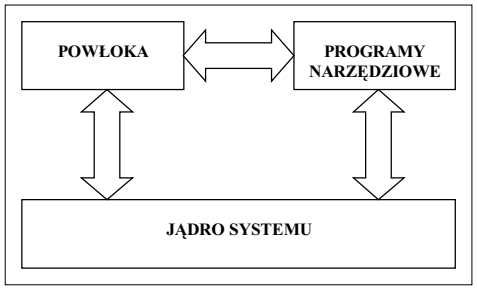
6. KONFIGURACJA POWŁOKI SYSTEMU LINUX

6.1. Zależności między powłoką, jądrem systemu a programami

Do podstawowych elementów systemu operacyjnego Linux należy:

- *powłoka* (shell, interpreter poleceń) - program pełniący funkcję interfejsu pomiędzy użytkownikiem i jądrem systemu, interpretujący polecenia oraz umożliwiający uruchamianie programów. Najczęściej wykorzystywaną powłoką w Linuksie jest Bash (*Bourne Again SHell*).
- *jądro systemu* - zbiór programów zarządzający procesami i zasobami systemu,
- programy narzędziowe.



Rys. 12. Ogólna struktura systemu operacyjnego Linux

Jednym ze sposobów komunikacji pomiędzy wymienionymi elementami systemu operacyjnego jest wykorzystanie zmiennych powłoki.

6.2. Zmienne powłoki

Zmienna to cecha posiadająca swoją nazwę i przyjmująca pewną wartość.

Polecenia wykorzystywane podczas pracy ze zmiennymi:

- *set* - wyświetlenie zmiennych powłoki (informacje wyświetlane są w postaci *nazwa_zmiennej=wartość_zmiennej*), np.

```
$ set
HOME=/home/nowakj
PWD=/home/nowakj/public_html
USER=nowakj
```

- *echo* - polecenie można wykorzystać do wyświetlenia wartości jednej zmiennej. Nazwę zmiennej należy poprzedzić znakiem dolara (np. $HOME, $USER), np.

```
$ echo ”informatyka”
informatyka

$ echo HOME
HOME

$ echo $HOME
/home/nowakj

$ echo $SHELL
/bin/bash
```

- operator „=” - zdefiniowanie nowej zmiennej lub zmiana wartości zmiennej już istniejącej (po obu stronach znaku równości nie wolno zamieszczać spacji), np.

```
$ UCZELNIA=”Uniwersytet Jagielloński”
$ UCZELNIA=”Uniwersytet Ekonomiczny”
$ UCZELNIA=”$UCZELNIA w Krakowie”
$ echo $UCZELNIA
Uniwersytet Ekonomiczny w Krakowie
```

- pojedyncze cudzysłowy (apostrofy: '...') maskują nazwy zmiennych i sprawiają, że nie pojawiają się ich wartości, np.

```
$ UCZELNIA="UEK w Krakowie"
$ echo 'Studiuje w $UCZELNIA'
```

- da wynik: Studiuje w $UCZELNIA

- podwójne cudzysłowy ("...") nie maskują nazw zmiennych, np.

```
$ UCZELNIA="UEK w Krakowie"
$ echo "Studiuje w $UCZELNIA"
```

- da wynik: Studiuje w UEK w Krakowie

- odwrotne cudzysłowy (`...`) przypisują zmiennym wynik polecenia (polecenie umieszczone w odwrotnych cudzysłowach jest wykonywane), np.

```
$ zmienna=`ls *.txt`
```

- polecenie *unset* - usuwanie zmiennych powłoki, np.

```
$ unset UCZELNIA
```

6.3. Przykładowe zmienne powłoki systemowej

Spośród wielu zmiennych powłoki, najczęściej wykorzystywane lub modyfikowane przez użytkowników są, zmienne:

- PS1 - zmienna przechowuje definicję tzw. *znaku zachęty* (monitu systemu), np.

```
$ PS1="Podaj polecenie:"
Podaj polecenie:
```

