UŻYTKOWANIE SYSTEMU OPERACYJNEGO UNIX

Spis treści

1. SYSTEM PLIKÓW 3

1.1. Pliki i katalogi 3
1.1.1. Katalog bieżący 3
1.1.2. Listing zawartości katalogu 4
1.1.3. Wzorce uogólniające 4
1.1.4. Prawa dostępu 6
1.1.5. Pliki ukryte 6
1.1.6. Tworzenie i usuwanie podkatalogów 6
1.1.7. Kopiowanie, usuwanie i przenoszenie plików 7
1.1.8. Tworzenie dowiązań 7

1.2. Programy pomocnicze 7
1.2.1. Podstawowe narzędzia operacji na plikach tekstowych 7
1.2.2. Narzędzia przeszukujące system plików 10

2. PROCESY 12

2.1. Lista procesów 12

2.2. Usuwanie procesów 13

2.3. Uruchomienie procesu w tle 13
2.3.1. Wejście/wyjście 13
2.3.2. Potoki 14

3. KOMUNIKACJA POMIĘDZY UŻYTKOWNIKAMI 15

3.1. Informacje o użytkownikach 15
3.1.1. Informacja o sobie samym 15
3.1.2. Informacja o zalogowanych użytkownikach 15
3.1.3. Informacja o dowolnym użytkowniku 17

3.2. Narzędzia komunikacji 19
3.2.1. Wysyłanie komunikatów 19
3.2.2. Dwustronna wymiana informacji 19
3.2.3. Blokowanie terminala 20
3.2.4. Poczta elektroniczna 20

4. ŚRODOWISKO PRACY W SYSTEMIE UNIX 22

4.1. Skrypty 22

4.2. Zmienne powłoki 22
4.2.1. Zmienne lokalne 23
4.2.2. Zmienne środowiskowe 23

4.3. Alias 24

4.4. Pliki konfiguracyjne 24

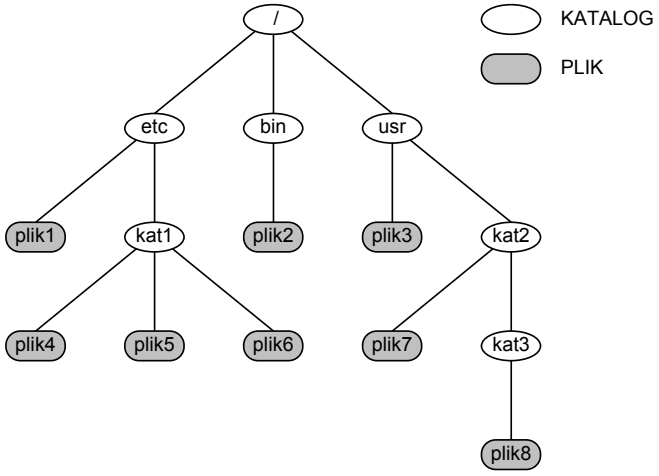
1. System plików

W systemie UNIX miejsce trwałego przechowywania danych nazywane jest systemem plików. Wyróżnia się następujące rodzaje plików:

- plik zwykły (ang. normal file) - elementarny zbiór danych identyfikowany przez system po swojej nazwie i położeniu w systemie plików
- katalog (ang. directory) - zawiera pliki (w tym również kolejne katalogi)
- urządzenie (ang. device)
- plik specjalny (link symboliczny, kolejka FIFO itp.)

1.1. Pliki i katalogi

System plików w systemie UNIX ma strukturę hierarchiczną. Początkiem systemu plików jest katalog główny zwany *root* i oznaczany */*. Stanowi on najwyższy poziom hierarchii. Kolejny poziom tworzą katalogi, w których zgrupowane są pliki i inne katalogi, stanowiące dalszy poziom hierarchii. Schematycznie wyraża to poniższy rysunek:



Nazwy plików są pojedynczymi wyrazami i mogą składać się z dowolnych liter, cyfr oraz kilku dozwolonych znaków, takich jak np. kropka lub znak łącznika (_). Na rysunku nazwą pliku umieszczonego na samym dole hierarchii jest wyraz plik8. Jest to tzw. *krótka nazwa pliku*. System UNIX rozróżnia małe i duże litery, więc nazwy plik8 i PLIK8 oznaczają dwa różne pliki. Położenie pliku wyznacza tzw. *ścieżka dostępu do pliku*, podająca wszystkie katalogi, jakie należy przejść aby znaleźć dany plik. W tym przykładzie będzie to: /usr/kat2/kat3/. Stąd *pełna nazwa pliku*, która jest połączeniem ścieżki dostępu i nazwy krótkiej, będzie tutaj: /usr/kat2/kat3/plik8.

Każdy użytkownik posiada swój katalog domowy, którego jest właścicielem. W tym katalogu może on przechowywać swoje własne pliki i zakładać podkatalogi.

1.1.1. Katalog bieżący

Katalog, w którym aktualnie pracujemy nazywa się katalogiem bieżącym. Pełną nazwę katalogu bieżącego otrzymamy po wydaniu polecenia **pwd**.

```
% pwd
/student/darek
%
```

1.1.2. Listing zawartości katalogu

Zawartością katalogu są nazwy i atrybuty plików. Do wyświetlenia zawartości katalogu bieżącego służy polecenie **ls**.

```
% ls
plik1        plik3        podkatalog1  skrypt
plik2        plik_inny    podkatalog2
%
```

```
% ls plik1
plik1
% ls /student/darek/plik2
/student/darek/plik2
%
```

```
% ls plik1 plik2
plik1  plik2
%
```

Jak każda nazwa pliku, może w poleceniu ls wystąpić, jako parametr, również nazwa katalogu. Wynikiem będzie wówczas zawartość tego katalogu

```
% ls podkatalog1
plik_pk1
% ls /student/darek/podkatalog2
plik_pk2
%
```

1.1.3. Wzorce uogólniające

Nazwy plików podawane jako parametry poleceń systemu UNIX (np. polecenia ls) mogą zawierać tzw. wzorce uogólniające, które mogą zastępować fragmenty, a nawet całości rzeeczywistych nazw.

