UNIX

SCO Unix

QNX

Be

SCO

SUN MICROSYSTEMS
SOLARIS™

SOLARIS™

Historia

- Historia UNIX'a jest długa i sięga czasów, gdy Ken Thompson, Denis Ritchie i inni zaczynali pracę minikomputerze PDP – 7 w roku 1969.
- Od tego czasu Unix pojawiał się w wielu wcieleniach, jako wersja handlowa, wariant akademicki czy też podstawa wysiłków normalizacyjnych.
  - Sixth Editio lub Version 6 (1975) - pierwsza szeroko dostępna wersja, przynajmniej w społeczności technicznej, i podstawa dla pierwszego unika Berkeley.
  - Xenix (1980). - wersja Microsoftu
  - System V (1983 - 1992) - jedna z najbardziej ekspansywnych wersji Unika z AT&T, twórcy systemu Unix.

Historia

- Berkeley Unix (wersja 4.2 w 1984 roku, 4.4 w 1993 roku) - opracowana na Uniwersytecie Berkeley; jedna z najważniejszych wersji Unika, z wieloma dodatkowymi cechami.
- Posix (1988 i dalej) - początek zestawu standardów zgodnych z IEEE.
- X/Open Portability Guides (XPG3 w 1990 roku, XPG4.2 w 1994 roku) - specyfikacja praktyczna, jednocząca pewną liczbę podstawowych standardów i praktyki przemysłowej (X/Open ostatecznie zdobył znak towarowy Uniksa).

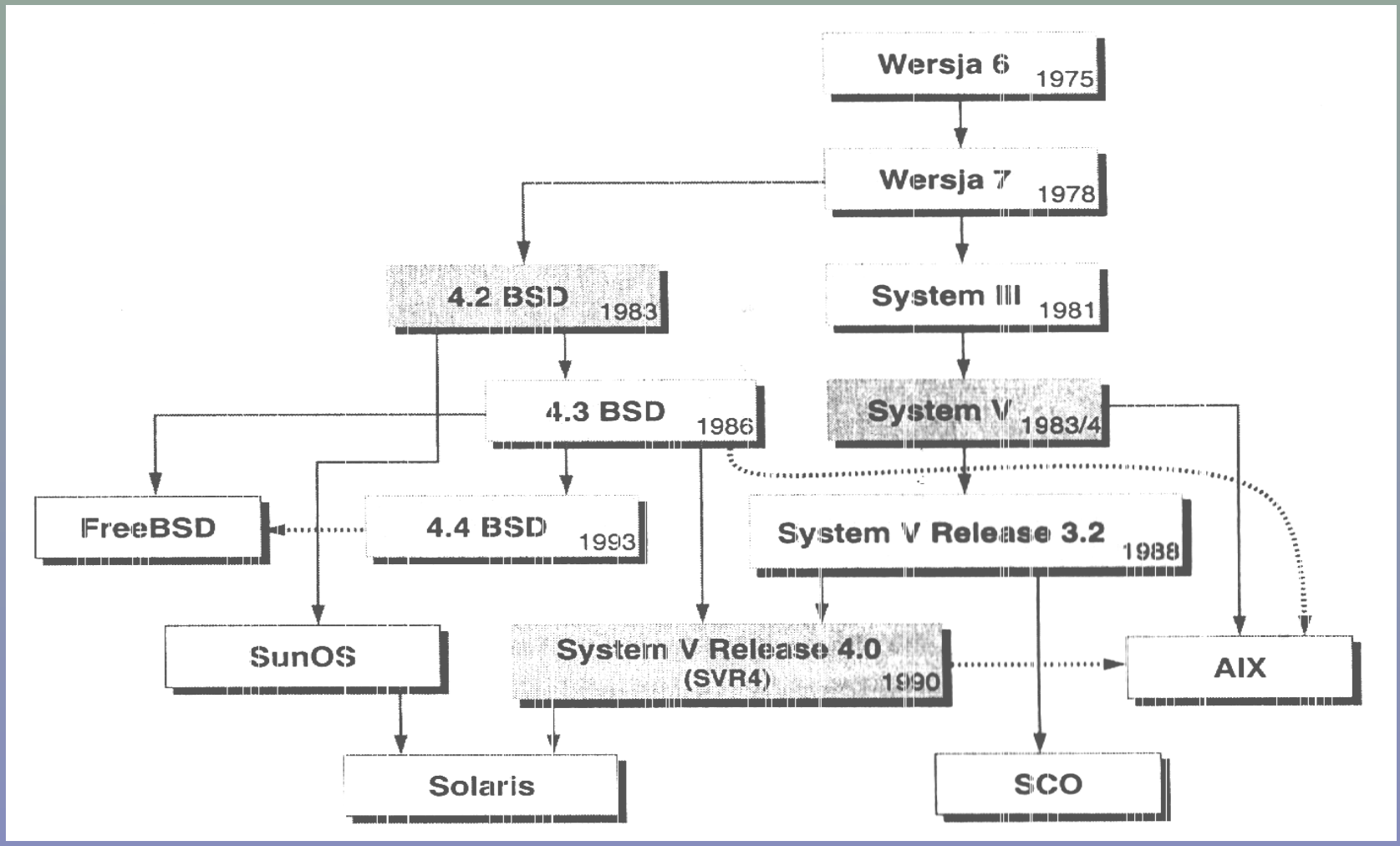
Przeznaczenie systemu

- System operacyjny, przeznaczony głównie dla komputerów pełniących rolę serwerów.
- Pomimo że powstał pod koniec lat 60. (1969), nadal jest jednym z częściej spotykanych w świecie systemów operacyjnych.
- Większość producentów superkomputerów dołącza do nich własną wersję tego systemu
- UNIX stanowi obecnie najbardziej popularne środowisko pracy szczególnie w zastosowaniach sieciowych gdyż cechuje go duża stabilność oraz bezpieczeństwo systemu przed nieautoryzowanym dostępem.

Najbardziej znane odmiany

- **AIX (**IBM**),**
- **A/UX (**Apple**),**
- **BSD (**Uniwersytet Kalifornijski w Berkley**),**
- **Linux,**
- **GNU (**Free Software Foundation**),**
- **HP/UX (**Hawlett-Packard**),**
- **SunOS (**Sun Microsystem**),**
- **Ultrix (**DEC**),**
- **Unicos (**Cray Corporation**),**
- **UNIX (**AT&T, SCO, Sun Microsystems**),**
- **Xenix (**SCO**).**

Dystrybucje



UNIX a Windows

- W porównaniu z Windows Unix jest bardzo stabilnym i oferującym duże możliwości w zakresie komend systemem operacyjnym.
- Obsługa, praca i administracja systemu klasy Unix wymaga od użytkownika przynajmniej dobrej znajomości systemu, podczas gdy obsługa Windows – co najwyżej podstawowej.
- Z tego względu Unix powinien być przede wszystkim stosowany w środowiskach o dużym obciążeniu systemu, wymagających stabilnej pracy i wysokiego poziomu bezpieczeństwa.