Użytkownik definiując znak zachęty może w nim zamieszczać specjalne kody, które są następnie zamieniane na odpowiednie wartości, np.

```
$ PS1="\h: \w \$"
```


Tabela 11. Kody wykorzystywane przy definiowaniu znaku zachęty

Kod	Znaczenie
\!	numer polecenia
\$	znak dolara ($) dla zwykłego użytkownika, dla użytkownika root znak hash (#)
\d	aktualna data
\s	nazwa powłoki
\t	aktualny czas
\u	nazwa użytkownika
\w	katalog bieżący
\h	nazwa komputera

- HOME - zmienna przechowuje ścieżkę dostępu do katalogu domowego użytkownika, np.

```
$ echo $HOME
/home/nowakj
```

- PATH - zmienna przechowuje tzw. *ścieżki poszukiwań*. Dwukropek znajdujący się na końcu zmiennej PATH oznacza, że przeszukiwany będzie także katalog bieżący.

```
$ echo $PATH
/usr/bin:/usr/sbin
```

Tabela 12. Zawartość zmiennej PATH a przeszukiwane katalogi

Zmienna PATH	Przeszukiwane katalogi
/usr/bin:/usr/sbin	/usr/bin /usr/sbin
/usr/bin:/usr/sbin:	/usr/bin /usr/sbin katalog bieżący

Nowe ścieżki dostępu do zmiennej PATH można dodać np. w ten sposób:

```
$ PATH=$PATH:$HOME/programy
```

- SHELL - zmienna przechowuje nazwę programu będącego interpreterem poleceń, np.

```
$ echo $SHELL
/bin/bash
```

- USER - zmienna przechowuje nazwę użytkownika, np.

```
$ echo $USER
nowakj
```

6.4. Uruchamianie kolejnych powłok

Użytkownik w systemie Linux ma możliwość uruchamiania kolejnych powłok, tzw. *powłok potomnych*, służy do tego polecenie *bash*. Do zamknięcia otwartej powłoki potomnej służy polecenie *exit*.

Przykładowo:

```
$ ...
   praca z pierwszą powłoką
$ bash
   uruchomienie drugiej powłoki
$ ...
   praca z drugą powłoką
$ exit
   zamknięcie drugiej powłoki
$ ...
   praca z pierwszą powłoką
```


Powłoka 1

```
$ ls
$ cp plik test.txt
$ bash
```

Powłoka 2

```
$ finger
$ bash
```

Powłoka 3

```
$ pwd
$ man cat
$ exit
```

```
$ echo $HOME
$ exit
$ vi test.txt
$ logout
```

Rys. 13. Praca z kilkoma powłokami

6.5. Lokalne i globalne zmienne systemowe

- *zmienna lokalna* to zmienna dostępna tylko w bieżącej powłoce,
- *zmienna globalna* to zmienna widoczna w bieżącej powłoce (powłoce macierzystej) i wszystkich powłokach potomnych. Zmienną globalną tworzy się za pomocą polecenia *export*, które może być wykorzystane przy tworzeniu nowej zmiennej lub do wyeksportowania zmiennej już istniejącej. Zmienne utworzone (wyeksportowane) w powłoce potomnej nie są widoczne w powłoce macierzystej.

Powłoka 1

```
$ UCZELNIA="UEK"
$ echo $UCZELNIA
UEK
$ bash
```

Powłoka 2

```
$ echo $UCZELNIA
```

zmienna UCZELNIA utworzona w Powłoce 1 nie została wyeksportowana, a więc nie jest widoczna w Powłoce 2

```
$ exit
$ export UCZELNIA="UEK"
$ echo $UCZELNIA
UEK
$ MIASTO="Kraków"
$ export MIASTO
$ echo $MIASTO
Kraków
$ bash
```

