Elementarz un*x'owy

Marek Czajko, Michał Zasada

1 Wstęp

Niniejszy dokument został przygotowany na potrzeby przedmiotu "Podstawy systemu HP-UX" a następnie zmodyfikowany by uogólnić przedstawiane w nim kwestie dla wszelkich odmian systemu operacyjnego un*x. Jednak opcje komend i przykłady pochodzą z tego systemu.

Materiały są przeznaczone w szczególności dla ludzi:

- planujących wykonywanie pod un*x'em podstawowych działań jako użytkownicy;
- nie mających wcześniej styczności z tym systemem;

Jednocześnie proszę o przesyłanie na adres mcj@witch.sggw.waw.pl wszelkich uwag mogących przyczynić się do poprawienia i rozbudowy tego dokumentu.

Oznaczenia stosowane w tym dokumencie:

- **czcionką o stałej szerokości, krojem "grubym" oznaczane są teksty wpisywane przez użytkownika;**
- **czcionką o stałej szerokości, krojem "grubym", podkreślonym oznaczane są teksty wpisywane przez użytkownika, ale nie wyświetlane na ekranie (np. hasła);**
- czcionką o stałej szerokości, krojem normalnym oznaczane są teksty wyświetlane przez system operacyjny oraz nazwy klawiszy na klawiaturze,
- [nawiasy_kwadratowe] w składni poleceń oznaczają argument lub opcję fakultatywną (podawaną bez nawiasów), brak nawiasów oznacza argument obligatoryjny,
- tekst podkreślony w składni poleceń oznacza, że należy go czymś zastąpić (a nie przepisywać jak leci :).

1.1 Co to jest system operacyjny?

System operacyjny (OS - Operating System) to oprogramowanie nadzorujące pracę komputera. Generalnie, system operacyjny składa się z jądra (ang. kernel) oraz programów użytkowych. Kernel przyjmuje zlecenia przesłane doń przez programy użytkowe, użytkowników itd. i wykonuje je przydzielając im zasoby komputera takie jak pamięć, czas procesora czy urządzenia zewnętrzne. Kernel działa zawsze. Jest pierwszym programem, który startuje po włączeniu komputera i ostatnim, który jeszcze działa, gdy system zostaje zatrzymany.

Istnieje duża ilość systemów operacyjnych. Należą do nich najbardziej znane jak DOS, OS/2 czy Windows 95. Całą grupę tworzą różne odmiany systemu un*x, do której to grupy należy także HP-UX.

1.2 Geneza un*x'a

System Un*x (pisany małą literą oznacza najczęściej pewien typ/grupę systemów operacyjnych) wywodzi się z systemu Multics, który powstawał w latach 60-tych w Bell Telephone Laboratories (oddział AT&T) przy współpracy z General Electric i Massachusetts Institute of Technology (MIT). Po zarzuceniu prac nad Multics'em kilku jego twórców (Ken Thompson, Rudd Canaday, Doug McIlroy, Joe Ossana i Denis Ritchie) opracowało w roku 1969 na jego bazie system, który poprzez przeciwieństwo (był dużo mniejszy i początkowo tylko na jednego użytkownika) nazwano (Brian Kernigham) od słowa unicus (jedyny) – „Unix". Początkowo miał na celu wykorzystanie maszyny PDP-7 firmy DEC. System wzbudził zainteresowanie, więc korzystając z zapotrzebowania administracji Laboratorium na dobre narzędzia do przygotowywania dokumentów autorzy mogli kontynuować nad nim pracę. Po opracowaniu, przez Dennis'a Ritchie języka C system przepisano w nowym języku (1973) z assemblera. Od tej chwili historia un*x'a jest

nierozerwalnie związana z C. Dzięki C możliwe stało się stosunkowo łatwe przeniesienie un*x'a na inne maszyny, co nastąpiło po raz pierwszy w 1977 roku. W początkach lat 70-tych un*x został bardzo tanio rozprowadzony na uniwersytetach amerykańskich gdzie zyskał dużą popularność.

Ostatecznie powstały dwie główne odmiany un*x'a: AT&T Unix (Unix System Laboratories, obecnie własność firmy Novell) i BSD (Berkeley Software Distribution, opracowany w latach 70-tych na Uniwersytecie Kalifornijskim w Berkeley) oraz zbiór dokumentów standaryzujących o nazwie POSIX. Wszystkie inne wersje są najczęściej połączeniem rozwiązań stosowanych w tych odmianach oraz dodatków wprowadzonych przez twórców konkretnej implementacji.

1.3 Charakterystyka un*x'a

Un*x ma swoje implementacje na różnych maszynach (początkowo był rozprowadzany z kodem źródłowym a i teraz jest wyposażany we wszystkie narzędzia potrzebne do tworzenia programów). Od Amigi (*Linux*) i PC (*FreeBSD, Linux, Solaris, SCO*), poprzez procesory Sparc (SUN - *Solaris, Linux*), MIPS (Silicon Graphics - *Irix, Linux*) i Alpha AXP (DEC - *Digital Un*x, Linux*) aż po duże maszyny IBM (*AIX*) i Cray (*Unicos, Solaris*) - wszędzie można spotkać un*x'a. Od swojego powstania przebył długą drogę i został znacznie rozbudowany. Jednak nadal można w nim znaleźć wiele pozostałości "prehistorycznych" jak:

- krótkie nazwy poleceń (spowodowane lenistwem twórców oraz wolnymi wówczas liniami transmisyjnymi,
- stare nazewnictwo (np. często występujące słowo print (ang. drukować) oznacza tu także wyświetlać; były to czasy, kiedy nie było monitorów)
- bardzo różniące się w filozofii rozwiązania (system tworzyło bardzo wielu ludzi, w tym także studentów)

Un*x jest systemem wielodostępnym i wielozadaniowym. Wielodostępność oznacza, że w tej samej chwili może, na tym samym komputerze, pracować wielu ludzi (siedząc przy nim lub łącząc się poprzez sieć). Wielozadaniowość pozwala na jednoczesne uruchamianie wielu programów. W system wbudowane są mechanizmy rozróżniania użytkowników (konta, hasła) oraz zabezpieczania plików przed niepowołanym dostępem (prawa dostępu). Możliwa jest także praca w sieci - potrzebne narzędzia są już w systemie. Wszystkie nowsze wersje un*x'a są wyposażone także w bardzo estetyczne, wygodne i łatwe do opanowania środowisko graficzne (X Window System). W każdym un*x'ie znaleźć można także kompilator i niezbędne biblioteki do języka C.

Jednak największym atutem (niektórzy uważają, że największą wadą) un*x'a jest jego silnie zmodularyzowana budowa. Oznacza to, że w systemie znajduje się bardzo dużo drobnych programów wykonujących niewielkie zadania. Łącząc wyniki poszczególnych programików możemy osiągać zdumiewające wyniki, nieosiągalne w innych systemach bez zakupu specjalizowanego oprogramowania. Nie należy też zapominać o bogatym języku skryptów, który ułatwia uzyskiwanie takich wyników.

Ostatnią cechą, jaką tu wymienimy, wyróżniającą un*x spośród innych systemów jest reprezentowanie urządzeń jako pliki. Jest nawet powiedzenie, że wszystko w un*x'ie jest plikiem. Oznacza to np., że znając plik reprezentujący terminal, przy którym siedzi inny użytkownik możemy wyświetlić na jego ekranie wiadomość zapisując ją po prostu do tego pliku. Zapisując tekst do pliku oznaczającego port z podłączoną drukarkę powodujemy powstanie wydruku. Nawet dyski, czy pamięć operacyjna mają swoją reprezentację w postaci plików. Zwykle jednak bezpośredni zapis czy odczyt z takich plików specjalnych wymaga sporej wiedzy o konkretnym urządzeniu i nie jest wymagany. Narzędzia systemowe dbają o to abyśmy nie musieli zajmować się takimi szczegółami.

1.4 Konto

Jak już wcześniej wspomniano, w un*x'ie, w celu umożliwienia weryfikacji danej osoby - a co się z tym wiąże jej praw do poszczególnych elementów systemu - każdemu użytkownikowi jest przypisany identyfikator. Całość elementów związanych z użytkownikiem określa się mianem konta (ang. account), na które składa się m.in. nazwa, hasło, grupa, wydzielony katalog na dysku itd., aby więc rozpocząć pracę z systemem trzeba mieć w nim założone konto. Konto zakłada administrator systemu, który w un*x'ie ma identyfikator "root". Konto root jest tworzone w momencie instalacji systemu. "root" ma nieograniczone prawa a jednocześnie jedynie on może zakładać nowe konta - chyba, że tak zmodyfikuje system, że będzie to mogła robić osoba o innym identyfikatorze.

Zwykle jeden człowiek ma jedno konto w systemie. Nie jest to jednak regułą. Konto może też być przypisane nie do człowieka a do funkcji, jaką wykonuje. Wszystko zależy od polityki instytucji oraz umowy z administratorem.

Aby zapewnić, że jeden człowiek nie wykorzystuje konta innego człowieka, konta zwykle mają przypisane hasła. Hasło nadaje administrator podczas tworzenia konta, może ono jednak być (i powinno) zmienione w każdej chwili przez danego użytkownika. Hasło jest znane jedynie użytkownikowi. Nikt inny nie może go odczytać (nawet administrator, choć może je zmienić). Jeśli użytkownik zapomni hasło, musi poprosić administratora, aby ustalił nowe.

Identyfikator użytkownika nie powinien być dłuższy niż 8 znaków. Jest często zlepkiem liter występujących w imieniu i nazwisku użytkownika lub w nazwie funkcji, jaką pełni. Jan Kowalski może mieć np. identyfikator jank.
Ostatecznie, aby rozpocząć pracę z systemem, administrator musi nam podać nasz identyfikator (nazwę konta) oraz początkowe hasło.

2 Sesja, pierwsze polecenia

2.1 Sesja

Wszystko, co robimy od momentu rozpoczęcia pracy z systemem aż do jego zakończenia nazywamy **sesją**. Sesję rozpoczynamy od **zalogowania się**, czyli podania systemowi naszego identyfikatora i hasła. Kończymy ją wpisując polecenie **`exit`** albo wciskając kombinację klawiszy `^d` (przytrzymując klawisz `Control` uderzamy klawisz litery "`d`") i, ewentualnie, `Enter`.

Należy bezwzględnie pamiętać o zakończeniu sesji, nawet, jeśli jedynie na chwilę oddalamy się od terminala.

Siedząc przy terminalu znakowym, po jego włączeniu, wciskamy klawisz `Return`, aby "zmusić" go do nawiązania połączenia z serwerem. Jeśli system nie zgłasza się - nie pojawia się napis **`login:`** - sprawa jest poważniejsza. Należy to zgłosić administratorowi. W przeciwnym wypadku wprowadzamy po dwukropku nasz identyfikator (w przypadku naszego Jana Kowalskiego będzie to napis **`jank`**) i wciskamy `Return`. Niezależnie od tego czy identyfikator został poprawnie wprowadzony, czy nie, pojawia się napis **`password:`** zachęcający nas do wpisania hasła. Wpisywane hasło nie jest pokazywane na monitorze. Po wciśnięciu `Return`, jeśli wszystko poszło dobrze, sesja jest rozpoczęta. Może się jeszcze pojawić pytanie systemu o typ terminala, przy jakim siedzimy:
`TERM=(vt100)`
Najczęściej spotykanym, choć jednym z najstarszych, typem terminala jest **`vt100`**. Należy pamiętać, że niewłaściwie podany typ terminala może całkowicie uniemożliwić pracę z un*x'em. Po chwili zgłasza się tak zwany shell (w najprostszym przypadku wyświetlając znak **`$`**) i możemy zacząć wprowadzać polecenia. Jeśli podaliśmy nieprawidłowy identyfikator lub hasło, shell się nie zgłasza a po ponownym wciśnięciu `Return` pojawia się znowu napis **`login:`**.

W przypadku pomyłki przy wprowadzaniu identyfikatora lub hasła nie należy go poprawiać - klawiatura nie jest jeszcze w pełni "sprawna". Wciskamy `Return` tyle razy (zwykle 1 lub 2) aż pojawi się znowu **`login:`** i zaczynamy zabawę od początku.

Pod adresem http://halina.univ.gda.pl/owner/instr1.html można znaleźć kilka informacji na temat procesu logowania się i podstawowych poleceń HP-UX.

2.2 Shell

Shell to un*x'owy interpreter poleceń. Ma znacznie bogatsze możliwości niż np. jego DOS'owy odpowiednik, `command.com`. Zadaniem shell'a jest umożliwienie wprowadzania poleceń z klawiatury, wykonywanie pewnych dodatkowych funkcji (jak np. podstawienie wartości pod zmienne, obsługa strumieni i potoków czy generowanie nazw plików na podstawie metaznaków) a wreszcie przekazywanie zinterpretowanych poleceń systemowi operacyjnemu (jądru) do wykonania.

Istnieje wiele różnych shell'i. Do najbardziej znanych należą `sh` (Bourne SHell), `csh` (C SHell), `ksh` (Korn SHell) czy `bash` (Bourne-Again SHell). Domyślnym shell'em w Linux'ie jest `bash`. Administrator może jednak zmienić shell dowolnemu użytkownikowi. Domyślnym znakiem zgłoszenia shell'i `sh`, `ksh` i `bash` jest **`$`** (dolar) a `csh` **`%`** (procent).