Charakterystyka systemu

- **wielodostępność** - tzn. może nadzorować równoczesną pracę wielu użytkowników komputera (liczba użytkowników zależy głównie od mocy obliczeniowej sprzętu) .
  - Użytkownik jest rozpoznawany przez identyfikator swojego konta w systemie (*userid*).
  - Konto użytkownika jest chronione prywatnym hasłem (*password*).
- **wielozadaniowość** - tzn. każdy użytkownik może zlecać wykonanie wielu zadań równocześnie (dokładniej - z podziałem czasu).

Charakterystyka systemu

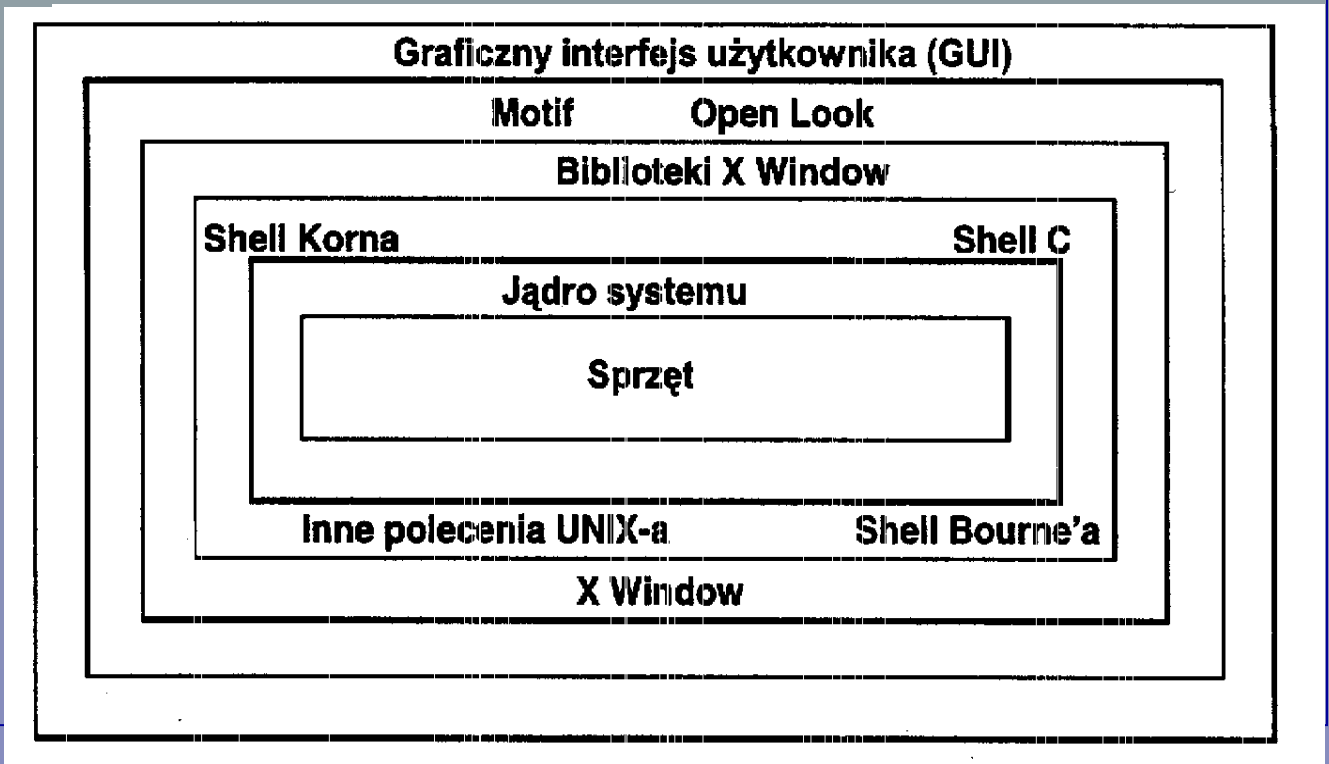
- **hierarchiczny, drzewiasty system plików**, z jednolitym potraktowaniem wszystkich typów plików,
- wykonywanie operacji wej/wyj niezależnie od typu urządzenia zewnętrznego,
- duża liczba programów narzędziowych, tj. kompilatorów języków programowania,
- duża liczba programów usługowych,
- przenaszalność oprogramowania dzięki zapisowi w C.

Charakterystyka systemu

- **Bezpieczeństwo systemu** wymaga aby mogli w nim pracować tylko zarejestrowani użytkownicy.
- Rejestracją użytkowników i ogólnym nadzorem nad systemem zajmuje się użytkownik ***root*** (administrator systemu).
- Jest on użytkownikiem mającym najszersze uprawnienia w systemie.

Architektura systemu

- Unix składa się z trzech komponentów, tj. :
  - » *kernela*,
  - » *shella*
  - » *systemu* plików



Architektura systemu

Kernel (jądro)

- Jądro systemu usytuowane jest nad sprzętem i ponosi odpowiedzialność za wszystkie jego działania.
- Jądro to zbiór najważniejszych podprogramów, zarządzających zasobami systemu.
- Zarządza pamięcią, systemem plików oraz programowymi i sprzętowymi urządzeniami.
- Zawiera też język niskiego poziomu, shell obsługujący (utrzymujący) procesy.
- Od chwili uruchomienia komputera jądro sprawuje nad nim pełną kontrolę.
- Użytkownik komunikuje się z jądrem przez powłokę.

Architektura systemu

- Wyróżnia się kilka metod konstrukcji jąder:
  - **jądro monolityczne** -_wszystkie zadania są wykonywane przez jądro, będące jednym, dużym programem działającym w trybie jadra
  - **mikrojadro** - w tej technice z monolitycznego jądra zostaje tylko jego podstawowa część, a części odpowiedzialne za bardziej wyrafinowane funkcje są wydzielone do funkcjonalnych bloków albo realizowane jako zwykłe procesy_w trybie użytkownika

Architektura systemu

- **nanokernel** - technika zbliżona do techniki mikrojądra, różnica w wielkości - nanokernel jest jeszcze mniejszy.
- **exokernel** - architektura będąca odmianą nanojądra. Cechą wyróżniającą jest możliwość zarządzania zasobami systemu przez nieuprzywilejowanego użytkownika, a rola jądra sprowadza się do zabezpieczania zasobów.
- **cachekernel** - w tej technice jądro systemu buforuje obiekty systemowe takie jak wątki czy przestrzenie adresowe tak jak sprzęt komputerowy buforuje pamięć.

