Marcin Tomana

SYSTEM UNIX

Wyższa Szkoła Informatyki i Zarządzania
Bielsko-Biała 2001

Marcin Tomana

SYSTEM UNIX

SŁOWO WSTĘPNE

Podręcznik ten przedstawia wiedzę, jaką powinien posiadać każdy, kto w jakikolwiek sposób ma do czynienia z systemem UNIX. Przedstawione zagadnienia przydatne są w pracy normalnego użytkownika, ale również każdy administrator serwera UNIX powinien, materiał zawarty w tym podręczniku, znać biegle i umieć go stosować. Nie przedstawiono w tym podręczniku informacji jak konfigurować i administrować systemy UNIX.

W podręczniku tym przedstawiono podstawowe usługi związane ze specyfiką internetu, a w szczególności protokołu TCP/IP, takie jak transmisja plików czy zdalna praca. Pokazano również jak pod systemem UNIX wykorzystywać pocztę elektroniczną oraz zagadnienia związane z podpisami elektronicznymi i szyfrowaniem poczty elektronicznej. Przedstawione zostały ogólne zasady funkcjonowania procesów jądra systemu oraz systemu plików. Szczegółowo zostały omówione zagadnienia związane z podstawami programowania w systemie UNIX w oparciu o powłoki systemu, aplikację 'sed' oraz 'awk'.

W obecnych czasach system UNIX wykorzystywany jest w zastosowaniach przemysłowych oraz powszechnie stosuje się go w internecie. Większość serwerów internetowych opartych o protokół TCP/IP wykorzystuje właśnie system UNIX. Biegła znajomość tego systemu jest bardzo poszukiwane na rynku pracy.

Prof. dr inż. Rościsław Buń

Spis Treści

1. Wprowadzenie

Niniejszy podręcznik pokazuje jak można wykorzystywać system UNIX w codziennej pracy normalnego użytkownika. Znajomość obsługi tak skomplikowanego systemu pozwala wykorzystywać go do bardzo różnych zadań.
Podręcznik ten nie jest przeznaczony bezpośrednio dla administratorów systemu UNIX. Jednak programy opisane tutaj muszą przez administratorów być bardzo dobrze znane. Całe „wnętrze" systemów UNIX opiera się na wielu skryptach, które przy pomocy małych programów konfigurują całe środowisko sieciowe. Administracja serwerów UNIX wymaga szerokiej znajomości tematyki sieci opartej o protokół TCP/IP, pewnych dodatkowych mechanizmów funkcjonujących w systemach UNIX, oraz w szczególności oprogramowania użytkowego opisanego w tym podręczniku.

System UNIX wykorzystywany jest głównie przez duże instytucję i firmy ze względu na bardzo dużą skalowalność tego systemu. Stosować go można w średnich firmach równie dobrze, jak w potężnych korporacjach opartych o klastry wieloserwerowe.

Najczęściej wykorzystywanym systemem operacyjnym w internecie jest UNIX i to dla niego najwięcej powstało aplikacji internetowych. Znajomość tego systemu umożliwia użytkownikom internetu lepsze wykorzystywanie możliwości jakie daje internet, a w szczególności wszelkich powszechnych usług takich jak poczta elektroniczna, transmisja plików czy programowanie aplikacji WWW ogólnie nazywanych ‘server-side’, czyli wykonywanych po stronie serwera.

2. Wstęp do systemów UNIX

2.1. Historia systemu i odmiany UNIX

Historia systemów UNIX sięga początków informatyki. Systemy UNIX istniały od początku systemów wielozadaniowych i można dzisiaj stwierdzić, że jest to najbardziej rozpowszechniony system na obecnych komputerach mimo, że na co dzień jest to trudne do zaobserwowania.

Sama nazwa UNIX oznacza nie konkretną implementację systemu operacyjnego, tylko całą rodzinę systemów, które korzystają z pewnej wspólnej idei. Praktycznie każdy producent większych komputerów posiada swój własny system operacyjny UNIX. Następstwem tego istnieją bardzo liczne odmiany systemów UNIX. Osoby pragnące znać system UNIX, muszą znać bardzo wiele wersji i odmian tego systemu. Praktycznie wszystkie systemy UNIX wykorzystują pewne wzorce, które przez lata rozwoju tych systemów dobrze ugruntowały się wśród producentów systemów UNIX. Różnice często wynikają z:

- różnic sprzętowych maszyn, na których eksploatowane są te systemy
- różnic w administracji szczegółowymi parametrami systemu
- różnic w środowiskach graficznych

Przyczyny te mogą odstraszać, lecz tak naprawdę te specyficzne dla wersji umiejętności potrzebną są tylko administratorom i nie są konieczne dla zdecydowanej większości osób zajmujących się UNIX'em.

Przykładów różnych odmian systemów operacyjnych UNIX można by wypisywać bardzo wiele. Przedstawiono tu tylko część tych najbardziej rozpowszechnionych w Polsce:

System UNIX	**Producent**	**Platforma**
Solaris	SUN	SUN, PC
SCO	SCO	PC
HP-UIX	HP	HP
IBM-AIX	IBM	IBM
QNX		Wiele
Linux		PC, Sun, Motorola

Rys. 1. Zestawienie przykładowych odmian systemów UNIX

W systemie UNIX praktycznie za podstawowy język używany do pisania zarówno jądra systemu jak i większości aplikacji, przyjęto język C. Wielu producentów sprzedając swoje systemy dostarcza wraz z nimi źródła systemu, co umożliwia ponowną kompilację niektórych aplikacji, czy samego jądra systemu. Dzięki temu może ono być bardziej zoptymalizowane dla konkretnej platformy sprzętowej.

Systemy UNIX generalnie są bardzo drogie i zazwyczaj wykorzystywane są do celów bardzo specyficznych np. Systemy Operacyjne Czasu Rzeczywistego (RTOS - Real Time Operating System). W Polsce wykorzystywane są raczej w dużych instytucjach np. bankach. Powszechnie systemy unixowe wykorzystywane są w internecie, stąd firmy zajmujące się internetem nastawione są głównie na ten system.

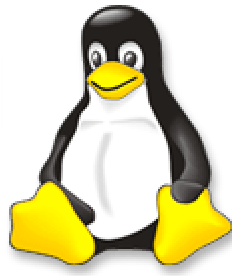
Systemy UNIX były raczej mało dostępne i niepopularne. Tę sytuację zmienił system Linux, który pojawił się jako darmowa wersja UNIX'a rozprowadzana przez Internet, posiadająca wszystkie elementy dużych rozbudowanych systemów UNIX'owych. Dzięki dużemu zainteresowaniu system Linux stał się bardzo rozpowszechnionym i łatwo dostępnym systemem. Jest wykorzystywany nawet w bardzo dużych przedsiębiorstwach, głównie jako oprogramowanie serwerów obsługujących fragmenty sieci połączonych z internetem.

System UNIX konfigurowany jest przez wiele różnych plików, których zapis często jest bardzo nieczytelny. W systemie UNIX istnieje polecenie man, dzięki któremu możliwy jest dostęp do bogatej dokumentacji opisującej wszystkie programy, pliki konfiguracyjne itp.

2.2. System operacyjny Linux

Linux jest nowoczesnym systemem operacyjnym. Posiada między innymi prawdziwą wielozadaniowość i wielowątkowość z pełną ochroną pamięci oraz dużą otwartość na wiele różnych platform sieciowych m.in. TCP/IP, IPX, SMB, Appletalk. Linux jest dostępny za darmo, na licencji GPL, co oznacza, że można go stosować za darmo także do celów komercyjnych, lecz jeżeli chce się go sprzedawać (np. tworząc zmodyfikowane wersje), trzeba udostępnić kod źródłowy. Pod względem wydajności, szybkości może spokojnie konkurować z drogimi komercyjnymi systemami operacyjnymi UNIX. Fenomenem jest sytuacja, iż system Linux na platformie SUN działa o wiele wydajniej niż oryginalny system SUN'a – Solaris. Ponadto, bardzo ważną zaletą jest pełna dostępność kodu źródłowego systemu i większości aplikacji. Linux posiada bardzo dużo użytkowników, którzy w internecie są bardzo aktywni i na wielu listach dyskusyjnych można znaleźć wysokiej klasy specjalistów, którzy pomogą rozwiązać problemy w konfiguracji czy użytkowaniu systemu.

Sama nazwa Linux dotyczy wyłącznie jądra systemu, zaś jądro z resztą oprogramowania systemowego nazywa się dystrybucją. Najbardziej znane dystrybucje to RedHat, Debian, Slackware czy też Polska PLD (Polish Linux Distribution).



Rys. 2. Logo systemu Linux - Pingwin

3. Usługi wykorzystywane w TCP/IP

3.1. Wstęp

W rozdziale tym opisane są ogólne zasady korzystania w systemie UNIX z podstawowych usług wykorzystywanych w sieciach TCP/IP. Przedstawiona jest tu możliwość jaka dają systemy UNIX – zdalna praca na serwerze. Pokazana jest najpowszechniejsza aplikacja – telnet, ale również aplikacja z szyfrowaną transmisją danych – ssh. Omówiona jest transmisja plików przy pomocy protokołu ftp oraz szyfrowanego protokołu ssh. Pokazana jest możliwość przeglądania i ściągania stron www w trybie tekstowy oraz szeroko pojete wykorzystanie poczty elektronicznej - aplikację, podpisy cyfrowe, szyfrowanie poczty.

Wszelkie usługi TCP/IP opisane są od strony użytkowej. Nie ma przedstawionych informacji, jak konfigurować i administrować takimi serwisami na serwerach.

3.2. Zdalna praca na serwerze unix

Program ‘telnet’ daje możliwość zdalnej pracy na serwerze UNIX. Po połączeniu się z serwerem, oprogramowanie serwera pyta się o użytkownika i hasło. Tylko po podaniu poprawnych danych możliwa jest zdalna praca.
Telnet może być wykorzystywany przez komputer oparty o Windows a także UNIX. Telnet jest usługą bardzo podobną do zdalnej pracy przez terminal. Terminal jednak korzysta z połączenia szeregowego RS232, zaś telnet „udaje” terminal korzystając z sieci TCP/IP. Klient telnetu jest bardzo prostym programem i występuje w każdym systemie operacyjnym obsługującym TCP/IP.

Wszyscy korzystający z protokołu telnet powinni zdawać sobie sprawę, że protokół ten przesyła hasło użytkownika w postaci niezaszyfrowanej przez sieć i

istnieje prosta możliwość podsłuchania tego hasła. Hasło to zazwyczaj jest hasłem umożliwiającym dostęp do innych, usług takich jak pop3 czy ftp.

Przykładowa sesja zamieszczona jest poniżej

```
unix1%~>telnet unix2.domena2.net
Trying 192.168.4.12.
Connected to unix2.domena2.net.
Escape character is '^]'.

SunOS 5.6

login: marcin
Password:
Last login: Fri Oct 26 09:11:50 from marcin-notebook.domena1.net
Sun Microsystems Inc.   SunOS 5.6       Generic August 1997
unix2%~>uname –a
SunOS unix2 5.6 Generic_105181-21 sun4u sparc SUNW,Ultra-1
```

Pracując na serwerze można zmienić hasło do konta pocztowego przy pomocy instrukcji ‘passwd’. Po wykonaniu polecenia należy wprowadzić bieżące hasło, następnie nowe hasło trzeba wprowadzić dla weryfikacji dwa razy.

W zależnośni od ustawień systemu wymagania dotyczące hasła oraz komunikaty mogą być bardzo różne. Niektóre systemy mogą mieć wbudowane słowniki słów kluczowych występujących najczęściej w hasłach i nie pozwolą wprowadzić hasła bazującego na tych słowach. Procedura zmiany hasła może być wtedy dość uciążliwia. Hasło jednak powinno być dość skomplikowane. Zagwarantuje to bezpieczeństwo systemu i innych usług takich jak ftp, ssh.

```
sun%/home/marcin >passwd
passwd:  Changing password for marcin
Enter login password:
passwd(SYSTEM): Sorry, wrong passwd
Permission denied
sun%/home/marcin >passwd
passwd:  Changing password for marcin
Enter login password:
New password:
passwd(SYSTEM): Password too short - must be at least 8 characters.
New password:
Re-enter new password:
passwd(SYSTEM): They don't match; try again.
New password:
Re-enter new password:
passwd (SYSTEM): passwd successfully changed for marcin
```

Rys. 3. Przykładowa sesja zmiany hasła

Wszelkie sesje można sprawdzać przy pomocy polecenia ‘last’. Polecenie to wyświetla informacje kiedy, jak długo oraz nazwę komputera z jakiego pracowano na podanym koncie. Wyświetlane są sesje telnet, ssh i ftp.

```
[root@tel AVP]# last marcin
marcin   pts/0        tv.wizja.net     Sat Nov 10 10:07   still logged in
marcin   pts/1        tel.wizja.net    Wed Nov  7 09:53 - 09:56  (00:02)
marcin   tty5                          Wed Nov  7 09:53   still logged in
marcin   pts/0        tv.wizja.net     Tue Nov  6 15:35 - 15:36  (00:01)
marcin   pts/0        tv.wizja.net     Tue Nov  6 14:19 - 14:20  (00:00)
marcin   ftpd13090    tel.wizja.net    Tue Nov  6 11:37 - 11:37  (00:00)
marcin   pts/0        tv.wizja.net     Mon Nov  5 16:42 - 16:54  (00:12)
marcin   pts/0        marcin-notebook. Mon Nov  5 09:36 - 13:16  (03:39)
marcin   pts/0        pl111.bielsko.cv Sun Nov  4 16:54 - 17:21  (00:26)
marcin   pts/1        marcin-notebook. Sat Nov  3 09:56 - 16:50  (06:53)
```

Rys. 4. Przykładowy spis sesji wyświetlany przy pomocy polecenia last

Pracując zalogowany jako dowolny użytkownik można przy pomocy polecenia ‘su’ uruchomić shell jako inny użytkownik oraz z jego uprawnieniami. Polecenie umożliwia dostęp dopiero po sprawdzeniu hasła. Wykonując polecenie ‘su — użytkownik’ uruchamiany jest tzw. login shell.

3.3. Zdalna praca z szyfrowaną transmisją danych

Usługa ssh jest bardzo podobna do telnet. Umożliwia również zdalną pracę na serwerze UNIX. Jest od usługi telnet znacznie lepsza ponieważ cała transmisja pomiędzy klientem a serwerem jest szyfrowana i nie ma możliwości podsłuchania hasła czy przesyłanych danych.

Podczas nawiązywania pierwszego połączenia pobierany jest specjalny klucz publiczny serwera, co umożliwia takie szyfrowanie danych, że tylko serwer będzie mógł odszyfrować transmisje danych.

Usługę ssh można również wykorzystywać do zaszyfrowanej transmisji plików na serwer.

Przykładowa sesja zamieszczona jest poniżej:

```
[marcin@trex marcin]$ ssh elan.elfin.pl
Warning: Permanently added the DSA host key for IP address '213.25.37.35' to
the list of known hosts.
Enter passphrase for key '/home/kadra/marcin/.ssh/identity':
marcin@elan.elfin.pl's password:
Permission denied, please try again.
```

```
marcin@elan.elfin.pl's password:
Last login: Fri Oct 26 10:33:02 2001 from pl232.bielsko.cvx.ppp.tpnet.pl
You have mail.
[marcin@elan marcin]$
```

Wykorzystując protokół SSH można również w bezpieczny sposób wgrywać i odbierać pliki z serwera. Pod unix do tego celu służy instrukcja ‘scp’.

Uproszczona składnia jest następująca:

scp [parametry] [[user@]host1:]file1 [...] [[user@]host2:]file2

Jako parametry można podać np. –r oznaczający kopiowanie włącznie z podkatalogami.

Poniżej zamieszczono przykładową sesję kopiowania pliku ze zdalnego komputera na komputer lokalny.

```
[root@tel marcin]# scp -r marcin@tv.wizja.net:public_html/named.txt .
marcin@tv.wizja.net's password:
named.txt            100% |*********************************| 42382
00:00
```

3.4. Transmisja plików - FTP

Bardzo często w sieci wykorzystywany jest protokół FTP umożliwiający transmisje plików pomiędzy komputerami. Jest to protokół, który służy raczej do okresowej transmisji, niż do ciągłego przesyłania danych. Obsługa programów jest interaktywna, tzn. wydaje się różne polecenia, które są przez system odpowiednio interpretowane.

Pod Windows istnieje bardzo wiele programów obsługiwanych graficznie, w których praca jest bardzo wygodna i przyjemna.

W każdym praktycznie systemie operacyjnym obsługującym TCP/IP istnieje obsługa protokołu FTP w postaci programu interpretującego polecenia.

Po uruchomieniu programu dostępne są następujące podstawowe polecenia:

open [serwer]

Polecenie otwiera połączenie z innym komputerem. Komputer ten musi być serwerem FTP. Nie może to być np. komputer Windows bez specjalnego oprogramowania.

Po zestawieniu połączenia system poprosi o podanie użytkownika i hasła.

W przypadku systemów Anonimowego FTP użytkownika podaje się ‘ftp’ lub ‘anonymous’, hasło dowolne (najczęściej w postaci swojego e-maila). Jeżeli zalogowanie się nie powiedzie to nie trzeba ponownie się łączyć z serwerem tylko wystarczy użyć polecenia user.

close

Zamyka połączenie z serwerem.

user [użytkownik]

Zmienia w ramach aktualnego połączenia zalogowanego użytkownika.

ls

dir

Operacje wyświetlające katalog na zdalnym serwerze.

cd [katalog]

Zmiana katalogu bieżącego na zdalnym serwerze.

bin

Ustawia tryb transmisji plików na binarny. Stosowany do transmisji plików graficznych oraz wszelkich plików binarnych. Transmisja taka gwarantuje, że żaden bajt w pliku nie jest modyfikowany.

UWAGA: w niektórych systemach FTP tryb bin nie jest trybem domyślnym i należy go specjalnie włączyć, inaczej w przypadku transmisji plików binarnych zostaną one uszkodzone.

ascii

Ustawia tryb transmisji plików tekstowych i tylko do takich powinien on być stosowany. Transmisja uwzględnia różnice w notacjach plików tekstowych w różnych systemach np. UNIX-DOS, Windows. Jest jedna różnica znacząca w tych systemach – zapis znaku końca linii.

get [plik]

Operacja ta umożliwia pobieranie pliku ze zdalnego serwera na lokalny komputer. W podstawowych systemach FTP instrukcją tą można pobrać tylko 1 plik bez

nazw wieloznacznych (np. ‘*’,‘?’). W większych systemach FTP operacja ta obsługuje pobieranie nawet całych katalogów z podkatalogami.

mget [pliki]

Operacja ta umożliwia pobieranie ze zdalnego serwera wiele plików jednocześnie. Można używać nazw wieloznacznych (np. ‘*’,‘?’)

put [plik]

Operacja ta umożliwia wysyłanie na zdalny serwer 1 pliku bez nazw wieloznacznych.

mput [plik]

Operacja ta umożliwia wysyłanie na zdalny serwer wiele plików jednocześnie z użyciem nazw wieloznacznych.