Powłoka 2

```
$ echo $UCZELNIA
AE
```

zmienna UCZELNIA wyeksportowana z Powłoki 1 jest widoczna w Powłoce 2

```
$ echo $MIASTO
Kraków
$ MIASTO="Gdynia"
$ export STOLICA="Warszawa"
$ exit
$ echo $MIASTO
Kraków
```

zmiana w Powłoce 2 wartości zmiennej MIASTO nie jest widoczna w Powłoce 1

```
$ echo $STOLICA
```

zmienna STOLICA utworzona w Powłoce 2 nie jest widoczna w Powłoce 1

```
$ logout
```

Rys. 14. Zmienne lokalne i globalne w powłokach

6.6. Pliki startowe

Użytkownik ma możliwość skonfigurowania systemu, w taki sposób, aby przy każdym logowaniu, uruchamianiu nowej powłoki lub kończeniu pracy, system automatycznie wykonywał polecenia zawarte w tzw. *plikach startowych*. Mechanizm ten jest wykorzystywany do:

- tworzenia lub modyfikowania zmiennych systemowych,
- tworzenia nowych nazw poleceń (aliasów),

- uruchamiania lub zamykania programów,
- wyświetlania komunikatów powitalnych lub pożegnalnych.

Wykorzystywane pliki startowe, to:

- *.bash_profile* - polecenia zawarte w tym pliku są wykonywane każdorazowo przy logowaniu się użytkownika,
- *.bashrc* - polecenia zawarte w tym pliku są wykonywane każdorazowo przy uruchamianiu nowej powłoki,
- *.bash_logout* - polecenia zawarte w tym pliku są wykonywane każdorazowo przy kończeniu pracy przez użytkownika.

6.7. Aliasy

Użytkownik ma możliwość definiowania nowych nazw dla poleceń, tzw. *aliasów*. Służy do tego polecenie *alias*, a składnia jego jest następująca:

```
$ alias nazwa=polecenie
```

lub

```
$ alias nazwa='polecenie'
```

Polecenie *alias* bez żadnych argumentów i opcji wyświetla wszystkie zdefiniowane przez użytkownika aliasy.

Możliwość definiowania aliasów jest przydatna kiedy użytkownik chce nadać np. łatwiejszą do zapamiętania nazwę jakiemuś poleceniu lub chce w skróconej postaci zapisać polecenie wraz z jego opcjami i argumentami.

Do usuwania zdefiniowanych aliasów służy polecenie *unalias*.

Przykładowe definicje i usuwanie aliasów:

```
$ alias katalog=ls
$ alias katalog='ls -al'
$ alias ls='ls -al'
$ unalias katalog
$ unalias ls
```

Jeżeli aliasy mają być dostępne w każdej nowej sesji użytkownika, to należy je zdefiniować w pliku *.bash_profile*.

6.8. Zmienne tablicowe

W powłoce bash można tworzyć jednowymiarowe **zmienne tablicowe**:

- elementy tablicy są indeksowane liczbami całkowitymi (od zera); indeksowanie nie musi być ciągłe (tzn. np. może być element o indeksie równym 5 a następny o indeksie równym 9),
- **tablica=(wartość1 wartość2 ... wartośćN)** - utworzenie zmiennej tablicowej, np.

```
$ kolory=(czerwony zielony niebieski)
```

  - *uwaga: pomiędzy elementami tablicy mają być tylko spacje.*

- **tablica[indeks]=wartość** - dodanie elementu o podanym *indeksie* do zmiennej *tablica*, np.

```
$ kolory=(czerwony zielony niebieski)
```

- **${tablica[indeks]}** - odwołanie się do wartości elementu o podanym *indeksie* ze zmiennej *tablica*, np.

```
$ echo "Moj ulubiony kolor: ${kolory[2]}"
$ Moj ulubiony kolor: niebieski
```


- **${#tablica[indeks]}** - odwołanie się do **długości** elementu o podanym *indeksie* ze zmiennej *tablica*, np.

```
$ echo "Nazwa tego koloru ma ${#kolory[2]} liter"
$ Nazwa tego koloru ma 9 liter
```