2.3 Wiersz poleceń

Po zalogowaniu się system (shell) wita użytkownika wspomnianym wyżej znakiem zgłoszenia (nazywanym też **prompt**'em lub **monit**'em), co oznacza, że komputer czeka na wydawanie komend. Komendy w un*x'ie to nic innego, jak programy, które uruchamia użytkownik. Nazwy programów są najczęściej skrótami od nazw angielskich mówiących, jakie zadanie wykonuje dany program (np. list => **`ls`**).
Konwencja przyjęta w un*x'ie zakłada, że wielkie i małe litery traktowane są przez system jako zupełnie różne, stąd np. zbiory: Plik.txt, pLik.txt, PLIK.txt, pliK.TXT czy t.p. traktowane są jako różne. Konwencja ta dotyczy również wydawanych systemowi poleceń - wszystkie one zawierają TYLKO małe litery.

Większość komend w systemie ma następującą składnię:

```
polecenie [opcja...] [argument...]
```

W takim zapisie nawiasy kwadratowe oznaczają, że dany element jest opcjonalny, podkreślenie, że trzeba go zastąpić odpowiednim tekstem, natomiast trzykropek, że może wystąpić więcej niż raz. Poszczególne pozycje muszą być oddzielone odstępem, (czyli przynajmniej jedną spacją lub tabulatorem) a oznaczają, co następuje:

- `polecenie` to nazwa pliku wykonywalnego (zwykle programu),
- `opcja` to modyfikator, który zmienia standardowe działanie polecenia, zwykle jest to pojedyncza litera poprzedzona znakiem `-` (minus),
- `argument` to zwykle ścieżka dostępu do pliku/katalogu lub inny argument nie zaczynający się od znaku `-`, jeśli argumentów jest więcej niż jeden, muszą być one oddzielone przynajmniej jedną spacją.

Działanie komend z różnymi opcjami i argumentami najlepiej prześledzić na przykładzie. W przypadku popełnienia błędu typograficznego można użyć klawisza `Backspace` w celu dokonania poprawki - nie należy się przejmować, jeśli kasowane znaki nie znikają z ekranu. Nowicjusz nie powinien też używać innych klawiszy pomocnych zwykle przy wprowadzaniu poprawek (np. strzałka w lewo).
Do wyświetlania zawartości katalogu bieżącego służy polecenie **`ls`**. Wynik działania samego polecenia (bez opcji i argumentów) może wyglądać następująco:

```
$ ls
index.html  plik.txt  prace  text.doc
$
```

Po wykonaniu tej komendy widać tylko tyle, że katalog bieżący zawiera cztery pliki lub katalogi (jeśli jest to nowe konto może się też nic nie pojawić, w takim wypadku wykonaj wcześniej komendę `touch plik.txt`). Nie wiadomo, czy wśród nich znajdują się katalogi, jaki rozmiar mają poszczególne zbiory, kto jest ich właścicielem itp. Aby uzyskać te informacje należy wydać komendę **`ls`** z opcją **`-l`** (długi - ang. **l**ong):

```
$ ls -l
total 8
-rw-r--r--   1 misioo   inni       46 Feb 23 12:21 index.html
-rw-r--r--   1 misioo   inni       75 Feb 23 12:21 plik.txt
drwxr-xr-x   2 misioo   inni       24 Feb 23 12:15 prace
-rw-r--r--   1 misioo   inni      137 Feb 23 12:22 text.doc
$
```

W tej chwili informacji na temat plików jest już znacznie więcej. Każda linia opisuje jeden plik lub katalog. Pierwszy znak w każdej linii mówi o typie pliku: znak `-` (minus) oznacza zwykły plik, litera `d` mówi, że jest to katalog. Dalsze informacje to:

- prawa do pliku (np. rw-r--r--)
- liczba dowiązań
- właściciel pliku
- grupa pliku
- rozmiar pliku w bajtach
- data i godzina utworzenia / ostatniej modyfikacji
- nazwa pliku

System operacyjny zakłada, że pierwszym słowem w linii jest nazwa polecenia. Jeśli więc np. pominiemy odstęp między **`ls`** a **`-l`** to shell wyświetli komunikat, że nie ma takiej komendy:

```
$ ls-l
ksh: ls-l:  not found
$
```

W oglądanym katalogu mogą znajdować się również pliki lub katalogi ukryte, których nazwa rozpoczyna się od "`.`" (kropki, ang. "dot files" - pliki "kropkowe"). Nie mają one żadnego specjalnego znaczenia dla systemu - nie są jedynie domyślnie wyświetlane. Aby wyświetlić również takie "przypadki" należy użyć polecenia **`ls`** z opcją **`-a`** (wszystkie - ang. **a**ll):

```
$ ls -a
.           .cshrc      .login      .sh_history   plik.txt    text.doc
..          .exrc       .profile    index.html    prace
$
```

Przy użyciu tej opcji pojawiają się pliki ukryte - w tym przypadku pliki konfiguracyjne i startowe dla danego użytkownika. Pojawia się również oznaczenie katalogu bieżącego (`.`) i nadrzędnego (`..`). Jednak znowu mamy tu tylko same nazwy - należy, więc połączyć działanie opcji **`-a`** i **`-l`**:

```
$ ls -a -l
```

```
total 22
drwxr-xr-x   3 misioo   inni      1024 Feb 23 12:15 .
drwxr-xr-x  18 root     root      1024 Feb 19 19:25 ..
-rw-r--r--   1 misioo   inni       818 Feb 19 19:17 .cshrc
-rw-r--r--   1 misioo   inni       347 Feb 19 19:17 .exrc
-rw-r--r--   1 misioo   inni       377 Feb 19 19:17 .login
-rw-r--r--   1 misioo   inni       446 Feb 19 19:17 .profile
-rw-------   1 misioo   inni       362 Feb 23 12:54 .sh_history
-rw-r--r--   1 misioo   inni        46 Feb 23 12:21 index.html
-rw-r--r--   1 misioo   inni        75 Feb 23 12:21 plik.txt
drwxr-xr-x   2 misioo   inni        24 Feb 23 12:15 prace
-rw-r--r--   1 misioo   inni       137 Feb 23 12:22 text.doc
$
```

Ten sam efekt uzyskuje się, gdy opcje podamy odwrotnie, tzn. wydamy komendę **`ls -l -a`**. Taki sposób dodawania opcji może być czasem uciążliwy, szczególnie w przypadku konieczności użycia wielu opcji. Dlatego też polecenie **`ls -a -l`** można podać również tak: **`ls -al`** lub tak: **`ls -la,`** co da identyczny efekt, jak oddzielne użycie opcji.

Ostatnim elementem składni poleceń są argumenty. W przypadku **`ls`** argumentem może być nazwa pliku (jeżeli chcemy wyświetlić informacje o danym pliku w bieżącym katalogu):

```
$ ls -la plik.txt
-rw-r--r--  1 misioo   inni        75 Feb 23 12:21  plik.txt
$
```

(`ls plik.txt` bez opcji jedynie potwierdzi istnienie takiego pliku) lub nazwa katalogu (jeżeli interesuje nas wyświetlenie zawartości katalogu innego niż bieżący):

```
$ ls -l /
total 10632
-rwxr-xr-x   1 root     other    2543616 Oct 26  1995 SYSBCKUP
drwxr-xr-x   4 root     other       3072 Oct 26  1995 bin
drwxr-xr-x  12 root     other       3072 Feb 19 18:58 dev
drwxr-xr-x  12 root     other       6144 Feb 23 14:31 etc
-rwxr-xr-x   1 root     sys      2543616 Oct 26  1995 hp-ux
dr-xr-xr-x   4 bin      bin         1024 Oct 26  1995 lib
drwxr-xr-x   2 root     root        8192 Oct  4  1994 lost+found
drwxr-xr-x  204 bin     bin         4096 Dec 15 17:28 system
drwxrwxrwx   2 root     root        3072 Feb 24 13:57 tmp
drwxr-xr-x  18 root     root        1024 Feb 19 19:25 users
drwxr-xr-x  41 root     root        1024 Jan 29  1996 usr
$
```

2.4 passwd

Poleceniem służącym do zmiany hasła jest **passwd**. Zmiana hasła jest najczęściej pierwszą czynnością po pierwszym zalogowaniu się na nowym koncie. Zmiana hasła jest też konieczna wówczas, gdy istnieje podejrzenie, że ktoś złamał nasze hasło i w ten sposób otworzył sobie drogę do systemu. Zasady bezpieczeństwa zalecają zmianę hasła co jakiś czas (np. co kilka miesięcy) jako zwykłą procedurę postępowania użytkownika.

Hasła:

- powinny mieć minimum 6 a maksimum 8 znaków długości,
- powinny być kombinacją dużych i małych liter oraz cyfr i innych znaków dostępnych na klawiaturze (jak . (kropka) czy ; (średnik),
- w skład hasła nie powinno wchodzić słowo ze słownika (w tym także innych krajów),
- w skład hasła nie powinny wchodzić znaki, których wewnętrzna reprezentacja jest inna w innym systemie operacyjnym (DOS, Windows, Mac, czy nawet inaczej skonfigurowany un*x); do takich należą polskie znaki diaktryczne,
- hasła nie powinny być nigdzie zapisywane (w tym także gdzieś w komputerze)

Procedura zmiany hasła wygląda następująco (wykonuje ją przykładowy użytkownik "misioo"):

```
$ passwd
Changing password for misioo
Old password:
```

System prosi najpierw o stare hasło - w ten sposób unika się możliwości zmiany hasła przez osobę, która dostała się do systemu bez podawania hasła - np. w chwili, gdy lekceważąc podstawowe wymogi bezpieczeństwa użytkownik oddala się od terminala nie wylogowując się.
W trakcie pisania - podobnie, jak przy logowaniu się do systemu - ze względów bezpieczeństwa hasło nie pojawia się. Po podaniu starego hasła system prosi o wprowadzenie nowego:

```
$ passwd
Changing password for misioo
Old password: str1hslo
New password: nwe0hslo
```

a następnie o jego potwierdzenie, aby uniknąć przypadkowych pomyłek typograficznych:

```
$ passwd
Changing password for misioo
Old password: str1hslo
New password: nwe0hslo
Re-enter new password: nwe0hslo
$
```

Powtórne wprowadzenie właściwego hasła powoduje rzeczywiste dokonanie zmiany hasła dla użytkownika i zakończenie działania komendy **passwd** - wyświetla się znak zgłoszenia systemu.

Jeżeli wprowadzić chcemy zbyt krótkie hasło - system na to nie pozwoli:

```
$ passwd
Changing password for misioo
Old password: str1hslo
New password: krt3
```

```
Password is too short - must be at least 6 characters
New password:
...
```

Taka sama sytuacja będzie miała miejsce, jeżeli hasło będzie zbyt podobne do starego (musi się ono różnić co najmniej trzema znakami):

```
$ passwd
Changing password for misioo
Old password: str1hslo
New password: abr1hslo
Passwords must differ by at least 3 positions
New password:
...
```

Jeżeli w trakcie wprowadzania starego hasła popełni się błąd, system powiadomi o tym:

```
$ passwd
Changing password for misioo
Old password: zle1hslo
Sorry.
$
```

Z kolei, jeżeli przy ponownym wpisywaniu nowego hasła popełniona zostanie pomyłka, pojawia się stosowny komunikat:

```
$ passwd
Changing password for misioo
Old password: str1hslo
New password: nwe0hslo
Re-enter new password: nwe0hsl0
They don't match; try again.
New password:
...
```

Wprowadzanie nowego hasła należy wówczas rozpocząć od nowa.

Jeżeli hasło zostało pomyślnie zmienione należy go używać od najbliższego logowania się do systemu.

2.5 Dokumentacja

Dokumentacja systemu dostępna jest w różnych formach zależnie od odmiany un*x'a:

- drukowanej (książki); jest to najlepsza dokumentacja; można ją nabyć niezależnie od samego systemu operacyjnego w formie pojedynczych pozycji, grup związanych z konkretnym aspektem systemu lub w całości;
- elektronicznej, rozprowadzanej na płytach CD i taśmach; zakres jest zwykle identyczny jak w przypadku książek; tę formę dokumentacji także zakupuje się oddzielnie;
- elektronicznej, rozprowadzanej wraz z systemem operacyjnym a dostępnej dla użytkownika za pomocą komendy **man** (podręcznik - ang. manual);

- elektronicznej, rozprowadzanej poprzez Internet; np. do systemu Linux tworzone są przez samych użytkowników tzw. dokumenty HOWTO

Komenda **man** oraz odpowiednia dokumentacja jest instalowana z każdą odmianą un*x'a. Nie obejmuje kompletu dostępnej dokumentacji systemu a jedynie szczegółowy opis wszystkich poleceń. Poza opisem poleceń zawiera też nieco dodatkowych informacji przydatnych programistom i administratorom. **"man"** nie nadaje się do nauki un*x'a choć jest doskonałym narzędziem pomocniczym w przypadku, gdy znamy już nazwę komendy a chcemy poszerzyć wiedzę na jej temat. Z uwagi na dostępność **man** zajmiemy się więc tylko tą formą podręczników.