Polecenia z ! na początku

Wszelkie polecenia, które wykonywane są przez lokalny komputer. Np. !dir wyświetli katalog bieżący z lokalnego komputera, a nie ze zdalnego serwera.

Pod systemami UNIX istnieje wiele programów klienta FTP o bardzo dużych możliwościach z przechowywaniem książki adresowej, z zapamiętywaniem ostatnio odwiedzanych katalogów i z wieloma ułatwieniami. Jednym z takich programów jest ncftp.

```
NcFTP Bookmark Editor

                                        Number of bookmarks:   6
  Open selected site:       <enter>     avp-db           ftp.kaspersky.pl
  Edit selected site:       /ed         icm-contrib      ftp.icm.edu.pl
  Delete selected site:     /del        icm-docs         ftp.icm.edu.pl
  Duplicate selected site:  /dup        icm-kernel       ftp.icm.edu.pl
  Add a new site:           /new        icm-redhat-rpms  ftp.icm.edu.pl
                                        proftp           ftp.proftpd.org
  Up one:                   <u>         ~
  Down one:                 <d>         ~
  Previous page:            <p>         ~
  Next page:                <n>         ~
                                        ~
  Capital letters selects first         ~
    site starting with the letter.      ~
                                        ~
  Exit the bookmark editor: <x>         ~
                                        ~
                                        ~
                                        ~
                                        ~
                                        ~
                                        ~
                                        ~
                                        ~
                                        ~

  ftp://ftp.icm.edu.pl/pub/linux/redhat/linux/7.2/en/os/i386/RedHat/RPMS/
```

Rys. 5. Przykładowy ekran z książką adresową programu ncftp

Wszyscy korzystający z protokołu FTP powinni zdawać sobie sprawę, że protokół ten przesyła hasło użytkownika w postaci niezaszyfrowanej przez sieć i istnieje prosta możliwość podsłuchania tego hasła. Hasło to zazwyczaj jest hasłem umożliwiającym dostęp do innych usług takich jak pop3,telnet czy ssh.
Przykładowa sesja zamieszczona jest poniżej

```
[marcin@tv marcin]# ftp tel.wizja.net
Connected to tel.wizja.net (212.160.214.3).
220 ProFTPD 1.2.2rc2 Server (WizjaNet FTP Server) [tel.wizja.net]
Name (tel.wizja.net:marcin):
331 Password required for marcin.
Password:
230 User marcin logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd linux
250 CWD command successful.
ftp> dir
227 Entering Passive Mode (212,160,214,3,129,138).
150 Opening ASCII mode data connection for file list
drwxrwxrwx   5 640      man          4096 Nov  6 14:46 amavisd-snapshot-
20010714
-rw-r--r--   1 root     root       290862 Nov  3 08:58 amavisd-snapshot-
20010714.tar.gz
-rw-r--r--   1 marcin   wizja       50826 Nov  3  2000 arc-5.21e-4.i386.rpm
drwxr-xr-x   2 root     root         4096 Nov  3 09:17 avp
drwxr-xr-x   4 root     1001         4096 Nov  3 09:07 avpselinux
-rw-r--r--   1 root     root      2754489 Sep 13 14:22
avpselinux3.0.135.2.tar.gz
drwxr-xr-x   3 root     root         4096 Nov  3 10:23 kavselinux
-rw-r--r--   1 root     root      4636423 Oct 16 08:50 kavselinux.tar.gz
-rw-r--r--   1 root     root        27126 Jul  7  1999 lha-1.14e-1.i386.rpm
-rw-r--r--   1 root     root     28561727 Oct 24 02:28 linux-2.4.13.tar.gz
-rw-r--r--   1 root     root        20596 Sep  7 17:59 perl-MIME-Base64-2.12-
6.i386.rpm
-rw-r--r--   1 root     root       371372 Jun 12 18:13 proftpd-1.2.2rc2-
1.i686.rpm
-rw-r--r--   1 root     root         5292 Jun 12 18:13 proftpd-inetd-
1.2.2rc2-1.i686.rpm
-rw-r--r--   1 root     root         5591 Jun 12 18:13 proftpd-standalone-
1.2.2rc2-1.i686.rpm
-rw-r--r--   1 root     root       109603 May 19  1999 rar-2.50-3.i386.rpm
-rw-r--r--   1 root     root        46467 Nov  7 13:10 sendmail.cf
-rw-r--r--   1 root     root        46307 Nov  7 13:10 sendmail.orig.cf
-rw-r--r--   1 root     root        56554 May 17  1999 zoo-2.10-4.i386.rpm
226 Transfer complete.
ftp> bin
200 Type set to I.
ftp> get rar*
227 Entering Passive Mode (212,160,214,3,129,140).
550 rar*: No such file or directory
ftp> mget rar*
mget rar-2.50-3.i386.rpm? y
227 Entering Passive Mode (212,160,214,3,129,144).
150 Opening BINARY mode data connection for rar-2.50-3.i386.rpm (109603
bytes).
226 Transfer complete.
109603 bytes received in 0.005 secs (2.1e+04 Kbytes/sec)
ftp> quit
221 Goodbye.
```

Rys. 6. Przykładowa sesja FTP

Należy pamiętać, że przy pomocy transmisji binarnej przegrywane pliki są w postaci niezmienionej. Jeżeli np. z poziomu systemu DOS albo Windows zostanie

wgrany plik tekstowy przy pomocy transmisji binarnej, to pod systemami UNIX plik ten może nie być czytelny. Dostępne są narzędzia takie jak ‘dos2unix’ czy ‘unix2dos’, które umożliwiają konwersje plików tekstowych pomiędzy systemami. Przykładowe wykonanie instrukcji zamieszczone jest poniżej:

```
[root@tv marcin]# dos2unix powiaty.txt
dos2unix: converting file powiaty.txt to UNIX format ...
```

Istnieje możliwość pobierania plików w tle. Najlepszym narzędziem do tego celu jest aplikacja ‘screen’ szerzej opisana w rozdziale 4.4 na stronie 31.

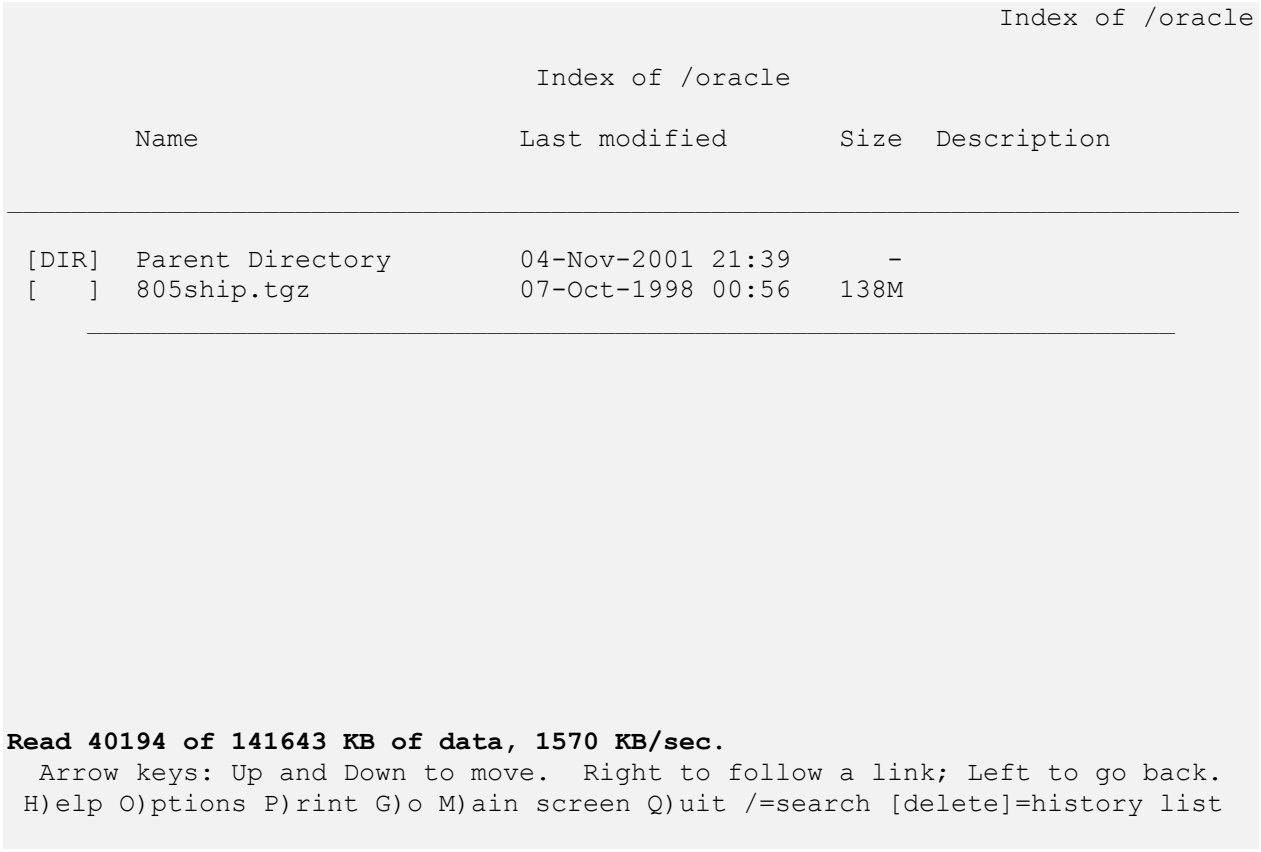
3.5. Przeglądanie stron WWW

W trybie tekstowym pod systemami UNIX istnieje możliwość przeglądania stron WWW. Niewidoczne są wtedy wszelkie elementy graficzne, lecz teksty zamieszczone na stronie da się odczytać. Wiele stron tworzonych jest tak, że da się zamieszczony na nich tekst dobrze czytać. Do przeglądania stron w trybie tekstowym służy aplikacja ‘lynx’.

```
                                                    System Rozkladu Zajec WSIiZ

                        Internetowy Serwis Rozkladu Zajęć

   Zajęcia dla wykładowcy
   Zajęcia na sale
   Zajęcia dla grupy
     ____________________________________________________________________________

   Rezerwacja tematu projektu
   Przegladanie terminow projektow
     ____________________________________________________________________________

   Rezerwacja na zajecia
   Obecnosc na zajeciach
     ____________________________________________________________________________

   Terminy sesji egzaminacyjnej
   Terminy sesji dla Wykladowcy
   Wydruk egzaminow poprawkowych
   [wsiLogo.gif]

                                                                 Scripts by M.Tomana

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Rys. 7. Przykładowa strona WWW wyświetlana przez aplikacje lynx

W przypadku spisu plików lub jeśli link wskazuje na jakiś plik, przy pomocy klawisza ‘d’ można uruchomić pobieranie takiego pliku (Download). Po odebraniu pliku program pyta się gdzie zapisać pobrany plik. Pobieranie takie można uruchomić w tle. Najlepszym narzędziem do tego celu jest aplikacja ‘screen’, szerzej opisana w rozdziale 4.4 na stronie 31.



Rys. 8. Pobieranie pliku przy pomocy aplikacji lynx

3.6. Poczta elektroniczna

Poczta elektroniczna na serwerze UNIX przechowywana najczęściej jest w katalogu /var/spool/mail w postaci pliku o nazwie takiej, jak nazwa użytkownika. Jest to plik tekstowy, który zawiera sekwencyjnie wszystkie maila użytkownika. Pliki takie mogą znajdować się również w innych katalogach użytkownika np. ~/mail. Obsługa tych plików jest standardowa i wykorzystywane są one przez wiele programów UNIX takich jak sendmail, mail, procmail, fetchmail, pine, elm.

Poczta elektroniczna może być pobierana przez programy, które są klientami POP3 np. Microsoft Outlook Express, Netscape Mail, Eudora. Programy te pobierają przez sieć TCP/IP zawartość skrzynki i przesyłają na komputer lokalny. Istnieje możliwość zdalnego czytania poczty elektronicznej po zalogowaniu się na serwer UNIX.

Istnieje wiele programów stworzonych do tego celu. Na wszystkich systemach UNIX znajduje się program mail. Obsługa tego programu jest interaktywna. Wydając jednoliterowe polecenia wykonujemy różne funkcje tego programu.

Przykładowe polecenia programu mail (wersja na SunOS 5.6)

+ następny mail

- poprzedni mail

d [n] usunięcie maila ze skrzynki o numerze n

q wyjście z programu

p wyświetlenie maila

Program mail można wykorzystywać również do wysyłania maili. Składnia jest następująca:

mail –s "temat wiadomosc" konto@domena

Po wykonaniu instrukcji program czyta z klawiatury treść maila i po wprowadzeniu kodu EOF (pod unix Ctrl+D) mail jest wysyłany do docelowego adresata.

Innym przykładem programu czytającego pocztę jest program ‘pine’. Jest to przykład programu bardzo rozbudowanego, umożliwiającego większość podstawowych operacji jakie są potrzebne w przypadku obsługi poczty elektronicznej.

Dostępna jest książka adresowa, obsługa wielu folderów, sortowanie i przeszukiwanie folderów z pocztą. Obsługa jest bardzo prosta i przejrzysta.

```
  PINE 4.33   MAIN MENU                        Folder: INBOX  1,072 Messages

            ?     HELP               -  Get help using Pine
            C     COMPOSE MESSAGE    -  Compose and send a message
            I     MESSAGE INDEX      -  View messages in current folder
            L     FOLDER LIST        -  Select a folder to view
            A     ADDRESS BOOK       -  Update address book
            S     SETUP              -  Configure Pine Options
            Q     QUIT               -  Leave the Pine program

   Copyright 1989-2001.  PINE is a trademark of the University of Washington.

? Help                       P PrevCmd                  R RelNotes
O OTHER CMDS  > [ListFldrs] N NextCmd                  K KBLock
```

Rys. 9. Przykładowy ekran programu pine do obsługi poczty elektronicznej

Następnym problemem występującym przy obsłudze poczty elektronicznej jest możliwość podejmowania pewnych akcji w momencie odebrania przez serwer maila dla konta pocztowego. Do tego celu wykorzystuje się program procmail. Każdy użytkownik może w swoim domowym katalogu stworzyć plik **.procmailrc**, w którym definiuje reguły, jakimi ma się kierować serwer w momencie odebrania maila dla tego użytkownika.

Plik ten zawiera deklaracje pewnych zmiennych oraz szereg reguł postępowania.

Zmienne definiują np. katalog z folderami (plikami) zawierającymi pocztę. Katalog ten może być np. ~/mail - wspólny z programem pine, co umożliwia rozrzucanie sobie poczty do różnych folderów.

Przykładowy plik wraz z opisem zamieszczono poniżej:

```
MAILDIR=/home/kadra/marcin/mail

# przeniesienie maili zawierajacych w polu To tekst 'blug@' do folderu blug
:0
* ^To.*blug@
blug

# przeniesienie maili zawierajacych w polu From tekstu 'news' do folderu news
:0
* ^From.*news
news

# przekazanie KOPI maila do skryptu
:0 c
| /home/kadra/marcin/sendsms

# NAJPROSTSZY PRZYKLAD FORWARDU MAILI NA TELEFON KOMORKOWY
# przekazanie KOPI maila jesli w temacie jest slowo 'pilne'
#  program formail zostawia tylko naglowek Subject i From i przekazuje maila
#  do wysylki na bramke smsowa
:0 c
* Subject: .*pilne
| formail -X Subject -X From | sendmail '601601601@sms.wsi.edu.pl'
```

Rys. 10. Przykładowy plik .procmail wraz z komentarzami

3.7. Wykorzystanie PGP w UNIX

PGP to system uwierzytelniania poczty elektronicznej. W skrócie umożliwia on:

- sprawdzenie czy faktycznie mail pochodzi od nadawcy,
- sprawdzenie czy podczas transmisji nie został zmodyfikowany,
- szyfrowanie zawartości maili.

Istnieją wersje PGP pod Windows a także dla systemu UNIX. Najbardziej rozbudowane programy pocztowe mają wsparcie dla PGP. Jednym z takich programów jest 'pine'.

Na systemach UNIX funkcjonuje pakiet GNU PGP, który można bardzo łatwo zintegrować z programem pine. Pakiet ten jest udostępniany na zasadch publicznej licencji GNU, więc nie ma żadnych problemów licencyjnych czy opłat. W pliku '/etc/pine.conf" lub '~/.pinerc' muszą wpisane być filtry, takie jak poniżej.

```
display-filters=_LEADING("-----BEGIN PGP MESSAGE-----")_ /usr/bin/gpg-check,
        _LEADING("-----BEGIN PGP SIGNED MESSAGE-----")_ /usr/bin/gpg-check

sending-filters=/usr/bin/gpg-sign,
        /usr/bin/gpg-encrypt _RECIPIENTS_,
        /usr/bin/gpg-sign+encrypt _RECIPIENTS_
```

Rys. 11. Filtry programu pine dla korzystania z pakietu GNU PGP

W tym momencie, jeśli będziemy czytali maile podpisane PGP, program pine automatycznie będzie weryfikował podpisy.

W przypadku wysyłania maili, po naciśnięciu kombinacji klawiszy ‘CTRL+X’, pojawi się możliwośc wysłania maila korzystając ze zdefiniowanych filtrów. Przy pomocy kombinacji klawiszy ‘CTRL+N’ można zmienić filtr i wysłać maila podpisanego elektronicznie przez PGP.

```
  PINE 4.33   COMPOSE MESSAGE                  Folder: INBOX  1,514 Messages

To      : ela@tomana.com
Cc      :
Attchmnt:
Subject : wazne informacje
----- Message Text -----
Przesylam Ci bardzo wazne informacje dotyczace ...



_____________________________________________________________
Marcin Tomana, tel. 0601 843199,(33) 818 95 95, fax. 0601 840166
email: marcin@tomana.net       www: http://www.marcin.tomana.net




Send message (filtered thru "gpg-sign+encrypt")?
 ? Help      Y [Yes]       ^P Prev Filter ^R Background
^C Cancel    N No          ^N Next Filter
```

Rys. 12. Przykładowy wybór filtra programu pine do podpisywania i szyfrowania poczty przy pomocy PGP

Do prawidłowego funkcjonowania programu pine i GnuPGP potrzebne jest stworzenie katalogów ‘.pinepgp’ oraz ‘.gnupg’

Do podpisywania potrzebna jest lokalna baza kluczy tzw. ‘keyring’, w którym umieszczone będą nasze klucze prywatne (dostępne tylko na hasło) oraz klucze publiczne zaufanych osób.

Do obsługi lokalnej bazy kluczy służy program ‘gpg’. Poniżej pokazano możliwe opcje programu gpg. Najważniejsze zaznaczono czcionką wytłuszczoną.