1.1.3.1. Elementy wzorców uogólniających

Element * zastępuje dowolny ciąg znaków

```
% ls *
plik1        plik2        plik3        plik_inny    skrypt

podkatalog1:
plik_pk1

podkatalog2:
plik_pk2
%
```

```
% ls plik*
plik1      plik2      plik3      plik_inny
%
```

```
% ls pl*
plik1      plik2      plik3      plik_inny
%
```

```
% ls p*
plik1        plik2        plik3        plik_inny

podkatalog1:
plik_pk1

podkatalog2:
plik_pk2
%
```

```
% ls pod*

podkatalog1:
plik_pk1

podkatalog2:
plik_pk2
%
```

Element **?** oznacza dowolny jeden znak

```
% ls plik?
plik1  plik2  plik3
%
```

Element **[ ]** zastępuje dokładnie jeden znak wybrany spośród podanego w nawiasach zbioru.

```
% ls plik[12]
plik1  plik2
% ls plik[13]
plik1  plik3
% ls plik[132]
plik1  plik2  plik3
% ls plik[1-3]
plik1  plik2  plik3
%
```

1.1.3.2. Przykłady łączenia wzorców

```
% ls p*1
plik1

podkatalog1:
plik_pk1
%
```

```
% ls p*[12]
plik1        plik2

podkatalog1:
plik_pk1

podkatalog2:
plik_pk2
%
```

1.1.4. Prawa dostępu

1.1.4.1. Pełna informacja o pliku:

Polecenie ls posiada kilka opcji podawanych po znaku -. Opcja **-l** pozwala poznać pełną informację o plikach w katalogu bieżącym:

```
% ls -l
total 56
-rw-r--r--   1 darek    student      136 Apr 10 19:16 plik1
-rw-r--r--   1 darek    student      136 Apr 10 19:19 plik2
-rw-r--r--   1 darek    student      136 Apr 10 19:20 plik3
-rw-r--r--   1 darek    student       18 Apr 10 19:25 plik_inny
drwxr-sr-x   2 darek    student      512 Apr 10 19:29 podkatalog1
drwxr-sr-x   2 darek    student      512 Apr 10 19:30 podkatalog2
-rw-r--r--   1 darek    student       13 Apr 10 19:26 skrypt
%
```

prawa dostępu | liczba dowiązań | właściciel | grupa | rozmiar | data i czas ostatniej modyfikacji | nazwa pliku

Prawa (trzy kolejne litery) podawane są dla właściciela pliku, użytkowników grupy i pozostałych użytkowników.
Prawa dostępu do plików zwykłych:

- r prawo do odczytu zawartości (umożliwia również kopiowanie),
- w prawo do zapisu (zmiany zawartości pliku lub usunięcia zawartości),
- x prawo do uruchomienia (dotyczy plików zawierających programy binarne lub skrypty).

Prawa dostępu do katalogów:

- r prawo do przeglądania (np. umożliwia wykonanie polecenia ls),
- w prawo do tworzenia, usuwania i zmiany nazw plików,
- x prawo dostępu do plików.

1.1.4.2. Zmiana praw dostępu:

Prawa dostępu do pliku mogą być zmienione tylko przez właściciela i użytkownika uprzywilejowanego o nazwie root. Do zmiany praw służy polecenie **chmod**. Przykłady użycia polecenia chmod:

- `chmod u+x plik` dodanie prawa wykonywania właścicielowi
- `chmod g-w plik` zabranie prawa zapisu grupie
- `chmod a+r plik` dodanie prawa odczytu wszystkim użytkownikom
- `chmod o=x plik` zmiana praw pozostałych użytkowników na prawo wykonywania
- `chmod o= plik` wyzerowanie praw pozostałych użytkowników
- `chmod 777 plik` nadanie wszystkich praw wszystkim użytkownikom (prawa w postaci ósemkowej)

1.1.5. Pliki ukryte

Utworzenie pliku ukrytego polegan na nadaniu nazwy zaczynającej się od kropki. Jakiekolwiek operacje na plikach z wzorcem uogólniającym * (kopiowanie cp *, usuwanie rm * itp.) nie dotyczą plików ukrytych. Wzorzec obejmujący pliki ukryte składa się z gwiazdki poprzedzonej kropką .*. W celu wyświetlenia plików ukrytych należy użyć opcji –a w poleceniu ls, np.:
ls -a, ls -al

1.1.6. Tworzenie i usuwanie podkatalogów

Katalog tworzy się poleceniem **mkdir**:

```
mkdir katalog
```

Pusty katalog można usunąć poleceniem **rmdir**:

```
rmdir katalog
```

W celu ususnięcia całego podrzewa (katalogu wraz z wszystkimi plikami i podkatalogami) używamy opcji –r w poleceniu rm:

```
rm –r katalog
```

1.1.7. Kopiowanie, usuwanie i przenoszenie plików

Do kopiowania plików służy polecenie **cp**. Polecenie to umożliwia:

- utworzenie kopi pliku w tym samym katalogu pod inną nazwą:

```
cp plik nowy_plik
```

- utworzenie kopi pliku pod tą samą nazwą w innym katalogu:

```
cp plik katalog
cp plik1 plik2 plik3 katalog
cp pl* katalog
```

- utworzenie kopi pliku pod pod inną nazwą w innym katalogu:

```
cp plik katalog/nowy_plik
```

Opcja –r w polecenie cp. umożliwia kopiowanie całych katalogów (z wszystkimi plikami i podkatalogami):

```
cp -r katalog1 nowy_katalog
```

Do usuwanie plików (w przypadku opcji –r również katalogów) służy polecenie **rm**:

```
rm plik
rm plik1 plik2
rm -i pl*
rm -r katalog
```

W celu przesunięcia pliku do innego katalogu lub zmiany nazwy pliku stosuje się polecenie **mv**:

```
mv plik katalog
mv plik nowa_nazwa
```

1.1.8. Tworzenie dowiązań

System UNIX dysponuje specyficznym mechanizmem, pozwalającym na występowanie tego samego pliku pod różnymi nazwami (w różnych katalogach). Umożliwia on wykorzystanie, np. w katalogach różnych użytkowników, wspólnego pliku, bez potrzeby oddzielnego kopiowania go i nieustannego aktualizowania kopii. Mechanizm ten nosi nazwę dowiązania.

Dowiązanie tworzone jest za pomocą polecenia **ln** (ang. link). Jego składnia przypomina składnię polecenia kopiowania:

```
ln nazwa_pliku nazwa_dołączenia
```

W wyniku dowiązania do istniejącego pliku, tworzony zostaje w hierarchii systemu plików „nowy" plik (dokładniej nowa nazwa), którego atrybuty i położenie na dysku jest identyczne z oryginałem. Liczba dowiązań (atrybut każdego pliku w UNIXie) do pliku zostaje zwiększona o jeden. Dowiązanie może zostać skasowane przez uprawnionego użytkownika. Zmniejsza to liczbę dowiązań o jeden.

Nie można dokonać dowiązania do katalogu oraz do pliku w innym systemie plików (na innej partycji).

Poleceniem **ln -s** można utworzyć dowiązanie symboliczne. Tworzony jest wówczas nowy krótki plik, który zawiera ścieżkę do oryginału. Wszystkie odwołania do tak utworzonego pliku są faktycznie kierowane do oryginalnego pliku.

Dowiązania symboliczne nie mają ograniczeń, tzn. można dowiązać symbolicznie katalog oraz plik z innego systemu plików.