Architektura systemu

Shell (powłoka)

- Interpreter poleceń użytkownika.
- Jest to język wysokiego poziomu, który może być używany do wykonywania komend systemu, lub też jako język programowania, do pisania skomplikowanych skryptów.
- Powłoka pośredniczy pomiędzy użytkownikiem a jądrem.
- Kiedy użytkownik rozpoczyna pracę (loguje się do sytemu podając poprawny identyfikator konta i hasło) system operacyjny uruchamia program powłoki, który pobiera polecenia do wykonania z klawiatury terminala użytkownika.

Architektura systemu

- Administrator przydziela każdemu użytkownikowi jego powłokę standardową (uruchamianą w chwili logowania się do systemu).
- Użytkownik może też uruchomić dowolną powłokę zainstalowaną w systemie.
- Najczęściej stosowanymi powłokami w różnych implementacjach systemu UNIX są:
  - powłoka Bourne'a (sh)
  - powłoka Korna (ksh)
  - powłoka C (csh)

  lub ich modyfikacje, np. powłoki: bash, zsh, tcsh.

Powłoka – podstawowe pojęcia

- Użytkownik może wprowadzać polecenia dla systemu operacyjnego na dwa sposoby:
  - wypisując je z klawiatury,
  - umieszczając ciąg poleceń w pliku tekstowym, a następnie używając nazwy tego pliku jako komendy (taki plik nazywany jest skryptem).
- Ciąg znaków wprowadzany przez użytkownika z klawiatury (lub linie skryptu) jest przez powłokę dzielony na słowa, tj. ciągi znaków rozdzielone którymś z następujących symboli:

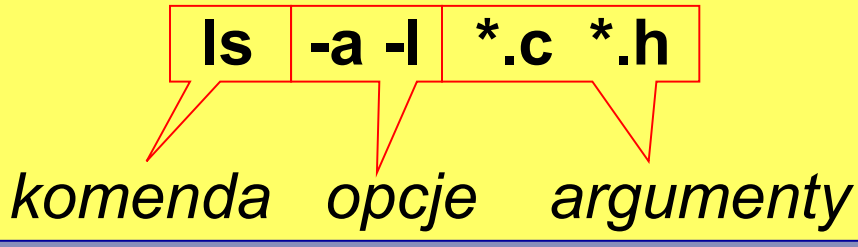
  ; & ( ) | < > nl (nowa linia) spacja (odstęp) TAB

  (powyższe symbole nazywa się metaznakami).

Powłoka – podstawowe pojęcia

- Pierwsze słowo w linii jest nazwą komendy, pozostałe słowa to parametry przekazywane komendzie, wśród których możemy wyróżnić:
  - opcje - zazwyczaj poprzedzane znakiem - (minus)
  - oraz argumenty.
- Niektóre komendy są wbudowane w powłokę (np. komenda *cd*), zazwyczaj jednak są to nazwy plików binarnych lub skryptów.

Przykład:



Powłoka – podstawowe pojęcia

- Linie rozpoczynające się znakiem # są liniami komentarza i są ignorowane przez powłokę.
- Komenda pobiera dane ze standardowego wejścia (*standard input*, w skrócie *stdin*), którym zazwyczaj jest klawiatura terminala.
- Wyniki działania komendy są wyprowadzane na standardowe *wyjście (standard output, stdout*), którym normalnie jest monitor terminala.
- Informacje o błędach są kierowane na standardowe wyjście diagnostyczne (*standard error, stderr*).

Powłoka – podstawowe pojęcia

- Kilka komend można połączyć w potok, rozdzielając je metaznakiem |,

Przykład: **ls -l | grep kowalski | more**

konstrukcja taka oznacza, że standardowe wyjście pierwszej komendy będzie standardowym wejściem drugiej komendy potoku, z kolei standardowe wyjście drugiej komendy będzie standardowym wejściem trzeciej komendy itd.

Powłoka – podstawowe pojęcia

- Ciąg potoków lub komend oddzielonych znakami ; (średnik) lub & (ampersand) tworzy listę komend.
  - Użycie jako separatora znaku średnika oznacza, że komenda występująca przed średnikiem musi się zakończyć zanim powłoka rozpocznie wykonywanie

Przykład:

```
# Zamiana nazw plików a i b:
cp a tmp ;  mv b a ;  mv tmp

# Uruchamia kompilację programu
# bigprog.c i  jednocześnie pozwala
# na edycję pliku sub.c.
cc bigprog.c & vi sub.c
```

Powłoka – podstawowe pojęcia

- Każda komenda zwraca pewna wartość całkowitoliczbową, która nazywamy statusem wyjścia komendy.
  - Wartość zero oznacza, że komenda zakończyła się sukcesem.
  - Wartość rożna od zera jest zwracana w przypadku błędów i sytuacji nietypowych.

Powłoka – zmienne powłoki

- Zmienne powłoki pozwalają przypisać nazwie symbolicznej pewien ciąg znaków lub liczbę całkowitą.
- Zmienną definiuje się instrukcją przypisania postaci :

zmienna=wartość

gdzie:

zmienna - jest nazwa zmiennej,

wartość - jest ciągiem znaków, który przypisuje się danej zmiennej.

Powłoka – zmienne powłoki

- $*zmienna* lub ${*zmienna*}
- -tej drugiej postaci używa się zazwyczaj wtedy, gdy powłoka może mieć problem z jednoznacznym określeniem nazwy zmiennej.
- Przykład1: *system*=UNIX
  echo To jest system $system
  # Zostanie wypisany tekst:
  To jest system UNIX
- Przykład 2: *tx* = *abc*
  echo ${txt}def
  # Jeśli nazwa zmiennej *txt* nie byłaby
  # zamknięta w nawiasy, to powłoka
  # szukałaby zmiennej *txtdef*

Powłoka – zmienne powłoki

- W powłoce, uruchamianej po zalogowaniu się użytkownika, jest zdefiniowanych wiele zmiennych (ich liczba i nazwy zależą od rodzaju powłoki).
- Ważniejsze zmienne predefiniowane przez system operacyjny lub powłokę to:

Zmienna	Przeznaczenie zmiennej
$	Numer procesu powłoki
PPID	Numer procesu rodzicielskiego
HOME	Prywatny katalog użytkownika
MAIL	Skrzynka pocztowa użytkownika
PWD	Katalog aktualny