```
[marcin@trex marcin]$ gpg --help
gpg (GnuPG) 1.0.4
Copyright (C) 2000 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Home: ~/.gnupg
Supported algorithms:
Cipher: 3DES, CAST5, BLOWFISH, RIJNDAEL, RIJNDAEL192, RIJNDAEL256, TWOFISH
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA, ELG
Hash: MD5, SHA1, RIPEMD160

Syntax: gpg [options] [files]
sign, check, encrypt or decrypt
default operation depends on the input data

Commands:

 -s, --sign [file]                make a signature
     --clearsign [file]           make a clear text signature
 -b, --detach-sign                make a detached signature
 -e, --encrypt                    encrypt data
 -c, --symmetric                  encryption only with symmetric cipher
     --store                      store only
 -d, --decrypt                    decrypt data (default)
     --verify                     verify a signature
     --list-keys                  list keys
     --list-sigs                  list keys and signatures
     --check-sigs                 check key signatures
     --fingerprint                list keys and fingerprints
     --list-secret-keys           list secret keys
     --gen-key                    generate a new key pair
     --delete-key                 remove key from the public keyring
     --delete-secret-key          remove key from the secret keyring
     --sign-key                   sign a key
     --lsign-key                  sign a key locally
     --edit-key                   sign or edit a key
     --gen-revoke                 generate a revocation certificate
     --export                     export keys
     --send-keys                  export keys to a key server
     --recv-keys                  import keys from a key server
     --import                     import/merge keys
     --list-packets               list only the sequence of packets
     --export-ownertrust          export the ownertrust values
     --import-ownertrust          import ownertrust values
     --update-trustdb             update the trust database
     --check-trustdb [NAMES]      check the trust database
     --fix-trustdb                fix a corrupted trust database
     --dearmor                    De-Armor a file or stdin
     --enarmor                    En-Armor a file or stdin
     --print-md algo [files]      print message digests

Options:

 -a, --armor                      create ascii armored output
 -r, --recipient NAME             encrypt for NAME
     --default-recipient NAME     use NAME as default recipient
```

```
     --default-recipient-self     use the default key as default recipient
 -u, --local-user                 use this user-id to sign or decrypt
 -z N                             set compress level N (0 disables)
     --textmode                   use canonical text mode
 -o, --output                     use as output file
 -v, --verbose                    verbose
 -q, --quiet                      be somewhat more quiet
     --no-tty                     don't use the terminal at all
     --force-v3-sigs              force v3 signatures
     --force-mdc                  always use a MDC for encryption
 -n, --dry-run                    do not make any changes
     --batch                      batch mode: never ask
     --yes                        assume yes on most questions
     --no                         assume no on most questions
     --keyring                    add this keyring to the list of keyrings
     --secret-keyring             add this secret keyring to the list
     --default-key NAME           use NAME as default secret key
     --keyserver HOST             use this keyserver to lookup keys
     --charset NAME               set terminal charset to NAME
     --options                    read options from file
     --status-fd FD               write status info to this FD
     --trusted-key KEYID          ulimately trust this key
     --load-extension FILE        load extension module FILE
     --rfc1991                    emulate the mode described in RFC1991
     --openpgp                    set all packet, cipher and digest options
to OpenPGP behavior
     --s2k-mode N                 use passphrase mode N
     --s2k-digest-algo NAME       use message digest algorithm NAME for
passphrases
     --s2k-cipher-algo NAME       use cipher algorithm NAME for passphrases
     --cipher-algo NAME           use cipher algorithm NAME
     --digest-algo NAME           use message digest algorithm NAME
     --compress-algo N            use compress algorithm N
     --throw-keyid                throw keyid field of encrypted packets
 -N, --notation-data NAME=VALUE   use this notation data

(See the man page for a complete listing of all commands and options)

Examples:

 -se -r Bob [file]          sign and encrypt for user Bob
 --clearsign [file]         make a clear text signature
 --detach-sign [file]       make a detached signature
 --list-keys [names]        show keys
 --fingerprint [names]      show fingerprints

Please report bugs to <gnupg-bugs@gnu.org>.
```

Rys. 13. Dostępne opcje programu gpg z pakietu GnuPGP

Zawsze należy stworzyć parę swoich kluczy a następnie wyeksportować nasz klucz publiczny albo na serwer albo do pliku, który następnie przekażemy zaufanym osobom. Przykładowa sesja generowania pary kluczy prywatnego i publicznego zamieszczona jest poniżej.

```
[marcin@trex marcin]$ gpg --gen-key
gpg (GnuPG) 1.0.4; Copyright (C) 2000 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
```

```
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

gpg: Warning: using insecure memory!
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
Your selection? 1
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024)
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1w
Key expires at Fri Nov 23 10:21:48 2001 CET
Is this correct (y/n)? y

You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Marcin Tomana
Email address: marcin@tomana.com
Comment:
You selected this USER-ID:
    "Marcin Tomana <marcin@tomana.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++++++++++++..+++++.++++++++++.++++++++++++++++++.++++++++++++++++++++
++++++++++++++++++++++.+++++++++++++++.++++++++++..+++++.+++++>...+++++..+++
++.............................................................................
..............................+++++
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
+++++++++++++++++++++++++++++++++....+++++.+++++++++++++++.+++++...++++++++
+++++++.+++++.++++++++++.++++++++++.++++++++++++++++++++.++++++++++.........+
++++^^^
public and secret key created and signed.
```

Rys. 14. Przykładowa sesja generowania klucza prywatnego i publicznego

3.8. Zagadnienia kontrolne

1. Zdalna praca przy pomocy telnet i ssh
2. Lista sesji telnet, ssh, ftp – last
3. Wykonywanie zadań jako inny użytkownik
4. Kopiowanie plików przy pomocy tekstowego narzędzia ftp oraz scp
5. Transmisja plików w trybie tekstowym
6. Przeglądanie stron WWW i pobieranie plików przez http przy pomocy lynx
7. Obsługa poczty elektronicznej przy pomocy mail i pine
8. Generowanie kluczy do podpisów elektronicznych oraz szyfrowania
9. Wykorzystanie PGP w programie pine

4. Procesy i zadania w unixie

4.1. Wstęp

W rozdziale tym przedstawiona jest ogólnie tematyka związana z działaniem procesów w UNIX. Pokazane są podstawowe operacje na procesach dostępne dla użytkownika. Zaprezentowane instrukcje i programy umożliwiają większe wykorzystanie mocy obliczeniowej serwera w przypadku pracy wielozadaniowej. Przedstawione są również metody wykonywania zadań w tle, co bardzo często wykorzystywane jest do ściągania plików z internetu. Pokazano narzędzie cron, które umożliwia regularne wykonywanie zadań w tle bez ingerencji użytkownika.

4.2. Pojęcie procesu

W systemie UNIX każdy program uruchamiany jest jako proces, który może powołać do życia następne procesy. Każdy proces ma jeden proces nadrzędny, który go powołał do życia. Przy zamykaniu procesu najpierw zamykane są wszelkie procesy potomne. Z każdym procesem związanych jest wiele informacji, do których dostęp możliwy jest poprzez instrukcję ps. Instrukcja ta ma różne parametry w zależności od systemu i platformy sprzętowej. O ile nie zostanie to w systemie zablokowane, użytkownik ma dostęp do informacji o procesach systemowych oraz innych użytkowników. Nie może jednak wpływać na inne procesy.

Dzięki instrukcji ps można zobaczyć aktualnie uruchomione procesy oraz takie informacje jak:

PID – (Process ID) – Unikalny ID procesu

PPID – (Parent Process ID) – ID procesu nadrzędnego

PRI – (Priority) – priorytet procesu (jeśli proces jest uruchamiany jako ważniejszy)

NI – (Nice value) – ujemny priorytet (jeśli proces jest uruchamiany jako nieobciążający procesor)

STIME – (Start Time) – czas rozpoczęcia działania procesu

TIME – (Time) – czas wykorzystania procesora przez proces

C – (CPU) – obciążenie procesora

W wielu systemach UNIX dostępny jest program o nazwie top, który umożliwia w trybie interaktywnym przeglądanie działających procesów.

```
10:33am  up 6 days, 14:55, 22 users,  load average: 0.22, 0.36, 0.55
166 processes: 164 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:  5.8% user,  1.9% system,  0.0% nice, 92.2% idle
Mem:  128068K av, 124732K used,   3336K free,  83172K shrd,   6564K buff
Swap: 128484K av,  43820K used,  84664K free                 55372K cached

  PID  PPID PRI  NI  SIZE SHARE %CPU %MEM STAT   TIME COMMAND
 1754  1738  14   0  1116   848  2.9  0.8 R      0:00 top
 1157   474   3   0  2676  1876  1.6  1.7 S      0:03 httpd
 1552   474   4   0  2644  1932  1.2  1.8 S      0:00 httpd
 1430   474   2   0  2544  1796  0.9  1.6 S      0:00 httpd
30456   347   1   0   508   292  0.3  0.3 S      0:00 /usr/local/sbin/sshd
 1711  5089   1   0  1192   864  0.3  0.9 S      0:00 proftpd: simenicp - dial-58.b
 1737   347   1   0   996   800  0.3  0.7 S      0:00 /usr/local/sbin/sshd
    1     0   0   0   120    48  0.0  0.0 S      0:02 init [3]
    2     1   0   0     0     0  0.0  0.0 SW     0:06 kflushd
    3     1   0   0     0     0  0.0  0.0 SW     0:23 kupdate
    4     1   0   0     0     0  0.0  0.0 SW     0:59 kswapd
    5     1   0   0     0     0  0.0  0.0 SW     0:00 keventd
  311     1   1   0   212   116  0.0  0.1 S      3:37 syslogd -m 0
  322     1   0   0   488   144  0.0  0.1 S      0:00 klogd
  338     1   0   0   192    96  0.0  0.0 S      0:00 crond
  347     1   0   0   260   124  0.0  0.1 S      0:41 /usr/local/sbin/sshd
  379     1   0   0  1360  1168  0.0  1.0 S      0:00 xntpd -A
  396     1   0   0  1264   244  0.0  0.4 S      0:04 /usr/sbin/dhcpd
  416     1   0   0  4536     0  0.0  0.0 SW     0:00 AvpDaemon
  417     1   0   0  4540     0  0.0  0.0 SW     0:00 AvpDaemon
  446     1   5   0   528   236  0.0  0.2 S      0:11 sendmail: accepting connectio
  474     1   0   0  1068   436  0.0  0.4 S      0:00 httpd
  523     1   0   0   164     0  0.0  0.0 SW     0:00 safe_mysqld
  558   523   0   0  2148   756  0.0  1.2 S      0:01 /usr/sbin/mysqld --basedir=/
  560     1   0   0   660    24  0.0  0.2 S      0:00 stunnel -d trex.wsi.edu.pl:99
  562     1   0   0   612    24  0.0  0.0 S      0:00 stunnel
  576     1   0   0   460   272  0.0  0.2 S      0:34 /usr/local/pgsql/bin/postmast
  595     1   0   0    68     0  0.0  0.0 SW     0:00 mingetty
```

Rys. 15. Działanie programu 'top'

4.3. Operacje na zadaniach

Bardzo często uruchomienie jednego programu powoduje uruchomienie wielu procesów. Jeżeli program został uruchomiony z linii poleceń to dla niego

przyznawany jest numer zadania. W każdym momencie z jednego terminala użytkownik może mieć uruchomionych i pracujących wiele zadań.

Przy pomocy znaku & na końcu linii można uruchamiać całe zadanie od razu w tle.

Na przykład wykonanie instrukcji

```
marcin@sun:/home/marcin>cp -r perl kopia &
[1] 10823
marcin@sun:/home/marcin>
```

spowoduje uruchomienie zadania w tle i od razu system wraca do linii poleceń wyświetlając informacje, że zawieszone zadanie ma numer [1] oraz PID głównego procesu jest 10823.

Po skończeniu zadania w tle na ekran wyświetlana jest informacja:

```
[1]    Done                          cp -r perl kopia
```

Jest to informacja, że zadanie zostało zakończone.

Do przeglądania aktualnych zadań służy instrukcja jobs

```
marcin@sun:/home/marcin>jobs
[1]  + Suspended                     man ps
[2]  - Running                       cp -r perl kopia
```

Każde zadanie posiada swój unikalny numer (w nawiasach kwadratowych) oraz status.

Status mówi czy zadanie pracuje, czy jest zatrzymane, czy jest zatrzymywane, czy się skończyło.

W trakcie działania zadania można zadanie przerwać i zamknąć przy pomocy instrukcji kill wykonując instrukcję:

```
kill %1
```

jeśli podajemy numer zadania lub

```
kill 10823
```

jeśli podajemy PID procesu głównego.

Czasem może zaistnieć sytuacja, że instrukcja kill bez żadnych dodatkowych parametrów nie potrafi zamknąć zadania. Należy wtedy uruchomić kill z parametrem –9.

```
kill –9 10823
```

Jeżeli uruchomiliśmy jakieś zadanie i chcemy go zatrzymać, to należy nacisnąć kombinację klawiszy ^Z (Ctrl+Z). Spowoduje to zawieszenie bieżącego zadania. Możemy następnie zadanie ponownie uruchomić (wznowić) przy pomocy instrukcji

```
fg %1
```

gdzie 1 oznacza numer zadania, jeżeli chcemy aby zadanie wróciło na pierwszy plan. Ostatnie zadanie można wznowić przy pomocy instrukcji fg lub fg %% lub %%.

Zadanie można również wznowić i uruchomić od razu w tle przy pomocy instrukcji

```
bg %1
```

gdzie 1 oznacza numer zadania. Ostatnie zadanie można również wznowić przy pomocy instrukcji bg %%, bg lub %%&

4.4. Wykonywanie zadań w tle

W poprzednim rozdziale pokazany został sposób wykonywania zadań w tle. W wielu systemach UNIX nie ma standardowo możliwości zamknięcia sesji shella, jeśli uruchomione są procesy podrzędne w tle. Sama standardowa praca procesów w tle jest również niewygodna w przypadku aplikacji, które wyświetlają coś na ekran, np. lynx czy ftp, ponieważ tekst ten jest wyświetlany na ekranie, na którym pracujemy i wykonujemy dalsze polecenia.

W celu usunięcia tych niedogodności stworzono specjalną aplikację ‘screen’, która potrafi dla określonego użytkownika uruchomić aplikację w tle przydzielając jej specjalny wirtualny ekran, gdzie aplikacja może spokojnie wyświetlać różne dane bez zakłócania pracy sesji nadrzędnej.

Wykorzystanie tej instrukcji jest bardzo proste. Zadanie, które byśmy normalnie wykonywali, należy poprzedzić instrukcją ‘screen’. Poniżej zamieszczony jest przykład pobierania pliku ze strony www przy pomocy aplikacji lynx.

```
screen lynx http://www.download.com/progs/vdub.zip
```

W tym momencie uruchamiany jest normalnie program lynx, który zaczyna pobierać plik. Można zaczekać aż się skończy, lecz można taką sesję zostawić w tle razem z aktualnym ekranem, na który aplikacja będzie wyświetlała informacje o postępie ściągania pliku. Należy nacisnąć kombinację klawiszy **CTRL+A**, a następnie klawisz **D**. Wrócimy w tym momencie do shella, a zadanie będzie wykonywało się w tle. Można w tym momencie całkiem się wylogować i rozłączyć nawet z internetem. Aplikacja screen zapewni nam działania aplikacji lynx na serwerze i sukcesywne ściąganie pliku.

W dowolnym momencie, jeśli chcemy sprawdzić stan zaawansowania ściągania pliku, lub czy operacja nie została przerwana z powodu jakiegoś błędu, możemy wrócić do ekranu pracującego w tle, wykonując polecenie

```
screen –r
```

W przypadku jeśli uruchomiliśmy w tle więcej aplikacji na różnych ekranach polecenie ‘screen –r’ pokaże pracujące ekrany w tle w postaci numerów procesów. W tym celu należy po poleceniu ‘screen –r’ podać jeszcze numer konkretnego procesu, który chcemy podglądnąć.

```
[marcin@trex marcin]$ screen -r
There are several suitable screens on:
        24134.pts-49.trex       (Detached)
        24145.pts-49.trex       (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
[marcin@trex marcin]$ screen -r 24134
```

Rys. 16. Powrót do ekranu pracującego w tle przy pomocy screen

Bardzo wygodnym narzędziem do pobierania plików z sieci jest program ‘wget’ albo w starszych wersja ‘getwww’. Program ten potrafi pobierać pliki zarówno z serwerów FTP oraz WWW. Potrafi wielokrotnie się łączyć w celu pobrania pliku.

Program wget w przypadku pobierania stron WWW potrafi pobierać również inne strony lub pliki zawarte na pobieranej głównej stronie. W ten sposób można pobrać cały serwis wraz z wszystkimi podstronami. Wykorzystanie aplikacji ‘screen’ wraz z programem ‘wget’ jest bardzo często stosowane.

Poniżej zamieszczony jest wynik działania polecenia ‘wget --help’ które zawiera wszelkie wykrzystywane przez ten program opcje.

```
GNU Wget 1.7, nie-interaktywny ¶ci±gacz sieciowy.
Użycie: wget [OPCJE]... [URL]...

Obowi±zkowe argumenty długich opcji s± też obowi±zkowe dla opcji krótkich.

Start:
  -V,  --version                   wy¶wietl wersję Wget'a i zakończ
                                   działanie.
  -h,  --help                      wy¶wietl t± pomoc.
  -b,  --background                przejdŁ w tło po wystartowaniu.
  -e,  --execute=KOMENDA           wykonaj komendę .wgetrc'.

Logging and input file:
  -o,  --output-file=FILE     log messages to FILE.
  -a,  --append-output=FILE   append messages to FILE.
  -d,  --debug                print debug output.
  -q,  --quiet                quiet (no output).
  -v,  --verbose              be verbose (this is the default).
  -nv, --non-verbose          turn off verboseness, without being quiet.
  -i,  --input-file=FILE      download URLs found in FILE.
  -F,  --force-html           treat input file as HTML.
  -B,  --base=URL             prepends URL to relative links in -F -i file.
       --sslcertfile=FILE     optional client certificate.
       --sslcertkey=KEYFILE   optional keyfile for this certificate.

Download:
       --bind-address=ADDRESS   bind to ADDRESS (hostname or IP) on local
host.
  -t,  --tries=NUMBER           set number of retries to NUMBER (0 unlimits).
  -O   --output-document=FILE   write documents to FILE.
  -nc, --no-clobber             don't clobber existing files or use .#
suffixes.
  -c,  --continue               resume getting a partially-downloaded file.
       --dot-style=STYLE        set retrieval display style.
  -N,  --timestamping           don't re-retrieve files unless newer than
local.
  -S,  --server-response        print server response.
       --spider                 don't download anything.
  -T,  --timeout=SECONDS        set the read timeout to SECONDS.
  -w,  --wait=SECONDS           wait SECONDS between retrievals.
       --waitretry=SECONDS      wait 1...SECONDS between retries of a
retrieval.
  -Y,  --proxy=on/off           turn proxy on or off.
  -Q,  --quota=NUMBER           set retrieval quota to NUMBER.