1.2. Programy pomocnicze

1.2.1. Podstawowe narzędzia operacji na plikach tekstowych

1.2.1.1. Edytor vi

Narzędzia operacji na tekstach w UNIXie można podzielić na dwie kategorie: edytory i pliki obróbki tekstów. Jako przykład edytora można wymienić program **vi** występujący we wszystkich odmianach UNIXa. Jest dzięki temu często wykorzystywany przez administratorów i zaawansowanych użytkowników. Program **vi** uruchamiamy poleceniem vi *plik*, gdzie *plik* jest nazwą pliku zawierającego edytowany tekst. Jeśli plik o podanej nazwie nie istniał jeszcze, to zostanie on utworzony. Zaraz po uruchomieniu vi znajduje się w *trybie komend*, w którym pojedyncze znaki lub ich sekwencje oznaczają komendy interpretowane przez edytor, np:

- i (*insert*) - nakazuje przejście do *trybu edycji* tekstu; tekst jest wstawiany od aktualnego miejsca położenia kursora
- a (*append*) - przejście do *trybu edycji* tekstu; tekst jest wstawiany od miejsca za aktualnym położeniem kursora

x	- kasowanie znaku w miejscu położenia kursora
X	- kasowanie znaku na lewo od kursora
4x	- skasowanie 4 kolejnych znaków od bieżącego położenia kursora
1G	- skok do pierwszej linii
6G	- skok do 6-tej linii
G	- skok do ostatniej linii
dd (*delete*)	- kasowanie całej linii
d5d	- skasowanie 5 linii począwszy od tej, w której znajduje się kursor
dG	- skasowanie linii od aktualnej, aż do ostatniej
/	- szukanie miejsca wystąpienia zadanego fragmentu tekstu
ZZ	- zapisanie aktualnie edytowanego pliku i opuszczenie programu.

Powrót z trybu edycji do trybu komend następuje po wciśnięciu klawisza Esc. Niektóre komendy wymagają dodatkowej interakcji z użytkownikiem, np. po wydaniu komendy szukania (klawisz [/]), należy podać ciąg znaków, który ma zostać znaleziony w edytowanym tekście i zatwierdzić go klawiszem [Enter]. Jeśli chcemy znaleźć kolejne wystąpienie tego samego ciągu znaków, naciskamy w trybie komend kolejno klawisze [/] i [Enter].

Szersze przedstawienie edytora vi odbędzie się na ćwiczeniach.

1.2.1.2. Obróbka tekstów

Oprócz edytora vi, w każdym systemie UNIX występuje program **more**. Służy on do wyprowadzania zawartości pliku na ekran. Wywołuje się go, podobnie jak vi, podając nazwę pliku. Program more wyświetla zawartość pliku, aż do zapełnienia ekranu, poczym wstrzymuje wyświetlanie, czekając na naciśnięcie klawisza. Następujące klawisze traktowane są jako komendy:

[Enter],[Return]	- wyświetla kolejną linię tekstu
[spacja]	- wyświetla kolejny ekran tekstu
[b] (*back*)	- wyświetla poprzedni ekran tekstu
[=]	- podaje numer linii znajdującej się u szczytu ekranu
[/]	- szukanie miejsca wystąpienia zadanego fragmentu tekstu
[v]	- uruchomienie edytora vi i automatyczne wczytanie przeglądanego pliku do edycji
[h] (*help*)	- krótki spis komend programu
[q] (*quit*)	- zakończenie przeglądania pliku

Trzy pierwsze komendy mogą zostać poprzedzone liczbą oznaczającą ilość jednostek, o jakie należy przesunąć przeglądany tekst, np. 3[spacja] przesunie tekst o trzy ekrany.

Program more możemy wywołać z kilkoma opcjami. Najważniejsze z nich to opcja -*n* (minus), gdzie *n* jest dowolną liczbą naturalną, definiująca ilość linii wyświetlanych jako jeden "ekran", oraz opcja +*n* (plus), podająca numer pierwszej wyświetlanej linii. Oto przykłady użycia kilku wersji tych opcji:

```
% more -10 plik1          (będzie wyświetlać po 10 linii tekst z pliku plik1)
% more +40 plik1          (wyświetlanie rozpocznie się od 40 linii)
% more +/Tekst plik1      (wyświetlanie rozpocznie się od pierwszej linii zawierającej słowo "Tekst")
% more -10 plik1 plik2    (wyświetli zawartości plików plik1 i plik2,  po 10 linii)
% more -10 plik[1-3]
%
```

W systemie UNIX dostępny jest również prostszy program przeglądania plików tekstowych - **pg**. Pozwala on wyświetlać tekst strona po stronie i rozumie następujące komendy:

[Enter]	- kolejna strona
4[Enter]	- strona nr 4
+2[Enter]	- przeskok o 2 strony do przodu
-3[Enter]	- cofnięcie się o 3 strony
h (help)	- spis komend
q (quit)	- koniec przeglądania

Oprócz dwóch powyższych, mamy do dyspozycji jeszcze trzeci program, który pozwala nam obejrzeć zawartość pliku na ekran. Program **cat** służy do konkatenacji plików i może zostać wykorzystany do wyprowadzenia zawartości plików na ekran terminala.

```
% cat plik1
% cat plik1 plik2
% cat plik[1-3]
%
```

Istnieją również programy pozwalające wyświetlać tylko początek (**head**) lub koniec pliku (**tail**). Wyświetlają one domyślnie 10 pierwszych / ostatnich linii tekstu, lecz opcja *-n* (minus) pozwala zmienić wartość domyślną:

```
% head plik1
% head plik[12]
% head -15 plik3
% tail -15 plik3
%
```

Klasycznym przykładem obróbki tekstów jest sortowanie. Program **sort** umożliwia uporządkowanie linii pliku tekstowego względem dowolnej kolumny (przez kolejną kolumnę rozumie się tu kolejne wyrazy w linii) w kolejności alfabetycznej lub liczbowej, rosnąco lub malejąco. Kryteria sortowania zależą od podanych w wywołaniu programu opcji.

Najistotniejsze opcje programu sort oznaczają:

+2	- pominięcie podczas sortowania pierwszych 2 kolumn w każdej linii (linie będą sortowane wg zawartości kolejnych kolumn)
-3	- ograniczenie sortowania do 3 kolumny (np. `sort +2 -4 plik` posortuje *plik* biorąc pod uwagę tylko kolumny trzecią i czwartą)
+3.1	- pominięcie 3 kolumn i 1 znaku kolumny następnej
-3.3	- pominięcie wszystkiego co znajduje się za 3 kolumnami i 3 znakami.
-n (*numeric*)	- sortowanie numeryczne
-d (*dictionary*)	- sortowanie słownikowe
-f	- duże i małe litery traktowane są identycznie
-r (*reverse*)	- odwrócenie porządku sortowania (sortowanie w kolejności malejącej)
-b (*blank*)	- pominięcie pustych znaków
-u (*unique*)	- usunięcie duplikatów linii
-o (*output*)	- zapisanie wyniku posortowania w pliku o podanej nazwie

Działanie tych opcji prześledzimy na poniższych przykładach:

```
% more plik_sort
c. 7763 John Rambo
b. 96   Dracula hrabia
a. 284  Sierotka Marysia

% sort plik_sort
a. 284  Sierotka Marysia
b. 96   Dracula hrabia
c. 7763 John Rambo

% sort -n +1 plik_sort
b. 96   Dracula hrabia
a. 284  Sierotka Marysia
c. 7763 John Rambo

% sort -nr +1 plik_sort
c. 7763 John Rambo
a. 284  Sierotka Marysia
b. 96   Dracula hrabia

% sort +3 -f plik_sort
b. 96   Dracula hrabia
a. 284  Sierotka Marysia
c. 7763 John Rambo

% sort -b +2 -3 plik_sort
b. 96   Dracula hrabia
c. 7763 John Rambo
a. 284  Sierotka Marysia

% sort -b +3.1 -3.3 plik_sort -o plik.posortowany
% more plik.po*
c. 7763 John Rambo
a. 284  Sierotka Marysia
b. 96   Dracula hrabia

%
```

1.2.2. Narzędzia przeszukujące system plików

W tym punkcie przedstawione zostaną dwa bardzo popularne narzędzia przeszukujące system plików.