Powłoka – zmienne powłoki

Zmienna	Przeznaczenie zmiennej
OLDPWD	Katalog sprzed ostatniej komendy cd
SHELL	Nazwa powłoki
RANDOM	Liczba losowa
PATH	Ścieżka przeszukiwań
SECONDS	Liczba sekund od rozpoczęcia pracy
PS1	Tekst wyświetlany jako znaki gotowości do pracy systemu
EDITOR	Edytor użytkownika
TERM	Nazwa terminala , z którego pracuje użytkownik

Powłoka – zmienne powłoki

- Zestaw komend można zapisać na dysku w postaci skryptu i wykonać go wypisując jego nazwę.
- Skrypt taki wykonuje się w osobnej kopii powłoki.
- Jeśli wywołując skrypt poda się w oprócz nazwy również parametry, to parametry te są dostępne wewnątrz skryptu jako zmienne o nazwach $1, $2, $3, ... $*n*.
- Zmienne te noszą nazwę zmiennych pozycyjnych.
- Liczba *n* zmiennych pozycyjnych jest dostępna jako zmienna $#.
- Zmienne powłoki mogą być lokalne, czyli obowiązujące tylko w danej powłoce, lub środowiskowe (eksportowane).

Powłoka – zmienne powłoki

- Zmienne środowiskowe obowiązują również w powłokach potomnych, tj. uruchomionych z danej powłoki (np. przez wykonanie skryptu lub jawne

Przykład:

```
aa=XXX
bb=YYY
export aa
ksh
echo $aa $bb
#  Zostanie wypisany tekst:  XXX (zmienna bb nie
# została wyeksportowana i nie jest dostępna w
# powłoce  potomnej)
```

Powłoka – substytucje

- Polecenia wypisywane przez użytkownika z klawiatury lub programy (skrypty) w języku powłoki przed próbą wykonania mogą być przez powłokę modyfikowane.
- Jedna z takich modyfikacji została już opisana powyżej: ciągi znaków poprzedzone znakiem $ są interpretowane jako zmienne powłoki i zastępowane wartościami odpowiednich zmiennych.
- Modyfikacje wykonywane przez powłokę nazywa się *substytucjami*.
- Użytkownik może zablokować wykonywanie niektórych substytucji (np. stosując przełączniki powłoki).

Powłoka – aliasy

- Powłoka sprawdza, czy pierwsze słowo komendy ma zdefiniowany *alias* (synonim).
- Alias, podobnie jak zmienna powłoki, to pewna nazwa, której przypisano ciąg znaków.

Przykład:

alias dir=ls **alias md=mkdir**

alias ll='ls -al' **alias llm='ll | more'**

W dwóch ostatnich przykładach znaki cudzysłowu pozwalają przypisać nazwom **ll** i **llm** ciągi znaków zawierające spacje.

Z ostatniego przykładu widać, że przy definicji aliasu można użyć innego aliasu.

Powłoka – substytucja ścieżki

Przykład:
Jeżeli prywatnym katalogiem użytkownika jest /home/janek, to:

~	⇒ /home/janek
~/doc	⇒ /home/janek/doc
~marek/bin	⇒ /home/marek/bin
~+/doc	⇒ $PWD/doc

Powłoka interpretuje również następujące konstrukcje rozpoczynające się od tyldy:

- **~user** - nazwa prywatnego katalogu użytkownika user
- **~+** - nazwa aktualnego katalogu
- **~-** - nazwa katalogu aktualnego przed ostatnią komendą cd

Powłoka – substytucja wyniku komendy

- Użycie konstrukcji o postaci:

$(lista)

Przykład 1:
Wyświetlenie listy plików, których właścicielem jest użytkownik:

ls -l | grep ` whoami `

Przykład 2:
Przeniesienie plików, których nazwy są wymienione w pliku *filelist* do katalogu dane:

mv $(cat filelist) dane/

` lista `

Powłoka – generacja nazw plików

- Jeżeli słowo stanowiące część komendy interpretowanej przez powłokę zawiera znaki

  *** ? [ ]**

  to słowo to zostanie potraktowane jako wzorzec i zastąpione przez powłokę listą słów (nazw plików lub katalogów) pasujących do tego wzorca.
- Przy generacji nazw plików obowiązują następujące reguły:
  - znak ***** oznacza dowolny ciąg znaków (również ciąg pusty),
  - znak **?** oznacza jeden dowolny znak,

Powłoka – generacja nazw plików

- konstrukcja **[abc]** oznacza dokładnie jeden znak

Przykłady:

a*z	- pasuje do nazw plików zaczynających się na literę „a” i kończących na literę „z”, np. *az, aaz, axyz*.
***.c**	- pasuje do nazw plików mających końcówkę „.c”, np. *prog.c, func.c*.
[aeou]la	- pasuje do nazw: *ala, ela, ola, ula*.
[A-Za-z]*	- pasuje do nazw plików zaczynających się od dowolnej litery.

Powłoka – cytowanie

- Powłoka może zinterpretować wprowadzane komendy niezgodnie z intencją użytkownika.
- Można temu zapobiec stosując mechanizm cytowania..
- Można stosować trzy sposoby cytowania:
  - **\x** - odwrócony ukośnik kasuje specjalne znaczenie znaku stojącego bezpośrednio za nim,
  - **'...'** - pojedyncze znaki cudzysłowu kasują specjalne znaczenie wszystkich znaków pomiędzy nimi,
  - **"..."** - podwójne znaki cudzysłowu kasują specjalne znaczenie wszystkich znaków pomiędzy nimi, za wyjątkiem znaku $ (substytucja zmiennych powłoki) i znaku \.

Powłoka – wyrażenia arytmetyczne

Przykład:

x=1; y=2 - przypisanie wartości liczbowych zmiennym x i y

let "x=x+1" - zwiększane wartości x o 1

echo x = $x - zostanie wypisane: x = 2

zapisu wyrażeń arytmetycznych:

let *wyrażenie*

lub

((*wyrażenie*))

(w przypadku stosowania drugiej podanej postaci należy pamiętać o odstępach wokół podwójnych nawiasów).