Katalogi:
  -nd  --no-directories              nie twórz katalogów.
  -x,  --force-directories           wymu¶ tworzenie katalogów.
  -nH, --no-host-directories         nie twórz katalogu o nazwie hosta.
  -P,  --directory-prefix=PREFIKS    zapisz pliki w PREFIKS/...
       --cut-dirs=LICZBA             ignoruj LiCZBA zdalnych katalogów

HTTP options:
       --http-user=USER      set http user to USER.
       --http-passwd=PASS    set http password to PASS.
  -C,  --cache=on/off        (dis)allow server-cached data (normally
allowed).
```

```
  -E,  --html-extension      save all text/html documents with .html
extension.
       --ignore-length       ignore `Content-Length' header field.
       --header=STRING       insert STRING among the headers.
       --proxy-user=USER     set USER as proxy username.
       --proxy-passwd=PASS   set PASS as proxy password.
       --referer=URL         include `Referer: URL' header in HTTP request.
  -s,  --save-headers        save the HTTP headers to file.
  -U,  --user-agent=AGENT    identify as AGENT instead of Wget/VERSION.
       --no-http-keep-alive  disable HTTP keep-alive (persistent
connections).
       --cookies=off         don't use cookies.
       --load-cookies=FILE   load cookies from FILE before session.
       --save-cookies=FILE   save cookies to FILE after session.

FTP options:
  -nr, --dont-remove-listing   don't remove `.listing' files.
  -g,  --glob=on/off           turn file name globbing on or off.
       --passive-ftp           use the "passive" transfer mode.
       --retr-symlinks         when recursing, get linked-to files (not
dirs).

Recursive retrieval:
  -r,  --recursive          recursive web-suck -- use with care!
  -l,  --level=NUMBER       maximum recursion depth (inf or 0 for infinite).
       --delete-after       delete files locally after downloading them.
  -k,  --convert-links      convert non-relative links to relative.
  -K,  --backup-converted   before converting file X, back up as X.orig.
  -m,  --mirror             shortcut option equivalent to -r -N -l inf -nr.
  -p,  --page-requisites    get all images, etc. needed to display HTML page.

Rekursywnie akceptuj/odrzuć
  -A,  --accept=LISTA               lista akceptowanych rozszerzeń.
  -R,  --reject=LISTA               lista odrzucanych rozszerzeń.
  -D,  --domains=LISTA              lista akceptowanych domen.
       --exclude-domains=LISTA      lista domen do odrzucenia
                                    oddzielonych przecinkami.
       --follow-ftp                 podłżaj za odno¶nikami FTP
                                    ze stron HTML.
       --follow-tags=LISTA          lista akceptowanych podłżanych HTML
  -H,  --span-hosts                 podczas pracy rekursywnej idŁ do
                                    obcych hostów
  -L,  --relative                   podłżaj wyłłczenie za relatywnymi
                                    odno¶nikami.
  -I,  --include-directories=LISTA  lista akceptowanych katalogów.
  -X,  --exclude-directories=LISTA  lista odrzucanych katalogów.
  -nh, --no-host-lookup             nie sprawdzaj wpisu DNS hostów.
  -np, --no-parent                  nie wychodŁ poza katalog - matkę.

Informacje o błędaj przesyłaj do <bug-wget@gnu.org>.
```

Rys. 17. Wynik wykonania polecenia wget --help

4.5. Tworzenie harmonogramów wykonywania zadań

W przypadkach gdy chcemy, aby jakieś zadania wykonywały się same automatycznie w określonych momentach dnia, tygodnia czy miesiąca, idealnym narzędziem do tego jest ‘cron’.

Do edycji listy zadań służy polecenie ‘crontab’. Poleceniem ‘crontab –l’ można wyświetlić aktualną listę zadań. Do edycji listy zadań służy polecenie ‘crontab –e’. Polecenie to otwiera, przy pomocy domyślnego edytora, plik z listą zadań. Po wyjściu z edytora lista zadań aktualizowana jest w systemie.

Domyślny edytor definiowany jest przez zmienną środowiskową EDITOR. Poniżej zamieszczono przykład ustawiający domyślny edytor do edycji na ‘vi’. Więcej szczegółów dotyczących zmiennych zamieszczono w rozdziale 6.3 na stronie 46.

```
EDITOR=vi
export EDITOR
```

Po wejściu do edycji można modyfikować listę zadań, która zorganizowana jest w taki sposób, że jedna linia oznacza jedno zadanie i przed zadaniem, oddzielone spacjami lub tabulatorami, występują ograniczenia minut, godzin, dni tygodnia, dni miesiąca oraz miesięcy. Poszczególne ograniczenia mogą występować w następującej formie:

- cyfra — oznacza tylko tą wartość
- cyfra1,cyfra2,... — oznacza wymienione wartości
- cyfra1-cyfra2 — oznacza zakres liczb
- * — oznacza wszelkie możliwe wartości
- /cyfra — oznacza liczby podzielne przez podaną wartość (istnieje tylko w niektórych implementacja ‘cron’

Oczywiście możliwe są różne kombinacje w/w zakresów, np. 1-7,10-22,30.

Poniżej zamieszczono przykładowy spis zadań wraz z komentarzem.

```
#minuty godziny dzien_miesiaca miesiac dzien_tygodnia   zadanie
0 23 * * * zadanie_codzienne
0 6,23 * * * zadanie_2x_dziennie
0 7-19 * * * zadanie_uruchamiane_od_7_do_19_co_godzine
0 6 1 * * zadanie_uruchamine_1_szego
0 6 * * 1 zadanie_uruchamiane_w_kazdy_poniedzialek
0 6 * * 6-7 zadanie_uruchamiane_w_weekend
0,15,30,45 8-12 * * 1-5 zadanie_uruchamiane_co_15_min_miedzy_8_a_12_od_pn_do_pt
```

4.6. Zagadnienia kontrolne

1. Parametry związane z każdym procesem (PID,PPID,PRI,STIME,TIME,CPU)
2. Pojęcie grupy procesów jako zadanie
3. Zawieszanie procesów
4. Praca procesów i zadań w tle
5. Zabijanie zadań
6. Wykonywanie zadań w tle z przechwytywaniem ekranu przy pomocy screen
7. Pobieranie plików ftp i http przy pomocy wget
8. Tworzenie harmonogramów wykonywania poleceń bez ingerencji użytkownika

5. Operacje kartotekowe i plikowe

5.1. Wstęp

W rozdziale tym pokazane są podstawowe operacje na plikach i katalogach. Przedstawione są również podstawowe zagadnienia, które różnią systemy UNIX od systemów DOS/Windows. Pokazane są wszelkie możliwości przetwarzania potokowego i strumieniowego oraz tzw. „skróty", czyli odwołania do plików i katalogów.

Przedstawiono również dość zagmatwany, lub po prostu odmienny od powszechnie znanych, system praw do plików i katalogów.

5.2. Operacje kartotekowe

Podstawowa operacja kartotekowa to wyświetlanie zawartości katalogu. Do tego celu służy instrukcja 'ls'. Bez żadnych parametrów wyświetla tylko nazwy plików i katalogów.

Dodatkowe, najczęściej stosowane parametry opisane są poniżej:

ls –l Wyświetla dodatkowe informacje (prawa, liczbę linków, właściciela, grupę, wielkość pliku)

ls –a Wyświetla pliki ukryte – nazwy plików zaczynające się od znaku '.' (kropka)

ls –d Jeżeli plik jest katalogiem, to wyświetla informacje o katalogu, a nie jego zawartość

ls –R Wyświetla pliki wraz z podkatalogami

Do tworzenia katalogów służy instrukcja 'mkdir'. Można dodatkowo podać wiele parametrów i stworzyć jednym wywołaniem kilka katalogów, np.:

```
mkdir  katalog  katalog/podkatalog  katalog/podkatalog2
```

Do usuwanie katalogów służy instrukcja ‘rmdir’. Instrukcja umożliwia skasowanie tylko pustych katalogów.

Usuwanie plików wykonuje się przy pomocy operacji ‘rm’. Można podawać więcej parametrów - plików. Można korzystać z nazw wieloznacznych opisanych szeroko w Rozdziale 6.4 na stronie 48. Usuwanie katalogów wraz z podkatalogami możliwe jest dzięki parametrowi ‘–r’.

```
rm –r katalog1 plik1 plik2
```

Kopiowanie plików wykonuje się przy pomocy operacji ‘cp’, która musi posiadać minimum 2 parametry - pliki/katalogi. Jeżeli parametrów plików/katalogów jest więcej, to ostatni parametr musi być katalogiem i do niego kopiowane są wszystkie poprzednie pliki/katalogi. Operacja kopiowania może być wykonana z parametrem ‘-r’, co umożliwi kopiowanie całych katalogów z podkatalogami.

Poniższy przykład skopiuje kat1 i plik1 do kat2

```
cp –r kat1 plik1 kat2
```

Do przenoszenia plików i zmiany nazwy istnieje jedna wspólna instrukcja ‘mv’. Jeżeli drugi parametr – plik/katalog znajduje się w katalogu bieżącym i są tylko dwa parametry – pliki/katalogi, to wykonywana jest operacja zmiany nazwy. W przypadku podania dwóch lub większej liczby parametrów – plików/katalogów ostatni parametr musi być istniejącym katalogiem i jest wykonywana operacja przeniesienia.

Bardzo przydatną operacją do przeglądania katalogu jest instrukcja ‘file’, która pokazuje również na podstawie analizy plików dodatkowe informacje o zawartości plików.

Poniższy przykład pokazuje działanie instrukcji ‘file’

```
sun%~>file *
bin:            directory
configi:        directory
dziwne:         directory
find:           directory
forward:        ascii text
hack:           directory
info.sh:        executable shell script
it_conf.cfg:    empty file
login:          c program text
login.mbr:      executable /bin/tcsh script
logout:         ascii text
```

```
man:            directory
nt:             directory
pass2html:      c program text
plik.txt:       ascii text
ranking:        directory
smb:            directory
system:         ascii text
test.sh:        executable shell script
testnet.sh:     executable shell script
top:            ELF 32-bit MSB executable SPARC Version 1, dynamically
linked,stripped
trust.txt:      ascii text
uptime.log:     ascii text
vmstat.log:     ascii text
```

Rys. 18. Wynik działania polecenia 'file'

5.3. Skróty do plików

W systemach unix'owych występują odwołania do innych plików, które nazywane są często linkami. Istnieją dwa typu skrótów – twarde i miękkie (hardlink i softlink).

Miękki link to zwykły odsyłacz do innego pliku w katalogu. Widoczne jest to przy wyświetlaniu katalogu, co jest widoczne poniżej:

```
sun%~>ls -l plik*
-rw-r--r--   1 marcin   wizja       240 Jun  9  2000 plik.txt
lrwxrwxrwx   1 marcin   wizja         8 Oct 26 11:50 plik2.txt -> plik.txt
lrwxrwxrwx   1 marcin   wizja        11 Oct 26 11:50 plik3.txt -> /etc/passwd
```

Linki miękkie tworzy się przy pomocy instrukcji 'ln' z parametrem '-s'.

```
ln -s  plik_istniejacy  nowy_link
```

Linki twarde to całkiem inny typ skrótów do plików. Link twardy w katalogu nie różni się niczym od początkowego właściciela. Wskazuje bezpośrednio na blok danych na dysku. Linków twardych w katalogu może być więcej i widoczne jest to przy wyświetlaniu katalogu.

```
sun%~>ls -l
total 258
drwxrwxrwx   2 marcin   wizja       512 Jun 18  1998 bin
drwxrwxrwx   2 marcin   wizja       512 Jun  3  1997 configi
drwxrwxr--   2 marcin   wizja       512 Sep  3  1999 dziwne
drwxrwxr--   4 marcin   wizja       512 Jul 13  1999 find
-rw-rw-r--   1 marcin   wizja        27 Sep  1  1999 forward
drwxr-xr-x   3 marcin   wizja       512 Jun 28  2000 hack
-rwxr-xr-x   1 marcin   wizja       169 May 31  1999 info.sh
-rw-rw-rw-   1 root     other         0 Dec 27  1999 it_conf.cfg
```

Dla przykładu do katalogu „find” istnieją 4 twarde linki. Pierwszy jest w aktualnym katalogu. Jeden zawsze jest w samym katalogu (pozycja ‘.’) oraz katalog „find” posiada 2 podkatalogi, w których są zapisy do katalogu nadrzędnego (pozycje ‘..’) czyli tego „find”.

Linki twarde tworzy się poleceniem ‘ln’ bez żadnych dodatkowych parametrów:

```
ln  istniejący_plik  nowy_link_twardy
```

5.4. Operacje na strumieniach

Każdy proces w UNIX’ie wykorzystywać może wiele plików. Standardowo otwieranych jest kilka, zwanych standardowymi strumieniami wejścia-wyjścia.

stdin – standardowy strumień wejściowy (domyślnie klawiatura) – służy często do pobierania danych,

stdout – standardowy strumień wyjściowy (domyślnie ekran) – służy najczęściej do wyświetlania danych, wyników,

stderr – standardowy strumień wyjściowy błędów (domyślnie ekran) – służy najczęściej do wyświetlania informacji o błędach programu,

Istnieje możliwość przekierowywania domyślnych urządzeń obsługujących te strumienie przez system.

Do przekierowania stdout służy znak ‘>’. Po nim powinien podany być plik, do którego mają być zapisane dane, które program wyświetla na stdout. Plikiem tym może być urządzenie – np. /dev/null – strumień pusty, /dev/tty1 – konsola, /dev/cua1 – port com.

Poniższy przykład, zamiast wyświetlać na ekran posortowany plik dane.txt, zapisuje go w pliku dane2.txt

```
sort dane.txt > dane2.txt
```

Poniższy przykład zapisuje efekty działania programu ps do pliku procesy.txt

```
ps –ef > procesy.txt
```

Znak ‘>>’ pełni bardzo podobną funkcję do znaku ‘>’ z taką różnicą, że jeżeli plik istnieje to dane są dopisywane na końcu pliku.

Znak ‘2>’ przekierowywuje (w shellach sh,bash) strumień stderr do podanego pliku.

Znak ‘<’ przekierowywuje strumień stdin, tak że dane nie są wprowadzane z klawiatury, tylko z podanego pliku.

Poniższy przykład pokazuje, że program sort nie czyta znaków z klawiatury tylko z pliku dane.txt

```
Sort < dane.txt
```

W przypadku czytania danych z klawiatury znak końca pliku wprowadzany jest przez CTRL+D. W przypadku gdy nie mamy możliwości wprowadzenia znaku CTRL+D – np. w skryptach shellowych, można to zrobić przy pomocy znaku ‘<<’. Podać należy wtedy frazę, która będzie przetworzona na znak końca pliku.

Przykład zamieszczony poniżej posortuje nazwiska wprowadzone z klawiatury

```
[marcin@trex wyniki]# sort << 'KONIEC'
> Ola
> Ala
> Jola
> Monika
> Joanna
> Ewa
> KONIEC
Ala
Ewa
Joanna
Jola
Monika
Ola
```

Bardzo często przekierowywanie strumieni wykorzystuje się jako jednoczesne przekierowanie stdout jednego programu na stdin drugiego, uruchamianego później. Do tego celu wykorzystywany jest znak ‘|’.

Programy wykorzystywane w takich celach nazywamy często filtrami.

Przykład zamieszczony poniżej pokazuje że instrukcja cat nie wyświetli na ekran pliku dane.txt tylko przekaże go jako stdin do programu sort, który dopiero dane przetworzone wyświetli na stdout – ekran

```
cat dane.txt | sort
```

Przykład poniżej pokazuje, że program ps nie wyświetla efektów swojego działania na ekran tylko przekazuje do programu grep jako stdin i dopiero ten program ten wysyła dane na stdout czyli ekran.

```
ps –ef | grep nfs
```

5.5. Wykonywanie warunkowe

W przypadku gdy w jednej linii poleceń chcemy wykonać kilka instrukcji, jedną po drugiej, to oddzielając te instrukcje znakiem ‘;’ spowodujemy, że wszystkie instrukcje wykonywane będą sekwencyjnie. Wykonanie poprzedniej nie będzie miało żadnego wpływu na działanie następnej.

Poniższy przykład najpierw skopiuje plik potem zmieni nazwę katalogu.

```
cp –r kat1 kopia; mv kat1 kat2
```

W przypadku gdy chcemy wykonanie następnej instrukcji uzależnić od wykonania poprzedniej, używać można znaków ‘&&’ lub ‘||’. W przypadku gdy chcemy, aby następna instrukcja wykonała się tylko jeśli poprzednia wykonała się dobrze to używamy znaków ‘&&’.

Poniższy przykład zmieni nazwę katalogu tylko jeśli operację kopiowania się powiodła.

```
cp –r kat1 kopia && mv kat1 kat2
```

W przypadku odwrotnym, gdy chcemy wykonanie kolejnej instrukcji uzależnić od błędnego wykonania poprzedniej instrukcji należy użyć znaków ‘||’.

Poniższy przykład wyświetli informacje, że katalog się nie stworzył nie wyświetlając standardowego komunikatu instrukcji mkdir

```
mkdir katalog 2> /dev/null  ||  echo "Katalog nie zostal stworzony"
```

Poniższy przykład pokazuje jak można wykorzystać warunkowe wykonywanie instrukcji do obsługi zarówno dobrego jak i złego wykonania instrukcji.

```
mkdir katalog 2> /dev/null && echo "Katalog zalozony" || echo "Blad"
```

Bardzo cenna jest możliwość grupowania instrukcji przy pomocy znaków '(' oraz ')'. Szereg instrukcji wykonywany jest wtedy w oddzielnej powłoce shell, co nie ma wpływu na zewnętrzne wywołanie.

Poniższy przykład pokazuje, że zmiana katalogu w ramach wykonania w '( )' nie ma wpływu na powłokę zewnętrzną.

```
pwd; (cd ..; ls); pwd
```

Poniższy przykład pokazuje, jak można grupować instrukcję.