1.2.2.1. grep

Program grep przegląda pojedynczy katalog w poszukiwaniu plików zawierających podany ciąg znaków. Wywołanie tego programu ma postać:

```
grep  wzorzec_tekstu  nazwy_plików
```

Dla przykładu polecenie grep "to jest" *pl** spowoduje przeszukanie wszystkich plików w bieżącym katalogu, których nazwy pasują do wzorca *pl** i z ich zawartości wypisze na ekranie te linie, które zawierają tekst `to jest`. Poszukiwany tekst jest podawany w postaci tzw. wyrażenia regularnego. Może być ono ujęte w znaki apostrofy lub cudzysłowiu i w większości przypadków nie ma między nimi różnicy. W wyrażeniach regularnych m.in. następujące znaki mają znaczenie specjalne:

- `.` - oznacza dowolny jeden znak tekstu
- `[abc]` - dowolny znak spośród podanych: a,b lub c
- `[a-z]` - dowolny znak z zakresu od a do z
- `[^aA]` - dowolny znak różny od a i A
- `[^a-z]` - dowolny znak spoza zakresu od a do z
- `^` - początek linii tekstu
- `$` - koniec linii tekstu

Jeśli poszukujemy ciągu znaków zawierającego znak specjalny, musimy zarządać od programu grep, by potraktował ten znak dosłownie. W tym celu poprzedzamy ów znak znanym nam już symbolem \, np. '\$' oznacza znak dolara, a nie koniec linii.

Jak większość programów UNIXowych, również grep przyjmuje kilka opcji. Najważniejsze z nich, to:

-i - powoduje, iż duże i małe litery nie są rozróżniane
-n - dla każdego pliku podaje numery linii, w których występuje szukany wzorzec
-l - wyświetla tylko nazwy plików, które zawierają szukany wzorzec
-v - wyświetla tylko te linie, które nie zawierają podanego wzorca
-w - poszukuje nie fragmentu tekstu, lecz całych wyrazów identycznych z podanym wzorcem

Oto kilka przykładów zastosowania programu grep:

```
% grep '^[0-9]' pl*                    (szuka w plikach pl* linii rozpoczynających się od cyfry)
% grep -n '[kK]$' pl*                  (podaje nazwy plików i numery linii kończących się na k lub K)
% grep "\*" podkatalog1/*              (szuka w plikach podkatalogu linii zawierających *)
% grep -lwv "plik" pl*                 (wyświetli nazwy plików nie zawierających wyrazów plik)
%
```

1.2.2.2. find

Program find przeszukuje drzewo katalogów systemu plików w poszukiwaniu plików spełniających zadane kryteria. Dla każdego znalezionego pliku podejmowana jest zdefiniowana akcja. Wywołanie tego programu ma postać:

```
find  katalog_początkowy  specyfikacja_plików  [akcja]
```

Specyfikacja pliku może obejmować następujące opcje:

-name "*wzorzec*" - nazwa pliku musi odpowiadać *wzorcowi*
-user *identyfikator* - szukane są tylko pliki, których właścicielem jest użytkownik o danym *identyfikatorze*
-group *grupa* - tylko pliki odpowiadające danej *grupie* właściciela
-atime *czas* - tylko pliki, do których ostatni dostęp nastąpił w ciągu podanego *czasu* (mierzonego w dniach)
-mtime *czas* - tylko pliki modyfikowane w ciągu podanego *czasu* (mierzonego w dniach)

Ewentualna akcja może obejmować jedną z następujących opcji:

-print - pełna nazwa każdego znalezionego pliku jest wyświetlana na ekranie
-exec *polecenie* \; - dla każdego znalezionego pliku wykonywane jest *polecenie*
-ok *polecenie* \; - dla każdego znalezionego pliku wykonywane jest *polecenie*, z uprzednim zapytaniem czy należy je dla tego pliku wykonać

W poleceniach tych można operować symbolem {}, w miejsce którego program find podstawiać będzie pełną nazwę znalezionego pliku.

Oto przykład zastosowania programu find, w celu odnalezienia plików o nazwie READ* znajdującego się gdzieś w katalogu bieżącym i wszystkich jego podkatalogach, należącego do użytkownika michal i nie wykorzystywanego od dwóch dni:

```
% find . -name "READ*" -user michal -atime +2 -print
%
```

2. Procesy

Każdy uruchomiony w systemie UNIX program nosi nazwę procesu. Wiele z procesów uruchamianych jest przy starcie systemu. Pozostałe są uaktywniane przez użytkowników. W momencie zarejestrowania się użytkownika w systemie uruchomiony zostaje jego pierwszy proces - powłoka interpretująca polecenia. Każde polecenie wykonania programu powoduje uaktywnienie kolejnego procesu. Procesy pracujące w systemie zebrane są w strukturę hierarchiczną. Powłoka jest procesem nadrzędnym dla wszystkich pozostałych procesów użytkownika, a procesy te są tzw. potomkami swojego nadrzędnego procesu - powłoki. Dowolny proces może uruchomić kolejny proces potomny i stać się macierzystym (nadrzędnym) wobec owego procesu potomnego. Wszystkie procesy w systemie są ponumerowane. Numery te są unikalne w danym cyklu życia systemu i każdy proces jest jednoznacznie identyfikowany przez swój numer (tzw. PID- Process IDentifier).

2.1. Lista procesów

Listę procesów dla aktualnej powłoki otrzymamy wywołując polecenie **ps**.

```
% ps
   PID    TTY  TIME CMD
 14285  pts/0  0:00 -csh
 14286  pts/0  0:00 ps
%
```

numer procesu, terminal, czas aktywności, nazwa

Pełne informacje o procesach aktualnej powłoki podaje ps –f (full):

```
% ps -f
    USER   PID  PPID   C    STIME    TTY  TIME CMD
  michal 14285 14267   0 14:44:03  pts/0  0:00 -csh
  michal 14287 14285   7 14:44:24  pts/0  0:00 ps -f
%
```

nazwa właściciela, numer procesu, numer procesu nadrzędnego, czas uruchomienia, terminal, czas aktywności, pełna nazwa procesu

