Instrukcja do zajęć laboratoryjnych

Język ANSI C (w systemie LINUX)

wersja: 2.0

Nr ćwiczenia:	dodatkowe 1	
Temat:	**Dystrybucja oprogramowania**	
Cel ćwiczenia:	Celem ćwiczenia jest przedstawienie najważniejszych programów do archiwizacji, kompresji oraz przygotowywania i korzystania z tzw. łat (programy: `tar`, `gzip`, `gunzip`, `compress`, `uncompress`, `diff` oraz `patch`)	
Wymagane przygotowanie teoretyczne:	Informacje podane na wykładzie. Podstawy pracy z systemem LINUX.	
Sposób zaliczenia:	Sprawozdanie w formie pisemnej.	[**X**]
	Pozytywna ocena ćwiczenia przez prowadzącego pod koniec zajęć.	[ ]

1. Konwencje przyjęte w instrukcji

`Czcionka o stałej szerokości`

Nazwy programów, poleceń, katalogów, wyniki działania wydawanych poleceń.

`Czcionka o stałej szerokości pogrubiona`

Podaje tekst, który należy dosłownie przepisać. W przypadku plików źródłowych wyróżnia istotniejsze fragmenty.

`Czcionka o stałej szerokości kursywą`

Tekst komentarza w przykładowych sesjach przy terminalu.

`Czcionka o stałej szerokości kursywą pogrubiona`

Wyróżnia istotniejsze fragmenty wyników działania wydawanych poleceń.

2. Uwagi wstępne

Chcąc w wygodny sposób dystrybuować stworzone przez nas oprogramowanie musimy znać podstawowe programy, które ten proces wspomagają. Musimy mianowicie wiedzieć jak

archiwizować pliki, jak je kompresować i dekompresować oraz wreszcie jak tworzyć tzw. łaty i jak je nakładać na posiadane pliki. Niniejsze ćwiczenie przybliża te zagadnienia.

3. Kompresja i dekompresja plików – polecenia gzip, gunzip, compress, uncompress

Tradycyjnym kompresorem w systemach uniksowych jest polecenie `compress` (i `uncompress` przy dekompresji). Zwyczajowo tak skompresowane pliki otrzymują końcówkę `.Z`. W LINUX-ie polecenie `compress` jest wypierane przez dużo wydajniejsze polecenie `gzip` (**G**NU **zip**). `gzip` potrafi dodatkowo rozpakowywać archiwa utworzone przez `compress`. Istnieje również polecenie `gunzip` (do dekompresji plików skompresowanych poleceniem `gzip`), choć na dobrą sprawę wykonuje ono dokładnie to samo co polecenie `gzip` wydane z opcją `-d`.

`gzip` działa w taki sposób, że kompresuje tylko pojedyncze pliki. Uruchomienie rekurencyjnej kompresji katalogu (opcja `-r`) spowoduje niezależne skompresowanie wszystkich plików w drzewie katalogu. Jeżeli teraz chcielibyśmy połączyć je w jeden plik archiwum, musimy użyć polecenia `tar`.

Należy również pamiętać o tym, że polecenie `gzip` nie tworzy na dysku nowych plików (skompresowanych) ale **zmienia (funkcjonalnie to samo co polecenie mv)** po prostu nie skompresowane pliki na ich skompresowane odpowiedniki. Odwrotnie dzieje się w przypadku dekompresji. Trzeba być tego świadomym, gdyż przykładowo skompresowanie wszystkich plików w katalogu `/bin` czy `/sbin` może uniemożliwić pracę systemu!

Uwaga: od pewnego czasu dostępny jest nowy program do kompresji o nazwie `bzip2`. Używa on bardziej wydajnego algorytmu i w konsekwencji pliki są lepiej kompresowane w krótszym czasie. Opcje linii poleceń są bardzo podobne do programu `gzip` (choć nie identyczne!). Szczegóły znajdziesz w dokumentacji systemowej `man`.