```
(mv kat kat_old && cp –r kat_old kat) && (echo "Kopia OK"; ls)
```

5.6. Prawa związane z plikami i katalogami

System praw w UNIX jest trochę inny niż w większości znanych systemów operacyjnych. Każdy plik i katalog ma ustawioną dla siebie maskę praw. Maska ta jest wyświetlana przy pomocy polecenia 'ls' np. 'ls -l'. Składa się ona z 10 znaków, z czego pierwszy oznacza typ pozycji w katalogu (d – katalog (directory), '-' – normalny plik, l – odsyłacz (link)). Następne 9 znaków podzielono na 3 grupy. Pierwsza grupa dotyczy praw właściciela pliku, druga dotyczy praw grupy pliku, ostatnia zaś praw wszystkich pozostałych użytkowników. Na standardowe prawa składają się możliwe literki: 'r', 'w', 'x'. Literka 'r' oznacza prawo odczytu dla plików, a dla katalogów możliwość wyświetlenia zawartości katalogu. Literka 'w' oznacza prawo zapisu dla plików i możliwość modyfikowania zawartości dla katalogu. Literka 'x' oznacza prawo wykonywania w przypadku pliku oraz prawo wejścia do katalogu w przypadku katalogów.

Do zmiany praw służy instrukcja 'chmod'. Posiada ona dwa tryby podawania żądanych praw. Pierwszy to podanie maski wszystkich praw. Dla każdej grupy praw wyznaczana jest liczba dziesiętna, która binarnie odpowiada ustawionym bitom określonych praw – 'r' to 2^2 czyli 4, 'w' to 2^1 czyli 2, 'x' to 2^0 czyli 1, np. wartość 6 oznacza prawa 'rw', wartość 5 oznacza prawa 'rx'.

W chmod prawa można podawać również jako frazę: [kto][operator][prawa], gdzie:

[kto] może zawierać wartości: ‘u’ – właściciel (user), ‘g’ – grupa (group), ‘o’ – pozostali (others), ‘a’ – wszyscy (all),
[operator] może zawierać wartości: ‘+’ – przy dodawaniu praw, ‘-‘ – przy odejmowaniu praw, ‘=’ – przy nadawaniu określonych praw
[prawa] może zawierać literki oznaczające prawa, czyli ‘r’, ‘w’, ‘x’.
Najczęściej używane są zapisy takie jak: ‘go+x’ dla dodania grupie i pozostałym prawa ‘x’,
‘o-w’ dla zabrania prawa ‘w’ pozostałym, ‘g=rw’ dla nadania grupie praw ‘rw’.
Poniższe dwa przykłady ustawiają prawa: rwxr-x—x

```
chmod 751 plik
chmod g-w,o-r plik   # przy zalozeniu ze wczesniej były rwxrwxr-x
```

Bardzo istotnym elementem tutaj jest możliwość zmiany właściciela i grupy dla pliku czy katalogu. Operacji tej nie może wykonać dowolna osoba. Składnia jest bardzo prosta

```
chown  nowy_wlasciciel   plik
chown  nowy_wlasciciel:nowa_grupa  plik
chgrp  nowa_grupa  plik
```

5.7. Zagadnienia kontrolne

1. Operacje na katalogach (przeglądanie, tworzenie, usuwanie)
2. Operacje na plikach w katalogu (kopiowanie, przenoszenie, zmiany nazw)
3. Identyfikacja zawartości – instrukcja file
4. Skróty do plików – linki twarde i miękkie
5. Operacje na strumieniach stdin, stdout, stderr
6. Wykorzystywanie programów jako filtry
7. Wykonywanie warunkowe instrukcji – operatory && i ||
8. Wykonywanie instrukcji w podrzędnej powłoce – operatory ( )
9. Prawa związane z plikami i katalogami

6. Podstawowe operacje powłoki UNIX

6.1. Wstęp

Rozdział ten przedstawia podstawowe informacje dotyczące działania powłoki systemu UNIX tzw. shell. Shell wykorzystywany jest cały czas podczas pracy w systemie UNIX i pełne wykorzystanie tego jest bardzo ważne i potrzebne. Zostały zaprezentowane możliwości podstawowych powłok wystepujących w systemach UNIX takie jak: zmienne środowiskowe i powłoki, uzupełnianie nazw plików, dostęp do historii. Szeroko pokazane zostały wszelkie mechanizmy podstawiania oraz cytowania.

6.2. Podstawowe powłoki używane w Unix'ie

Unix, po uruchomieniu usługi telnet, po przeprowadzeniu identyfikacji (nazwa konta oraz poprawne hasło) uruchamia proces login, który następnie na podstawie danych zapisanych w pliku /etc/passwd uruchamia pierwszy proces dla tego użytkownika. Dzięki temu można np. umożliwić użytkownikowi dostęp z tego konta tylko np. do programu poczty lub bezpośrednio do protokołu ppp. Zmienić domyślny program uruchamiany po zalogowaniu może tylko administrator serwera.

Najczęściej konto podłączone jest do tzw. shell'a czyli programu powłoki, dzięki któremu można uruchamiać i wykonywać wszelkie polecenia systemu unix. Umożliwia to zdalną pracę na serwerze, co w praktyce daje bardzo duże możliwości, np. zdalnego administrowania serwerem. Na unix'ie powstało bardzo wiele różnych shell'i. Znajomość ich wszystkich nie jest potrzebna. Do najbardziej popularnych należą dwa z nich: sh oraz csh i na nich się w tym skrypcie skupimy. Istnieją również bardziej rozbudowane wersje shell'a umożliwiające np. lepszą obsługę historii poleceń (na strzałkach) czy łatwą zmianę tzw. prompt'a linii

poleceń. Dla shell'a sh jest to bash, zaś dla csh jest to tcsh. Do pisania skryptów opartych o shell wystarczają całkowicie shell'e podstawowe (sh,csh). Różnice pomiędzy tymi dwoma shell'ami są dość istotne i przyjęło się że częściej do obsługi interaktywnej pracy (wykonywanie poleceń unixa) wykorzystuje się csh, zaś do pisania skryptów raczej sh. Oczywiście nie ma żadnych przeciwwskazań w innym wykorzystywaniu tych shell'i.

6.3. Zmienne środowiskowe i zmienne powłoki

Każdy shell obsługuje tzw. zmienne. Istnieją różne typy zmiennych w shell'ach. Każdy proces (również shell) posiada określoną pamięć do przechowywania tzw. zmiennych środowiskowych. Zmienne te zawierają różne informacje systemowe wykorzystywane przez programy. Zmienne środowiskowe (environment) są kopiowane do wszystkich procesów potomnych i każdy proces może modyfikować tylko swoje zmienne środowiskowe a nie ma dostępu do zmiennych procesu nadrzędnego.

Dostęp do wartości zmiennej jest w shell'ach sh i csh identyczny - wystarczy przed nazwą zmiennej wstawić znak $. Np. żeby wyświetlić zawartość zmiennej PATH wystarczy napisać instrukcję:

```
echo $PATH
```

Istnieją również zmienne specyficzne dla danego shell'a - są to tzw. zmienne shell'owe. Istnieją one tylko w tym danym procesie i nie są dziedziczone do procesów potomnych. Dostęp do tych zmiennych jest identyczny jak do zmiennych środowiskowych tzn. przez użycie znaku $.

Sposoby definiowania zmiennych są specyficzne dla określonych shell'i i tak dla shell'a sh listę wszystkich zmiennych wyświetla się przy pomocy polecenia:

```
set
```

Nowe zmienne deklaruje się w sposób następujący:

```
ZMIENNA=wartosc
```

W przypadku gdy wartość zmiennej zawiera kilka słów najlepiej zacytować całą jej zawartość:

```
ZMIENNA="wartosc - kilka slow"
```

W shellu sh zmienne te są standardowo zmiennymi shell'owymi (lokalnymi) - istniejącymi tylko w tym procesie. Żeby stworzyć zmienną środowiskową należy taką zmienną wyeksportować przy pomocy polecenia export.

```
export ZMIENNA
```

W tym momencie zmienna ta staje się zmienną środowiskową i widoczna będzie w procesach potomnych.

W shell'u csh istnieją dwie oddzielne tablice zmiennych - zmiennych shell'owych i zmiennych środowiskowych. Zmienne shell'owe wyświetla się przy pomocy polecenia:

```
set
```

Nowe zmienne shell'owe deklaruje się w następujący sposób:

```
set zmienna=wartosc
```

Zaś zmienne środowiskowe wyświetla oraz deklaruje się przy pomocy oddzielnego polecenia:

```
setenv
```

Nowe zmienne środowiskowe (dziedziczone do procesów potomnych) deklaruje się w następujący sposób:

```
setenv ZMIENNA wartosc
```

Jeśli np. chcemy dopisać coś na zmiennej PATH, która jest zmienną środowiskową odpowiedzialną za przeszukiwane ścieżki przy wydawaniu poleceń, to należy to zrobić w następujący sposób:

W shell'u sh

```
PATH="$PATH:/usr/sbin"
```

W shell'u csh

```
setenv PATH "$PATH:/usr/sbin"
```

Zmienne mogą być wyświetlane w tekstach cytowanych przy pomocy cudzysłowu " lecz już nie przy pomocy cudzysłowu ' (szczegóły patrz rozdział 6.8 na stronie 52)

```
echo "Witam cie $user w pracy"
```

6.4. Uzupełnianie nazw plików

Powszechną zaletą wielu systemów operacyjnych jest wieloznaczny dostęp do plików, czyli użycie pewnego wzorca (maski) oznaczającej wiele plików.

Powszechnym znakiem używanym w maskach plików jest znak *, który oznacza wiele dowolnych znaków. Poniżej podano kilka przykładów takich typowych masek.

a* oznacza wszystkie pliki zaczynające się na literkę a (nie A)

*.txt oznacza wszystkie pliki które kończą się tekstem ".txt"

_ oznacza wszystkie pliki które zawierają w swojej nazwie znak _

Następnym znakiem wieloznacznym jest znak ?, który oznacza dokładnie jeden dowolny znak.

??? oznacza pliki o dowolnej nazwie, lecz długości 3 znaków.

zbior??.txt oznacza pliki składające się z tekstu "zbior", dwóch dowolnych znaków i teksty ".txt".

Następne nazwy wieloznaczne można używać przy pomocy nawiasów []. Każdy nawias klamrowy w nazwie oznacza dokładnie jeden znak z możliwości podanych w nawiasie. W nawiasie mogą być wyszczególnione znaki lub podany zakres znaków.

[abc]* oznacza pliki, które zaczynają się na literkę a lub b lub c

zbior[139].txt oznacza pliki o nazwach: zbior1.txt lub zbior3.txt lub zbior9.txt

zbior1[1-3].txt oznacza pliki o nazwach: zbior11.txt lub zbior12.txt lub zbior13.txt

zbior[12][0-9].txt oznacza pliki o nazwach: zbior10.txt ... zbior19.txt, zbior20.txt ... zbiort29.txt

zbior[1-35].txt oznacza pliki o nazwach: zbior1.txt lub zbior2.txt lub zbior3.txt lub zbior5.txt

a nie pliki: zbior1.txt ... zbior35.txt co by się mogło wydawać.

Można używać również znaku ^, który oznacza negację.

^*a* oznacza wszystkie pliki, które nie zawierają w nazwie literki a

[^A-Z]* oznacza wszystkie pliki, które nie zaczynają się na dużą literkę

Dodatkowo w csh wprowadzono nawiasy klamrowe, które można używać do zapisu plików, które w określonym miejscu mogą zawierać określone różne ciągi znaków

{zbior,plik}.txt oznacza pliki o nazwach: zbior.txt lub plik.txt

*.{dat,idx} oznacza pliki, które kończą się tekstem ".dat" lub ".idx"

Bardzo istotne jest tu zrozumienie mechanizmu działania podstawiania nazw plików. Otóż podstawianie to jest wykonywane przez sam shell, jeszcze zanim jakiekolwiek polecenie zacznie działać np. wpisanie polecenia:

```
cp *.txt kopia
```

nie spowoduje przekazania do poleceniach cp dwóch parametrów tylko wielu, albo może tylko jednego - co skończy się błędem. Otóż shell najpierw przejrzy katalog, następnie zmieni w linii poleceń tekst "*.txt" na odpowiednie nazwy plików np. "ola.txt ala.txt plik.txt" i dopiero wykona instrukcje cp już jako:

```
cp ola.txt ala.txt plik.txt kopia
```

Ważne w tym momencie jest to, że cp potrafi taką składnię obsłużyć. Jedyne ograniczenia to takie, że ostatni parametr musi być katalogiem.

Ograniczeniem tu jest jednak brak możliwości wykonania np. instrukcji cp *.txt *.asc umożliwiającej skopiowanie wielu plików do plików o innych nazwach co było możliwe np. pod DOS'em a w unix'ie nie ma racji bytu.

Ważne też jest to że trzeba pamietać o ostatnim parametrze, bo np. składnia

```
cp *.txt
```

w systemie DOS/Windows mówi żeby system skopiował wszystkie pliki kończące się na ".txt" do katalogu bieżącego nawet jeśli nie jest on jawnie podany, zaś to

samo pod unixem może oznaczać, w zależności ile mamy plików w katalogu, albo błąd (1 plik lub więcej niż 2) albo, w przypadku gdy w katalogu są dokładnie dwa pliki kończące się na ".txt", jeden zostanie skopiowany do drugiego i stracimy jego zawartość.

6.5. Wykorzystywanie poleceń z historii

Shell csh, tcsh, bash zapamiętuje wszystkie wprowadzane polecenia i umożliwia dostęp i wykorzystywanie ich. Jest to bardzo cenny mechanizm przy wprowadzaniu wielu bardzo rozbudowanych poleceń. Historie poleceń wyświetlić można przy pomocy polecenia history. Wielkość bufora pamiętającego listę poleceń można definiować przy pomocy zmiennej history. Dostęp do poprzednich poleceń jest realizowany przez wyrażenie, w którym występuje znak "!" i tak np.

!3 oznacza polecenie o indeksie 3

!-3 oznacza polecenie o indeksie o 3 mniejszym niż bieżące

!pico oznacza ostatnie polecenie, które zaczynało się od słowa pico

!! oznacza całe poprzednie polecenie

!* oznacza wszystkie parametry z poprzedniego polecenia

!$ oznacza ostatni parametr z ostatniego polecenia

!^ oznacza pierwszy parametr z ostatniego polecenia.

Wyrażenia te bardzo przydają się w praktyce np.:

```
marcin@mthome:/home/marcin>co test4.sh test
co: Command not found.
marcin@mthome:/home/marcin>cp !*
cp test4.sh test
```

lub np.

```
marcin@mthome:/home/marcin>cp test4.sh test/ttt.sh
marcin@mthome:/home/marcin>chmod 755 !$
chmod 755 test/ttt.sh
marcin@mthome:/home/marcin>ls -l !$
ls -l test/ttt.sh
-rwxr-xr-x   1 marcin   test      117 Nov 17 14:33 test/ttt.sh
marcin@mthome:/home/marcin>vi !$
```

Bardzo często stosowana jest również składnia: ‘^s1^s2^’, która oznacza zastąpienie w ostatnim poleceniu tekstu s1 na tekst s2.

6.6. Podstawianie nazw poleceń

Mechanizm ten jest bardzo specyficznym typem podstawiania. Jest automatycznie wykonywany jeśli shell w liście tzw. alias'ów znajdzie wpis identyczny z aktualnie wykonywanym poleceniem.

Listę aktualnych aliasów można zobaczyć przy pomocy polecenia alias.

```
marcin@mthome:/home/marcin>alias
h       history 20
ll      ls -l
search  find . -name "!*" -print
```

Najprostszy alias h oznacza, że jeśli zostanie wykonane polecenie h to system podmieni linie poleceń na history 20.

Drugi alias ll oznacza, że jeśli zostanie wykonane polecenie ll to system wykona ls -l, oraz jeśli zostanie wpisane ll *.txt to system wykona ls -l *.txt

Trzeci alias jest bardziej skomplikowany i działa tylko w shellu csh i odwołuje się przez wyrażenie z ! do historii. Jako ostatnie polecenie z historii pamiętane jest tu pierwotne polecenie bez zmian i wykonanie polecenia 'search dane.txt' spowoduje wykonanie polecenia 'find . -name "dane.txt" -print'. Podstawianie to jest wykonywane jako pierwsze, więc np. zapis 'search *.txt' spowoduje wykonanie 'find . -name "*.txt" -print' i znak * będzie traktowany jak zwykły znak a nie jak nazwa wieloznaczna i mechanizm podstawiania nazw plików już nie zadziała co by można było przypuszczać patrząc na pierwotną linię: search *.txt

6.7. Wykonywanie instrukcji w linii poleceń

Oba shell'e umożliwiają wykonywanie instrukcji w linii poleceń - czyli jeszcze przed wykonaniem całej linii poleceń. Wykorzystanie tego bardzo cennego mechanizmu jest bardzo proste i wystarczy wstawić dane polecenie w apostrofy odwrócone ` (klawisz nad tabulatorem).

Mechanizm ten najprościej przedstawić na prostym przykładzie, który wyświetla ile jest aktualnie zalogowanych użytkowników:

```
echo "Aktualnie jest zalogowanych `who|wc -l` uzytkownikow"
```

Zanim zostanie wykonane polecenie echo wykonywane jest najpierw polecenie `who|wc -l` i jego wynik wstawiany jest do linii poleceń a następnie dopiero jest uruchamiane polecenie echo.

Należy pamiętać, że wstawienie apostrofów odwróconych ` w apostrofy normalne ' spowoduje najzwyklejsze wyświetlenie ich na ekranie jak normalne znaki.

6.8. Używanie cudzysłowów, apostrofów i cytowanie

W unixe można używać kilku metod cytowania. Ważne jest aby umieć dobrze rozróżniać sposoby działania poszczególnych.

Do normalnego cytowania zwykłych tekstów używa się cudzysłowów ". Cudzysłów ten ma następujące znaczenie:

- wyłącza mechanizm podstawiania nazw plików
- nie wyłącza mechanizmu podstawienia zmiennych
- nie wyłącza mechanizmu podstawienia poleceń z historii
- nie wyłącza mechanizmu wykonywania poleceń w linii poleceń

Innym mechanizmem cytowania jest apostrof '. Apostrof ma następujące znaczenie:

- wyłącza mechanizm podstawiania nazw plików
- wyłącza mechanizm podstawienia zmiennych
- nie wyłącza mechanizmu podstawienia poleceń z historii
- wyłącza mechanizm wykonywania poleceń w linii poleceń

Zagadnienie to można przeanalizować od strony konkretnych mechanizmów podstawiania.

Gdy chcemy, aby mechanizm podstawiania nazw plików działał, nie możemy używać żadnych cudzysłowów, i na odwrót, jeśli chcielibyśmy jawnie podać znak np. * to należy go zacytować.

Następujący przykład umożliwia stworzenie katalogu o nazwie ze znakiem *

```
marcin@mthome:/home/marcin>mkdir "jacek*"
marcin@mthome:/home/marcin>ls -l
```

```
total 280
-rwxr-xr-x   1 marcin   sys          169 May 31 15:58 info.sh
drwxrwxr--   2 marcin   sys          512 Nov 17 15:19 jacek*
drwxrwxr--   2 marcin   sys          512 Nov 13  1997 kredyt
```

Mechanizm podstawiania zmiennych (wyrażenia z $) nie działa tylko w apostrofach, czyli jeśli chcemy wyświetlić zawartość zmiennej to należy użyć cudzysłowów, zaś gdy chcemy wyświetlić znak $ to trzeba użyć apostrofów.

```
marcin@mthome:/home/marcin>echo $user
marcin
marcin@mthome:/home/marcin>echo "$user"
marcin
marcin@mthome:/home/marcin>echo '$user'
$user
```

W sytuacji gdy chcemy wyświetlić znak $ i użyjemy cudzysłowów to system zignoruje to albo nawet pokaże błąd.

```
marcin@mthome:/home/marcin>echo 'Towar kosztowal $20'
Towar kosztowal $20
marcin@mthome:/home/marcin>echo "Towar kosztowal $20"
Towar kosztowal
```

Mechanizm wykonywania poleceń w linii poleceń nie działa w apostrofach identycznie jak w przypadku wyrażeń z $.

```
marcin@mthome:/home/marcin>echo "W katalogu jest `ls|wc -w` plikow"
W katalogu jest       32 plikow
marcin@mthome:/home/marcin>echo 'W katalogu jest `ls|wc -w` plikow'
W katalogu jest `ls|wc -w` plikow
```

Z wyrażeniami z wykrzyknikiem jest problem ponieważ nie cytuje ich zarówno cudzysłów jak i apostrof. W sytuacji jeśli chcemy wyświetlić znak ! i chcemy mechanizm podstawiania z historii wyłączyć należy przed cytowanym znakiem wstawić znak "\".