"Strony" podręczników man są podzielone na sekcje ponumerowane od 1 do 9:

- `1` - polecenia użytkownika
- `1m` - polecenia obsługi systemu, głównie dla administratora, choć jest tam wiele poleceń przydatnych dla "zwykłego" użytkownika
- `2` - wywołania systemowe, dla programistów
- `3` - procedury biblioteczne, dla programistów
- `4` - formaty plików, dla administratora i programistów
- `5` - różne udogodnienia, dla wszystkich po trochu
- `6` - gry
- `7` - pliki specjalne, dla administratora i programistów
- `8` - brak, zawartość przeniesiona do sekcji 1m
- `9` - słownik pojęć, dla wszystkich, choć najbardziej przydatne nowicjuszowi

Szczególnie przy wyższych numerach sekcji ich przeznaczenie może się nieco różnić pomiędzy odmianami un*x'a. Składnia polecenia **man** to:

```
man [sekcja] strona
```

gdzie pod

- `sekcja` *można* podstawić jedno z podanych wyżej oznaczeń (bez minusa - to nie jest opcja tylko argument); jeśli nie podamy sekcji a w kilku sekcjach występuje taka sama nazwa strony, polecenie man pokaże nam stronę z sekcji o najniższym numerze;
- `strona` *należy* podstawić nazwę odpowiedniej strony podręcznika; w przypadku poleceń nazwa ta jest identyczna jak nazwa polecenia (oczywiście bez opcji i argumentów);

Przyjrzyjmy się układowi strony podręcznika na przykładzie polecenia **users**:

```
$ man users


 users(1)
users(1)


 NAME
      users - compact list of users who are on the system


 SYNOPSIS
      users [-c]


 DESCRIPTION
```

```
      users lists the login names of the users currently on the system in
a
      compact, one-line format.  In the HP Clustered environment, the -c
      option can be used to list the login names of all users in the
cluster
      (see glossary(9)).

      The login names are sorted in ascending collation order (see
      Environment Variables below).

 EXTERNAL INFLUENCES
    Environment Variables
      LC_COLLATE determines the order in which the output is sorted.

      If LC_COLLATE is not specified in the environment or is set to the
      empty string, the value of LANG is used as a default.  If LANG is
not
      specified or is set to the empty string, a default of ``C'' (see
      lang(5)) is used instead of LANG.  If any internationalization
      variable contains an invalid setting, users behaves as if all
      internationalization variables are set to ``C'' (see environ(5)).

 AUTHOR
      users was developed by the University of California, Berkeley and
HP.

 FILES
      /etc/utmp

 SEE ALSO
      who(1).

 Hewlett-Packard Company            - 1 -  HP-UX Release 9.04: November
1993

$
```

W pierwszej linii występuje zapis `users(1)` gdzie `users` to oczywiście nazwa polecenia (a w ogólności nazwa strony podręcznika man) a `1` to numer sekcji, w której znajduje się opis. Na końcu widzimy też numer strony. Podawane jest to tylko w celach pomocniczych (byśmy wiedzieli jak duży jest dany opis) i nie ma nic wspólnego z położeniem w dokumentacji danego opisu (zwanego stroną niezależnie od tego ile faktycznie stron zajmie na wydruku).

Każda strona składa się z kilku części. Nie wszystkie części występują przy każdym opisie. Poniżej przedstawione są nazwy ważniejszych części i ich krótkie charakterystyki.

- `NAME` (*nazwa*) - zawiera nazwę polecenia i krótki jego opis;
- `SYNOPSIS` (*składnia*) - składnia polecenia; prawie identyczna jak w tych materiałach do ćwiczeń (poza podkreśleniami w składni poleceń, które wprowadziliśmy dla lepszego zilustrowania);
- `DESCRIPTION` (*opis*) - szczegółowy opis polecenia, opcji i argumentów;
- `EXAMPLES` (*przykłady*) - przykłady użycia polecenia;
- `FILES` (*pliki*) - pliki związane w jakiś sposób z danym poleceniem;
- `SEE ALSO` (*patrz także*) - odwołania do innych stron związanych z danym poleceniem lub innej dokumentacji zawierającej dodatkowe informacje;
- `DIAGNOSTICS` (*diagnostyka*) - wyjaśnia komunikaty błędów, jakie mogą pojawić się przy korzystaniu z danego polecenia (ale tylko te, które są podawane przez same polecenie a nie np. shell);
- `BUGS` (*błędy*) - opis przyczyn powodujących problemy (tego nie znajdziesz w dokumentacji programów firmy Microsoft);
- `WARNINGS` (*ostrzeżenia*) - na co zwrócić baczną uwagę;
- `AUTHOR` (*autor*) - twórca lub twórcy polecenia;

Podczas oglądania wyniku działania polecenia man można używać klawiszy `Enter` do przewinięcia opisu o jedną linię, spacji do przewinięcia o wysokość ekranu a litery `q` do zatrzymania działania polecenia.

Jako przykład nazwy strony, która występuje w kilku sekcjach można przytoczyć **`mount`**. Polecenie

`man mount`

pokaże nam opis funkcji `mount` z sekcji `2` (dla programisty) natomiast

`man 1m mount`

opis polecenia **`mount`** a więc to, czego szukaliśmy. Sekcja `1m` jest tu wyjątkiem od reguły i występuje przy sortowaniu po sekcji `2`.

By dowiedzieć się więcej na temat samej komendy man trzeba wpisać polecenie

`man man`

Przy każdej sekcji jest też jej krótki opis dostępny po wydaniu komendy

`man` `sekcja` **`intro`**

gdzie `sekcja` jest oczywiście oznaczeniem interesującej nas sekcji natomiast

`man manuals`

wymienia wszystkie drukowane podręczniki.

2.6 Podstawowe komendy

2.6.1 whoami

Składnia: `whoami`

Polecenie `whoami` (ang. who am i - kim jestem) wyświetla nazwę użytkownika, który wykonał tę komendę. Przydaje się, gdy użytkownik używa jednocześnie w danej chwili kilku kont a zapomniał, z którego wykonuje daną komendę (to ma sens :).

Przykład (polecenie wykonuje użytkownik, który ma konto o nazwie mcj):

```
$ whoami
```

```
mcj
$
```

2.6.2 date

Uproszczona składnia: `date`

Polecenie date (ang. date - data) bez żadnych opcji powoduje wyświetlenie aktualnej daty i czasu. Dodatkowe opcje służą m.in. do zmiany sposobu wyświetlania.

Przykład:

```
$ date
Wed Feb 25 18:28:46 CET 1998
$
```

Mamy więc środę (`Wed`), 25 lutego (`Feb 25`) `1998` roku, godzinę `18:28` i `46` sekund w środkowo-europejskiej strefie czasowej (`CET` - Central European Time).

2.6.3 uname

Składnia: `uname [-a]`

Polecenie `uname` wyświetla nazwę systemu operacyjnego. Najużyteczniejszą opcją jest `-a` (ang. all - wszystko), która powoduje uzupełnienie wyniku o wiele użytecznych informacji o systemie operacyjnym i komputerze. Między innymi podaje też nazwę komputera.

Każdy komputer z un*x'em ma swoją nazwę nadawaną mu zwykle podczas instalacji systemu operacyjnego. Nazwę tę oczywiście może zmienić w każdej chwili.

Przykład:

```
$ uname -a
HP-UX boguslaw A.09.04 B 9000/806 1324576089 16-user license
$
```

Dowiadujemy się z tego, że pracujemy w systemie `HP-UX` w wersji `9.04`. Komputer nazywa się `boguslaw` i jest to model `9000/806`. System zakupiono z licencją na `16` jednocześnie pracujących użytkowników (nie mylić z liczbą kont, która może być dowolnie duża). `1324576089` to sprzętowy numer identyfikacyjny komputera (nieprzydatne).

2.6.4 who

Uproszczona składnia: `who [-T] [-H]`

Polecenie `who` (ang. who - kto) wyświetla listę aktualnie pracujących w systemie (zalogowanych) użytkowników. Posiada wiele opcji pozwalających uzyskiwać dodatkowe informacje. Prawdopodobnie najbardziej przydatne z tych opcji to:

- `-T` powodująca poszerzenie zakresu wyświetlanych informacji o użytkowniku oraz
- `-H` (ang. headers - nagłówki) dodająca nagłówek do każdej wyświetlanej kolumny

Przykładowy wynik działania polecenia

```
$ who -TH
NAME       LINE         TIME           IDLE    PID  COMMENTS
root     + console      Feb 25 17:02    .     1042  system console
user1    + ttyp2        Feb 25 17:12   0:38  23420  IIIp313st1.sggw.
```

```
user2    + ttyp4        Feb 25 16:16   .    23150  dce1-les.sggw.wa
user3    + ttyp5        Feb 25 16:17  1:32  23152  dce1-les.sggw.wa
mcj      + ttypb        Feb 25 17:27  0:03  23451  giswitch.sggw.wa
$
```

oznacza, że aktualnie w systemie pracuje 5 użytkowników o nazwach `root`, `user1`, `user2`, `user3` i `mcj`. `user1` zalogował się zdalnie, z komputera `IIIp313st1.sggw.waw.pl`, 25 lutego (`Feb 25`) o `17:12`, a ostatnio wpisał coś na klawiaturze `38` sekund temu. Administrator (`root`) siedzi w tej chwili przy konsoli (ang. `console`), czyli głównym terminalu sterującym.

3 System plików, katalogi

3.1 Wprowadzenie

System plików jest sposobem organizacji danych. Nazwa ta jest stosowana zarówno do rozwiązania na poziomie systemu operacyjnego (np. system plików FAT w DOS, HFS w HP-UX czy ext2 w Linux) jak i do sposobu zorganizowania informacji z punktu widzenia użytkownika. Zajmiemy się tu jedynie tym drugim przypadkiem.

Dane w większości systemów operacyjnych przechowywane są w plikach, pliki natomiast w katalogach. Katalogi mogą zawierać także następne katalogi często zwane podkatalogami. Dzięki temu powstaje hierarchiczna struktura katalogów - tzw. **hierarchiczny system plików** (zaprojektowany na Uniwersytecie Kalifornijskim w Berkeley).

Nie istnieje plik, który można by umieścić poza tą strukturą. W efekcie dany nośnik (np. dysk) musi mieć już założony jakiś katalog, aby można było na ten nośnik zapisać nasz plik. Istnieje więc pojedynczy katalog zwany **katalogiem głównym** (ang. **root** directory - katalog korzeń), w którym zawiera się cały system plików. Angielska nazwa katalogu głównego wynika z faktu, że system plików przypomina odwrócone drzewo, w którym katalogi spełniają rolę gałęzi a pliki liści. Drzewo jest odwrócone, więc katalog główny (korzeń) znajduje się na jego szczycie. Katalog główny jest oznaczany "`/`" (znakiem dzielenia). Taka jest jego nazwa i jest to jedyny katalog, którego nazwy nie można zmienić.

W un*x'ie nie ma oznaczeń dysków. Istnieje tylko jedna struktura katalogów, która obejmuje wszystkie dyski (nawet te, które są zamontowane w innych komputerach).

Struktura katalogów un*x'a jest bardzo rozbudowana - instalowany system nie jest umieszczany w jednym katalogu (jak np. DOS czy Windows 95) lecz w bardzo wielu. Istnieje duże podobieństwo w układzie i nazewnictwie preinstalowanych katalogów pomiędzy różnymi odmianami tego systemu. Ingerencja w tą strukturę wymaga sporej wiedzy od administratora. Nie jest ona jednak oczekiwana od użytkownika. Każdemu użytkownikowi podczas tworzenia konta jest zakładany **katalog użytkownika** (zwanym też **katalogiem domowym** od ang. "home directory" lub **macierzystym**). Katalog domowy należy tylko do tego jednego użytkownika, który jako jedyny może tam tworzyć nowe katalogi i umieszczać pliki. Nikt poza nim (i oczywiście administratorem) nie ma możliwości usunięcia czegokolwiek z tego katalogu (chyba, że administrator postanowi inaczej, ale to rzadki przypadek). W Linux'ie katalogi użytkowników są zakładane domyślnie w katalogu home (ang. users - użytkownicy) a w HP-UX w katalogu users. Katalogi użytkowników mają zwykle taką samą nazwę jak nazwa konta. Czasami może też być jakiś katalog pośredni dla oznaczenia grupy użytkowników. Po zalogowaniu, katalog użytkownika staje się jednocześnie **katalogiem bieżącym**. Inaczej mówiąc - rozpoczynamy pracę będąc w swoim katalogu domowym.

3.2 Katalog bieżący

Jest to ten katalog, w którym się w danej chwili znajdujemy, w którym pracujemy. Większość komend wiąże swoje działanie z katalogiem bieżącym. Np. komenda **`ls`** wpisana bez argumentów wyświetla właśnie zawartość katalogu bieżącego. Ograniczenie działania komend do katalogu bieżącego ma swój głęboki sens. Jeśli w 2 różnych katalogach mamy 2 pliki o tych samych nazwach to np. komenda kasowania pliku usunie jedynie ten z nich, który znajduje się w katalogu bieżącym, o drugim zaś nic nie będzie wiedziała. Niedostateczne rozumienie tej zasady może sprawiać kłopot użytkownikowi - jeśli wpisze naszą komendę kasowania w katalogu *nie* zawierającym kasowanego pliku, otrzyma jedynie komunikat typu "*nie ma takiego pliku*", bez żadnych dodatkowych wyjaśnień, że chodzi tu tylko o katalog bieżący - ten fakt jest z góry zakładany.