Pełne informacje o wszystkich procesach uzyskamy łącząc opcję -f z opcją -e (every process):

```
% ps -ef
    USER   PID  PPID   C    STIME    TTY  TIME CMD
    root     1     0   0   Apr 17      -  5:55 /etc/init
    root  1920     1   0   Apr 17      -  2:07 /etc/cron
    root  3658  4672   0   Apr 17      -  0:04 /etc/syslogd
    root  5206  4672   0   Apr 17      -  0:03 /etc/inetd
                                      ...
  michal 14280 14267   0 14:44:03  pts/1  0:00 -csh
  michal 14285 14267   2 14:44:03  pts/0  0:00 -csh
  michal 14288 14285   6 14:44:27  pts/0  0:00 ps -ef
%
```

nazwa właściciela, numer procesu, numer procesu przodka, czas uruchomienia, terminal, czas aktywności, pełna nazwa procesu

2.2. Usuwanie procesów

Dowolny proces możemy może zostać usunięty z systemu przez jego właściciela. Służy do tego celu polecenie **kill** *pid*, wysyłające do procesu o podanym numerze *pid* sygnał przerwania pracy; np.

```
% kill 14290
```

spowoduje wysłanie sygnału zakończenia (sygnał nr 15 – SIGTERM) do procesu o numerze 14290. Poleceniem kill możemy wysyłać do procesów dowolne sygnały, w szczególności sygnał zabicia (nr 9 – SIGKILL), czyli bezwarunkowe przerwanie pracy; przykładowo komenda

```
% kill -9 14290
```

spowoduje bezwarunkowe zabicie procesu 14290.

Aktualnie uruchomiony proces możemy również przerwać z terminala, naciskając kombinację klawiszy **^C** (Control C), co powoduje wysłanie sygnału przerwanie SIGINT.

2.3. Uruchomienie procesu w tle

Procesy uruchamiane poleceniem wydanym z klawiatury terminala pracują na tzw. pierwszym planie. Powłoka czeka na zakończenie procesu i dopiero po tym fakcie jest gotowa na przyjęcie kolejnych poleceń od użytkownika. Można jednak proces uruchomić w tle. Wówczas powłoka utworzy nowy proces potomny, będący powłoką, której nakaże wykonanie zadanego polecenia, a sama powróci do stanu gotowości na kolejne polecenia. W ten sposób użytkownik może wydać kilka poleceń zanim pierwsze z nich się zakończy. Polecenie jest uruchomione w tle, jeśli po ostatnim parametrze następuje znak **&.**

```
% ls -l &
% find . -name "READ*" -user michal -atime +2 -print &
%
```

Aktualnie uruchomiony proces można także zatrzymać klawiszem **^Z**. Spowoduje to zastopowanie tego procesu. Możemy taki zastopowany proces wprowadzić do wykonania (kontynuacji) w tle poleceniem **bg** (*background*), a nawet przywrócić po dowolnym czasie z powrotem na pierwszy plan poleceniem **fg** (*foreground*), pod warunkiem jednak, że pomiędzy tymi poleceniami nie uruchomimy w tle niczego innego.

W systemie UNIX możemy jednym poleceniem uruchomić kilka procesów, oddzielając poszczególne z nich średnikiem:

```
% date; pwd; ls -l
%
```

Podane w ten sposób programy są wykonywane wg kolejności, jeden po drugim. Taką sekwencję procesów możemy również wprowadzić w tło:

```
% (date; pwd; ls -l) &
%
```

Dobrym zwyczajem jest ujmowanie poleceń w nawiasy, chociaż nie we wszystkich odmianach UNIXa jest to konieczne.

2.3.1. Wejście/wyjście

Każdy proces domyślnie korzysta z tzw. standardowego wejścia danych - klawiatury terminala, z którego uruchomiona dany proces oraz z tzw. standardowego wyjścia danych, którym jest ekran tego terminala. Tak np. proces sort wyniki sortowania domyślnie podaje na standardowe wyjście, a jeśli nie podamy mu nazwy pliku wejściowego, oczekuje danych ze standardowego wejścia. Podobnie działa większość innych programów. Poznany wcześniej program **cat** pozwala nam łączyć zawartości wielu plików ze sobą, np. polecenie

```
% cat plik1 plik2
```

wyświetli na standardowym wyjściu zawartość plików plik1 i plik2. Możemy jednak dokonać tzw. przeadresowania wyjścia, kierując wynik do podanego pliku, np. w poleceniu

```
% cat plik1 plik2 > plik12
```

zawartość plików plik1 i plik2 zostanie zapisana w pliku plik12.

Wykorzystywane mogą być następujące symbole :

< *plik* — przeadresowanie standardowego wejścia, czyli pobranie danych wejściowych z *pliku*

> plik — przeadresowanie standardowego wyjścia, czyli utworzenie pliku i zapisanie w nim tego co proces wypisałby na standardowym wyjściu

>> *plik* — przeadresowanie standardowego wyjścia, z dopisaniem informacji do istniejącego *pliku*.

W szczególności, bezparametrowe polecenie

```
% cat
```

spowoduje przepisanie danych wczytywanych ze standardowego wejścia, czyli klawiatury, na standardowe wyjście, czyli ekran, a polecenie

```
% cat > plik4
```

spowoduje zapisanie do pliku plik4 danych wczytywanych z klawiatury.

Jako kolejny przykład proponujemy rozważenie różnic w wynikach poniższych poleceń:

```
% (date; pwd; ls -l) > plik5
%  date; pwd; ls -l  > plik6
%
```

2.3.2. Potoki

Przeadresowanie wejścia/wyjścia możemy wykorzystać w celu utworzenia potoków, których bierze udział jednocześnie kilka procesów. Każdy kolejny proces w potoku czyta dane z wejścia, które zostało przeadresowane na wyjście procesu poprzedniego. Oto przykłady potoków:

```
% ls -al | more    (proces ls podaje wynik procesowi more, który w efekcie wyświetla listing strona po stronie)
% who | sort              (podaje posortowaną listę użytkowników pracujących w systemie)
% ps -ef | grep csh      (szuka na liście procesów linii zawierających słowo csh)
% ls -l /usr/bin | sort -bnr +4 -5 | head (proszę odgadnąć rezultat)
%
```

3. Komunikacja pomiędzy użytkownikami

3.1. Informacje o użytkownikach

3.1.1. Informacja o sobie samym

3.1.1.1. id

Polecenie id zwraca numeryczny identyfikator użytkownika i grupy, do której został zaliczony przy otwarciu konta, wraz z odpowiednimi nazwami.

```
% id
uid=1091(darek) gid=101(staff)
%
```

identyfikator i nazwa użytkownika — identyfikator i nazwa grupy

Jeżeli identyfikatory rzeczywisty i efektywny są różne, zwracane są oba.

```
% ls -l ./id
-rws--s--x   1 root     other      11264 May 08 11:47 ./id
% ./id
uid=1091(darek) gid=101(staff) euid=0(root) egid=1(other)
%
```