```
marcin@mthome:/home/marcin>cd
marcin@mthome:/home/marcin>echo "czesc !!"
echo "czesc cd"
czesc cd
marcin@mthome:/home/marcin>echo "czesc \!\!"
czesc !!
```

Cytowanie jednego znaku przy pomocy \ jest bardzo przydatne i można go również stosować np. do wyrażeń z $

```
marcin@mthome:/home/marcin>echo $user musi zaplacic \$20
marcin musi zaplacic $20
```

6.9. Formatowanie wyświetlania

Do lepszego formatowania wyników używa się instrukcji printf, która działa jak identyczna instrukcja w języku C. Składnia printf bardzo często pojawia się w unixie ponieważ unix przejął wiele dobrych mechanizmów z języka C. Printf występuje jako samodzielne polecenie, sposób wyświetlania wyników w find, instrukcja w awk oraz w perl'u.

Pierwszy parametr to tekst formatujący, w którym zdefiniowane są wszystkie wartości w postaci wyrażeń z % i tak:
%s oznacza tekst (%20s oznacza tekst ustawiony na 20 znakach do prawej strony, %-30s oznacza tekst ustawiony na 30 znakach do lewej strony).
%d oznacza liczbę (%10d oznacza liczbę ustawioną na 10 znakach do prawej strony)
%f oznacza liczbę zmienno pozycyjną (%10f oznacza liczbę ustawioną na 10 znakach do prawej strony, %10.2f oznacza liczbę ustawioną na 10 znakach z 2 miejscami po przecinku do prawej strony).
Przydaje się to bardzo często do formatowania liczb zmienno pozycyjnych, np.:

```
marcin@home:/home/marcin>printf "Obciazenie serwera przez ostatnie 15min.:
%.1f\n" `uptime | awk -F, '{print $6}'`
Obciazenie serwera przez ostatnie 15min.: 0.5
```

Należy pamiętać, że printf samodzielnie nie przechodzi do następnej linii, więc trzeba to zrobić jawnie przez użycie znaku nowej linii - \n

6.10. Operacje matematyczne

W większości systemów UNIX występuje program ‘bc’ umożliwiający wykonywanie podstawowych operacji matematycznych. Program działa w taki sposób, że czyta z klawiatury działania matematyczne i wyświetla wyniki.
Przykład zamieszczono poniżej.

```
sun%~>bc
2*(450+340+120)
1820
^D
sun%~>
```

Można również przesyłać tekst jako stdin do polecenia:

```
sun%~>echo "2*(450+340+120)" | bc
1820
sun%~>
```

6.11. Zagadnienia kontrolne

1. Deklaracja i wykorzystanie zmiennych w shell'ach sh i csh
2. Zmienne środowiskowe i zmienne powłoki
3. Mechanizm uzupełniania nazw plików – znaki *,?,[,],{,},^
4. Wykorzystanie poleceń z historii
5. Mechanizm podstawiania nazw poleceń – instrukcja alias
6. Wykonywanie instrukcji w linii poleceń
7. Cytowanie przy pomocy cydzysłowów ", ', ` oraz znaku \
8. Formatowanie wyświetlania instrukcją printf
9. Operacje matematyczne przy pomocy instrukcji bc

7. Operacje na plikach tekstowych

7.1. Wstęp

Rozdział ten przedstawia podstawowe operacje na zawartości plików. Opisane zostały najczęściej wykorzystywane programy do edycji plików, a w szczególności „nietypowy" edytor 'vi' powszechnie wykorzystywany w systemach UNIX.

Druga część rozdziału poświecona jest przeszukiwaniu zawartości plików oraz zawartości tekstowej plików. Pokazano narzędzia takie jak 'sed' czy 'awk', bardzo szeroko opisywane w literaturze, dające możliwości takie jak rozbudowane języki programowania.

7.2. Edytory plików tekstowych

W unixie istnieje wiele programów do edycji plików. Praktycznie na wszystkich dystrybucjach unixa występuje program o nazwie vi. Program ten w porównaniu z programami do jakich jesteśmy dziś przyzwyczajeni, jest dość archaicznym narzędziem i wielu użytkowników po prostu nie może przejść pierwszego etapu umiejętności skorzystania z niego. Dzisiejszy użytkownik przyzwyczajony jest do programów, które obsługuje się intuicyjnie. Przykładem takiego programu jest program pico, który nie zawsze jest w standardowych instalacjach wielu systemów operacyjnych i należy go najczęściej ręcznie doinstalować.

Program pico obsługuje się bardzo intuicyjnie i po uruchomieniu na dole ekranu pojawia się pasek (widoczny poniżej) najczęściej używanych opcji dostępnych przy pomocy klawisza ctrl (control).

```
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

Nie ma sensu tutaj szczegółowo omawiać tych opcji ponieważ zasada obsługi tego programu jest bardzo intuicyjna i jest podobna do innych spotykanych w różnych systemach operacyjnych. Edytowany plik tekstowy najlepiej wpisywać w linii poleceń przy uruchamianiu programu. Program sam wczyta dany plik i domyślnie będzie pamiętał jego nazwę przy zapisie.

Program vi ma oprócz podstawowej zalety, że jest bardzo powszechny, jedną często decydującą zaletę, dzięki której do dziś wiele osób z niego korzysta - otóż jest to program bardzo szybki. Wiele osób stwierdzi, że nawet najwolniejszy edytor tekstowy jest bardzo szybki przy dzisiejszych komputerach, ale nie przy dzisiejszych łączach, z którymi mamy do czynienia w internecie. Często łącze jest tak wolne, że odświeżanie niepotrzebne ekranu powoduje bardzo niekomfortową pracę z plikiem.

Program ten po uruchomieniu jest w tak zwanym trybie poleceń i nawigacji a nie jak większość programów od razu w trybie edycji tekstu. Tryb poleceń i nawigacji odznacza się tym, że możemy przy pomocy specjalnych poleceń określonych konkretnymi literkami, poruszać się po pliku oraz dokonywać na nim pewnych zmian. Wejście do trybu wprowadzania danych odbywa się też przy pomocy specjalnego polecenia - klawisza. Wyjście z trybu wprowadzania danych odbywa się przez naciśnięcie klawisza ESC.

Poniżej zamieszczono większość najczęściej potrzebnych poleceń:

strzałki - nawigacja po dokumencie (również klawisze: h(←), j(↓), k(↑), l(→)
i - przejście do trybu wprowadzania przed aktualnym znakiem
I - przejście do trybu wprowadzania na początku linii
a - przejście do trybu wprowadzania za aktualnym znakiem
A - przejście do trybu wprowadzania na końcu linii
o - przejście do trybu wprowadzania za bieżącą linią
O - przejście do trybu wprowadzania przed bieżącą linią
C - zmiana reszty linii od bieżącego miejsca
D - usunięcie reszty linii od bieżącego miejsca
S - zamiana bieżącej linii
s – zamiana bieżącego znaku
J - łączenie linii
cw - przejście do trybu edycji - zmiana aktualnego słowa

x - kasowanie znaku
dw - kasowanie słowa
dd - kasowanie linii
u - cofnięcie poprzedniej operacji
^B, ^F - przewijanie ekranu w górę, w dół o całą stronę
nG - skok do linii - gdzie n to jej numer
n| - skok do kolumny n
^G - pokazuje aktualny numer linii oraz kolumny
H, L - skok do pierwszej, ostatniej linii na ekranie
0, $ - skok do początku, końca linii
ZZ lub :x - wyjście z zapisem
:q! - wyjście bez zapisu
:w - zapis pliku
:w nazwa - zapis w pliku o nazwie nazwa
/text - szukanie słowa text w dokumencie
?text - szukanie słowa text w dokumencie w tył
n - następne szukanie w przód
N - następne szukanie w tył
yy - skopiowanie linii do bufora
p – wstawienie tekstu z bufora za znakiem (linią)
P - wstawienie tekstu z bufora przed znakiem (linią)
v - zaznaczenie bloku (potem y - skopiowanie go do bufora, d - usunięcie)
V - zaznaczenie bloku liniowego
^V - zaznaczanie bloku kolumnowego

Należy również wspomnieć, że przed większością poleceń można wstawić liczbę oznaczającą ilość wykonywanych operacji, np. 3dd usunie 3 kolejne liczby, 5J połączy 5 kolejnych linii, 20x usunie 20 kolejnych znaków.

W systemach Linux występuje wersja programu vi o nazwie vim, która ma zaimplementowanych bardzo wiele mechanizmów dla programistów m.in. podświetlanie składni dla skryptów shell'owych, w perlu, czy stron html. Podświetlanie składni włącza się przez polecenie:

:syntax on

Jeżeli chcemy, aby podświetlanie składni było automatycznie włączane, należy stworzyć plik o nazwię .vimrc w swoim katalogu domowym o następującej składni:

```
if &t_Co > 1
  syntax on
endif
```

Z innych ciekawych rozszerzeń istnieje możliwość obsługi równocześnie wielu okien.

:split plik – otwiera nowy plik w nowym okienku

^WW – przełączanie się pomiędzy oknami.

7.3. Podstawowe operacje na plikach

Do podstawowych operacji na plikach zaliczyć trzeba w pierwszej kolejności możliwość wyświetlenia zawartości pliku. Służy do tego celu polecenie cat, które umożliwia wyświetlenie na stdout (ekran) danych z plików lub z stdin (klawiatura). Jako parametry podaje się nazwy plików. Przy pomocy przekierowania stdout można np. wprowadzić wynik z klawiatury do pliku:

```
cat > plik.txt
...
^D
```

Należy pamiętać, że znakiem końca pliku jest ^D (Ctrl+D).

Można również wprowadzać z klawiatury do momentu wprowadzenia określonego słowa, tak jak pokazano poniżej.

```
cat << KONIEC > plik.txt
...
...
KONIEC
```

Ta składnie często jest wykorzystywana w skryptach do wyświetlenia większej ilości linii na ekran.

Wyświetlać można również wiele plików na raz i wtedy używa się do określenia stdin znaku "-". Poniższy przykład pokazuje jak połączyć kilka plików z klawiaturą. W pliku pismo.txt będzie najpierw plik naglowek.txt, potem to co zostanie wprowadzone z klawiatury a na końcu zawartość pliku podpis.txt.

```
cat naglowek.txt - podpis.txt > pismo.txt
...
^D
```

Bardzo często potrzebne jest posortowanie zawartości pliku. Do tego celu służy polecenie sort, które można użyć z następującymi parametrami:

-r umożliwia sortowanie w odwróconej kolejności

-k x umożliwia sortowanie przez porównywanie słowa nr x oddzielanych standardowo spacjami

-n stosuje się jeśli słowo porównywane jest liczbą (normalnie "10" jest mniejsze od "9", zaś porównując jako liczby to 10 jest większe niż 9)

Na przykład dla pliku o nazwie nazwiska.txt z danymi:

```
Jas Fasola 29
Michal Nowakowski 22
Jacek Olgiert 17
Ignacy Moscicki 38
```

następująca instrukcja posortuje te linie wg wieku od najstarszego i wynik zapisze w pliku spis.txt:

```
cat nazwisko.txt | sort -k 3 -n -r > spis.txt
```

Następna bardzo cenna instrukcja to polecenie uniq, umożliwiające usuwanie linii powtarzających się. Istotne jest aby linie te znajdowały się obok siebie w tym pliku. Uniq w takiej sytuacji zostawi tylko jedną z nich na ekranie.

Uniq umożliwia również przy pomocy parametru -c policzenie ile razy dana linia wystąpiła, np.

```
cat << KONIEC | sort | uniq -c | sort -r -n
Jas
Michal
Tomek
Michal
Krzys
Jas
Michal
KONIEC
   3 Michal
   2 Jas
   1 Krzys
   1 Tomek
```

Bardzo często trzeba policzyć ile w danym strumieniu, pliku występuje linii, słów, znaków. Do tego celu idealnym programem jest wc. Ma on zasadniczo trzy parametry:

-w liczy słowa

-c liczy znaki

-l liczy linie

Narzędzie to jest bardzo cenne do wyciągnięcia różnych informacji z systemu, np. żeby stwierdzić ile jest plików w katalogu można policzyć linie w poleceniu ls:

```
ls -l | wc -l
```

albo żeby stwierdzić ile osób jest zalogowanych, można wykonać:

```
who | wc -l
```

Bardzo często używa się tej instrukcji uruchamiając ją w linii poleceń:

```
echo "Aktualnie jest zalogowanych `who|wc -l` uzytkownikow"
```

Do wyświetlenia początkowego fragmentu pliku bardzo przydatna jest instrukcja head. Standardowo wyświetlanych jest 10 pierwszych linii pliku (lub strumienia), lecz może to być ustalone przy pomocy parametru -n np.:

```
marcin@home:/home/marcin>ps -ef | head -n 5
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0   Oct 21 ?        0:01 sched
    root     1     0  0   Oct 21 ?        0:30 /etc/init -
    root     2     0  0   Oct 21 ?        0:00 pageout
    root     3     0  1   Oct 21 ?       252:09 fsflush
```

Instrukcja tail umożliwia wyświetlenie końcowych części pliku.

Standardowo tail wyświetla 10 ostatnich linii. Przy pomocy następującej składni można wyświetlić np. 20 ostatnich linii:

```
cat plik.txt | tail -20l
```

Instrukcja ‘tail’ umożliwia również wyświetlenie linii od jakiejś określonej do końca np. żeby wyświetlić linię od 20 do końca, należy wpisać:

```
cat plik.txt | tail +20l
```

Dla tych dwóch instrukcji istnieje bardzo wiele różnic na różnych platformach unix - dlatego bardzo często trzeba zajrzeć do dokumentacji (man head, man tail) w celu sprawdzenia poprawności parametrów.

Bardzo często przydaje się instrukcja ‘cut’ do wycinania z każdej linii jakiegoś jej fragmentu. Fragmentem mogą być np. znaki w linii pomiędzy dwoma kolumnami, lub poszczególne słowa oddzielone podanymi znakami.

Żeby wyświetlić poszczególne kolumny można użyć parametru -c wraz z zakresem kolumn

```
finger|cut -c 10-30,60-100
```

Polecenie to może również odwoływać się do słów oddzielonych określonymi znakami, np.

```
marcin@home:/home/marcin>uptime|cut -d "," -f 3
  7 users
```

Parametr -d określa znak oddzielający słowa zaś par. -f specyfikuje, które pola ma wyświetlić.

Bardzo częstym problemem jest zamiana w pliku (lub strumieniu) określonego tekstu na inny. Do tego celu idealna jest instrukcja sed. Możliwości tej instrukcji są bardzo duże. Sed umożliwia operacje na liniach przy pomocy odpowiednich predefiniowanych makr oznaczonych literkami. Poszczególne instrukcje mogą być oddzielane średnikami. Poprzedzane mogą być adresem (zakresem) definiującym, na których liniach ma daną instrukcję wykonywać. Jako adres można również używać wyrażeń regularnych zapisanych pomiędzy znakami '/' (szczegóły patrz rozdział: 7.4 na stronie 63)

Zasada działania jest taka, że najpierw każda instrukcja jest wstawiana do tzw. bufora, na którym wykonywane są instrukcje, po czym to co zostaje w buforze zostaje wyświetlone na ekran.

W celu wyświetlenia tylko linii numer 3, należy usunąć wszystkie linie a linie numer 3 jawnie wyświetlić przed usunięciem:

```
cat plik.txt | sed '3p;d'
```

Literka p oznacza wyświetlenie bufora na ekran, zaś literka d oznacza wyczyszczenie bufora.

Gdybyśmy chcieli wyświetlić wszystkie linie oprócz linii 3 to wystarczyłoby napisać:

```
cat plik.txt | sed '3d'
```

Operacji p już nie trzeba robić, bo jest ona robiona zawsze na końcu automatycznie.

Oto zestawienie najcenniejszych poleceń:

p wyświetla aktualny bufor na ekran
d czyście aktualny bufor
s/text1/text2/ zamienia w buforze tekst text1 na text2

Bardzo przydatne jest wyrażenie s, które można stosować do wszelkich zamian tekstu, np.:

```
finger | sed '1d;s/Nov/Paz/'
```

Z polecenia finger usuwa 1 linie oraz zamienia tekst Nov na Paz.

7.4. Wyrażenia regularne

Wyrażenia regularne wykorzystywane są przez wiele programów i języków programowania. W UNIX'ie występują w takich aplikacjach jak grep,sed,awk. Występują również w takich językach programowania jak perl, tcl czy php.

Wyrażenia regularne to teksty – tzw. nazwy wieloznaczne, które można dopasować zazwyczaj do wielu tekstów. Proste wyrażenia regularne występują w rozszerzeniach nazw plików – znak '*' oznacza dowolne znaki, znak '?' oznacza jeden dowolny znak.

W podstawowych wyrażeniach regularnych występuje bardzo wiele znaków, które mają specjalne znaczenie. Jeżeli chcemy użyć znaku specjalnego jako normalnego znaku bez specjalnego znaczenia należy znak taki zacytować. Wstawia się przed tym specjalnym znakiem, który chcemy zacytować tzw. backslash – '\' – np. tekst '\.' oznacza zwykły znak '.' (kropka).

Lista specjalnych znaków zamieszczona jest poniżej wraz ze szczegółowym opisem.

. jeden dowolny znak

? poprzedni znak może nie występować

[] JEDEN znak z zakresu, [ynYN],[a-zA-Z],[13-68] – oznacza znaki:1,3,4,5,6,8

[^] jeden znak nie występujący w zakresie, [^0-9], [^A-ZŁÓĄŚĆŹŻĘŃ]

* poprzedni znak może występować wiele raz lub wcale

+ poprzedni znak może występować jeden raz lub więcej

{n} poprzedni znak musi występować dokładnie n razy

| operator lub

() grupowanie – np. "(abc){2}" oznacza "abcabc"

Zastosowanie wyrażeń regularnych jest bardzo proste. Służą do sprawdzania czy podany tekst np. w analizowanej linii pasuje do określonego podanego wyrażenia regularnego. W podstawowym zastosowaniu zwracana jest tylko wartość logiczna oznaczająca, że gdzieś w przeszukiwanym tekście dopasowano wyrażenie regularne.

Jeżeli chcemy, aby wyrażenie miałoby być dopasowane do początku lub końca tekstu, to należy użyć poniższych znaków.

^ początek tekstu

$ koniec tekstu

7.5. Przeszukiwanie zawartości plików

Polecenie grep służy do szukania w pliku bądź strumieniu linii, które zawierają określony tekst. Tekst ten jest podstawowym parametrem tego programu. Poniższy przykład zostawi, z tego co wyświetla finger, tylko te linie, które zawierają tekst pc43

```
finger | grep pc43
```

Jeśli chcemy aby grep wyświetlił tylko te linie, które nie zawierają określonego tekstu to należy użyć parametru -v, np.:

```
finger | grep -v pc43
```

Drugim poleceniem analogicznym jest polecenie egrep, które działa identycznie z jednym wyjątkiem, że jako parametr można podać wyrażenie regularne opisane w rozdziale 7.4 na stronie 63.

7.6. Rozbudowane operacje tekstowe – awk

Do bardziej zaawansowanej obróbki plików tekstowych stworzony został program awk. Program ten porównać można z normalnym językiem programowania. Posiada instrukcje warunkowe, pętle, własne funkcje, obsługę zmiennych, obsługę tablic.