3.3 Ścieżki dostępu

To samo pojęcie występuje także w innych systemach operacyjnych. **Ścieżka dostępu** do pliku lub katalogu to inaczej droga, jaką należy przebyć po drzewie katalogów, aby przejść z jednego jego miejsca (katalogu) do drugiego. W ścieżce wymieniamy po kolei nazwy wszystkich katalogów, które "prowadzą" do danego pliku lub katalogu. Inaczej mówiąc, gdybyśmy zamierzali użyć komendy **`cd`**, aby przejść do danego katalogu to najszybciej byśmy to zrobili podając w komendzie **`cd`** te właśnie katalogi, które należą do danej ścieżki dostępu.

Wyróżniamy dwa rodzaje ścieżek dostępu: względne i bezwzględne.

Ścieżki bezwzględne zaczynają się zawsze od nazwy katalogu głównego "/" i identyfikują jednoznacznie dany plik/katalog niezależnie od tego, w którym katalogu w tej chwili się znajdujemy. Przykładem może być ścieżka `/users/plik.txt`. Użyta np. w komendzie kasowania spowoduje usunięcie pliku `plik.txt` umieszczonego w katalogu `users`, który znajduje się w katalogu głównym.

Ścieżki względne nigdy nie zaczynają się od nazwy katalogu głównego i identyfikują plik/katalog z punktu widzenia katalogu bieżącego. Jeśli znajdujemy się w katalogu "`/`" to ścieżka `users/plik.txt` oznacza ten sam plik, co w powyższym przykładzie. Jednak, gdy katalogiem bieżącym jest, np. `/users` to taka ścieżka odpowiada ścieżce bezwzględnej `/users/users/plik.txt`. Sama nazwa pliku lub katalogu jest przykładem najkrótszej możliwej ścieżki względnej - jest to ścieżka z katalogu bieżącego do pliku/katalogu, który się w nim właśnie znajduje.

Jak widać w powyższych przykładach, nazwy katalogów w ścieżkach oddzielamy znakiem "`/`" (dzielenia) a więc odwrotnie niż w DOS. Należy jednak pamiętać, że w ścieżkach bezwzględnych pierwszy znak "`/`" to nazwa katalogu głównego a nie po prostu znak rozdzielający.

Ścieżek dostępu używamy wszędzie tam gdzie należy podać nazwę katalogu lub pliku (np. w argumentach poleceń) a dany plik/katalog nie znajduje się w katalogu bieżącym. Wybieramy najczęściej ten rodzaj ścieżki (względna/bezwzględna), który wymaga w danej chwili mniejszej ilości znaków do wpisania (bądźmy leniwi). Zależy to od tego, jak "daleko" od siebie są katalog bieżący i docelowy. Ścieżki bezwzględne są najpewniejsze, względne natomiast wprowadzono, aby ułatwić życie leniwemu użytkownikowi.

3.4 Pliki a katalogi

"Wszystko w un*x'ie jest plikiem". Jest nim więc także katalog. Mimo że katalogi są inaczej traktowane przez różne komendy to jednak są to pliki o określonej zawartości. Jest to tabela, która w każdym wierszu zawiera, w poszczególnych kolumnach, informacje na temat każdego pojedynczego pliku (a więc i katalogu). Do informacji tych należą m.in. nazwa pliku, oznaczenie typu (plik zwykły, specjalny, katalog), prawa dostępu, nazwa właściciela, data modyfikacji,

rozmiar i kilka innych. Jeśli więc nazwa pliku znajduje się w tej tabeli, mówimy, że plik znajduje się w tym katalogu.

Rozumienie powyższego rozwiązania znacznie ułatwia posługiwanie się komendami operującymi na strukturze katalogów oraz prawami dostępu.

W każdej tablicy katalogu (poza /) znajdują dwa wpisy o nazwach `.` (kropka) i `..` (dwie kropki). Jako że zaczynają się od kropki nie są zwykle pokazywane. Powstają one automatycznie w momencie zakładania katalogu. "`.`" oznacza katalog bieżący (wbrew pozorom czasami się przydaje) a "`..`" **katalog nadrzędny**, czyli ten, w którym znajduje się katalog z owym wpisem. Obu oznaczeń używa się we względnych ścieżkach dostępu (choć "`.`" znacznie rzadziej).

3.5 Katalogi

Co można zrobić z katalogami?

- dowiedzieć się, jaka jest nazwa katalogu bieżącego (komenda **pwd**)
- co jest w katalogu bieżącym lub jakimkolwiek innym (**ls**)
- zmienić katalog bieżący (**cd**)
- utworzyć (**mkdir**)
- usunąć (**rmdir**)
- zmienić nazwę (**mv**)

Po rozpoczęciu sesji użytkownik znajduje się w swoim katalogu domowym, który jest mu przypisany w momencie zakładania konta. W katalogu tym użytkownik ma pełne prawa, tzn. może w nim tworzyć pliki i katalogi, modyfikować je, usuwać, kopiować, nadawać prawa itd.

Polecenia systemowe znajdują się w katalogach: `/bin`, `/usr/bin` i `/usr/contrib/bin`. Wiedza ta zwykle nie jest potrzebna, ponieważ system sam znajduje komendę w odpowiednim katalogu po wpisaniu jej nazwy przez użytkownika.

3.6 pwd

Do sprawdzania, w jakim katalogu użytkownik znajduje się w danym momencie (jaki katalog jest katalogiem bieżącym) służy polecenie **`pwd`**. Jego wynik dla użytkownika **`misioo`** zaraz po jego zalogowaniu będzie następujący:

```
$ pwd
/users/misioo
$
```

3.7 ls

Aby przejrzeć zawartość katalogu bieżącego (o czym było już wspomniane w części poświęconej wydawaniu poleceń) należy wydać polecenie **`ls`**:

```
$ ls
index.html  plik.txt    prace       text.doc
$
```

Wspominaliśmy również o możliwości wydawania tego polecenia z opcjami (**`ls -l, ls -al`**. Odpowiednikiem polecenia **`ls -l`** jest **`ll`**:

```
$ ll
total 8
-rw-r--r--   1 misioo   inni       46 Feb 23 12:21 index.html
```

```
-rw-r--r--   1 misioo   inni       75 Feb 23 12:21 plik.txt
drwxr-xr-x   2 misioo   inni       24 Feb 23 12:15 prace
-rw-r--r--   1 misioo   inni      137 Feb 23 12:22 text.doc
$
```

Dzięki temu można skrócić tę dość często wydawaną komendę. Polecenie **`ll`** można uzupełniać o wszelkie inne opcje właściwe **`ls`**. Należy tu też zwrócić uwagę na wielkość katalogu `prace`, podaną w 5-tej kolumnie. Nie jest to wielkość wszystkich zawartych w nim plików, lecz wielkość, jaką zajmuje plik katalogu. Do znaczenia innych kolumn jeszcze wrócimy.

Ciekawą modyfikacją polecenia **`ls`** jest wydanie go z opcja **`-F`**, co umożliwia nam stwierdzenie, czy pośród plików w katalogu bieżącym znajdują się katalogi, a nie jest dla nas niezbędnym znajomość pozostałych opisów plików. Ma to szczególne znaczenie wtedy, gdy plików w katalogu jest dużo, a my (na razie :) nie wiemy, jak sobie z tym poradzić:

```
$ ls -F
index.html   plik.txt     prace/       text.doc
$
```

Krótszą formą polecenia **`ls -F`** jest **`lsf`**. Tak samo jak przy **`ll`** można do niego dodawać inne opcje.

Zauważyć można (czego nie widać przy samej komendzie **`ls`**), że plik `prace` to tak naprawdę jest katalog (`prace/`). Katalogi oznaczane są znakiem `/` a pliki wykonywalne (programy) znakiem `*` (gwiazdka).
Wspominaliśmy również, że w katalogu mogą znajdować się pliki ukryte - te uwidaczniają się po wydaniu polecenia **`ls`** z opcją **`-a`** lub **`-f`** :

```
$ ls -a
.             .cshrc       .profile     .vueprofile  prace
..            .exrc        .sh_history  index.html   text.doc
.Xauthority   .login       .vue         plik.txt
$
```

Widzimy tu też specjalne "kropkowe" katalogi: `.` i `..`.

W praktyce, najczęściej używanymi komendami do przeglądania zawartości katalogów są **`lsf`** i **`ll`**. Dla utrwalenia: **`lsf -l`** oraz **`ll -F`** wykonują dokładnie to samo. Często też podaje się przy tych poleceniach argument, czyli ścieżkę dostępu do katalogu.

3.8 cd

Do zmiany katalogu bieżącego służy polecenie **`cd`** (od ang. **c**hange **d**irectory - zmień katalog), które wydawać można z argumentami lub bez. Argumentem polecenia **`cd`** jest katalog docelowy, do którego chcemy się przemieścić (który chcemy uczynić katalogiem bieżącym). Prześledźmy działanie tej komendy z podstawowymi argumentami.
Argument **`..`** oznacza katalog znajdujący się wyżej w hierarchii katalogów. Wydanie więc polecenia

```
$ cd ..
$
```

oznacza przeniesienie się do katalogu, /users, co sprawdzamy komendą **`pwd`**:

```
$ pwd
/users
```

```
$
```

Aby powrócić do katalogu domowego (i to niezależnie od miejsca, w którym się aktualnie znajdujemy) należy wydać komendę **cd** bez żadnych argumentów (komendę wykonuje użytkownik misioo):

```
$ cd
$ pwd
/users/misioo
$
```

Wspomnieliśmy już wcześniej, że argumentem polecenia **cd** jest ścieżka prowadząca do katalogu, który chcemy uczynić bieżącym. Ścieżka ta może być podana albo względem katalogu bieżącego (ścieżka względna) albo w sposób bezwzględny (tzn. od katalogu głównego). Aby znaleźć się w katalogu /etc (czyli w podkatalogu etc katalogu /) należy wydać polecenie z odpowiednim argumentem:

```
$ cd /etc
$ pwd
/etc
$
```

Rzeczywiście znaleźliśmy się w katalogu /etc! Odpowiada to poruszaniu się z katalogu /users/misioo w następujący sposób:

```
$ pwd
/users/misioo
$ cd ../../etc
$ pwd
/etc
$
```

Oznacza to: zmień katalog na nadrzędny w stosunku do /users/misioo, czyli na /users, potem o jeszcze jeden wyżej, czyli do /, a z tego miejsca - do katalogu etc. Widać, że w niektórych przypadkach podawanie ścieżek bezwzględnych może być zdecydowanie krótsze. Wydając komendę **cd** bez argumentów można znów w prosty sposób znaleźć się w katalogu domowym.

3.9 Nazwy plików i katalogów

Aby nie tylko poruszać się po istniejącej strukturze, ale i ją modyfikować (oczywiście w zakresie dopuszczonym przez administratora, a więc w praktyce - w obrębie katalogu domowego), musimy omówić podstawowe reguły nazywania plików i katalogów oraz sposoby tworzenia i usuwania katalogów. Nazwy plików (a więc i katalogów) w un*x'ie mogą być o wiele dłuższe, niż np. w systemie DOS. Mogą one zawierać wielkie i małe litery, liczby, dowolną ilość kropek, myślniki, podkreślenia, przecinki itd. Mogą zawierać nawet 255 znaków! W katalogu /home/misioo utworzyłem "dziwny" plik, który to ilustruje:

```
$ ls
index.html
plik.txt
prace
text.doc
```

```
to.jest.1-z-przykladow.ze.pliki-w.UnIxIe.moga.miec-wiecej.NIZ_8_znakow
$
```

Rozszerzenia, jakie mogą zawierać nazwy plików, mają znaczenie tylko informacyjne i porządkujące - dla systemu nie ma to większego znaczenia. Teoretycznie dowolny znak może wystąpić w nazwie. Niektóre jednak znaki (jak `&`, `>`, `<`, `!`, `*`, `?`, `[`, `]`, spacja, tab, `@`, `#`, `$`, `^`, `(`, `)`, `'`, `"`, `` ` ``, `|`, `/`, `\` i `;`) mają specjalne znaczenie dla shell'a więc lepiej ich unikać. W nazwach plików nie należy używać polskich znaków - może to znacznie utrudnić posługiwanie się nimi. Nie należy stosować `-` (minusa) ani `+` (plusa) jako pierwszego znaku nazwy - polecenia potraktują taką nazwę jako zestaw opcji a nie argument. Kropka na pierwszej pozycji spowoduje utworzenie pliku ukrytego.