Na poniższym przykładzie zobaczymy jak działają omawiane tu polecenia:

```
Rozglądamy się po dysku.
$ ls
ddpro.c  global.c  host.c  host.h  hoster.c  imalloc.c  lmsg.c  lpvm.c
lpvmcat.c

Kopresujemy wszystkie pliki w bieżącym katalogu.
$ gzip *

Stwierdzamy, że oryginalne pliki zostały ZASTĄPIONE (!) wersjami
spakowanymi. Wersji nie skompresowanych NIE MA już na dysku !
$ ls
ddpro.c.gz   host.c.gz  hoster.c.gz   lmsg.c.gz  lpvmcat.c.gz
global.c.gz  host.h.gz  imalloc.c.gz  lpvm.c.gz


Zastosowanie opcji -l wyświetla raport dotyczący rozmiaru i współczynnika
kompresji już skompresowanych plików. Zastosowanie dodatkowo opcji -v daje
raport rozszerzony o datę i czas modyfikacji oryginalnego pliku, metodę
kompresji oraz sumę kontrolną CRC.
$ gzip -l *
         compressed        uncompressed  ratio uncompressed_name
              20939               72788  71.3% ddpro.c
               5070               17814  71.7% global.c
               7397               23181  68.2% host.c
               2026                4989  59.9% host.h
               7700               23379  67.2% hoster.c
```

```
                4090                12865  68.4% imalloc.c
                1573                 3641  57.5% lmsg.c
               27593                92832  70.3% lpvm.c
                4063                11665  65.4% lpvmcat.c
               80451               263154  69.4% (totals)
$

$ gzip -lv *
method  crc     date  time           compressed        uncompressed  ratio uncompressed_name
defla 77346186 Sep 27 01:35              20939               72788  71.3% ddpro.c
defla 618506f1 Feb  8 00:14               5070               17814  71.7% global.c
defla 3fdac573 Sep 27 01:35               7397               23181  68.2% host.c
defla 296f2bf6 Sep 27 01:35               2026                4989  59.9% host.h
defla d7234385 Sep 27 23:25               7700               23379  67.2% hoster.c
defla ada51b27 Mar 24 21:16               4090               12865  68.4% imalloc.c
defla 76e42620 Jun 26 00:08               1573                3641  57.5% lmsg.c
defla 6065f230 Sep 25 23:20              27593               92832  70.3% lpvm.c
defla b687c979 Feb  8 00:14               4063               11665  65.4% lpvmcat.c
                                         80451              263154  69.4% (totals)
$
```

Stwierdzamy, że uzyskaliśmy około 3-krotny stopień kompresji.
263154/80451=3.27

Z powrotem rozpakowujemy pliki.

```
$ gzip -d *
```

Sprawdzamy, czy się udało. Okazuje się, że tak.

```
$ ls
ddpro.c  global.c  host.c  host.h  hoster.c  imalloc.c  lmsg.c  lpvm.c
lpvmcat.c
$
```

Mamy również możliwość (w niewielkim w sumie zakresie) wpływać na stopień kompresji. Liczba 1 oznacza najszybszą, ale też i najgorszą kompresję, a liczba 9 odwrotnie. Domyślnie przyjmowana jest liczba 6. Różnice jak poniżej widać są jednak stosunkowo niewielkie i zwykle domyślna wartość 6 jest wystarczająca.

```
$ ls -l ddpro.c
-rw-r--r--    1 artur    students    72788 Sep 27  2001 ddpro.c

$ gzip -1 ddpro.c
$ ls -l ddpro.c.gz
-rw-r--r--    1 artur    students    25195 Sep 27  2001 ddpro.c.gz

$ gzip -9 ddpro.c
$ ls -l ddpro.c.gz
-rw-r--r--    1 artur    students    20877 Sep 27  2001 ddpro.c.gz

$ gzip ddpro.c
$ ls -l ddpro.c.gz
-rw-r--r--    1 artur    students    20939 Sep 27  2001 ddpro.c.gz
```

4. Archiwizujemy pliki – polecenie tar

Z pomocą polecenia `tar` (skrót od **t**ape **ar**chiver) tworzymy oraz rozpakowujemy pliki archiwów. Archiwum należy rozumieć jako plik, w którym „zapakowano” jeden lub więcej plików źródłowych znajdujących się w jednym lub więcej katalogach – ich ilość jest w zasadzie nieograniczona. Paczką taką jest zdecydowanie łatwiej zarządzać, niż na przykład kilkoma tysiącami plików, które tą paczkę tworzą.