Powłoka – konstrukcje sterujące

- Powłoki udostępniają użytkownikowi wiele konstrukcji sterujących, pozwalających na warunkowe wykonywanie poleceń, zautomatyzowanie wykonanie powtarzających się komend itp.
- Konstrukcje sterujące powodują, że język powłoki jest w pełni funkcjonalnym językiem programowania.
  - Instrukcje warunkowe
  - Instrukcje wyboru
  - Pętle

Powłoka – konstrukcje sterujące

Instrukcja warunkowa

- Instrukcja warunkowa pozwala wykonać zestaw (listę) komend, jeśli spełniony jest warunek określony przez użytkownika.
- Najprostszą postacią instrukcji warunkowej jest:

if *warunek* then *lista* fi

występujący w instrukcji warunek, od którego uzależnione jest wykonanie zestawu komend *lista*, to zazwyczaj wyrażenie o następującej postaci:

[[*wyrażenie_warunkowe*]]

Powłoka – konstrukcje sterujące

Instrukcja warunkowa

- Instrukcja warunkowa pozwala wykonać zestaw (listę) komend, jeśli spełniony jest warunek określony przez użytkownika.
- Najprostszą postacią instrukcji warunkowej jest:

if *warunek* then *lista* fi

występujący w instrukcji warunek, od którego uzależnione jest wykonanie zestawu komend *lista*, to zazwyczaj wyrażenie o następującej postaci:

[[*wyrażenie_warunkowe*]]

Powłoka – konstrukcje sterujące

Instrukcja warunkowa (cd.)

gdzie *wyrażenie_warunkowe* może być jedna z niżej podanych konstrukcji:

-a f*name* - plik lub katalog o nazwie *fname* istnieje

-d *fname* - *fname* jest nazwą katalogu

-f *fname* - *fname* jest nazwa zwykłego pliku

-w *fname* - użytkownik ma prawo zapisu pliku *fname*

-x *fname* - użytkownik ma prawo wykonania pliku *fname*

napis* = *wzorzec - napis i wzorzec są zgodne

wyr1* -eq *wyr2 - wartości wyrażeń *wyr1* i *wyr2* są równe;

Powłoka – konstrukcje sterujące

Instrukcja warunkowa (cd.)

zamiast **-eq** (równe) można użyć:

-ne (różne),

-lt (mniejsze),

- gt (większe),

-le (mniejsze lub równe),

- ge (większe lub równe)

war1* || *war2 - alternatywa warunków *war1* i *war2*

war1* && *war2 - koniunkcja warunków *war1* i *war2*

!*war* - negacja warunku *war*

Powłoka – konstrukcje sterujące

Instrukcja warunkowa (cd.)

- Innym sposobem określenia warunku jest komenda:

  test *warunek*

  lub (równoważnie)

  [*warunek*]

  Komenda *test* zwraca wartość zero jeśli warunek jest spełniony i wartość różna od zera, gdy nie jest

  Przykład:

  ```
  if  test -f /tmp/xxx  then
          rm  /tmp/xxx
  fi
  ```

  będzie spełniony.

Powłoka – konstrukcje sterujące

Instrukcja warunkowa (cd.)

- Można użyć bardziej rozbudowanych postaci instrukcji warunkowej:

if *warunek* then
** *lista1***
else *lista2* fi

if *warunek1* then l*ista1*
elif *warunek2* then l*ista2*
elif *warunek3* then *lista3*
...
else *lista* fi

Powłoka – konstrukcje sterujące

Instrukcja wyboru

Przykład:

```
case $2 in
        y* ) echo Yes ;;
        n* ) echo No  ;;
         *  ) echo ??? ;;
esac
```

jeśli drugi parametr pozycyjny zaczyna się na literę y, to zostanie wypisane słowo Yes; jeśli zaczyna się on na literę n, zostanie wypisane słowo No. Jeśli parametr ten zaczyna się innym niż y lub n znakiem, zostanie wypisane ???.

wyrażeń *słowo* i *wzorzec2*, itd.

Powłoka – konstrukcje sterujące

Pętle

- Pętla służy do wielokrotnego wykonania ciągu instrukcji.
- W powłokach występują trzy podstawowe rodzaje pętli: **for**, **while** i **until**.
- Pętla **for** pozwala wykonać ciąg instrukcji sciśle określoną liczbę razy.
- Ogólna postać instrukcji jest następująca:

for *zmienna* in *słowo1 słowo2* ...
do
lista
done

Powłoka – konstrukcje sterujące

Pętle

Zmiennej *zmienna* przypisywana jest wartość *słowo1*, i dla tak zdefiniowanej zmiennej wykonywane są komendy *lista*. Następnie zmienna przyjmuje wartość *słowo2* i ponownie wykonywana jest *lista*, itd.

Przykład:

```
for i in a*
do
        mv $i Afiles/
done
```

przenosi wszystkie pliki o nazwach zaczynających się na literę *a* do katalogu *Afiles*

Powłoka – konstrukcje sterujące

Pętle

- Pozostałe dwa rodzaje pętli pozwalają na wykonanie ciągu poleceń *lista* w zależności od tego, czy warunek (którym może być wyrażenie warunkowe, komenda lub nawet lista komend) jest spełniony.
- W instrukcji **while** warunek sprawdza się na początku, i *lista* jest wykonywana tak długo, jak długo warunek jest spełniony.
- W pętli **until** warunek sprawdzany jest na końcu, i pętla wykonywana jest tak długo, jak długo warunek nie jest spełniony.

Powłoka – konstrukcje sterujące

Pętle

```
for zmienna in słowo1 słowo2 ...
do
  lista
done
```

```
while warunek
do
  lista
done
```

```
until warunek
do
  lista
done
```

W liście komend występującej po słowie do można użyć komend:

break - przerwanie pętli

continue - powrót do początku pętli

Powłoka – funkcje

- Ciąg poleceń można zdefiniować (np. na początku skryptu) jako funkcję, a następnie wielokrotnie wykorzystywać tę funkcję w różnych miejscach skryptu z różnymi parametrami.
- Funkcje można zdefiniować w jeden z następujących sposobów:

  function *fname* { *lista*; }

  lub

  ***fname* () { *lista*; }**

  gdzie *fname* jest nazwą funkcji.

Powłoka – inne konstrukcje

- Listę komend można zamknąć w nawiasy okrągłe:

  (lista)

  lub klamrowe

  { lista; }

  w pierwszym przypadku lista jest wykonywana w osobnym środowisku,

  w drugim w tym samym środowisku co powłoka macierzysta (w powłoce Bourne'a konstrukcje te maja nieco inne działanie).

Powłoka – przyporządkowanie strumieni

- Przy otwarciu pliku dane pamiętane są w tablicy plików, a indeks do tej tablicy nazywamy *deskryptorem pliku*.
- Zazwyczaj programy wykonujące operacje wejścia/wyjścia określają plik lub urządzenie będące źródłem bądź odbiorcą informacji nie poprzez nazwę pliku, ale przez deskryptor tego pliku.
- Następujące deskryptory są zarezerwowane:
  - **0** standardowe wejście (standardowo związane z klawiaturą terminala),
  - **1** standardowe wyjście (standardowo związane z monitorem terminala),
  - **2** standardowe wyjście diagnostyczne (standardowo związane z monitorem terminala).