W skrypcie awk występuje jeden główny blok wykonywany dla każdej linii. Można stworzyć dodatkowe bloki BEGIN i END, które wykonywane są na początku i na końcu tylko raz. Przed każdym blokiem można podać zakres w postaci wyrażenia: /tekst/ co oznacza, że przetwarzane będą tylko te linie, które pasują do podanego wyrażenia regularnego. W ramach bloku może wystąpić wiele instrukcji – oddzielać je należy znakiem ‘;’.

Poniższy przykład pokazuje wykorzystanie tego typu bloków wraz z ograniczeniami zakresu oraz wykorzystanie zmiennych. Wyrażenie /^-/ oznacza linie zaczynające się od znaku – czyli zawierające pliki, $9 w tym przypadku to nazwa pliku, $5 to wielkość pliku.

```
ls –l |awk 'BEGIN{print "Spis plikow z katalogu"} /^-/{print $9; suma+=$5} \
END {print "Suma plikow :”,suma;}'
```

Każda linia dzielona jest na słowa standardowo oddzielane białymi znakami (spacje, tabulatory). Dostęp do poszczególnych słów możliwy jest dzięki zmiennym $1,$2,$3 itd. $0 oznacza całą przetwarzaną linię. Znak oddzielający słowa można zmienić wykonując awk z parametrem –F, jak podano poniżej.

```
awk –F: '{print $1}' /etc/passwd
```

Instrukcje mogą również być instrukcjami warunkowymi. Składnia jest następująca:

```
if (warunek) { instrukcje; ... } { instrukcje jeśli błąd; ... }
```

Poniższy przykład wykorzystuje instrukcje date i jeżeli jest po 16:30 to wyświetla odpowiedni komunikat.

```
date "+%H%M" | awk '{if ($1>1630) {print "Co ty jeszcze w pracy robisz ?"}}'
```

7.7. Zagadnienia kontrolne

1. Edytor tekstowy pico
2. Edytor tekstowy vi
3. Sortowanie zawartości plików
4. Operacje na identycznych liniach w plikach – instrukcja uniq
5. Liczenie linii, znaków, wyrazów w plikach

6. Zaawansowane operacje na plikach tekstowych przy pomocy programu sed
7. Przeszukiwanie zawartości plików z wykorzystaniem wyrażeń regularnych
8. Bloki funkcjonalne w awk – BEGIN, END, dla każdej linii
9. Składnia programowania w awk
10. Wykorzystanie „słów” w awk

8. Archiwizacja plików

8.1. Wstęp

W rozdziale tym opisano operację związane z ogólnie pojętą archiwizacją. Szczegółowo został przedstawiony program find, umożliwiający przeszukiwania struktury katalogów w celu znalezienia plików lub katalogów spełniających bardzo różne warunki. Sprawdzane mogą być nazwy, atrybuty, prawa, właściciele i wiele inny parametrów związanych z plikami.

Pokazane jest podstawowe narzędzie do tworzenia i obsługi archiwów – tar. Narzędzie to stosowane jest powszechnie również do obsługi urządzeń taśmowych do nagrywania kopi bezpieczeństwa. Pokazano również programy do kompresji plików takie jak compress i gzip oraz.

Aplikacje te i narzędzia przydatne są administratorom serwerów tak dobrze jak zwykłym użytkownikom.

8.2. Przeszukiwanie katalogów

Do przeszukiwania katalogu służy instrukcja ‘find’. Składnia tego programu jest prosta:

find [katalogi] [opcje]

gdzie [katalogi] to lista katalogów, które mają być przeszukiwane, [opcje] to lista przełączników informujących, jakie kryteria musi spełniać szukany plik.

Opcje mogą być następujące:

-name "wzorzec"

Nazwa pliku musi spełniać określony wzorzec, np. "*.c", "raport????.*"

-user "użytkownik"

Właściciel pliku musi być taki jak podany

-depth poziom

Określa ile poziomów w głąb struktury katalogów ma wchodzić

-exec "komenda"

Określa komendę jaka ma być uruchomiona po dopasowaniu maili do poprzednich opcji

Podany koniec komendy określony jest jako zacytowany znak ‘;’ czyli ‘\;’. Dopasowaną nazwę pliku można wykorzystywać przez wstawienie znaków ‘{}’.

-type typ

Określa typ pozycji w katalogu. ‘d’ oznacza katalog, ‘f’ oznacza plik, ‘l’ oznacza link,

-perm prawa

Określa prawa jakie musi posiadać szukany plik

-o

Pozwala wprowadzać operację logiczną ‘lub’ na poprzednim i następnym warunku.

Poniższy przykład tworzy pliki .phps dla wszystkich plików .php

```
find . -type f -name "*.php" -exec cp {} {}s \;
```

Poniższy przykład wyświetla wszystkie pliki w podkatalogach /etc, które posiadają prawo zapisu dla wszystkich

```
find /etc –perm –2 –type f
```

Poniższy przykład wyświetla wszystkie podkatalogi katalogu bieżącego

```
find . –type d –depth 1 –exec ls –ld {} \;
```

Poniższy przykład wyświetli wszystkie pliki .php i .cgi

```
find . –name "*.php" –o "*.cgi"
```

8.3. Operacje na archiwach plików

W systemach UNIX można tworzyć archiwa plików. Do tego celu służy program ‘tar’. Program ten nie kompresuje danych, tylko skleja wiele plików w jeden duży. Dlatego jeżeli chcemy stworzyć archiwum tar, musimy mieć minimalnie więcej miejsca niż zajmują same pliki.

Program tar służy generalnie do przerzucania danych z dysku na nośniki taśmowe, gdzie zapis jest sekwencyjny i obsługa jest na poziomie plików. Program tar właśnie tworzy jeden plik na tasiemce (streamer'ze) z archiwum plików.
Do obsługi tasiemki składnia jest następująca:
tar c pliki – zrzuca na tasiemkę podane pliki (do jednego pliku na tasiemce)
tar x – odtwarza z tasiemki streamer'a plik, na którym tasiemka jest ustawiona i tworzy pliki, które znajdowały się w tym archiwum.
Program tar potrafi również tworzyć archiwum na dysku. Wykorzystywany jest wtedy parametr 'f', po którym należy podać nazwę archiwum (zwyczajowo *.tar).
Poniższy przykład tworzy plik arch.tar zawierający wszystkie pliki „*.php" z podkatalogów

```
tar cf arch.tar  `find . -name "*.php"`
```

W wersji GNU tar istnieje parametr 'z', który umożliwia tworzenie archiwów skompresowanych przy pomocy gzip.

8.4. Kompresja plików w unixie

Standardowe programy do kompresji pod UNIX nie działają tak, jak programy pod DOS, Windows. Dwa najpopularniejsze 'compress' i 'gzip' działają w taki sposób, że jeden plik, który podany jest jako parametr, kompresują i tworzą nowy plik skompresowany o nazwię "*.gz" lub "*.Z". W przypadku jeżeli podamy więcej plików do skompresowania, każdy będzie kompresowany osobno i powstanie tyle plików skompresowanych ile zostało podanych do kompresji.

Do typowego tworzenie archiwów skompresowanych wykorzystuje się instrukcje tar z połączeniem gzip lub compress.
I tak poniższy przykład tworzy skompresowane archiwum, które normalnie można by uzyskać sekwencją instrukcji 'tar cf arch.tar *; gzip arch.tar'

```
tar cf - * | gzip -c > arch.tar.gz
```

Znak ‘-‘ oznacza stdout czyli ekran. Parametr ‘-c’ programu gzip pozwala na czytanie znaków nie z pliku lecz z stdin czyli z klawiatury. Standardowo operacja strumieni pozwala tworzyć skompresowane archiwum w locie.

Rozkompresować i rozpakować pliki z tak stworzonego archiwum można zrobić w sposób następujący:

```
gzip –d –c arch.tar.gz | tar xf –
```

Analogiczne parametry i zastosowanie ma compress. Jest programem występującym na większej liczbie systemów UNIX, jest zauważalnie wolniejszy niż gzip.

8.5. Zagadnienia kontrolne

1. Szukanie plików spełniających określone warunki
2. Wykonywanie operacji na znalezionych plikach i katalogach
3. Obsługa napędów taśm magnetycznych tzw. streamer’ów
4. Tworzenie archiwów danych
5. Kompresja plików i archiwów danych

9. Informacje systemowe Unix

9.1. Wstęp

W rozdziale tym przedstawione jest jak w systemach UNIX wyświetlic można informacje na temat systemu plików, pracy systemu, użytkowników oraz ich bieżącej pracy. Przedstawione są również metody komunikacji z innymi aktualnie pracujacymi użytkownikami.

Pokazane jest również, jak sprawdzać i interpretować ograniczenie miejsca w systemie plików nakładane na użytkowników.

9.2. Informacje o systemie

Czasami, zwłaszcza na systemach gdzie korzysta się z shell'i sh lub csh i nie jest widoczna w linii poleceń nazwa serwera, na którym pracujemy. Można to sprawdzić przy pomocy polecenia hostname.

```
#hostname
sun
#
```

Przydatne również jest polecenie uname, które pokazuje nawet platformę sprzętową, na jakiej aktualnie pracuje serwer.

```
sun%/home/marcin >uname -a
SunOS wizja  5.7 Generic_106541-16 sun4u sparc SUNW,Ultra-4
```

O aktualnej pracy serwera można dużo powiedzieć na podstawie programu ‘uptime’. Widoczny jest czas jak długo chodzi komputer od ostatniego restartu, widoczna jest liczba aktualnie pracujących użytkowników oraz obciążenie serwera – aktualne, przez ostatnie 5 min. i 15 min. Obciążenie serwera jest podane w mikrosekundach, których procesor potrzebuje na obsłużenie wszystkich procesów.

```
sun%/home/marcin >uptime
  6:28pm  up 28 day(s), 12:10,  16 users,  load average: 0.09, 0.12, 0.17
```

Następną instrukcją informacyjną jest instrukcja ‘date’, która pokazuje aktualną datę i godzinę.

```
#date
Wed Oct 31 18:33:50 MET 2001
```

9.3. Informacje dotyczące systemu plików

System plików w UNIXie jest inaczej skonstruowany niż w systemach takich jak DOS czy Windows. Dla użytkownika widoczna jest jedna spójna struktura katalogów, a urządzenia dodatkowe (cdrom, stacja dysków, napędy sieciowe) umieszczone są w odpowiednich katalogach. Cały dysk lokalny może być podzielony na oddzielne partycje, które mogą być podmontowane w określonych katalogach. Jest to bardzo istotny aspekt przy dobrym projektowaniu instalacji systemu od początku. Przepełnienie miejsca jak i ograniczanie miejsca może dotyczyć jednej partycji (jednego FileSystem'u). I tak katalogi, takie jak np. /var/spool/mail gdzie przechowywane są skrzynki z pocztą użytkowników lub /home gdzie przechowywane są katalogi domowe użytkowników, mogą mieć niezależne ograniczenie miejsca. Również w przypadku przepełnienia się jakiegoś filesystemu będzie to miało wpływ na działanie tylko określonych programów. Do przeglądania informacji o aktualnym podziale i aktualnie widocznych (zamontowanych) filesystemach dostępne jest polecenie ‘df’.

```
[root@trex httpd]# df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda5               303251    240205     47385  84% /
/dev/sda6              3168155   2789608    214686  93% /home
/dev/sdb1              8566007   6699854   1421958  82% /home/stud
/dev/sda14              202182    124018     67724  65% /tmp
/dev/sda8               792800    671692     80144  89% /usr
/dev/sda10              497667    346803    125162  73% /var
/dev/sda11              497829    232027    240100  49% /var/lib
/dev/sda13              202182     24972    166770  13% /var/log
/dev/sda7              1981000    839747   1038841  45% /var/spool/mail
/dev/sda12              303251       430    287160   0% /var/spool/mqueue
/dev/sda9               497829    148045    324082  31%
/var/lib/pgsql/pg_xlog
```

Rys. 19. Wynik wyświetlania polecenia ‘df’

Bardzo często przydatnym programem do sprawdzania ile zajmują pliki w określonej strukturze podkatalogów, jest program du, którego efekt widać poniżej

```
wizja %/home/5mtom >du -sk .
52589   .
wizja %/home/5mtom >du -sk `find . -type d`
52589   .
12      ./configi
15      ./bin
11      ./nt
7       ./nt/dns
...
```

9.4. Ograniczenia miejsca na dysku

W systemach UNIX istnieje możliwość ograniczania przestrzeni dyskowej użytkownikom przy pomocy pakietu ‘quota’. Ograniczenie miejsca może być realizowane tylko i wyłącznie w ramach jednego systemu plików. Oznacza to, że może zaistnieć sytuacja, że użytkownik w katalogu np. ‘/home’ nie będzie miał ograniczenia miejsca, a w katalogu ‘/home/studenci’ ograniczenie już będzie działało. Aktualny stan wykorzystania ograniczeń dostępny jest dla każdego użytkownika po wykonaniu polecenia ‘quota’. Poniżej zamieszczone są wyniki działania poleceń ‘df” oraz ‘quota’, żeby użytkownik mógł się zorientować w jakich katalogach ma ograniczenie miejsca.

```
[marcin@trex marcin]$ df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda5               303251    240091     47499  83% /
/dev/sda6              3168155   2993761     10533 100% /home
/dev/sdb1              8566007   6485708   1636104  80% /home/stud
/dev/sda14              202182      1058    190684   1% /tmp
/dev/sda8               792800    671765     80071  89% /usr
/dev/sda10              497667    346812    125153  73% /var
/dev/sda11              497829    239803    232324  51% /var/lib
/dev/sda13              202182    118948     72794  62% /var/log
/dev/sda7              1981000    853134   1025454  45% /var/spool/mail
/dev/sda12              303251       488    287102   0% /var/spool/mqueue
 [marcin@trex marcin]$ quota
Disk quotas for user marcin (uid 103):
     Filesystem  blocks   quota   limit   grace   files   quota   limit
grace
      /dev/sda6 1244280 3000000 3000000           14652       0       0
      /dev/sdb1      38 3500000 3500000               2       0       0
     /dev/sda14       1       0       0               1 3000000  300000
      /dev/sda7   11821  311000  310000               1       0       0
```

Rys. 20. Wynik działania polecenia quota pokazujące ograniczenie miejsca

9.5. Informacje o innych użytkownikach

W systemach UNIX podstawowe informacje o użytkownikach przechowywane są w pliku /etc/passwd, do którego każdy ma dostęp (tylko do odczytu). W pliku tym dawniej przechowywano zaszyfrowane hasła – teraz do tego celu służą inne pliki. Z każdym użytkownikiem związane są następujące informacje:

- konto użytkownika - używane do usług pop3, ssh, telnet, ftp
- nazwa użytkownika, np. Imię i Nazwisko
- używany shell - uruchamiany po wejściu ssh, telnet. Najczęściej /bin/tcsh, /bin/bash, /bin/false – w przypadku gdy chcemy aby konto nie miało shella, /bin/passwd w przypadku, gdy chcemy aby po wejściu ssh dało się tylko zmienić hasło.
- katalog domowy – oznaczany również jako '~'. Wykorzystywany jest przez wiele usług i wiele programów do przechowywania plików konfiguracyjnych i danych – np. domyślna strona użytkownika przechowywana jest w katalogu ~/public_html. Jest to standardowo katalog, w którym użytkownik ma prawa zapisu.

Dostęp do tych informacji możliwy jest również przy pomocy programu 'finger', który umożliwia również proste przeszukiwanie po nazwisku lub imieniu użytkownika.

Poniższy przykład wyświetla informacje dotyczące konta marcin

```
sun%~>finger marcin
Login name: marcin                      In real life: Tomana Marcin
Directory: /home/marcin                 Shell: /bin/tcsh
On since Oct 30 08:33:49 on pts/0 from bb_marcin2.bielsko.firma.com.pl
Mail last read Fri Oct 26 09:37:42 2001
No Plan.
```

Poniższy przykład pokazuje, że w przypadku, gdy finger dopasuje wiele kont, wtedy wyświetla informacje o nich wszystkich kolejno na ekran. Z wszystkich linii zostawiamy tylko linie zawierające słowo 'Login'.

```
[marcin@trex marcin]$ finger Anna | grep Login
Login: anna                             Name: Nowak Anna
Login: annaz                            Name: Anna Zeler
Login: annasz                           Name: Szymala Anna
Login: sernik                           Name: Semik Anna
Login: ab0111                           Name: Walus Anna
```

```
Login: ab0121                           Name: Zeler Anna
Login: szklodae                         Name: Kloda Anna
Login: kalinowska                       Name: Kalinowska Anna
Login: sitomane                         Name: Tomanek Anna
Login: szknopee                         Name: Knopek Anna
Login: siwawrze                         Name: Wawrzeczko Anna
Login: szzimmee                         Name: Zimmer Anna
Login: szburye                          Name: Bury Anna
Login: energetyka                       Name: Kloda Anna
```

Każdy użytkownik ma możliwość stworzenia sobie pliku '.plan' w swoim katalogu domowym, w którym zapisze pewne informacje o sobie, o tym czym się zajmuje itp. Plik ten jest wyświetlany, gdy ktoś przy pomocy finger będzie chciał zobaczyć informacje o nas.

Instrukcja finger bez podania parametrów wyświetla listę aktualnie zalogowanych użytkowników, jak to jest widoczne poniżej:

```
[marcin@trex marcin]$ finger
Login     Name               Tty      Idle  Login Time   Office     Office Phone
deexter   Moczek Blazej       pts/0    1:37  Oct 31 16:59 (gw2.bielsko.msk.pl)
dorcart   Misiarz Artur      *pts/60   303d  Oct 22 14:44
krolik    Kroliczek Tomasz    pts/5    1:05  Oct 31 17:42 (pa98.ruda-slaska-bykowina.sdi.tpnet.pl)
marcin    Tomana Marcin       pts/11         Oct 31 18:46 (pa151.bielsko.cvx.ppp.tpnet.pl)
sibanasj  Banas Jaroslaw     *pts/41   303d  Oct 31 17:28 (pf162.czechowice.sdi.tpnet.pl)
sikastea  Kastelik Andrzej    pts/58     9d  Oct 22 13:23 (trex.wsi.edu.pl)
```

Jest kilka bardzo podobnych instrukcji, działających w różnych systemach unixowych. Do najciekawszych należą:

Instrukcja 'w' wyświetla listę zalogowanych użytkowników oraz jakie aktualnie zadania robią.

