik|more
done
```

TCSH

```
#
foreach plik (*.c)
        echo plik $plik
        cat $plik|more
end
```

Skrypt sprawdzający, który z dwóch plików jest dłuższy

BASH:

```
if [ $# -ne 2 ]
then
echo blad: zla liczba parametrow
exit
fi
echo liczba argumentow=$#
if [ ! –f $1 ]
then
        echo blad: nie ma pliku $1
        exit
fi
if [ ! –f $2 ]
then
        echo blad: nie ma pliku $2
        exit
fi
d1=`cat $1|wc –c`
d2=`cat $2|wc –c`
echo $d1
echo $d2
if [ $d1 –gt $d2 ]
then
        echo „wiekszy jest plik $1”
else
        echo „wiekszy jest plik $2”
fi
```

TCSH

```
#
if ($#argv !=  2) then
      echo zla liczba parametrów
      exit
endif
echo liczba parametrów = $#argv
if (! -e $1) then
      echo Nie ma pliku $1
      exit
endif
if (! -e $2) then
      echo Nie ma pliku $2
      exit
endif
set d1=`wc -c  $1`
set d2=`wc -c $2`
echo $d1
echo $d2
if ($d1[1] > $d2[1]) then
echo Plik $1 jest dłuższy
else
echo Plik $2 jest dłuższy
endif
```