Powłoka – przyporządkowanie strumieni

- Dzięki deskryptorom możliwa jest bardzo łatwa zmiana przyporządkowania strumieni we/wy.
- Powłoka jest wyposażona w specjalne mechanizmy ułatwiające takie operacje.

Zmiana standardowego wejścia:

prog ... <fname

konstrukcja ta spowoduje, że standardowym wejściem programu *prog* będzie nie klawiatura terminala, ale plik o nazwie *fname*.

Powłoka – przyporządkowanie strumieni

Zmiana standardowego wyjścia:

prog ... >fname

konstrukcja spowoduje, że standardowym wyjściem programu *prog* będzie nie ekran monitora, ale plik o nazwie *fname*. Zamiast pojedynczego można użyć podwójnego znaku większości:

prog ... >>fname

spowoduje to, że standardowe wyjście programu *prog* zostanie dopisane na końcu pliku *fname*.

Powłoka – przyporządkowanie strumieni

Zmiana standardowego wyjścia diagnostycznego:

prog ... 2>fname

użycie tej konstrukcji spowoduje, że standardowe wyjście diagnostyczne programu *prog* będzie skierowane do pliku o nazwie *fname*.

- Mechanizm zmiany przypisania standardowych strumieni wejścia/wyjścia można stosować do strumieni opisanych dowolnymi deskryptorami, np. zapis:

prog ... <&d1 >&d2

oznacza, że wejściem komendy *prog* będzie plik o deskryptorze d1, a jego wyjściem plik o deskryptorze d2.

Powłoka – skrypty

- Jeżeli skrypt ma być wykonywany podobnie jak każda inna komendą, tj. przez napisanie nazwy skryptu z listą parametrów, to skrypt taki winien mieć nadany atrybut **x**.
- Skrypt można wykonać także w następujący sposób:

  sh <*skrypt*

  (zamiast sh można użyć nazwy innej powłoki).
- W obu przypadkach skrypt zostanie wykonywane w osobnej powłoce.
- Oznacza to m.in. że w czasie wykonywania skryptu nie będą widoczne nie wyeksportowane wcześniej zmienne powłoki, a zmienne zdefiniowane wewnątrz skryptu znikną po jego zakończeniu.

Powłoka – skrypty

- Skrypt może być wykonany w ramach aktualnej powłoki za pomocą specjalnej komendy **.** (kropka):

  . skrypt ...

  (w ten sposób można wykonać także skrypt, któremu nie nadano atrybutu x).
- Parametry przekazywane w linii komendy są dostępne wewnątrz skryptu jako zmienne powłoki **$1**, **$2**, ... itd.
- Użycie wewnątrz skryptu komendy

  shift

  spowoduje przesunięcie o jedną pozycję, tj. **$1** będzie drugim argumentem z linii komendy, **$2** - trzecim itd.

Powłoka – skrypty

- Programista tworząc skrypt może narzucić powłokę, która ma wykonać skrypt.
- Należy w tym celu podać w pierwszej linii nazwę tej powłoki (pełną ścieżkędostępu) poprzedzoną znakami **#!**, np.:

#!/bin/ksh

Powłoka – konfiguracja

- Niektóre aspekty sposobu działania powłoki można zmieniać za pomocą tzw. przełączników powłoki.
- Przełączniki takie są ustawiane za pomocą komendy:

  set +o *opcja*

  set -o *opcja*

  znak **+** oznacza wyłączenie przełącznika *opcja*,
  znak **-** jego włączenie.
- Liczba i znaczenie przełączników są zależne od powłoki.

Powłoka – konfiguracja

- Ważniejsze przełączniki to:
  - **allexport** - automatyczne eksportowanie zmiennych powłoki
  - **ignoreeof** - ignorowanie Ctrl-D (zakończenie pracy tylko komendą exit)
  - **noexec** - sprawdzana jest poprawność syntaktyczna komend (bez ich wykonania)
  - **noglob** - powłoka nie generuje nazw plików
  - **nounset** - niezdefiniowana zmienna powłoki jest traktowana jako błąd
  - **monitor** - pozwala na użycie komend sterowania zadaniami

Powłoka – konfiguracja

- Administrator systemu bądź użytkownik może utworzyć skrypty startowe, tj. skrypty zawierające komendy, które maja być automatycznie wykonane po zalogowaniu się użytkownika (rozpoczęciu pracy powłoki).
- Skrypty takie mają ustalone w danej powłoce nazwy.
  - Powłoka Bourne'a: **/etc/profile**, **$HOME/.profile**
  - Powłoka Korna: **/etc/profile**, **$HOME/.profile**, **$ENV**
  - Powłoka zsh: **/etc/zprofile**, **$HOME/.zprofile**

Powłoka – konfiguracja

- Pliki znajdujące się w katalogu /etc są tworzone przez administratora i zwykły użytkownik nie ma na nie wpływu.
- Zawierają one zwykle komendy, które muszą być wykonane podczas logowania się każdego użytkownika, takie jak np.
  - wyświetlenie motd,
  - ustawienie ścieżki systemowej
  - sprawdzenie poczty.

Powłoka – konfiguracja

- Pozostałe skrypty są tworzone przez użytkownika w jego prywatnym katalogu i z reguły ustawiają parametry powłoki odnoszące się do tego użytkownika, np.
  - rodzaj terminala,
  - prywatne katalogi w ścieżce dostępu,
  - nazwa używanego edytora ASCII,
  - aliasy itp.
- Pliki $HOME/.profile ($HOME/.zprofile, $HOME/.zlogin) wykonywane są (podobnie jak pliki z katalogu /etc) tylko przez powłokę loginową.
- Pliki wskazywane przez zmienną powłoki ENV są wykonywane w oddzielnej, nowej powłoce.

Procesy

- Proces unixowy jest wykonywalnym programem załadowanym do pamięci operacyjnej komputera, wraz z pewnym środowiskiem (zestawem zmiennych z wartościami).
- Jądro Unixa utrzymuje tablicę wszystkich istniejących procesów wraz z zestawem informacji o nich, np. zestawem otwartych plików, maską sygnałów procesu, priorytetem, i innymi.
- Procesy tworzone są zawsze przez klonowanie istniejącego procesu funkcją **fork**, zatem każdy proces posiada tzw. proces nadrzędny, albo rodzicielski (*parent process*), który go utworzył.

Procesy

- Wyjątkiem jest proces numer 1, utworzony przez jądro Unixa, i wykonujący program ***init***, który jest protoplastą wszystkich innych procesów w systemie.
- Podproces dziedziczy i/lub współdzieli pewne atrybuty i zasoby procesu nadrzędnego, a gdy kończy pracę Unix zatrzymuje go do czasu odebrania przez proces nadrzędny statusu podprocesu (wartości zakończenia).