3.10 Tworzenie i kasowanie katalogów - mkdir i rmdir

Do tworzenia nowych katalogów służy polecenie **`mkdir`** (z ang. **m**a**k**e **dir**ectory - "utwórz katalog"). Argumentem tego polecenia jest nazwa katalogu, który chcemy utworzyć (warto później sprawdzić, czy to faktycznie działa):

```
$ mkdir przyklad
$ lsf
index.html          prace/              text.doc
plik.txt            przyklad/
$
```

Katalog ten można uczynić bieżącym:

```
$ cd przyklad
$ pwd
/users/misioo/przyklad
$
```

Zwróćmy uwagę, że argumentem polecenia **`cd`** jest tu po prostu nazwa katalogu - oznacza to, że katalog, do którego się przemieszczamy, znajduje się w katalogu bieżącym. Identyczny efekt da wydanie komendy z innymi argumentami (za każdym razem wydawana jest komenda **`pwd`**, aby sprawdzić, w jakim katalogu znajduje się użytkownik `misioo`:

```
$ pwd
/users/misioo
$ cd ./przyklad
$ pwd
/users/misioo/przyklad
$ cd
$ pwd
/users/misioo
$ cd /users/misioo/przyklad
$ pwd
/users/misioo/przyklad
$
```

W pierwszym przypadku podajemy systemowi wprost, że ma zmienić katalog na `przyklad` znajdujący się w katalogu bieżącym (oznaczanym przecież jako `..` W drugim przypadku (po

powrocie do katalogu domowego) argumentem polecenia **cd** jest bezwzględna ścieżka do katalogu `przyklad`. Przykłady te pokazują, w jakich przypadkach można opuszczać cześć ścieżki do danego katalogu.
Usunięcie katalogu (pod warunkiem, że nie zawiera on żadnych plików) wykonuje się przy pomocy polecenia **rmdir** (z ang. **rem**ove **dir**ectory - "usuń katalog"). Argumentem komendy jest (podobnie, jak przy tworzeniu katalogu) nazwa katalogu (ewentualnie wraz ze ścieżką), który chcemy usunąć:

```
$ lsf
index.html          prace/              text.doc
plik.txt            przyklad/
$ rmdir przyklad
$ lsf
index.html          prace/
plik.txt            text.doc
$
```

Katalog `przyklad` został pomyślnie usunięty. Nazwę katalogu do usunięcia można podawać również razem ze ścieżką dostępu (**rmdir ./przyklad** lub **rmdir /users/misioo/przyklad**), co da taki sam efekt. Trzeba jednak pamiętać, że żaden katalog nie może zostać usunięty, jeśli znajduje się między katalogiem głównym i bieżącym. Nie zadziała więc np. komenda **rmdir ..** lub **rmdir .**. **rmdir** usuwa tylko puste katalogi. Jeśli katalog nie jest pusty trzeba z niego wcześniej usunąć wszystkie pliki. Można też posłużyć się poleceniem **rm**, które potrafi usuwać niepuste katalogi, ale o tym później.

3.11 mv

Polecenie mv (ang. **m**o**v**e - ruszaj) służy do zmiany nazwy pliku lub katalogu. Jego uproszczona składnia to:

```
mv [-i] nazwa_oryginalna nazwa_docelowa
```

Należy z nim bardzo uważać, ponieważ jest to polecenie stosowane także do przenoszenia plików między katalogami. Aby zmieniło nazwę musimy podać dokładnie 2 argumenty a oba muszą być katalogami lub oba plikami. Przydatną opcją jest `-i`, która powoduje, że w razie "nieprzewidzianych okoliczności" `mv` poprosi nas o potwierdzenie wykonania operacji. Przykład:

```
$ mv przyklad proba
$ lsf
index.html          prace/              text.doc
plik.txt            proba/
$
```

4 System plików, pliki

4.1 Plik

Plik jest ciągiem informacji/danych zapisanych na dysku, posiadającym następujące atrybuty:

- nazwa
- rozmiar

- data ostatniej modyfikacji zawartości
- typ (katalog, plik zwykły lub specjalny)
- prawa dostępu
- właściciel i grupa
- liczba dowiązań (mówi ile jest różnych ścieżek dostępu do tego pliku)

Wszystkie te cechy pokazuje komenda **`ll`** (**`ls -l`**). Opisaliśmy ją poprzednich rozdziałach. Tymczasem dowiedzmy się, co można z plikiem zrobić:

- obejrzeć zawartość (komendy **`cat`**, **`more`**, **`tail`**, **`head`**)
- zrobić kopię (skopiować) (**`cp`**)
- zmienić nazwę (**`mv`**)
- przenieść do innego katalogu (**`mv`**)
- usunąć (**`rm`**)
- utworzyć inną nazwę tego samego pliku (**`ln`**)
- utworzyć plik (**`cat`**, **`touch`**, **`vi`**, **`pico`** i inne programy)
- zmienić jego zawartość (**`vi`**, **`pico`** i inne programy)

4.2 Oglądanie zawartości pliku

4.2.1 cat

Do wyświetlania na ekranie zawartości pliku służy polecenie **`cat`** (od ang. con**cat**enate), które wydajemy w następujący sposób:

```
cat plik
```

Argumentem polecenia jest więc nazwa pliku, którego zawartość chcemy obejrzeć. Jeżeli jako argument podamy nazwy kilku plików, system połączy je i wyświetli na ekranie jako jeden długi plik. Przy poleceniu `cat` nie jest konieczne używanie żadnych dodatkowych opcji. Przykład użycia polecenia **`cat`** może wyglądać tak:

```
$ cat test.txt
12
13
[...]
33
34
i jeszcze byloby kilka, ale ich nie ma...
$
```

Niestety - początek tekstu przewinął się przez ekran, więc nie ma możliwości zobaczenia, jak się plik zaczyna. Do przeglądania zawartości pliku "strona po stronie" służy polecenie **`more`**.

4.2.2 more

Polecenie `more plik...` (ang. more - więcej) umożliwia wyświetlenie zawartości pliku podzielonej na strony mieszczące się na ekranie:

```
$ more test.txt
Policzymy teraz linie...
1
```

```
2
[...]
21
22
--More--(50%)
```

Na dole ekranu pojawia się napis `--More--(50%)`, który informuje, że została wyświetlona tylko część pliku (50%). Następną stronę obejrzymy naciskając `spację`. Przesunięcie tekstu o jedną linię w dół nastąpi po naciśnięciu klawisza `Enter`. Wyjście z trybu oglądania do shell'a uzyskamy po naciśnięciu litery `q`. Są to więc takie same klawisze, jakie pojawiły się przy omawianiu polecenia **`man`**.

4.2.3 tail

W niektórych przypadkach chcielibyśmy obejrzeć tylko końcówkę pliku (kilka ostatnich linii). W tym przypadku posłużyć się można poleceniem `tail [-x] plik` (od ang. tail - ogon). Polecenie bez opcji wyświetla ostatnie 10 linijek tekstu:

```
$ tail test.txt
26
27
28
29
30
31
32
33
34
i jeszcze byloby kilka, ale ich nie ma...
$
```

Liczbę tę można zmodyfikować podając jako opcję ilość linijek do wyświetlenia:

```
$ tail -3 test.txt
33
34
i jeszcze byloby kilka, ale ich nie ma...
$
```

4.2.4 head

`head [-x] plik` (od ang. head - głowa, nagłówek) jest poleceniem podobnym do **`tail`**, ale wyświetlającym początkowe linie pliku (domyślnie 10):

```
$ head test.txt
Policzymy teraz linie...
1
2
```

```
3
4
5
6
7
8
9
$
```

I w tym przypadku opcją polecenia jest liczba linii do wyświetlenia:

```
$ head -2 test.txt
Policzymy teraz linie...
1
$
```

4.3 Kopiowanie (cp)

Jak w każdym systemie operacyjnym jedną z podstawowych operacji na plikach jest ich kopiowanie. Do tego celu służy komenda **cp** (od ang. **c**o**p**y - kopiuj). Składnia polecenia jest następująca:

```
cp plik [plik...] cel
```

Jako argumentów należy więc użyć nazwy kopiowanego pliku lub plików oraz nowej nazwy pliku lub nowego miejsca, gdzie kopia ma się znajdować. W wyniku działania polecenia uzyskujemy:

```
$ ll
total 4
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 index.html
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni        24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni       161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni         0 Mar  5 08:20 text.doc
$ cp test.txt kopia.pliku
$ ll
total 6
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 index.html
-rw-r--r--   1 misioo   inni       161 Mar 10 20:14 kopia.pliku
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni        24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni       161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni         0 Mar  5 08:20 text.doc
$
```

4.4 Zmiana nazwy i przenoszenie (mv)

Do fizycznego przenoszenia pliku w inne miejsce lub pod inną nazwę służy polecenie **mv** (od ang. **m**o**v**e - przenieś). Składnia polecenia jest podobna, jak przy **cp**

```
mv plik [plik...] cel
```

```
$ mv kopia.pliku nowy.plik
$ ll
total 6
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 index.html
-rw-r--r--   1 misioo   inni       161 Mar 10 20:14 nowy.plik
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni        24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni       161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni         0 Mar  5 08:20 text.doc
$
```

Na przykładzie widać, że `kopia.pliku` występuje teraz pod nazwą `nowy.plik`. Stąd polecenie służyć może również do zmiany nazwy pliku.

4.5 Usuwanie (rm)

Usuwanie plików możliwe jest dzięki poleceniu **rm** (od ang. **re**m**o**ve - usuń, wymaż). Składnia polecenia jest następująca:

```
rm [-i] plik...
```

Wydanie polecenia bez opcji powoduje usuniecie pliku bez ostrzeżenia, stąd też należy używać tej komendy z rozwagą (tym bardziej, że w HP-UX nie ma polecenia typu undelete czy t.p.):

```
$ ls
index.html  nowy.plik   plik.txt    prace     test.txt   text.doc
$ rm nowy.plik
$ ls
index.html  plik.txt    prace       test.txt    text.doc
$
```

Jeżeli chcemy zabezpieczyć się (przynajmniej częściowo) przed np. błędami typograficznymi (szczególnie przy kasowaniu dużej ilości plików), warto używać opcji **-i** (od ang. **i**nteractive - interaktywnie), która powoduje wyświetlanie pytania o potwierdzenie operacji przy każdym kasowanym pliku:

```
$ rm -i kasowany.plik
kasowany.plik: ? (y/n) y
$
```

Usunięcie kasowanego pliku nastąpi dopiero po potwierdzeniu naszego zamiaru przez `y` (yes).

4.6 Tworzenie nowych nazw (ln)

Ciekawą własnością un*x'a jest możliwość tworzenia kilku nazw tego samego pliku (lub katalogu), przy czym nazwy te mogą znajdować się w zupełnie innym katalogu niż plik macierzysty. Takie połączenie nazw z plikiem nosi nazwę `linku`. Rozpoznać je można przy okazji wyświetlania zawartości katalogu po oznaczeniu `l` znajdującym się na pierwszym miejscu opisu typu pliku lub/oraz liczbie wyświetlanej w 2-giej kolumnie. W zależności od potrzeb możliwe jest tworzenie linków "twardych" (hard link) oraz symbolicznych. Od tej chwili zmiany dotyczące pliku dotyczą

również linków i odwrotnie.
Do tworzenia nowych nazw plików (katalogów) służy polecenie **ln** (od ang. **li**n**k** - łącz). Składnia polecenia, które tworzy nową nazwę `plik2` dla pliku `plik1` wygląda następująco:

```
ln [-s] plik1 plik2
```

Polecenie **ln** bez opcji tworzy "hard link", natomiast dodanie opcji **-s** pozwala na utworzenie linku symbolicznego:

```
$ ll
total 4
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 index.html
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni        24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni       161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni         0 Mar  5 08:20 text.doc
$ ln -s index.html link
$ ll
total 6
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 index.html
lrwxr-xr-x   1 misioo   inni        10 Mar 11 11:06 link -> index.html
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni        24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni       161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni         0 Mar  5 08:20 text.doc
$
```

Usunięcie pliku `index.html`, na który wskazuje `link` nie powoduje usunięcia samego linku symbolicznego (ale zawartość pliku jest już stracona). Jeżeli ponownie utworzymy plik o nazwie `index.html`, to `link` nadal będzie na niego wskazywał.
Nieco inaczej to wygląda w przypadku tworzenia linku "twardego". Usunięcie pliku `index.html` również nie spowoduje usunięcia linku `link.twardy`, ale w przypadku ponownego utworzenia pliku `index.html` `link.twardy` nie będzie linkiem do pliku `index.html`, lecz nadal zawierał będzie to, co znajdowało się w STARYM pliku `index.html`:

```
$ ll
total 6
-rw-r--r--   1 misioo   inni        11 Mar 11 11:18 index.html
-rw-r--r--   1 misioo   inni         0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni        24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni       161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni         0 Mar  5 08:20 text.doc
$ ln index.html link.twardy
$ ll
total 8
```

```
-rw-r--r--   2 misioo   inni          11 Mar 11 11:18 index.html
-rw-r--r--   2 misioo   inni          11 Mar 11 11:18 link.twardy
-rw-r--r--   1 misioo   inni           0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni          24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni         161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni           0 Mar  5 08:20 text.doc
$ rm index.html
$ ll
total 6
-rw-r--r--   1 misioo   inni          11 Mar 11 11:18 link.twardy
-rw-r--r--   1 misioo   inni           0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni          24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni         161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni           0 Mar  5 08:20 text.doc
$ touch index.html
$ ll
total 6
-rw-r--r--   1 misioo   inni           0 Mar 11 11:21 index.html
-rw-r--r--   1 misioo   inni          11 Mar 11 11:18 link.twardy
-rw-r--r--   1 misioo   inni           0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni          24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni         161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni           0 Mar  5 08:20 text.doc
$
```