Polecenie `tar` ma ogromną liczbę opcji wiersza poleceń (wydaj polecenie `man tar` a sam się przekonasz, ile ich jest). Nam tak naprawdę potrzebnych będzie tylko kilka podstawowych poleceń. Na poniższym przykładzie zobaczymy jak działa polecenie `tar`:

```
Rozglądamy się po katalogu bieżącym. Mamy w nim pięć plików (w tym jeden
ukryty), dwa podkatalogi, a w każdym z nich po jednym pliku.
$ ls -laR
-rw-r--r--    1 artur    students     1234 Nov  7 12:17 .bashrc
-rw-r-----    1 artur    students      761 Oct 11 07:47 calc.c
-rw-r-----    1 artur    students       82 Oct 11 07:47 hello.c
drwxr-xr-x    2 artur    students     4096 Nov 12 16:48 kat1
drwxr-xr-x    2 artur    students     4096 Nov 12 16:48 kat2
-rw-r-----    1 artur    students      164 Oct 28 13:24 pl.c
-rw-r--r--    1 artur    students      158 Oct 28 13:26 pleng.c

$ ls  -laR ./kat1
drwxr-xr-x    2 artur    students     4096 Nov 12 16:48 .
drwxr-xr-x    4 artur    students     4096 Nov 12 16:49 ..
-rw-r-----    1 artur    students     2368 Nov  6 12:28 Makefile

$ ls -laR ./kat2
-rw-r-----    1 artur    students      303 Oct 11 07:47 Makefile.hello

-- TWORZENIE ARCHIWUM --
Tworzymy archiwum (opcja -c od create), korzystamy z opcji -f wskazując,
że tworzymy archiwum w postaci pliku dyskowego oraz wyświetlamy obiekty,
które są ładowane do archiwum (opcja -v). Wykorzystujemy znak kropki jako
alias bieżącego katalogu. Zauważmy, że tar domyślnie tworzy archiwum
rekurencyjnie (z podkatalogami) oraz dołącza ukryte pliki oraz ukryte
katalogi (jeżeli takowe są).
$ tar -cvf moje_arch.tar .
./
./hello.c
./.bashrc
./kat1/
./kat1/Makefile
./pl.c
./pleng.c
./calc.c
./kat2/
./kat2/Makefile.hello
tar: ./moje_arch.tar: file is the archive; not dumped

Komunikat „./moje_arch.tar: file is the archive; not dumped” otrzymany po
utworzeniu archiwum mówi nam, że utworzono plik moje_arch.tar, ale bez
zrzucania na taśmę („not dumped”).


Wyświetlamy wielkość archiwum. Widzimy, że jest ono dużo większe niż suma
wielkości obiektów w nim umieszczonych. Różnica w wielkości archiwum oraz
plików źródłowych będzie się zmniejszała, gdy archiwum będzie składało się
z większej ilości plików i katalogów (cóż, plik tar zawiera pewną ilość
informacji „systemowych”, czyli w pewnym sensie nadmiarowych).
Zapamiętajmy: tworzenie archiwum oraz kompresja plików to dwa zupełnie
różne pojęcia!
$ ls -l moje_arch.tar
-rw-r--r--    1 artur    students    20480 Nov 12 17:12 moje_arch.tar

-- WYŚWIETLANIE ZAWARTOŚCI --
Wypisujemy zawartość archiwum. Opcja -v daje nam „szeroki” raport.
$ tar -tvf moje_arch.tar
drwxr-xr-x artur/students    0 2003-11-12 17:11:28 ./
-rw-r----- artur/students   82 2003-10-11 07:47:56 ./hello.c
```

```
-rw-r--r-- artur/students 1234 2003-11-07 12:17:12 ./.bashrc
drwxr-xr-x artur/students    0 2003-11-12 16:48:02 ./kat1/
-rw-r----- artur/students 2368 2003-11-06 12:28:10 ./kat1/Makefile
-rw-r----- artur/students  164 2003-10-28 13:24:35 ./pl.c
-rw-r--r-- artur/students  158 2003-10-28 13:26:25 ./pleng.c
-rw-r----- artur/students  761 2003-10-11 07:47:54 ./calc.c
drwxr-xr-x artur/students    0 2003-11-12 16:48:06 ./kat2/
-rw-r----- artur/students  303 2003-10-11 07:47:57 ./kat2/Makefile.hello
```