Procesy – kończenie pracy procesu

- Zakończenie procesu unixowego może być normalne, wywołane przez zakończenie funkcji **main()**, lub funkcji **exit()**. Wtedy następuje:
  - wywołanie wszystkich handlerów zarejestrowanych przez funkcję **atexit()** ,
  - zakończenie wszystkich operacji wejscia/wyjścia procesu i zamkniecie otwartych plików,
- Można spowodować normalne zakończenie procesu bez kończenia operacji wejścia/wyjścia ani wywoływania handlerów atexit, przez wywołanie funkcji **_exit()** .
- Zakończenie procesu może być anormalne, przez wywołanie funkcji **abort**, lub otrzymanie sygnału (funkcje **kill()**, **raise()**)).

Procesy – status procesu

- W chwili zakończenia pracy proces generuje kod zakończenia, tzw. status, który jest wartością zwróconą z funkcji **main()** albo **exit()**. W przypadku śmierci z powodu otrzymania sygnału kodem zakończenia jest wartość **128+nr sygnału**.
- Rodzic normalnie powinien po śmierci potomka odczytać jego kod zakończenia wywołując funkcję systemową **wait()** (lub **waitpid()**).
- Gdy proces nadrzędny żyje w chwili zakończenia pracy potomka, i nie wywołuje funkcji **wait()**, to potomek pozostaje (na czas nieograniczony) w stanie zwanym *zombie* .

Procesy – status procesu

- Istnieje mechanizm adopcji polegający na tym, że procesy, których rodzic zginął przed nimi (sieroty), zostają adoptowane przez proces numer 1, init.
- Init jest dobrym rodzicem (chociaż przybranym) i wywołuje okresowo funkcję **wait()** aby umożliwić poprawne zakończenie swoich potomków.

Procesy – grupy procesów

- **Grupa procesów** - to wszystkie podprocesy uruchomione przez jeden proces nadrzędny. Każdy proces w chwili utworzenia automatycznie należy do grupy procesów swojego rodzica.
- Pod pewnymi względami przynależność do grupy procesów jest podobna do przynależności do partii politycznych. Każdy proces może:
  - założyć nową grupę procesów (o numerze równym swojemu PID),
  - wstąpić do dowolnej innej grupy procesów,
  - włączyć dowolny ze swoich podprocesów do dowolnej grupy procesów (ale tylko dopóki podproces wykonuje kod rodzica).

Procesy – stany procesów

- **Wykonywalny (*runnable*)** — proces w kolejce procesów gotowych do wykonania
- **Uśpiony (*sleeping*)** — proces czeka na coś (np. na dane do przeczytania, na sygnał, na dostep do zasobu, lub dobrowolnie śpi na określony okres)
- **Zatrzymany (*stopped*)** — proces gotowy do wykonywania lecz zatrzymany na skutek otrzymania sygnału SIGSTOP lub SIGTSTP, może to być skutkiem dobrowolnego żądania procesu, celowego wysłania sygnału (np. przez użytkownika), lub odwołania się procesu pracującego w tle do terminala sterujacego; wznowienie pracy procesu następuje po otrzymaniu sygnału SIGCONT

Procesy – stany procesów

- **Wymieciony (*swapped out*)** — proces usunięty okresowo z kolejki procesów gotowych do wykonywania wskutek działania algorytmu obsługi pamięci wirtualnej; stan wymiecenia nie jest właściwym stanem procesu — może on być w jednym z trzech poprzednich stanów; wymiecenie procesu wykonywalnego oznacza brak dostatecznej ilości pamięci fizycznej na jednoczesne skuteczne wykonywanie wszystkich procesów wykonywalnych
- **Zombie** — proces czeka na zakończenie

System plików

- **Dane i programy** przechowywane w systemie tworzą tzw. system plików.
- W systemie UNIX wyróżnia się następujące rodzaje plikw:
  - **plik zwykły** – ciąg bajtów;
  - **katalog** – informacje o innych plikach (nazwy i atrybuty);
  - **plik specjalny** – reprezentujący urządzenie;
  - **dowiązania** (ang. link) – czyli inna nazwa pliku zapisanego w systemie.

System plików

- Nazwa pliku może składać się z dowolnych znaków mających reprezentację graficzną (litery i cyfry, znaki interpunkcji itp.) oraz niektórych znaków specjalnych.
- Długość nazwy pliku jest zwykle ograniczona do 255 znaków (w starszych implementacjach UNIX'a - do 14 znaków).
- Wszystkie dane o pliku, poza nazwą, są przechowywane w strukturze systemowej zwanej i-węzłem (*ang. i-node*).

System plików – ścieżka dostępu

- Opis lokalizacji pliku w drzewie katalogów.
- Llista nazw katalogów oddzielonych znakiem **/** i zakończona nazwą pliku.
- Istnieją dwa rodzaje ścieżek dostępu do pliku:
  - **ścieżka bezwzględna**, zaczynająca się od głównego katalogu, np.

    /home/student/ula/doc/sales.95.raport
  - **ścieżka względna**, podająca położenie pliku względem katalogu roboczego (np. jeśli katalogiem roboczym jest **/home/student**, to ścieżka względna do podanego wyżej pliku będzie:

    ../ula/doc/sales.95.raport

System plików – ścieżka przeszukiwań

- Ciąg nazw katalogów zapisanych w zmiennej środowiskowej PATH.
- Jeżeli użytkownik napisze z klawiatury nazwę komendy, to system operacyjny będzie poszukiwał programu o podanej nazwie w kolejnych katalogach tworzących ścieżkę przeszukiwań.
- W odróżnieniu od niektórych innych systemów operacyjnych, katalog roboczy w ogóle nie jest przeszukiwany (chyba, że występuje w sposób jawny w ścieżce przeszukiwań).

System plików – atrybuty

- Dla każdego pliku istnieje struktura zwana ***i-node*** zawierająca wiele informacji o pliku (w sensie obiektu istniejącego na dysku):
  - typ pliku: zwykły, specjalny (urządzenie), kartoteka;
  - 9 bitów praw dostępu i 3 bity dodatkowe;
  - długość pliku w bajtach;
  - numer właściciela;
  - numer grupy;
  - czas ostatniej modyfikacji pliku;
  - czas ostatniej modyfikacji i-node'u;
  - czas ostatniego dostępu do pliku;
  - liczba odwołań z różnych kartotek systemu plików.