4.7 Tworzenie plików (touch, cat)

Nowe pliki mogą być tworzone na wiele sposobów. Jednym z nich jest wykorzystanie polecenia `touch plik`.... Argumentem polecenia jest nazwa pliku, który chcemy utworzyć. Jeżeli pliku o podanej nazwie nie ma - zostanie on utworzony jako pusty, natomiast, jeżeli `plik` już istnieje - zostanie zmieniona jego data ostatniego dostępu:

```
$ lsf
index.html   plik.txt     prace/     test.txt    text.doc
$ touch nowy.plik
$ ll
total 4
-rw-r--r--   1 misioo   inni           0 Mar 11 11:21 index.html
-rw-r--r--   1 misioo   inni           0 Mar 11 11:46 nowy.plik
-rw-r--r--   1 misioo   inni           0 Mar  5 08:19 plik.txt
drwxr-xr-x   2 misioo   inni          24 Mar  5 09:43 prace
-rw-r--r--   1 misioo   inni         161 Mar 10 19:19 test.txt
-rw-r--r--   1 misioo   inni           0 Mar  5 08:20 text.doc
```

```
$ touch text.doc
$ ls -l text.doc
-rw-r--r--   1 misioo   inni          0 Mar 11 11:46 text.doc
$
```

Inną możliwością utworzenia pliku jest wykorzystanie znanego już polecenia **cat** w postaci:

```
cat > plik
```

Tym razem tworzony plik nie musi być pusty - system czeka bowiem na wprowadzenie tekstu, który zawierać ma nowy plik. Wprowadzanie tekstu kończymy kombinacją klawiszy Control i d (czyli ^D), co przez system rozumiane jest jako koniec pliku:

```
$ cat > plik.z.cat
Ala ma kota ^D
$ ls
index.html  plik.txt    prace       text.doc
nowy.plik   plik.z.cat  test.txt
$ cat plik.z.cat
Ala ma kota
$
```

Nowoutworzony plik.z.cat zawiera dokładnie to, co wprowadziliśmy do niego z klawiatury.

Kolejnym, chyba najczęściej używanym, sposobem tworzenia plików jest posłużenie się edytorem tekstu, np. dostępnym praktycznie w każdym systemie un*x'owym vi czy instalowanym wraz z programem pocztowym pine programem pico.

4.8 Edycja (vi, pico)

Najczęściej używanym edytorem dostępnym w un*x'ie jest **vi** (czyt. wi aj ;)). Jego popularność nie wynika bynajmniej z prostoty obsługi, ale z faktu, że znajduje się on w każdym un*x'ie i, wbrew pozorom, posiada naprawdę bardzo duże możliwości. Aby utworzyć lub wyedytować plik należy użyć polecenia vi wydanego z argumentem - nazwą pliku:

```
vi plik
```

Znacznie prostszym programem jest pico. Uruchamia się go podobnie jak vi:

```
pico plik
```

5 Prawa dostępu, znowu shell

5.1 Właściciele, grupy i prawa dostępu

Un*x, jako system przeznaczony dla wielu użytkowników, posiada wbudowane mechanizmy pozwalające na określanie przez administratora i użytkowników, kto ma prawo korzystać z plików i katalogów. Mechanizmy te oparte są na prawach dostępu do plików i katalogów oraz na informacji o tym, kto jest ich właścicielem i do jakiej grupy należą. Przynależność do grup pozwala na sterowanie prawami dostępu dla większej liczby użytkowników. Dane te uzyskać można przy pomocy znanego już polecenia **ll**:

```
$ ll
total 4
```

```
drwxr-xr-x   2 misioo   inni        24 Mar 18 06:13 katalog
-rw-r--r--   1 misioo   inni       133 Mar 18 06:13 plik.txt
-r--------   1 misioo   inni         0 Mar 18 06:21 top.secret
$
```

Poszczególne znaki pierwszej kolumny każdej linii informują o:

- typie pliku (pierwszy znak, nie ma nic wspólnego z samymi prawami),
- prawach właściciela do pliku/katalogu (trzy następne),
- prawach do pliku/katalogu dla danej grupy (trzy następne),
- prawach do pliku/katalogu dla pozostałych użytkowników (trzy ostatnie).

Nazwa właściciela jest wyświetlana w trzeciej kolumnie a nazwa grupy w czwartej. Pole zapisu praw ma zawsze 9 znaków. W zapisie tym przyjęta została następująca konwencja:

- `r` (ang. **r**ead) oznacza prawo do czytania pliku (a więc i katalogu),
- `w` (ang. **w**rite) oznacza prawo do zapisu/modyfikacji,
- `x` (ang. e**x**ecute) oznacza prawo do wykonywania,
- `-` (minus) oznacza brak określonego prawa (zapisu, czytania lub wykonywania).

W poszczególnych miejscach zapisu może się pojawić albo jeden ze znaków posiadania danego prawa (`r`, `w` lub `x`) albo znak `-` . Tak więc na pozycji 1-szej, 4-tej i 7-mej może być tylko `r` lub `-`, na pozycji 2-giej, 5-tej, lub 8-mej `w` lub `-` a na pozycjach 3-ciej, 6-tej i 9-tej tylko `x` lub `-`. W rzeczywistości mogą pojawiać się jeszcze inne literki, ale w codziennej pracy się ich nie stosuje.

Poglądowy rysunek wyjaśniający znaczenie kolejnych znaków znaleźć można w podręczniku systemowym w rozdziale dotyczącym komendy **`ls`** (**`man ls`**).

Prawa dostępu mają nieco inne znaczenie dla plików i katalogów:

- Prawo do **czytania** w odniesieniu do pliku oznacza, że jego zawartość może być przez użytkownika wyświetlona np. za pomocą polecenia `cat`. To samo prawo w odniesieniu do katalogu oznacza, że użytkownik ma prawo do poznania zawartości katalogu, czyli wykonania polecenia **`ls`**. Zwróćmy uwagę na fakt, że jeżeli katalog traktujemy jako plik zawierający informacje o zawartych w nim plikach, to jest to zupełnie logiczne.
- Prawo do **zapisu** dla pliku oznacza, że zawartość pliku może być przez użytkownika modyfikowana. To samo prawo w odniesieniu do katalogu oznacza, że użytkownik może w nim tworzyć nowe pliki czy katalogi lub usuwać istniejące. I jeżeli znowu zakładamy, że traktujemy katalog jako plik z informacjami o plikach w nim zawartych, prawo do zapisu to tak naprawdę prawo do modyfikacji tych informacji, czyli do modyfikacji pliku-katalogu.
- Prawo do **wykonywania** dla pliku oznacza, że może on być wykonywany ;) np. jako program lub skrypt (nie musi mieć specjalnego rozszerzenia jak w DOS). W odniesieniu do katalogu prawo to pozwala na uczynienie go katalogiem bieżącym (poleceniem `cd`). Brak tego prawa powoduje, że nie można do katalogu "wejść".

Nowotworzone pliki i katalogi posiadają pewne domyślne prawa dostępu (ustalone przez administratora) oraz określonego właściciela i grupę. Właścicielem pliku/katalogu staje się ten, kto go tworzy. Grupa pliku będzie taka, jak grupa tworzącego plik/katalog (w rzeczywistości nie jest to takie oczywiste, ponieważ dany użytkownik może należeć do kilku grup). Żeby sprawdzić swoją przynależność, można użyć polecenia **`id`**, które wyświetla podstawowe informacje o użytkowniku:

```
$ id
```

```
uid=300(misioo) gid=203(inni)
$
```

czyli `uid` (**u**ser **id**entification number) i (w nawiasie) nazwę użytkownika oraz gid (**g**rup **id**entification number) i nazwę grupy, do której należy użytkownik. `uid` i `gid` nie mają większego znaczenia dla zwykłego użytkownika.

Nowy plik zostanie utworzony z prawami `rw-r--r--`, co oznacza, że właściciel (`misioo`) ma prawo do czytania i modyfikacji pliku a domyślnie nie ma prawa do jego wykonywania (choć może to zmienić), członkowie grupy `inni` mają prawo do czytania pliku i nie mają prawa do jego modyfikacji i wykonywania a pozostali użytkownicy (czyli nie-`misioo` ;) i nie należący do grupy `inni`) mają też tylko prawo do czytania (bez prawa do zapisu i wykonywania).

Nowy katalog zostanie utworzony z prawami `rwxr-xr-x`, co oznacza, że użytkownik (właściciel) ma pełne prawa do katalogu (prawo do czytania, modyfikacji i wykonywania, czyli do przeglądania jego zawartości, tworzenia i kasowania w nim plików oraz uczynić go bieżącym/wykonać polecenie cd) a grupa i pozostali użytkownicy systemu mają prawo do czytania i wykonywania.

5.2 Zmiana praw do plików/katalogów - chmod

Aby system praw dostępu miał sens zmiana praw do plików i katalogów jest przywilejem właściciela i administratora. Do zmiany praw służy polecenie **`chmod`** (od ang. **ch**ange **mod**e). Polecenia tego używać można dwojako: w sposób symboliczny i bezwzględny.
Zmiana uprawnień w **sposób symboliczny** ma następującą postać ogólną:

```
chmod kto operator uprawnienia nazwa
```

Zmiana wymaga więc określenia:

- czyje uprawnienia chcemy zmienić,
- jak chcemy je zmienić,
- na jakie uprawnienia chcemy dokonać zmian,
- w stosunku, do jakiego pliku/katalogu.

Do określenia, czyich uprawnień dotyczyć ma modyfikacja, służą następujące znaki:

- **`u`** (user) - użytkownik, właściciel,
- **`g`** (group) - grupa,
- **`o`** (others) - inni użytkownicy,
- **`a`** (all) - wszyscy (users, group i others).

Zmianę określa się za pomocą operatorów poprzedzających określone prawa (r,w lub x):

- **`+`** oznacza dodanie prawa,
- **`-`** oznacza cofnięcie prawa,
- **`=`** oznacza ustalenie nowego prawa bez względu na stan poprzedni.

Zmiana uprawnień w **sposób bezwzględny** ma następującą postać ogólną:

```
chmod uprawnienia nazwa
```

Zmiana wymaga więc określenia:

- na jakie uprawnienia chcemy dokonać zmian,
- w stosunku, do jakiego pliku/katalogu.

Zmiany uprawnień dokonuje się od razu dla wszystkich (u, g i o) za pomocą liczby (np. 755), którą tworzy się według następującego schematu:
Każdemu prawu przypisane są określone wartości:

- prawo do czytania dla właściciela = 400
- prawo do zapisu dla właściciela = 200
- prawo do wykonywania dla właściciela = 100
- prawo do czytania dla grupy = 40
- prawo do zapisu dla grupy = 20
- prawo do wykonywania dla grupy = 10
- prawo do czytania dla pozostałych = 4
- prawo do zapisu dla pozostałych = 2
- prawo do wykonywania dla pozostałych = 1

Bezwzględne prawo do pliku/katalogu uzyskuje się dodając do siebie poszczególne wartości, np. prawo rwxr-xr-x zapisuje się: 400+200+100+40+10+4+1=**755**, prawo r-x------ uzyskuje się: 400+100=**500**.
Można przedstawić to w postaci diagramu (źródło - arexus@tande.com):

```
+-+-+-+-+-+-+-+-+
|r|w|x|r|w|x|r|w|x| pełne uprawnienia dla wszystkich
+-+-+-+-+-+-+-+-+
 4+2+1 4+2+1 4+2+1
   7     7     7    czyli chmod 777 plik0


+-+-+-+-+-+-+-+-+
|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
```

Polecenie **chmod** można wykonywać z parametrem '-R' (z ang. recursive), czyli wszystkie pliki i podkatalogi katalogu, który jest argumentem polecenia uzyskają takie same, określone przez użytkownika, prawa.

5.3 Zmiana właściciela pliku/katalogu - chown

Jak wspomniano wcześniej, istotnym elementem mechanizmów kontroli praw dostępu jest pojęcie właściciela pliku/katalogu i podział na grupy. Atrybuty te mogą być również zmienione. Do zmiany właściciela (lub właściciela i grupy) pliku służy polecenie **chown** (z ang. **ch**ange **own**er - zmień właściciela). Ogólna postać polecenia jest następująca:

```
chown user:grupa nazwa
```

Istnieje tu jednak pewne ograniczenie - trzeba być właścicielem by zmienić właściciela.

5.4 Zmiana grupy pliku/katalogu - chgrp

Do zmiany grupy pliku/katalogu służy polecenie **chgrp** (z ang. **ch**ange **gr**ou**p** - zmień grupę). Polecenie to ma następującą postać:

```
chgrp grupa nazwa
```

Nie trzeba być w grupie, aby zmienić grupę (trzeba być owner'em, czyli właścicielem).