-- KASOWANIE PLIKÓW Z ARCHIWUM --
Aby usunąć plik z archiwum musimy użyć długiej opcji --delete i określić nazwę pliku (plików, katalogów), który chcemy usunąć.
Uwaga: operacja usuwania plików nie posiada wygodnej jednoliterowej opcji - MUSIMY użyć opcji --delete (dwa znaki minusa plus nazwa delete).

```
$ tar --delete -f moje_arch.tar ./.bashrc ./kat1 ./kat2
```

Patrzymy, czy udało się usunąć wskazane pliki i katalogi.

```
$ tar -tf moje_arch.tar
./
./hello.c
./pl.c
./pleng.c
./calc.c
```

-- MODYFIKACJA ARCHIWUM --
Do utworzonego archiwum możemy łatwo dodać kolejne pliki oraz katalogi. Używamy tutaj opcji -r lub -u. Opcja -r mówi programowi tar, aby ten dodał określone pliki na końcu pliku tar. Opcja -u natomiast po prostu uaktualnia pliki w archiwum nowymi wersjami. Oczywiście jeżeli plik nie istnieje jeszcze w archiwum, to zostanie do niego dodany.

```
$ tar -rf moje_arch.tar .bashrc
```

Sprawdzamy, czy plik został dodany.

```
$ tar -tf moje_arch.tar
./
./hello.c
./pl.c
./pleng.c
./calc.c
.bashrc
```

-- TWORZENIE ARCHIWUM SKOMPRESOWANEGO --
Tworzymy nowe archiwum. Tym razem używamy opcji -z, która powoduje, że archiwum po utworzeniu jest natychmiast kompresowane (domyślnym programem kompresującym jest gzip). Zauważmy, że tym razem archiwum ma „sensowną" wielkość.
Zwyczajowo skompresowane pliki archiwum mają nazwy zakończone na .gz lub .tgz lub .tar.gz

```
$ tar -czf moje_arch.tar.gz .
$ ls -l m*
-rw-r--r--    1 artur    students    20480 Nov 12 17:27 moje_arch.tar
-rw-r--r--    1 artur    students     2925 Nov 12 17:33 moje_arch.tar.gz
```

Pamiętajmy, że jeżeli chcemy przejrzeć skompresowane archiwum, musimy to zrobić używając opcji -z.

```
$ tar -tzf moje_arch.tar.gz
./
./hello.c
./pl.c
./pleng.c
./calc.c
```

-- MODYFIKOWANIE ARCHIWUM SKOMPRESOWANEGO --

```
Uwaga: nie można usuwać plików ze skompresowanego pliku tar.
Skompresowanego archiwum nie można też aktualizować (opcja –u) oraz
dołączać nowych plików (opcja –r).
$ tar –rf moje_arch.tar.gz host.c
tar: This does not look like a tar archive
tar: Skipping to next header
tar: Error exit delayed from previous errors

$ tar –rzf moje_arch.tar.gz host.c
tar: Cannot update compressed archives
tar: Error is not recoverable: exiting now

$ tar --delete –zf moje_arch.tar.gz calc.c
tar: Cannot update compressed archives
tar: Error is not recoverable: exiting now

-- ROZPAKOWYWANIE ARCHIWUM --
Chcąc rozpakować pliki z archiwum używamy opcji –x. Oczywiście jeżeli
rozpakowujemy skompresowane archiwum musimy użyć dodatkowo opcji –z.
$ tar –xzf moje_arch.tar.gz
$ ls
calc.c  hello.c  moje_arch.tar.gz  pl.c  pleng.c
```