System plików – prawa dostępu

- Prawo wykonania przez użytkownika lub grupę użytkowników określonych operacji na pliku lub katalogu.
- Prawa dostępu są zwykle zapisywane w postaci 9-znakowego kodu: **rwxrwxrwx**
- Znaczenie poszczególnych kodów jest następujące:
  - **r** - prawo odczytu pliku prawo odczytu zawartości katalogu
  - **w** - prawo zapisu do pliku prawo zmiany zawartości katalogu (utworzenia lub usunięcia pliku)
  - **x** - prawo wykonania pliku prawo przejścia przez katalog

System plików – prawa dostępu

- Pierwsza grupa znaków rwx odnosi się do praw właściciela, druga grupy, a trzecia - pozostałych użytkowników.

Przykład:

```
$ ls -l
total 5
drwxr-x--- 7 jasio users 512  Jul 11 22:47 katalog1
drwxr-xr-x 2 jasio users 512  Jul 11 02:22 katalog2
drwxr-x--- 5 jasio users 512  Jul 11 23:11 katalog3
drwxr-xr-x 9 jasio users 512  Jul 11 12:30 katalog4
-rw-r--r-- 1 jasio jasio 159  Dec 22 15:15 plik1
```

System plików – katalogi systemowe

- **/bin, /usr/bin** - komendy dla użytkowników.
- **/etc, /usr/etc, /sbin, /usr/sbin** - pliki systemowe i konfiguracyjne, programy systemowe i narzędzia
- **/dev** - pliki specjalne reprezentujące urządzenia.
- **/tmp, /usr/tmp** - katalog na pliki tymczasowe.
- **/lib, /usr/lib** - biblioteki, pliki nagłówkowe.
- **/lost+found** - pliki odzyskane podczas sprawdzania spójności systemu plików.
- **/usr/local** - dodatkowe oprogramowanie zainstalowane w systemie.
- **/var** - zmieniające się pliki systemowe (logi, kolejki).
- **/home** - prywatne katalogi użytkowników.

Podstawowe polecenia

Dokumentacja elektroniczna

- Użytkownik systemu UNIX ma do dyspozycji systemy dokumentacji elektronicznej, które dostarczają opisy składni i zastosowań wszystkich dostępnych poleceń.
- Podstawowym podręcznikiem elektronicznym jest **man pages**, podręcznik ten wywołuje się poleceniem:

man [-s *nr_sekcji*] *temat*

gdzie:

- *te*

stru macje,

- *nr* nce

Ma naczeń.

Przykłady:

man man

man ls

man -s l zsh

Podstawowe polecenia

Operacje na plikach i katalogach

- **ls** - wyświetla liste plików z katalogu roboczego

```
# ls
TIMEZONE                labelit                 rc1.d
```

```
# ls -l c*
lrwxrwxrwx   1 root     root          29 Dec 12  2003 cachefspack
-> ../lib/fs/c
```

```
# ls -F d*
d2u*            dd*             devreserv*      diffmk*         dispgid*
dos2unix*
daps*           deroff*         df@             dircmp*         dispuid*        du*
date*           devattr*        diff*           dirname*        dmesg@
dumpcs*
dc*             devfree*        diff3*          disable@        domainname*
dumpkeys*
```

```
-r-xr-xr-x   1 bin      bin          9388 Jul 16  1997 cat
-r-xr-xr-x   4 root     bin         38672 Jun  2  2005 catman
```

katalogu bieżącym

Podstawowe polecenia

Operacje na plikach i katalogach

- **cd** *kat* - zmienia katalog roboczy na *kat*
- **cd ..** - przejście do wyższego katalogu (rodzica)
- **cd /** - wejście do głównego (root) katalogu
- **pwd** - wyświetla nazwę katalogu roboczego
- **mkdir** *kat* - zakłada w katalogu roboczym podkatalog *kat*
- **rmdir** *kat* - usuwa katalog *kat* (jeżeli jest pusty)
- **cp** *pl1 pl2* - kopiuje zawartość pliku *pl1* do nowego pliku *pl2*

Podstawowe polecenia

Operacje na plikach i katalogach

- **cp** *pl1 pl2 pl3 kat* - kopiuje pliki *pl1, pl2, pl3* do katalogu *kat*
- **rm** *pl1 pl2* - kasuje pliki *pl1, pl2*
- **mv** *pl1 pl2* - zmienia nazwę pliku *pl1* na *pl2*
- **mv** *pl1 pl2 pl3 kat* - przenosi pliki *pl1, pl2, pl3* do katalogu *kat*
- **ln** *pl1 nazwa2* - tworzy łącznik (tzw. "twardy") *nazwa2* do pliku *pl1*
- **ln** *pl1 pl2 pl3 kat* - tworzy łącznik do plików *pl1, pl2, pl3* w katalogu *kat*

Podstawowe polecenia

Operacje na plikach i katalogach

- **ln** -s *sciezka skr* - tworzy łącznik symboliczny ("miękki") o nazwie *skr*, zastępujący ścieżkę dostępu *sciezka*
- **find** *kat -name pl* - wyszukanie plików o nazwie *pl* w drzewie katalogów zaczynającym się od katalogu *kat*

Podstawowe polecenia

Prawa dostępu

np. polecenie:

chmod g+r *plik1*	- nadaje grupie, możliwość czytania pliku (kat.) *plik1*,
chmod u+x *plik2*	- nadaje właścicielowi, prawa wykonywania pliku plik2,
chmod o-x *plik3*	- odbiera pozostałym (other) użytkownikom, prawa do wykonywania pliku *plik3*,
chmod g=rwx *plik4*	- nadaje pełne uprawnienia grupie, do której należy właściciel,
chmod ugo-x *plik5*	- odbiera wszystkim prawa do wykonywania pliku plik5.

Podstawowe polecenia

Prawa dostępu

- Można też za pomocą polecenia **chmod**, nadawać uprawnienia w formie oktalnej np.:

  chmod 740 *plik1*

  nadaje właścicielowi uprawnienia do czytania, pisania i wykonywania pliku, grupie tylko to czytania a innym zabiera wszystkie uprawnienia.
- Obliczyć wartość oktalną uprawnień jest bardzo łatwo:

  |r|w|x|r|w|x|r|w|x| pełne uprawnienia dla wszystkich

  4+2+1 4+2+1 4+2+1 $\rightarrow$ 7 7 7 czyli chmod 777 plik0

Podstawowe polecenia

Prawa dostępu

- |r|w|x|r|w|-|-|-|-| pełne uprawnienia dla właściciela, prawo czytania i pisania dla grupy
- 4+2+1 4+2+0 0+0+0
- 7 6 0 czyli chmod 760 plik0
- |r|-|-|-|-|-|r|w|-| uprawnienie do czytania dla właściciela oraz prawo do czytania i pisania dla innych użytkowników
- 4+0+0 0+0+0 4+2+0
- 4 0 6 czyli chmod 406 plik0