- **${tablica[*]}** lub **${tablica[@]}**- odwołanie się do **wszystkich** elementów ze zmiennej *tablica*, np.

```
$ echo "Moje kolory: ${kolory[*]}"
$ Moje kolory: czerwony zielony niebieski
```

- **${#tablica[*]}** lub **${#tablica[@]}**- odwołanie się do **liczby elementów** ze zmiennej *tablica*, np.

```
$ echo "Liczba elementów tablicy: ${#kolory[*]}"
$ Liczba elementów tablicy: 3
```

- **unset tablica[indeks]** - usunięcie elementu o podanym *indeksie* ze zmiennej *tablica*, np.

```
$ unset kolory[2]
```

- **unset tablica[*]** lub **unset tablica[@]** lub **unset tablica[]**- usunięcie całej zmiennej *tablica*, np.

```
$ unset kolory[*]
```

7. SKRYPTY POWŁOKI – POLECENIA I STEROWANIE

7.1. Polecenia i struktury sterujące w skryptach

- **read** - odczytanie wiersza ze standardowego wejścia i przypisanie go zmiennej, np.

```
echo -n "Podaj swoje imie: "
read zmienna
```

- **test** lub **[ ]**- porównanie wartości dwóch argumentów

```
test 2 -eq „$liczba”
[ 2 -eq „$liczba” ]
```

- sprawdzenie czy w zmiennej $liczba *jest wartość* 2.

```
[ „$zmienna” = „abc” ]
```

- uwaga: nie wolno zapomnieć o znakach spacji znajdujących się po obu stronach znaku „=” oraz po obu stronach nawiasów kwadratowych.

Do polecenia *test* i *[ ]* można użyć:

- operatory porównania stałych: „-gt” - większe niż; „-lt” - mniejsze niż; „-ge” - większe lub równe; „-le” - mniejsze lub równe; „-eq” - równość; „-ne” - nierówność.
- porównania napisów: „-z” - testowanie pustego ciągu; „-n” - testowanie wartości napisu, „=” - identyczność napisów; „!=” - nieidentyczność napisów; „str” - sprawdzenie czy ciąg znaków nie jest ciągiem pustym.
- operatory logiczne: „-a” - operator AND; „-o” - operator OR; „!” - operator NOT.

- operatory testowania plików: „-f” - plik istnieje i jest zwykłym plikiem; „-s” - plik nie jest pusty; „-r” - możliwe jest odczytanie pliku; „-w” - możliwe jest zapisanie do pliku; „-x” - możliwe jest wykonanie pliku; „-d” - nazwa pliku jest nazwą katalogu.

- **let** - instrukcja służy do wykonywania działań matematycznych. Mogą być wykorzystane następujące operatory matematyczne: *, /, +, -, % (modulo); operatory porównania: >, <, >=, <=, = (przypisanie), == (porównanie), != (nierówność), & (AND), | (OR), ! (NOT), np.

```
$ let „a=2*5”
$ echo $a
$ 10

$ let „a=2>=5”
$ echo $a
$ 0
```

- **$((*wyrażenie*))** - wykonywanie działań matematycznych, np.

```
$ echo $((55-44))
```

- **if ... then ... fi, if … then … else … fi** - funkcja warunkowa, np.

```
if [ "$plik" = "" ]
then
    plik="dane.txt"
fi
```

- **until … do … done** - pętla until działa dopóki polecenie testujące jest fałszywe, np.

```
until test -r "$plik"
```

```
do
    echo "Plik \"$plik\" nie istnieje"
    echo -n "Podaj nazwę pliku: "
    read plik
done
```

- **while ... do ... done** - pętla while działa dopóki polecenie testujące jest prawdziwe, np.

```
  while let "licznik<=liczbawierszy+1"
do
...
done
```

- **for *zmienna* in *lista* … do ... done** - pętla for wykonuje polecenia kolejno dla wartości podanych na liście, np.

```
for zmienna in Monika Ewa Maria
do
  echo $zmienna
done
```

- **for ((*wyrażenie1; warunek; wyrażenie2*)) do ... done** - pętla for wykonuje *wyrażenie1*, następnie sprawdza *warunek*, jeżeli jest prawdziwy wykonuje blok poleceń i oblicza *wyrażenie2*, np.

```
for ((I=10; I<=10; I++))
do
  echo $I
done
```


- **continue** - powoduje przejście do następnej iteracji pętli**, break -** powoduje przerwanie wykonywania pętli.