3.1.1.2. who am i

Polecenie who am i zwraca informację o użytkowniku, mającym aktywną sesję na terminalu, na którym wydawane jest to polecenie.

```
% who am i
darek      ttyp8         May 07 14:06
%
```

nazwa użytkownika — terminal — data i czas zalogowania

Terminal jest nazwą pliku specjalnego, reprezentującego urządzenie, znajdującego się w katalogu /dev. Poniższy wydruk przedstawia listing wybranych plików, reprezentujących terminale.

```
% ls -l /dev/ttyp?
crw--w----   1 darek    terminal  58,  0 May 08 12:18 /dev/ttyp0
crw-------   1 inf37664 terminal  58,  1 May 08 11:42 /dev/ttyp1
crw-------   1 inf37656 terminal  58,  2 May 08 11:32 /dev/ttyp2
crw-------   1 inf37646 terminal  58,  3 May 08 12:18 /dev/ttyp3
crw-------   1 inf37653 terminal  58,  4 May 08 12:07 /dev/ttyp4
crw-------   1 root     terminal  58,  6 May 08 11:58 /dev/ttyp6
crw-------   1 inf00004 terminal  58,  7 May 08 12:18 /dev/ttyp7
crw-------   1 inf37646 terminal  58,  8 May 08 12:17 /dev/ttyp8
%
```

Użytkownik darek (pierwsza pozycja w listingu) wyraził zgodę na otrzymywanie komunikatów (patrz pkt. B3 Blokowanie terminali), w związku z czym jest prawo zapisu (write) dla grupy. W pozostałych plikach nie ma prawa zapisu dla grupy, co oznacza, że inni użytkownicy nie wyrazili zgody na otrzymywanie komunikatów.

3.1.2. Informacja o zalogowanych użytkownikach

3.1.2.1. who

Polecenie who zwraca informacje o wszystkich użytkownikach pracujących w systemie.

```
% who
inf37718   ttyp0        May 07 13:24
inf37642   ttyp1        May 07 14:23
inf37722   ttyp2        May 07 13:25
inf37642   ttyp3        May 07 14:21
inf40765   ttyp4        May 07 14:04
inf37721   ttyp5        May 07 13:28
michu      ttyp6        May 07 13:43
inf37698   ttyp7        May 07 13:31
darek      ttyp8        May 07 14:06
inf37651   ttyp9        May 07 14:15
root       ttyp10       May 07 14:23
%
```

nazwa użytkownika | terminal | data i czas zalogowania

3.1.2.2. finger

Polecenie finger, podobnie jak who, może zwrócić dane o zalogowanych użytkownikach, które obejmują jednak więcej informacji, jak wynika z poniższego listingu.

```
% finger
Login       Name                TTY Idle    When           Office
inf37718 Marzena Rabiega       *p0        Tue 13:24   Informatyk  2 Rok I-3
inf37642 Pawel Augustyniak     *p1        Tue 14:23   Informatyk  2 Rok I-1
inf37722 Monika Roszak         *p2        Tue 13:25   Informatyk  2 Rok I-3
inf37642 Pawel Augustyniak     *p3        Tue 14:21   Informatyk  2 Rok I-1
inf40765 Tomasz Motlawski      *p4    24  Tue 14:04   Informatyk  1 Rok I-3
inf37721 Pawel Robakowski      *p5        Tue 13:28   Informatyk  2 Rok I-3
michu    Michal Szychowiak     *p6    39  Tue 13:43   rm 409      tel 782378
inf37698 Aneta Musielak        *p7        Tue 13:31   Informatyk  2 Rok I-3
darek    Dariusz Wawrzyniak     p8        Tue 14:06   rm. 410     tel. 782378
inf37651 Michal Czajkowski     *p9     4  Tue 14:15   Informatyk  2 Rok I-1
root     SCO UNIX on wega sup   p10       Tue 14:23
inf33778 Marek Flejszman       *p11       Tue 14:25   Informatyk  3 Rok I-2
%
```

nazwa użytkownika | prawdziwe nazwisko | czas zalogowania | dodatkowe informacje

czas braku aktywności terminala

Polecenia finger można też użyć do uzyskania informacji o użytkownikach pracujących w innym systemie, do którego jest dostęp przez sieć. Należy wówczas jako argument podać adres poprzedzony znakiem @.

```
% finger @syriusz.cs.put.poznan.pl
[syriusz.cs.put.poznan.pl]
Login       Name               TTY Idle    When    Where
szyper   Marcin Szyprowski      co   30 Mon 10:51
wozniak  Piotr Wozniak p.430   p2 4:00 Mon 09:27  draco
darek    Dariusz Wawrzyniak p  p3   28 Tue 16:24  wega
szyper   Marcin Szyprowski     p4 2:25 Mon 10:51  :0.0
maciejl  Maciej Lewandowski    p5      Tue 16:27  carina.cs.put.po
wozniak  Piotr Wozniak p.430   p6   42 Tue 14:40  spika
lis      Tomasz Lis p.414L te  p7    4 Tue 16:51  pc3.sc.cs.put.po
maciejl  Maciej Lewandowski    p8      Tue 16:28  carina:0.0
%
```

adres terminala

3.1.2.3. whodo

Polecenie whodo zwraca informacje o użytkownikach i uruchomionych przez nich procesach.

```
% whodo
Wed May  8 11:42:36 1996
wega

ttyp0    darek    11:02
    ttyp0   28049    0:03 csh
    ttyp0   27116    0:00 whodo

ttyp1    inf37664 11:33
    ttyp1   20192    0:03 sh

ttyp2    inf37656 11:23
```

terminal — *nazwa użytkownika* — *czas zalog.*

```
     terminal   identyfikator    czas    nazwa
                    procesu     aktyw.  procesu
    ttyp2   28218    9:55 a.out
    ttyp2   28219    0:00 a.out

ttyp3    inf37646 11:34
    ttyp3      17    0:03 sh
    ttyp3   19416    0:00 vi

ttyp4    inf37653 11:42
    ttyp4   20673    0:00 tset
    ttyp4   26630    0:04 sh

ttyp5    jurek    11:42
    ttyp5   19379    0:03 login
%
```

3.1.3. Informacja o dowolnym użytkowniku

Polecenie finger może zwrócić dokładną informację o użytkowniku istniejącym w systemie (nawet jeśli nie jest zalogowany), jeśli w parametrach polecenia zostaną podane jakieś konkretne informacje identyfikujące danego użytkownika, typu nazwa, rzeczywiste imię lub nazwisko (patrz. poniższy listing).

```
% finger michu
Login name: michu       (messages off)  In real life: Michal Szychowiak
Office:  rm 409,  tel 782378
Directory: /staff/michu                 Shell: /bin/csh
On since May  7 15:12:20 on ttyp0              2 minutes 18 seconds Idle Time
Project:  motto: to err is human, to forgive - divine
Plan:
****************************************
*    ___     ___                       *
*  [   ]   [   ]                       *
*  /   \   /   \                       *
* /  /\ \ / /\  \ ichal Szychowiak      *
*[___] [___] [___]                      *
*  e-mail:                             *
*     szychowiak@pozn1v.put.poznan.pl  *
*  Michal.Szychowiak@cs.put.poznan.pl  *   _,_/|
*                                      *   \o.O;
*                          ********** * =(___)=
****************************************     U
```

```
% finger Wawrzyniak
Login name: darek       (messages off)  In real life: Dariusz Wawrzyniak
Office:  rm. 410,  tel. 782378
Directory: /staff/darek                 Shell: /bin/csh
On since May  9 12:45:53 on ttyp2
No Plan.

Login name: inf35860                    In real life: Pawel Wawrzyniak
Office:  Informatyk,  1 Rok I-2
Directory: /student/inf35860            Shell: /bin/sh
Never logged in.
No Plan.

Login name: inf37733                    In real life: Piotr Wawrzyniak
Office:  Informatyk,  2 Rok I-4
Directory: /student/inf37733            Shell: /bin/sh
Never logged in.
No Plan.

Login name: inf37732                    In real life: Artur Wawrzyniak
Office:  Informatyk,  2 Rok I-4
Directory: /student/inf37732            Shell: /bin/sh
Never logged in.
No Plan.
%
```