Instrukcja 'whodo' wyświetla listę procesów zalogowanych użytkowników.

```
# whodo
Tue Oct 20 09:04:46 MET 2001
BB-sun

console      root     13:22
    console        190    0:00 sh

pts/0        marcin    8:33
    pts/0        11849    0:00 tcsh
    pts/0        16895    0:00 mail

pts/1        root      9:04
    pts/1        16954    0:00 sh
    pts/1        17082    0:00 whodo
```

Instrukcja 'users' wyświetla listę zalogowanych użytkowników w postaci listy kont

```
[marcin@trex marcin]$ users
deexter dorcart krolik marcin sibanasj sikastea
```

Instrukcja ‘who’ wyświetla listę zalogowanych użytkowników wraz z informacją na jakich terminalach pracują lub skąd się zalogowali.

```
# who
root       console      Oct 15 13:22
marcin     pts/0        Oct 30 08:33    (bb_marcin2.bielsko.firma.com.pl)
root       pts/1        Oct 30 09:04    (bb_marcin2.bielsko.firma.com.pl)
```

Instrukcja ‘who’ wykorzystywana jest również do wyciągnięcia informacji o aktualnej sesji.

```
# who am i
root       pts/1        Oct 30 09:04    (bb_marcin2.bielsko.firma.com.pl)
```

9.6. Komunikacja z innymi użytkownikami

Najczęściej stosowaną formą komunikacji jest poczta elektroniczna. Jest to forma tzw. „nieagresywna”, co daje możliwość przeczytania wiadomości od nadawcy w wolnej chwili, a nie dokładnie w momencie, w którym dzwoni telefon czy ktoś nas bezpośrednio odwiedza.

Czasami potrzebna jest bezpośrednia forma komunikacji. Do tego celu wykorzystywana może być usługa IRC, lecz istnieją o wiele prostsze i standardowo wbudowane w systemy unix’owe mechanizmy. Najbardziej kulturalny to program ‘talk’. Jeżeli użytkownik chce porozmawiać z innym użytkownikiem, który również jest zalogowany to musi go wywołać. Wywoływanie użytkownika jest bardzo proste – ‘talk uzytkownik’. Po takim wywołaniu na ekranie użytkownika, z którym chcemy porozmawiać pojawi się komunikat w takiej lub podobnej formie:

```
Message from Talk_Daemon@sun at 9:28 ...
talk: connection requested by jacek@sun.
talk: respond with:  talk jacek@sun
```

Jeśli użytkownik docelowy chce z nami porozmawiać to musi również napisać polecenie ‘talk użytkownik’. W tym momencie ekran obu użytkowników dzielony jest na dwie części i można pisać tekst, który będzie wyświetlany na ekranie drugiego użytkownika.

Okienko zamyka się przy pomocy klawiszy Ctrl+C.

```
[Connection closing. Exiting]
Czesc
Potrzebuje od Ciebie nastepujace informacje w sprawie tego wniosku.
1.zaswiadczenie od lekarza
2.twoje oswiadczenie ze nie zalegasz z oplatami.
to wszystko

Najlepiej dzis jeszcze.

To do zobaczenia.

Ŕ-------------------------------------------------------------------------Ů
Czesc
ok.

Kiedy Ci to mam dostarczyc ?

to zalatwione.

Czesc.
```

Rys. 21. Przykładowy ekran rozmowy przy pomocy programu 'talk'

Istnieje jeszcze metoda przekazania informacji bezpośrednio na ekran użytkownika przy pomocy instrukcji 'write'. Po poleceniu podajemy konto użytkownika oraz z klawiatury wprowadzamy treść, która ma być wyświetlana na ekranie.

Operacja ta wykorzystywana jest często przez administratorów, jeśli chcą przekazać jakieś informacje użytkownikom, takie jak np. że serwer będzie przestartowywany, czy bardzo obciążony.

Przed tego typu komunikatami, które przerywają prace, zabezpieczyć się można przy pomocy polecenia 'mesg', podając porametr 'y' kiedy chcemy, lub 'n' kiedy nie chcemy ich otrzymywać.

9.7. Zagadnienia kontrolne

1. Informacje o systemie
2. Informacje o obciążeniu systemu
3. Interpretacja tabeli zamontowanych systemów plików
4. Określanie rozmiaru plików i katalogów
5. Określanie ograniczeń miejsca w ramach systemów plików

6. Informacje o innych użytkownikach
7. Komunikacja z innymi użytkownikami

10. Programowanie przy użyciu shell'a

10.1. Wstęp

Rozdział ten przedstawia pełne wykorzystanie najczęściej stosowanych powłok w systemach UNIX – csh i sh oraz rodzimych tcsh i bash. Powłoki te posiadają bardzo duże możliwości programowania i wykorzystania ich jako podstawowe narzędzie do tworzenia skryptów automatyzujących podstawowe działania jak i wszelkie operacje systemowe. Każdy administrator systemu UNIX powinien mieć bardzo duże umiejętności programowania głównie powłoki ‘sh’, ponieważ jako podstawowy w systemach UNIX jest praktycznie zawsze wykorzystywany przez wszelkie skrypty systemowe UNIX.

10.2. Struktura skryptu w unixie

Skrypty to specjalne pliki, które zawierają atrybut x, który umożliwia użytkownikowi uruchamiać je jak normalne programy.

```
marcin@mthome:/home/marcin>ls -l
-rw-rw-r--   1 marcin   sys         1481 Oct 22 10:42 akcept.txt
-rwxr-x---   1 marcin   sys          318 Jul 11  1997 pass2html
-rwxr-xr-x   1 marcin   sys          119 Jul 15 11:13 test.sh
-rwxrwxr-x   1 marcin   sys          408 Dec  8  1998 testnet.sh
```

Atrybut x ustawia się przy pomocy polecenia chmod (patrz rozdział 5.6 na stronie 43).

```
marcin@mthome:/home/marcin>chmod 755 test.sh
```

Ważne jest aby w pierwszej linii skryptu pojawiła się linia komentarza zawierająca program, który ma być użyty do interpretacji skryptu. Zwolni to użytkownika z problemu przebywania w określonym shell'u przy wykonywaniu skryptu. I tak skrypt może być programem shell'a sh, csh albo dowolnego innego języka programowania np. perl'a. Przy uruchamianiu, system samodzielnie najpierw

przeczyta pierwszą linie skryptu i wykorzysta odpowiedni program do interpretacji skryptu. Program ten musi być podany razem z całą ścieżką dostępu, np.:

```
#!/bin/sh
# dalsza czesc skryptu
```

Linie zaczynające się od znaku # są normalnie ignorowane i traktowane jako komentarze.

Każda linia jest wykonywana jakby była instrukcją danego języka. Wszystkie shell'e automatycznie mają możliwość wykonywania wszelkich programów zewnętrznych, o których była mowa w poprzednich rozdziałach.

W pisaniu programów przy użyciu shell'a najczęściej korzysta się z sh, rzadziej z csh. Nie używa się do pisania skryptów raczej shell'i bash, tcsh ponieważ rozszerzenia tych shell'i głównie dotyczą udogodnień w linii poleceń i paru kosmetycznych rzeczy, o które rzadko kiedy chodzi w skryptach.

10.3. Instrukcje warunkowe i pętle w shell'u sh, bash

Ogólna składnia instrukcji warunkowej if jest następująca:

```
if warunek
then
      instrukcje...
elif warunek
then
      instrukcje...
else
      instrukcje...
fi
```

Warunkiem może być dowolna instrukcja zwracająca kod wykonania chociażby np. "ls *.txt" jeśli nie znajdzie plików, to oprócz wyświetlenia komunikatu o błędzie, zwróci kod błędu, który można tu sprawdzić.

```
if ls *.txt
then
      echo "Wszystko dobrze"
else
      echo "Nie znalazl plikow"
fi
```

Bardzo cenną instrukcją często tu wykorzystywaną jest instrukcja test, która umożliwia sprawdzenie różnych warunków np.:

```
if test $user="marcin"
```

```
then
      echo "Witaj Marcin"
fi
```

Instrukcje test domyślnie zastąpiono w sh nawiasami kwadratowymi: [ ] stąd można powyższy przykład również zapisać:

```
if [ $user="marcin" ]
then
      echo "Witaj Marcin"
fi
```

W przypadku warunków bardziej skomplikowanych można używać innych operatorów:

s1 = s2 teksty identyczne
s1 **!=** s2 teksty różny
s1 **-eq** s2 liczby równe
s1 **-ne** s2 liczby różne
s1 **-gt** s2 liczba większa niż
s1 **-ge** s2 liczba większa lub równa niż
s1 **-lt** s2 liczba mniejsza niż
s1 **-le** s2 liczba mniejsza lub równa niż

Istnieje również grupa operatorów sprawdzająca pewne warunki na plikach, np. żeby sprawdzić czy dany katalog istnieje można użyć operatora -d:

```
if [ ! -d public_html ]
then
      mkdir public_html && echo "Stworzylem katalog"
fi
```

Oto lista najpopularniejszych operatorów tego typu

-d s1 czy istnieje katalog s1
-s s1 czy istnieje niezerowy plik s1
-f s1 czy istnieje plik s1
-r s1 czy plik s1 da się czytać
-w s1 czy plik s1 da się zapisać
-x s1 czy plik s1 da się wykonywać

W shell'u sh można korzystać z kilku pętli tak jak w większości języków programowania. Najczęściej wykorzystywana jest pętla for o następującej składni

```
for zmienna in zbiór
do
      instrukcje...
done
```

Gdzie zmienna jest podana bez znaku $. Zbiór zaś to oddzielone spacjami słowa - dzięki czemu można łatwo użyć zapisu np. *.txt co oznacza wszystkie pliki kończące się tekstem ".txt" a robione to jest tak, że shell takie jedno słowo "*.txt" zamieni na odpowiedni ciąg słów oddzielonych spacjami odpowiadającymi nazwom plików w katalogu.

Następujący przykład wyświetli początkowe fragmenty wszystkich plików pasujących do wzorca "*.txt"

```
for plik in *.txt
do
      head $plik
done
```

Należy pamiętać, że dostęp do zawartości zmiennej plik uzyskuje się używając znaku $ przed nazwą oraz, że zapis plików musi być bez cudzysłowów bo "*.txt" (w cudzysłowach) nie spowoduje podstawienie przez shell nazw z katalogu i pętla ta wykona się tylko dla pliku o nazwie: "*.txt" gdzie znak * będzie traktowany jak zwykły znak.

Oczywiście listę słów można podać jawnie, np.

```
for p in 01 02 03 04
do
      echo "Dzien $p/11/99"
done
```

Istnieje jeszcze jeden typ pętli - pętle działające tak długo, jak spełniony będzie warunek. Ogólna składnia pętli jest nastepująca:

```
while warunek
do
      instrukcje...
done
```

Pętla ta najlepiej nadaje się do wszelkich skryptów sprawdzających jakieś zdarzenia co piewien czas. Poniższy przykład przedstawia skrypt, który się uaktywni dopiero jak wszyscy wyjdą z systemu i będzie zalogowany tylko 1 użytkownik. Instrukcja sleep odczekująca określony czas bardzo się tu przydaje wpływając na mniejsze obciążenie systemu.

```
while [ `who|wc -l` -gt 1 ]
do
      sleep 5
done
# teraz mozna cos robic ...
```

10.4. Instrukcje warunkowe i pętle w shell'u csh, tcsh

W shell'u csh lub tcsh składnia jest inna. Shell ten jest żadko wykorzystywany do pisania skryptów mimo tego, że ma bardziej przyjazną składnię (bardziej podobną do języka C) oraz ma trochę więcej możliwości.

Składnia instrukcji if jest następująca:

```
if (warunek) then
     instrukcje...
else if (warunek) then
      instrukcje...
else
      instrukcje...
endif
```

Warunki definiuje się identycznie jak w sh z jedyną różnicą, że należy je wstawiać w nawiasach okrągłych: (). Oprócz tego liczby można porównywać operatorami > i < a nie -gt i -le. W sh te operatory były zarezerwowane tylko dla wartości tekstowych i porównywania wg kodów ascii.

W csh pętla for nie istnieje, lecz istnieje jej identyczny odpowiednik - pętla foreach. Zasada działania jest identyczna jak pętla for w sh.

```
foreach name (list)
      instrukcje...
end
```

Wprowadzono również w csh dodatkową pętlę, która umożliwia powtórzenie kilku instrukcji określoną liczbę razy.

```
repeat liczba command
```

Pętla ‘while’ również, tak jak instrukcja warunkowa ‘if’, nie ulega większym zmianom. Warunki deklaruje się jedynie w nawiasach okrągłych ().

```
while (warunek)
      instrukcje...
end
```

10.5. Parametry przekazywane do skryptów

Każdy program można uruchamiać z wieloma parametrami. Również skrypty mają wbudowany mechanizm obsługi parametrów. I tak zapis $1 oznacza

pierwszy parametr, $2 oznacza drugi itd. Jako parametr rozumie się tu słowa oddzielone spacjami nie cytowanymi, czyli wywołanie:

```
skrypt.sh "To jest jeden parametr"
```

przekazuje do skryptu tylko jeden parametr ($1) o wartości „To jest jeden parametr” ponieważ cudzysłów cytuje spacje wewnątrz.

Dodatkowo w skrypcie obsługiwany jest zapis $0, który oznacza linie poleceń, która została użyta do wywołania programu.

Bardzo przydatna do obsługi listy parametrów jest instrukcja shift, która przesuwa parametry z n+1 na n. W ten sposób można obsłużyć zmienną liczbę parametrów, np. poniższy skrypt spowoduje usunięcie po kolei wszystkich podanych parametrów wyświetlając informację dodatkowe.

```
#!/bin/sh
while [ "$1" != "" ]
do
  echo "Usuwam $1 ..."
  rm $1
  shift
done
```

10.6. Czytanie danych z klawiatury

W przypadkach, gdy skrypt musi przeczytać coś z klawiatury, przydatne jest polecenie ‘read’. Wykorzystanie polecenia read jest bardzo proste – należy podać tylko jeden parametr – nazwę zmiennej, do której ma być przypisany tekst wprowadzony przez użytkownika z klawiatury. Przykład poniżej pokazuje najprostsze zastosowanie polecenia ‘read’.

```
$ read zmienna
Marcin Tomana
$ echo $zmienna
Marcin Tomana
```

Poniższy skrypt pokazuje zastosowanie polecenia ‘read’ w interaktywnym dialogu z uzytkownikiem.

```
#!/bin/sh
echo "Czy uruchomic robienie kopi ?"
read odp
if [ "$odp" = "t" ] || [ "$odp" = "T" ]
then
  echo "Kopiuje dane..."
```

```
  cp -r `find -name "*.htm*"` kopie
fi
```

Poniżej zamieszczono wynik działania skryptu.

```
sun%/home/mtom >./read.sh
Czy uruchomic robienie kopi ?
n
sun%/home/mtom >./read.sh
Czy uruchomic robienie kopi ?
t
Kopiuje dane...
```

10.7. Zagadnienia kontrolne

1. Prawa skryptów do wykonywania
2. Wybór interpretera skryptu
3. Instrukcje warunkowe i pętle w shell'u sh i csh
4. Parametry przekazywane do skryptów z linii poleceń
5. Czytanie danych z klawiatury

11. Przykładowe ćwiczenia

11.1. Ćwiczenia

Ćwiczenie 1.

Wyświetlić, przetwarzając efekt działania polecenia ‘ls –l’, spis dziesięciu największych plików z katalogu w kolejności od największego.

Ćwiczenie 2.

Skopiować wszystkie pliki z podkatalogów pasujące do nazwy "*.jpg"

Ćwiczenie 3.

Wyświetlić imiona i nazwiska wszystkich kobiet o imieniu „Ewa” posiadających konta na lokalnym serwerze

Ćwiczenie 4.

Wyświetlić listę nazwisk z pliku /etc/passwd należących do grupy "stud" zdefiniowanej w /etc/group

Ćwiczenie 5.

Stworzyć alias sesje, który pokaże komputery z których ostatnio logował się użytkownik

Ćwiczenie 6.

Napisać instrukcję, która stworzy katalog ‘kopie’ o ile taki nie istnieje

Ćwiczenie 7.

Napisać instrukcję, która wyświetli komunikat „czesc” na ekranie dowolnego uzytkonika. Instrukcja ta w przypadku przesłania komunikatu ma wyświetlić tekst „pozdrowienie dostarczone”. W przypadku nieprzesłania komunikatu instrukcja ma wyświetlić tekst „pozdrowienie nie dostarczone” oraz ma nie wyświetlać standardowych komunikatów o błędach instrukcji ‘write’.

Ćwiczenie 8.

Znaleźć wszystkie pliki o rozszerzeniu „*.php”, które zawierają tekst „mysql_connect”

Ćwiczenie 9.

Wyswietlić komunikat "system przeciążony", gdy obciażenie systemu przez ostatnie 5 min. jest większe niż 1.5 – drugi parametr ‘load average’ wyświetlany przez instrukcję uptime

11.2. Rozwiązania ćwiczeń

Ćwiczenie 1.

```
ls -l | sort -k 5 -n -r|head -n 10
```

Ćwiczenie 2.

```
cp –r `find –name "*.jpg" –type f` kopia
```

Ćwiczenie 3.

```
finger ewa | grep Login | awk '{print $2" "$3}'
```

lub

```
cut –d: -f5 /etc/passwd | grep Ewa
```

Ćwiczenie 4.

```
awk –F: "{if (\$4==`awk -F: '/^stud:/{print \$3}' /etc/group`) {print \$5} }"
/etc/passwd
```

Pamiętać należy o niezapętlaniu cudzysłowów, czyli np. żeby w środku cudzysłowów "..." nie użyć ponownie cudzysłowów ". Znak '\' cytuje następny znak – w tym przypadku znak '$', który normalnie byłby interpretowany przez shell a nie przez awk.

Ćwiczenie 5.

```
alias sesje='last $USER | head | cut -c24-40 | sort | uniq'
```

Zmienna $USER zawiera nazwę aktualnie zalogowanego użytkownika.

Ćwiczenie 6.

```
test –d kopie || mkdir kopie
```

Ćwiczenie 7.

```
echo "Czesc"|write konto 2> /dev/null && echo "Komunikat dostarczono" || echo
"Komunikat nie dostarczono"
```

Ćwiczenie 8.

```
grep "mysql_connect" `find . -name "*.php*"`
```

Ćwiczenie 9.

```
uptime | awk -F, '{if ($5>1.5) {print "System przeciazony"} }'
```

12. Literatura

1. Robin K. Burk, David Horvath, “UNIX - Internet. Księga eksperta”, Helion, 1999
2. Sun Educational Services, “Solaris 7. System Administration I”, Sun Microsystems, 1999.
3. Sun Educational Services, “Solaris 7. System Administration II”, Sun Microsystems, 1999.
4. Unix man documentation: sh, csh, vi, regexp, sed, awk
5. SunGuru Service. http://www.sunguru.com
6. Linda Lamb, Arnold Robbins, “Edytor vi”, Helion, 2001, Przedruk z wydawnictwa O’REILLY.
7. Arnold Robbins, „Edytor vi. Leksykon kieszonkowy”, Helion, 2001, Przedruk z wydawnictwa O’REILLY.
8. Tim Parker, „Linux. Księga eksperta”, Helion, 1999.
9. Linux Documentation Project. http://www.linuxdoc.org
10. Arnold Robbins, „sed i awk. Leksykon kieszonkowy”, Helion, 2001.
11. Marcin Tomana, „Użytkowanie systemu UNIX”, Skrypt Wyższej Szkoły Informatyki i Zarządzania, 2001.

Wszelkie uwagi dotyczące niniejszego podręcznika
bardzo proszę przesyłać pocztą elektroniczną na adres:
marcin@tomana.net