5.5 Użyteczne funkcje shell'a

Shell ma wiele wbudowanych funkcji, które znacznie ułatwiają pracę z systemem. O wszystkich tych możliwościach można poczytać wydając komendę **man ksh** (lub **man inny_shell**). Poniżej zostaną omówione najbardziej użyteczne z nich (choć wybór był bardzo trudny :) w ksh. Shell **bash** ma takie same możliwości z tym, że niektóre operacje można wykonać znacznie prościej.

5.5.1 Generowanie nazw plików

Generowanie nazw plików (określane jako "pathname expansion" lub "file name generation") to funkcja shell'a, która pozwalają na krótsze zapisanie nazw plików (a więc i katalogów) we wpisywanych komendach. Znaki generujące są interpretowane przez shell, dzięki czemu możemy ich używać z dowolnymi komendami (a nie jak w DOS, gdzie dana komenda oddzielnie musi być tak napisana, aby je sama rozpoznawała). Generalnie, zamiast wpisywać w danej komendzie (np. kopiowania) wszystkie nazwy plików, używamy znaków generujących. Znaków tych jest kilka jednak my przedstawimy tu tylko dwa: `?` (znak zapytania) i `*` (gwiazdka). O innych można poczytać po wydaniu polecenia **man ksh**.

- `?` zastępuje dowolny pojedynczy znak występujący w nazwie pliku
- `*` zastępuje 0 lub więcej znaków

Oba znaki nie zastępują "`.`" (kropki) jeśli jest to pierwsza kropka w nazwie pliku. Poza tym jednym wyjątkiem zastępują wszystkie inne kropki (a więc inaczej niż w DOS). Można jest stosować oddzielnie albo łącznie, mogą też wystąpić wielokrotnie w jednej komendzie.

Zanim zobaczymy jak to działa utwórzmy kilka plików:

```
$ touch 1a 2a ba a ab abc abb test.txt
$ mkdir Mail
$ touch Mail/moje_listy
$ ls
1a        Mail      ab        abc       test.txt
2a        a         abb       ba
$
```

Wykorzystując znak `?` wpiszmy teraz kilka komend dla plików o nazwach złożonych:

- z dwóch znaków,

```
$ ls ??
1a  2a  ab  ba
```

```
$
```

- z dwóch znaków, ale o drugiej literze "a" ,

```
$ cd ..
$ ll mcj/?a
-rw-r--r--   1 mcj      informix       0 Mar 18 16:42 1a
-rw-r--r--   1 mcj      informix       0 Mar 18 16:42 2a
-rw-r--r--   1 mcj      informix       0 Mar 18 16:42 ba
$
```

- z trzech znaków, ale gdy środkowy to "c"

```
$ rm ?c?
rm: ?c? non-existent
$
```

W pierwszym przypadku shell znalazł wszystkie dwuznakowe pliki i przekazał kernel'owi do wykonania komendę: `ls 1a 2a ab ba`. W drugim sprawę skomplikowaliśmy używając w poleceniu ścieżki względnej - tak, generowanie nazw działa we wszystkich możliwych ścieżkach dostępu. W trzecim natomiast shell nie znalazł plików ze środkową literą "c", więc nic nie podstawił i uruchomił komendę `rm ?c?` (tak jak wpisaliśmy) a polecenie to napisało nam, że pliku o nazwie ?c? nie ma.

A teraz kilka przykładów z *:

```
$ ls a*
a    ab   abb  abc
$
```

To było łatwe.

```
$ ls *a*
1a    2a    a     ab    abb   abc   ba


Mail:
moje_listy
$
```

To już bardziej skomplikowane. `*a*` zostało zastąpione przez `1a 2a a ab abb abc ba Mail`. Zwróć uwagę, że `*` przed a powoduje też podstawianie nazw zaczynających się od `a` (spójrz jeszcze raz na definicję gwiazdki powyżej). Poza tym, w bieżącym katalogu jest też podkatalog `Mail` (też ma `a`) więc polecenie `ls` wyświetliło dodatkowo jego zawartość (1 plik o nazwie `moje_listy`).

```
$ ls *a
1a  2a  a   ba
$
```

Tu mamy wszystkie pliki, które kończą się na `a` (w DOS podobna operacja się nie uda).

```
$ ls .*rc a*
```

```
.cshrc   .exrc    a        ab       abb      abc
$
```

Tu wyświetliliśmy, poza tymi zaczynającymi się na `a`, także pliki ukryte, kończące się na `rc`.

Aby sprawdzić, że kropka nie jest traktowana tak jak w DOS, wykonajmy trzy poniższe polecenia:

```
$ ls *.txt
test.txt
$ ls *txt
test.txt
$ ls *
1a         2a         a          ab         abb        abc        ba
test.txt

Mail:
moje_listy
$
```

Zadanie 1: Czym różni się zapis `??*` od `*???`
Zadanie 2: Dlaczego **`ls .*`** wyświetla aż tyle nazw plików i katalogów?

5.5.2 Edycja wiersza poleceń

W przypadku ksh wymagana jest podstawowa znajomość poleceń edytora `vi`, aby wykorzystać tę funkcję (w `bash` wystarczy używać klawiszy strzałek, `Home`, `End`, `Backspace` i `Delete`). Jeśli w trakcie wprowadzania polecenia zapragniemy je poprawić możemy zawsze użyć klawisza `Backspace`, aby skasować poprzedzające kursor znaki aż do miejsca, które chcemy poprawić. Potem dopisujemy całą resztę polecenia.

Jeśli jednak **znamy** `vi` wciskamy klawisz `Esc`, aby przejść do trybu wydawania komend `vi` i poprawiamy nasze polecenie za ich pomocą. Niezbędnym minimum są komendy (litery):

- **`h`** - przesunięcie kursora w lewo
- **`l`** - przesunięcie kursora w prawo
- **`i`** - przejście do edycji wiersza poleceń przed literą gdzie stoi kursor (by coś dopisać)
- **`a`** - przejście do edycji wiersza poleceń za literą gdzie stoi kursor (by coś dopisać)
- **`x`** - skasowanie znaku, na którym stoi kursor

W trakcie edycji danego polecenia możemy dowolną ilość razy przechodzić między trybem edycji (litery **`a`** i **`i`**) oraz trybem wydawania komend `vi` (`Esc`). Wciśnięcie klawisza `Enter` spowoduje wykonanie edytowanego polecenia niezależnie od tego czy jesteśmy w trybie edycji czy wydawania komend `vi`. Uwaga: nie używaj klawiszy strzałek - nie będą działać.

Jeśli **nie znamy** `vi` a przypadkowo wcisnęliśmy `Esc` to chyba najbezpieczniej jest wcisnąć literę **`A`** (duże `a`) co spowoduje przeskok kursora na koniec wiersza i wyjście z trybu wydawania komend `vi`. Teraz możemy jak zwykle użyć `Backspace` by poprawić polecenie lub `Enter`, aby je wykonać.

5.5.3 Historia poleceń

Jest to właściwie rozszerzenie funkcji opisanej powyżej. `ksh` (i inne shell'e) zapamiętuje każdą wykonywaną przez użytkownika komendę w ukrytym pliku w katalogu użytkownika. Do pliku

tego nie ma potrzeby zaglądać. Po przejściu do trybu wydawania komend `vi` (klawisz `Esc`) możemy ich używać do wyświetlania po kolei wszystkich zapamiętanych przez shell poleceń (w `bash` posługujemy się klawiszami strzałek góra/dół). Służą do tego dwie komendy (litery):

- **`k`** - każde wciśnięcie litery `k` spowoduje wyświetlenie coraz wcześniej wprowadzonego przez użytkownika polecenia
- **`l`** - każde wciśnięcie litery `l` spowoduje wyświetlenie polecenia wprowadzonego po tym, które jest w danej chwili wyświetlane (to na wypadek gdybyśmy się, za pomocą **`k`**, cofnęli za daleko)

Tak więc wciśnięcie sekwencji klawiszy **`Esc k k l`** spowoduje wyświetlenie ostatnio wprowadzonego polecenia.

Gdy już wyświetliliśmy polecenie, o które nam chodziło możemy je wykonać ponownie (wciskając `Enter`) lub zmodyfikować je wg procedury z poprzedniego rozdziału (już nie trzeba wciskać `Esc` by przejść do trybu wydawania komend, choć nic się nie stanie, jeśli to zrobimy).

Uwaga: zarówno edycja wiersza poleceń jak i historia mogą nie działać, jeśli administrator zmienił standardowe ustawienia systemu (choć zwykle takimi drobiazgami się nie zajmuje).

5.5.4 Uzupełnianie nazw plików

W trakcie wprowadzania polecenia możemy dwukrotnie wcisnąć klawisz `Esc` (w niektórych shell'ach, jak `bash`, funkcję tę pełni klawisz `Tab`), aby shell dopisał brakujące znaki w niedokończonej przez nas nazwie pliku (lub katalogu). Jeśli znaki, które wprowadziliśmy nie pozwalają na identyfikację tylko jednego pliku (jest kilka plików o tym właśnie początku) usłyszymy sygnał dźwiękowy. Jednak, jeśli da się cokolwiek dopisać (nawet, jeśli nie wszystko) to shell to zrobi. W tym momencie możemy też wcisnąć sekwencję klawiszy `Esc =` (`Esc` a potem znak równości), aby shell pokazał nam wszystkie pliki o tym właśnie początku. Dopisujemy teraz tyle znaków, aby shell mógł coś dopisać (niekoniecznie wszystko) i znowu wciskamy `Esc Esc`. W efekcie proces wpisywania przez nas znaków i (po `Esc Esc`) uzupełniania nazwy trwa aż uzyskamy pełną nazwę danego pliku lub katalogu. Procedura ta dotyczy także ścieżek dostępu, w których poszczególne nazwy katalogów możemy poddawać operacji uzupełniania. W poniższym przykładzie operacja uzupełniania ścieżki dostępu do pliku `pli2czek` (polecenie **`rm`**) wykonywana jest w jednej linii (nigdzie nie ma `Enter`'a) jednak została rozbita na kilka linii dla ułatwienia zrozumienia.

```
$ touch pliczek pli2czek <Enter>
$ rm /use <Esc> <Esc>
$ rm /users/mc <Esc> <Esc>
$ rm /users/mcj/pli2 <Esc> <Esc>
$ rm /users/mcj/pli2czek <Enter>
$
```

5.5.5 Skróty "~" i "~user"

Shell dysponuje kilkoma skrótami, które znowu ułatwiają nam wpisywanie długich ścieżek dostępu. Skróty te to:

- `~` oznacza katalog domowy użytkownika, który wpisuje ten znak w linii poleceń
- `~user` oznacza katalog domowy użytkownika o nazwie user

Przykładowo, użytkownik `miecio` znajduje się dowolnym katalogu (np. głównym) a chce skopiować do swojego katalogu domowego plik o nazwie `zdzisio.txt` znajdujący się w katalogu domowym użytkownika `zdzisio`. Miecio nie musi nawet pamiętać gdzie jest jego ani Zdzisia katalog domowy. Wykonuje więc komendę:

```
$ cp ~zdzisio/zdzisio.txt ~/
$
```

6 Procesy

6.1 Proces

Procesem nazywamy uruchomiony program (choć jest to uproszczenie). Dla rozróżnienia - program nie uruchomiony jest po prostu plikiem przechowywanym na dysku. Dochodzimy więc do stwierdzenia, że un*x zajmuje się nie tylko plikami, ale także procesami - nie wszystko jest plikiem, część obiektów jest procesami. Prawie wszystkie komendy, jakie do tej pory ćwiczyliśmy powodowały powstawanie nowych procesów.
Ale jak właściwie proces powstaje? Prześledźmy to (w uproszczeniu) na przykładzie komendy ls.

1. Komendę tę wpisujemy w shell'u. Jest on więc tzw. **procesem macierzystym**, ponieważ na jego bazie powstaje nowy proces. A chyba nikt nie wątpi, że sam shell jest także procesem? Przecież jest to uruchomiony program.
2. Po wprowadzeniu komendy **`ls`** shell tworzy w pamięci operacyjnej kopię samego siebie wraz z tzw. **środowiskiem**. Do środowiska należą takie rzeczy jak zmienne oraz informacje o prawach dostępu danego użytkownika.
3. Następnie shell zastępuje swoją kopię przez program `ls`. Powstaje w ten sposób nowy proces `ls`. Jest to tzw. **proces potomny** shell'a.
4. Proces ten znajduje się w środowisku, w jakim umieścił go shell. Dziedziczy więc np. wszystkie prawa (i ograniczenia), jakie posiada użytkownik uruchamiający komendę. Każdy proces ma więc także swojego właściciela tak jak to jest w przypadku plików.
5. W tej chwili proces `ls` wykonuje się. Robi to, do czego został stworzony a więc przegląda katalog i wyświetla na ekranie jego zawartość. W tym czasie shell czeka na jego zakończenie. Czasem mówimy, że "śpi". To właśnie dlatego nie mamy w tym czasie dostępu do linii poleceń i nie możemy nic napisać.
6. Po zakończeniu swojego działania kernel sygnalizuje shell'owi, że proces `ls` się już skończył a następnie usuwa go z pamięci operacyjnej. Shell na ten sygnał się budzi i prezentuje nam nowy znak dolara. Możemy wpisać następną komendę.