Analogicznie można uzyskać informacje o użytkownikach w innych systemach. Należy wówczas podać informację identyfikującą (nazwę użytkownika, rzeczywiste nazwisko itp.) wraz z adresem sieciowym.

```
% finger darek@syriusz.cs.put.poznan.pl
[syriusz.cs.put.poznan.pl]
Login name: darek                       In real life: Dariusz Wawrzyniak p.410
tel.782-378
Directory: /syriusz/staff/darek         Shell: /bin/csh
Last login Tue Apr 30 10:34 on ttyp4 from clarissa
New mail received Tue May  7 15:26:24 1996;
  unread since Mon May  6 20:51:38 1996
No Plan.
```

3.2. Narzędzia komunikacji

3.2.1. Wysyłanie komunikatów

Polecenie write umożliwia wysłanie komunikatu do innego użytkownika pracującego w systemie. Komunikat jest dowolnym tekstem, który jest wprowadzany po wydaniu polecenia write <nazwa>. Treść komunikatu przekazywana jest po każdej zakończonej linii (czyli po naciśnieciu klawisza ENTER). Kombinacja klawiszy Control D jest informacją o zakończeniu wpisywania treści komunikatu. Na poniższych dwóch listingach pokazany jest odpowiednio terminal nadawcy i adresata komunikatu.

```
# write darek
 Czesc Darku!
#
```

```
%
 Message from michu on wega (ttyp10) [ Tue May 07 14:29:51 ] ...
 Czesc Darku!
(end of message)

%
```

Podobnie do polecenia write działa polecenie wall. W tym przypadku nie podaje się jednak adresata, a komunikat wysyłany jest do wszystkich użytkowników pracujących w systemie, którzy nie mają zablokowanych terminali (patrz pkt. 3). Poniższe dwa listingi przedstawiają odpowiednio terminal nadawcy i odbiorcy. Przekazywany komunikat trafia też na terminal nadawcy.

```
# wall
TEST

Broadcast Message from root (ttyp10) on wega   May 07 17:10 1996...
TEST

#
```

```
%
Broadcast Message from root (ttyp10) on wega   May 07 17:10 1996...
TEST
```

3.2.2. Dwustronna wymiana informacji

Jednoczesne przesyłanie informacji w obu kierunkach między dwoma użytkownikami pracującymi w systemie umożliwia polecenie talk. Na kolejnych dwóch listingach przedstawiony jest odpowiednio terminal użytkownika inicjującego "rozmowę" oraz użytkownika, z którym ma być nawiązana "rozmowa". "Rozmowa" może zostać nawiązana dopiero wówczas, gdy drugi z wymienionych użytkowników wyda polecenie zgodnie z podpowiedzią systemu (w tym konkretnym przypadku talk root@wega.cs.put.poznan.pl). Ekran (okno) terminala każdego z komunikujących się użytkowników jest wówczas dzielone na dwie części, z których jedna służy do przekazywania informacji, a druga do jej odbioru. Informacja jest przekazywana zaraz po jej wprowadzeniu na terminalu. Rozmowę może przerwać dowolna ze stron przez naciśnięcie klawisza przerwania procesu (Control C lub DEL, w zależności od terminala).

```
% talk darek
%
Message from Talk_Daemon@wega.cs.put.poznan.pl at 14:34 ...
talk: connection requested by michu@wega.cs.put.poznan.pl.
talk: respond with:  talk michu@wega.cs.put.poznan.pl
```

3.2.3. Blokowanie terminala

Ponieważ otrzymywanie komunikatów lub informacji o próbie nawiązania rozmowy może przeszkadzać i być niepożądane, użytkownik może się zakazać przyjmowania komunikatów, blokując terminal poleceniem mesg n. Odblokowanie terminala następuje przez wydanie polecenia mesg y.

3.2.4. Poczta elektroniczna

Poczta elektroniczna umożliwia dostarczenie listu elektronicznego do adresata, niezależnie od tego, czy jest on zalogowany w systemie. Konieczne jest posiadanie konta i wystarczająca ilość miejsca na dysku (lub partycji dysku), gdzie przechowywane są listy elektroniczne. O posiadaniu listów w skrzynce lub otrzymaniu nowych listów użytkownik jest informowany w czasie logowania.

Zarówno do wysyłania poczty elektronicznej jak i do przeglądania własnej skrzynki służy polecenie mail.

W celu wysłania listu należy wydać polecenie mail z nazwą adresata. Jeżeli konto adresata znajduje się w innym systemie niż konto nadawcy, należy podać również adres sieciowy tego systemu. Treść listu może być wprowadzona z klawiatury lub przygotowana w pliku i przekazana na standardowe wejście procesu wykonującego program mail.

Poniższy listing przedstawia wysłanie listu do użytkownika o nazwie darek, mającego konto w tym samym systemie, w którym ma konto nadawca. Treść listu oraz temat (ang. subject) wprowadzane są z klawiatury (tekst napisany pogrubioną czcionką jest wyświetlany przez system). W celu zakończenia redagowania listu należy wcisnąć kombinację klawiszy Control D, wówczas pojawi się pytanie o adresatów, do których ma zostać wysłana kopia listu (Cc - carbon copy). Podanie tych adresatów nie jest konieczne. Po podaniu (lub nie podaniu) adresatów kopii i wciśnięciu Enter nastąpi wysłanie listu.

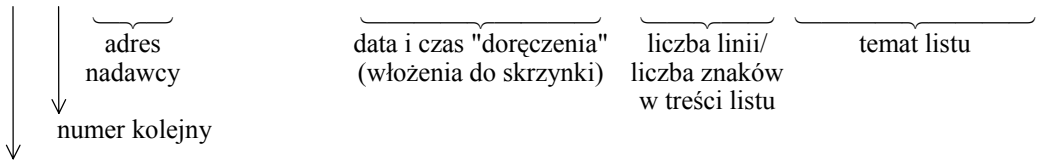
```
# mail darek
Subject: Pozdrowienia

Pozdrowienia z pokoju obok
root
Cc: michu
#
```

Poniższy listing przedstawia wysłanie listu do użytkownika o nazwie darek, mającego konto w systemie o adresie syriusz.cs.put.poznan.pl. Treść listu została przygotowana w pliku o nazwie pozdrowienia, którego zawartość została wyświetlona poleceniem more przed wysłaniem listu. Temat listu został podany w opcji -s polecenia mail.

```
% more pozdrowienia
Pozdrowienia z pokoju obok
root
% mail -sPozdrowienia darek@syriusz.cs.put.poznan.pl < pozdrowienia
%
```