- **case *tekst* in *wzorzec1*) … ;; *wzorzec2*) … ;; esac** - struktura sterująca case dopasowuje wartość *tekst* do jednego z kilku *wzorców*. Jeżeli wzorzec pasuje, wykonywane jest skojarzone z nim polecenie (polecenia), np.

```
case „$zmienna” in
„start”)
  echo „Start Programu”
  ;;
„stop”)
  echo „Koniec Programu”
  ;;
*)
  echo „Kontynuacja Programu”
  ;;
esac
```

- **function *nazwa_funkcji ()*** {**... return *wartość***} - tworzenie funkcji, np.

```
function przedstawSie ()
{
  echo “Nazywam sie: $1 $2”
  echo “Numer albumu: $3”
  return 0
}

przedstawSie Jan Kowalski 153445
```

- *wywołanie funkcji w skrypcie.*

7.2. Wyrażenia regularne

Wyrażenia regularne (ang. *regular expressions - regex*):

- są to wzorce, które opisują ciągi (łańcuchy) znaków,
- mogą określać zbiór pasujących łańcuchów, mogą również wyszczególniać istotne części łańcucha,
- stanowią integralną część narzędzi systemowych takich jak sed, grep, edytorów tekstu (np. vim), języków programowania przetwarzających tekst (np. awk, perl).

- Polecenie *grep* z parametrem *-E* daje możliwość rozszerzonego interpretowania wyrażeń regularnych.
- Podstawowe elementy (symbole) wyrażeń regularnych to:
  - „^" - początek ciągu znaków,
  - „$" – koniec ciągu znaków,
  - „." – dowolny znak,
  - [xyz] – dowolny znak spośród zamieszczonych na liście („x", „y", „z"),
  - [x-y] – dowolny znak z przedziału od x do y,
  - [[:alnum:]], [[:alpha:]], [[:digit:]], [[:lower:]], [[:upper:]] – dowolny jeden znak z podanej klasy (odpowiednio: znaki alfanumeryczne, litery, cyfry, małe litery, duże litery),

  Uwaga: znak „^" podany w zakresie neguje jego występowanie – np. [^1-6] oznacza, że znak nie może być cyfrą z przedziału 1 – 6.
  - „{n,m}" – poprzedzający element powtarza się od n do m razy,
  - „{n}" – poprzedzający element powtarza się n razy,

  - „{n,}" – poprzedzający element powtarza się od n lub więcej razy,
  - „*" – poprzedzający element (znak, wzorzec) występuje zero lub więcej razy,
  - „+" – poprzedzający element występuje przynajmniej raz,
  - „?" – poprzedzający element występuje zero razy lub jeden raz.

Przykład

```
$ cat plik.txt | grep –E
„^K[RA]{1,2}[[:digit:]]{4,5},.{3,20}$"
```

Poszukiwany wiersz tekstu zaczyna się od dużej litery „A" następnie jest jedna lub dwie duże litery spośród dostępnych „R" i „A", następnie jest 4 lub 5 cyfr, następnie przecinek „," i dowolny znak występujący od 3 do 20 razy. Znak „$" oznacza koniec wiersza.

Przykładowe wiersze tekstu odpowiadające wzorcowi to:

„KR23456, Fiat Punto"

„KRA9832, Ford Focus"

„KAA1234, błędna rejestracja"

- Podstawowe informacje o wyrażeniach regularnych można znaleźć w opisie polecenia *grep* (*man grep, http://www.digipedia.pl/man/doc/view/grep.1/*).