Zdarzają się jednak sytuacje wyjątkowe. Jeśli proces potomny się zakończy a macierzysty nie zostanie o tym powiadomiony nazywamy go **zombi** (ang. zombie). Jeśli natomiast przed zakończeniem procesu potomnego proces macierzysty zniknie to taki osierocony proces nazywamy właśnie **sierotą** (ang. orphan). Takie przypadki destabilizują pracę systemu operacyjnego (szczególnie zombi), co może powodować spowolnienie jego pracy.

Czy wszystkie polecenia powodują powstawanie procesów potomnych? Nie, ponieważ nie wszystkie są plikami na dysku. Niektóre polecenia są wbudowane w shell. Należą do nich np. `cd`, `pwd` i `echo` (choć zestaw ten może być różny w różnych shell'ach i odmianach un*x'a). Wykonywanie poleceń **wewnętrznych** shell'a nie powoduje powstawania nowych procesów. Polecenia **zewnętrzne**, czyli po prostu programy, zwykle znajdują się w katalogach `/bin` i `/usr/bin` a ich uruchomienie powoduje powstawanie procesów. Istnieje także możliwość, że uruchomienie jednego programu powoduje powstanie więcej niż jednego procesu.

6.2 ps

Zobaczmy teraz, czego możemy dowiedzieć się o procesach. Podstawowym poleceniem jest tu `ps`, obecne we wszystkich odmianach un*x'a (choć niestety z różnymi opcjami). Pokazuje ono tzw. stan procesów (ang. **p**rint **s**tatus). Uproszczona składnia:

```
ps [-ef] [-u użytkownik]
```

Bez opcji i parametrów pokazuje ono listę procesów przypisanych do terminala, przy którym siedzi wykonujący to polecenie (nie musi być ich właścicielem, choć najczęściej jest).

```
$ ps
   PID TTY      TIME COMMAND
 21842 ttyp2    0:00 ksh
 21853 ttyp2    0:00 ps
$
```

Widać 2 procesy w kolumnie `COMMAND`:

- `ksh` to oczywiście nasz shell
- `ps` to właśnie wykonujący się proces `ps`

Kolumna `PID` pokazuje tzw. identyfikator procesu (ang. **p**rocess **id**entification, number), czyli unikalny numer przypisywany procesowi przez kernel w momencie jego powstawania (procesu, nie kernel'a). Dzięki temu numerowi jądro odróżnia jedne procesy od drugich. `TTY` (ang. **t**ele**ty**pe - dalekopis) to nazwa pliku oznaczającego nasz terminal. W ten sposób jądro odróżnia terminale od siebie. Każdemu nowozalogowanemu użytkownikowi przypisywany jest inny plik terminala - ta informacja może się nam jeszcze przydać.

Czy możemy się więcej dowiedzieć o każdym z tych procesów? Oczywiście, używając opcji `-f` (ang. **f**ull - pełny), która oznacza prawie (wbrew nazwie) pełny opis poszczególnych procesów.

```
$ ps -f
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
     mcj 21842 21841  1 18:35:23 ttyp2    0:00 -ksh
     mcj 21859 21842  5 18:49:47 ttyp2    0:00 ps -f
$
```

Dodatkowo widzimy teraz nazwę właściciela procesu (w kolumnie `UID`), `STIME` (ang. **s**tart **time**), czyli czas lub datę powstania procesu oraz pełną komendę, która doprowadziła do powstania danego procesu (przynajmniej dla `ps`). Najważniejsza jednak w tej chwili jest kolumna `PPID` (ang. **p**arent **p**rocess **id**entification number) oznaczająca `PID` procesu macierzystego. Gdy przyjrzymy się dokładniej kolumnom `PID` i `PPID` to stwierdzimy, że `PID` shell'a jest identyczny jak `PPID` procesu `ps`. Oznacza to, że shell jest procesem macierzystym `ps` a `ps` jest procesem potomnym shell'a.

Co jest jednak procesem macierzystym shell'a i gdzie to drzewo genealogiczne się zaczyna? Wyświetlmy wszystkie procesy w systemie posługując się opcją `-e`. Użyjmy jednocześnie opcji `-f` dla wzbogacenia pokazywanych informacji. Lista procesów jest z pewnością długa. Użyjmy więc podanego niżej przykładu, aby oglądać ją tak jak przy korzystaniu z polecenia `more`:

```
$ ps -ef | more
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
    root     0     0  0  Jan  1  ?        0:36 swapper
    root     1     0  0  Feb  6  ?        0:03 init
```

```
    root 23671     1  0  Feb 25  console  0:00 /etc/getty console console
    root   187     1  0  Feb  6  ?        0:00 DIAGMON
      lp    73     1  0  Feb  6  ?        0:01 lpsched
    root   136     1  0  Feb  6  ?        0:33 /etc/rbootd /dev/lan0
    root    81     1  0  Feb  6  ?        0:00 /etc/nktl_daemon 0 0 0 -1
0 1 -2
    root   114     1  0  Feb  6  ?        0:13 /etc/inetd
    root   105     1  0  Feb  6  ?        0:00 /etc/rlbdaemon
     www 14102 14101  0  Feb 19  ?        0:00 httpd: idle
    root   139     1  0  Feb  6  ?        0:00 /etc/cron
    root   141     1  0  Feb  6  ?        0:00 /etc/ptydaemon
    root   421     1  0  Feb  6  ?        0:04 /usr/vue/bin/vuelogin
    root   419   418  0  Feb  6  ?        0:04 /etc/dtcnmp -l 1
    root   418     1  0  Feb  6  ?        0:01 /etc/dtcnmd /dev/lan0
    root   176     1  0  Feb  6  ?        0:05 /etc/syslogd
    root   184     1  0  Feb  6  ?        0:00 /etc/envd
     mcj 21871 21842  6 19:07:08 ttyp2    0:00 ps -ef
     mcj 21842 21841  1 18:35:23 ttyp2    0:00 -ksh
      lp 21740    73  0 16:12:24 ?        0:00 lpsched
    root 21841   114  0 18:35:22 ttyp2    0:00 telnetd -8
$
```

Większość procesów wycięto, aby nie zaciemniać obrazu. Jak widać, poza nielicznymi wyjątkami (swapper'em nie będziemy się przejmować), procesem macierzystym dla wielu jest `init` o numerze `1`. Po kolei uruchamia on przy starcie systemu różne programy, które są ważne dla jego prawidłowego działania (systemu, nie init'a). Kilka z nich pozwalają nam się zalogować i jako pierwszy nasz proces uruchamiają shell. Większość z tych ważnych procesów należy do użytkownika root. Jak widać nie trzeba więc wpisywać komendy, aby proces do nas należał - istnieją możliwości automatycznego uruchamiania programów. Pochodzenie poszczególnych procesów powoduje powstanie pewnej formy drzewa procesów - tak, jak w przypadku drzewa katalogów. Jeśli dokładnie prześledzimy `PID` i `PPID` procesów to stwierdzimy, że dla wielu z nich proces macierzysty już się nie wykonuje. Nie oznacza to wcale, że są one sierotami. Wiele programów jest napisane w specjalny sposób lub zostały uruchomione w specjalny sposób, dzięki czemu zakończenie procesu macierzystego nie ma żadnego wpływu na ich prawidłowe działanie.

Ciekawostką w powyższym przykładzie są oznaczenia w kolumnie `TTY`. `?` (znak zapytania) oznacza, że dany proces nie jest przypisany do żadnego terminala, na żadnym, więc nie zobaczymy wyników jego działania. Prawdopodobnie procesy te zostały więc uruchomione automatycznie a nie "z ręki". `"console"` oznacza konsolę, czyli terminal przeznaczony głównie do wykonywania zadań administracyjnych.

I ostatnia z co ciekawszych opcji umożliwiająca obejrzenie procesów wybranego użytkownika: `ps -u użytkownik` lub od razu z dodatkowymi opisami (tu kolejność opcji ma znaczenie, ponieważ nazwa użytkownika jest parametrem opcji `-u` a nie argumentem `ps`):

```
$ ps -fu misioo
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
```

```
  misioo 22201 22188  0 15:04:14 ttyp2    0:00 lynx
http://giswitch.sggw.waw.pl/
  misioo 22188 22179  0 15:02:32 ttyp2    0:00 -ksh
$
```

Resztę opisu opcji i kolumn polecenia ps można jak zwykle poznać wydając komendę `man ps`.

6.3 kill

W niektórych sytuacjach zdarza się, że chcielibyśmy zatrzymać jakiś proces. Sytuacja takie mogą mieć miejsce zwłaszcza wtedy, gdy jakiś proces "zawiesił się" i nie ma z nim żadnej komunikacji. Do zatrzymania procesu służy polecenie **`kill`** (z ang. **kill** - zabij). Jego ogólna postać wygląda następująco:

```
kill [-sygnal] PID
```

Zanim jednak zaczniemy zatrzymywać procesy, uruchommy jakiś mało ważny, na którym moglibyśmy bezpiecznie trenować:

```
$ sleep 500 &
[1]     22233
$
```

Aby zatrzymać proces musimy znać jego identyfikator (PID), który uzyskać można poleceniem **`ps`**:

```
$ ps
   PID TTY      TIME COMMAND
 22222 ttyp2    0:00 telnetd
 22233 ttyp2    0:00 sleep
 22223 ttyp2    0:00 ksh
 22234 ttyp2    0:00 ps
$
```

Polecenie **`kill 22233`** wysyła do procesu o `PID` równym `22233` sygnał `TERM` (z ang. **term**inate - przerwij, zatrzymaj), co w większości przypadków wystarcza do jego zakończenia. Polecenie to bez podania opcji (odpowiada to sygnałowi - opcji `-15`) pozwala na w miarę "kulturalne" zakończenie procesu - zamyka on otwarte pliki itp. Efektem działania tak wydanego polecenia będzie:

```
$ kill 22233
[1] + Terminated              sleep 500 &
$
```

Proces uruchomiony komendą `sleep 500 &`, o identyfikatorze `22233` został przerwany.
Jeżeli proces po próbie jego zatrzymania poleceniem bez opcji nadal funkcjonuje, konieczne jest użycie polecenia `kill` z opcją **`-9`**. Wysyła ono sygnał `KILL` - zabij.

```
$ kill -9 22233
[1] + Killed                sleep 500 &
$
```

Proces został "zabity" (przerwany) bezwarunkowo. Oznacza to, że nie "uporządkował" on swego otoczenia (np. nie pozamykał pootwieranych plików, jeśli jakieś otworzył). Stąd (ze względu na możliwość utraty danych czy zachwiania równowagi systemu) opcji tej należy używać jak najrzadziej.

kill służyć może także do przekazywania informacji procesom. Np. opcja **-1** to zwykle "zrestartuj"; odpowiada ona opcji **-HUP** (z ang. **h**ang **up**). Widać, że przy podawaniu poleceniu kill opcji, jako sygnałów można używać tak numerów, jak i nazw sygnałów. Wiele sygnałów może być ignorowanych przez procesy, KILL zwykle nigdy. Pełna lista sygnałów jest dostępna po wydaniu polecenia man 5 signal.

Podobnie, jak przy prawach dostępu do plików i katalogów, można "zabijać" tylko własne procesy. Próba zabicia nie swojego procesu (niezależnie od opcji - sygnału) kończy się następująco:

```
$ ps -ef
c03 22274 22273  0 16:27:44 ttyp8    0:00 -ksh
mcj 22081 22080  0 14:38:56 ttyp3    0:00 -ksh
$ kill -9 22274
kill: 22081: permission denied
$
```

Oczywiście nie trzeba chyba nikogo przekonywać, że ograniczenie to nie obowiązuje administratora (użytkownika o identyfikatorze root).

I jeszcze jedna uwaga: jeżeli zajdzie konieczność zatrzymywania procesów, trzeba zabijać je poczynając od ostatniego potomka. Taki sposób postępowania jest konieczny, aby nie powstawały sieroty (czyli procesy bez procesu macierzystego). Który proces jest czyim potomkiem można sprawdzić obserwując ich PID i PPID (to nie jest takie trudne).
Polecenie **kill** jest niezmiernie skutecznie. Mogą się jednak zdarzyć (na szczęście rzadkie) przypadki, gdy procesu nie można zabić. Jedyną radą na to jest wówczas interwencja bardzo doświadczonego administratora (czasami ma to związek z samym sprzętem) lub - co pomaga na wiele bolączek ;) - restart systemu (choć jak zwykle jest to najgorsze wyjście z sytuacji, może też się skończyć tym, że system już nie wystartuje).

7 Dodatek

Niestety z powodu braku czasu nie zdążyliśmy opisać tu poleceń związanych z drukowaniem oraz, być może, najważniejszego zagadnienia, decydującego o ogromnej wszechstronności i giętkości un*x'a: strumieniach i potokach.

7.1 Odpowiedzi na zadania:

- Odpowiedź na zadanie 1: Niczym, zarówno ??* jak i *??* oznaczają wszystkie, co najmniej dwuznakowe nazwy.
- Odpowiedź na zadanie 2: .* oznacza także . (samą kropkę), czyli katalog bieżący oraz .. czyli katalog nadrzędny - zawartość obu będzie więc dodatkowo wyświetlana.