Wydanie polecenia mail bez parametrów oznacza, że nastąpi wyświetlenie zawartości skrzynki, jak to przedstawia kolejny listing. Każdy z listów w skrzynce można odczytać, zapisać do pliku, usunąć, wysłać odpowiedź na ten list do nadawcy itp.

```
% mail
SCO System V Mail (version 3.2)  Type ? for help.
"/usr/spool/mail/darek": 2 messages 1 new 2 unread
>N  2 root             Tue May  7 15:27   11/305   Pozdrowienia
 U  1 root             Tue May  7 14:39   15/279   Dysk!
&
```



N - nowo odebrany list
U - nie odczytany list

4. Środowisko pracy w systemie Unix

4.1. Skrypty

Skrypty są plikami tekstowymi, zawierającymi polecenia dla powłoki (shell'a). W skryptach mogą też występować konstrukcje programotwórcze (pętle, instrukcje warunkowe, instrukcje wyboru) oraz możliwe jest przekazywanie parametrów z linii poleceń. Zawartość przykładowego skryptu przedstawia poniższy listing. Zadaniem skryptu jest utworzenie podkatalogu HOME2 w katalogu domowym i skopiowanie wszystkich plików z katalogu domowego do utworzonego podkatalogu HOME2.

```
% more copy_home
cd
mkdir HOME2
cp * HOME2
%
```

W celu wykonania poleceń zawartych w skrypcie należy uruchomić proces powłoki i przekazać nazwę skryptu jako parametr, jak to pokazano na poniższym przykładzie.

```
% csh copy_home
cp : <Chess> directory
cp : <ExMake> directory
cp : <HOME2> directory
cp: cannot open id
cp : <ipc> directory
cp : <potoki> directory
cp : <pres> directory
cp : <proc> directory
cp : <sig> directory
cp : <skrypty> directory
cp : <temp> directory
%
```

Inną możliwością jest nadanie prawa x (execute) do pliku będącego skryptem i podanie nazwy skryptu jako polecenia dla powłoki. System sam uruchomi wówczas jako proces potomny nową powłokę, która zinterpretuje zawartość skryptu (patrz przykład poniżej).

```
% chmod u+x copy_home
% copy_home
cp : <Chess> directory
cp : <ExMake> directory
cp : <HOME2> directory
cp: cannot open id
cp : <ipc> directory
cp : <potoki> directory
cp : <pres> directory
cp : <proc> directory
cp : <sig> directory
cp : <skrypty> directory
cp : <temp> directory
%
```

4.2. Zmienne powłoki

W powłoce można zdefiniować zmienne, których wartości (lub samo istnienie) wpływają na zachowanie się powłoki oraz uruchamianych przez nią procesów. Ze względu na zakres dostępności zmienne te dzielą się na:

lokalne - dostępne tylko w powłoce i nie przekazywane do procesów potomnych,

środowiskowe - dostępne w powłoce oraz we wszystkich procesach potomnych, w szczególności w procesach powłoki, uruchamianych po zdefiniowaniu odpowiednich zmiennych.

Do wartość zmiennej powłoki można się odwołać przez podanie nazwy zmiennej, poprzedzonej znakiem $.

Znaczenie wybranych zmiennych powłoki
home - ścieżka do katalogu domowego użytkownika, domyślny argument polecenia cd,
HOME - zmienna, na podstawie której definiowana jest wartość zmiennej home,
PATH - katalogi przeszukiwane w celu znalezienia plików z programami,
TERM - nazwa typu terminala,
MAIL - nazwa ścieżkowa pliku, który jest skrzynką pocztową.

4.2.1. Zmienne lokalne

Do definiowania zmiennych lokalnych służy polecenie set. Poniższy przykład przedstawia użycie polecenia set do zmiany znaku zachęty.

```
% set prompt='C:\> '
C:\>
```

Polecenie set bez parametrów zwraca na standartowe wyjście wartości wszystkich zmiennych lokalnych (patrz przykład poniżej).

```
C:\> set
LOGTTY  /dev/ttyp0
_d      ()
argv    ()
cdspell
history 20
home    /staff/darek
ignoreeof
noclobber
path    (/bin /usr/bin /staff/darek/bin .)
prompt  C:\>
shell   /bin/csh
status  0
C:\>
```

4.2.2. Zmienne środowiskowe

Do definiowania zmiennych środowiskowych służy polecenie setenv. Poniższy przykład przedstawia użycie polecenia setenv do zmiany katalogu domowego. Wartością zmiennej lokalnej home w potomnej powłoce csh jest /staff/darek/skrypty pomimo, że w powłoce macierzystej zmienna home miała wartość /staff/darek.

```
% setenv HOME /staff/darek/skrypty
% echo $home
/staff/darek
% csh
% echo $home
/staff/darek/skrypty
%
```

W celu otrzymania nazw wszystkich zmiennych środowiskowych należy użyć polecenia env.

```
% env
HOME=/staff/darek
PATH=/bin:/usr/bin:/staff/darek/bin:
LOGNAME=darek
TERM=xterm
HZ=100
TZ=CEU-1CEU,M3.5.0,M9.5.0
SHELL=/bin/csh
MAIL=/usr/spool/mail/darek
HUSHLOGIN=FALSE
%
```

4.3. Alias

Polecenie alias umożliwia nadanie alternatywnej nazwy poleceniu lub sekwencji poleceń. Można w ten sposób dostosować nazwy do własnych przyzwyczajeń lub skrócić długie nazwy lub sekwencje poleceń, które są często używane. Polecenie unalias usuwa nazwę alternatywną.

W poniższym przykładzie nadano alternatywną nazwę dir poleceniu listowania katalogu ls -l.

```
% alias dir 'ls -l'
% dir skrypty
total 14
-rwx------   1 darek    staff         76 Jan 05 10:38 licz
-rwx------   1 darek    staff        124 Jan 05 11:29 licz2
-rwx------   1 darek    staff          9 Jan 12 10:11 opcje
-rwx------   1 darek    staff         54 Feb 14 13:40 rodz_pliku
-rwx------   1 darek    staff         46 Jan 05 12:23 ts
-rwx------   1 darek    staff         53 Jan 12 10:48 zmienna
-rwx------   1 darek    staff         40 Jan 12 10:48 zmienna2
%
```

4.4. Pliki konfiguracyjne

Pliki konfiguracyjne są skryptami o charakterze specjalnym. Dla powłoki csh istotne znaczenie mają:

.cshrc - skrypt wykonywany każdorazowo przy uruchomieniu nowego procesu powłoki,

.login - skrypt wykonywany zaraz po zalogowaniu,

.logout - skrypt wykonywany tuż przed wylogowaniem (po wydaniu komendy exit, kończącej sesję).

Skrypty te są uruchamiane automatycznie. Skrypty .cshrc i .login zawierają najczęściej definicje zmiennych i aliasów, a skrypt .logout może być wykorzystany np. do uruchomienia poleceń porządkujących, typu usuwanie plików tymczasowych.