5. Tworzenie i montowanie łat – polecenia diff, patch

5.1. Uwagi wstępne

Program `diff` służy do porównywania plików (głównie tekstowych, ale potrafi też porównywać pliki binarne). Z jego pomocą tworzymy tzw. *plik różnicowy*, który zawiera informacje o tym, w jaki sposób (w których wierszach) różnią się dwa badane pliki.

Przy porównywaniu plików tekstowych najbardziej naturalną metodą badania jest porównywanie całych wierszy w plikach a nie poszczególnych znaków. Gdy dwa badane wiersze nie są identyczne fakt ten w odpowiedni sposób odnotowywany jest w pliku różnicowym. Dysponując plikiem różnicowym oraz jednym z porównywanych plików można odtworzyć drugi plik. Do tego odtwarzania służy drugie z omawianych w tym rozdziale poleceń, a mianowicie polecenie `patch`.

Omawiany tu mechanizm używany jest w LINUX-ie najczęściej do aktualizacji plików źródłowych (typowym przykładem są pliki źródłowe jądra systemu). Gdy wprowadzone w plikach źródłowych zmiany nie są wielkie nie ma sensu udostępniać pełnych kodów źródłowych – wystarczy udostępnić (np. w internecie) tylko plik różnicowy (tzw. łata; ang. *patch*) i plik taki już wystarcza do uaktualnienia posiadanych kodów źródłowych do najbardziej aktualnej wersji (myślimy tu o zmianach ilościowych a nie merytorycznych, bo te drugie oczywiście mogą być bardzo nawet znaczące, mimo zmiany zaledwie kilku linii kodu!). Jeżeli uświadomimy sobie, że najnowsza wersja jądra systemu LINUX (numer 2.4.22; instrukcja powstaje 10.11.2003) zajmuje ok. 30MB a jego uaktualnienie (patch dla wersji 2.4.18) tylko 5MB to oczywistym staną się zalety plików różnicowych.

Aby bliżej poznać omawiany tu mechanizm rozważmy następujący przykład. Niech plik A będzie starą wersją programu a plik B jego wersją najnowszą. Wówczas możliwe jest wygenerowanie pliku różnicowego R i zachodzą następujące zależności, schematycznie przedstawione w poniższej tabeli:

Symbol	Opis
B−A=R	Utworzenie pliku różnicowego (tzw. łaty)
B=A+R	Nową wersję pliku A (identyczna z plikiem B) otrzymamy, gdy na plik A „nałożymy” łatę R
A=B−R	Przywracamy pierwotną zawartość plikowi A („zdejmujemy” łatę)

Program `diff` może generować pliki różnicowe w kilku różnych, *de facto* ustandaryzowanych, formatach: *normalnym*, *GNU unified* oraz *kontekstowym*. W niniejszej instrukcji dokładnie omówiony zostanie format normalny. Pozostałe dwa są w istocie tylko pewną modyfikacją tego pierwszego.

Szczegółowy i bardziej formalny opis formatu normalnego można znaleźć w systemie pomocy `info`, wydając polecenie `info diff` (tekst pomocy można też przesłać do pliku tekstowego poleceniem `info diff --output=nazwa_pliku --subnodes`) oraz w dokumentacji `man`, wydając polecenie `man diff`. Tutaj ograniczymy się tylko do kilku przykładów.

5.2. Przykład 1

Jako pierwszy rozważmy plik A zawierający trzy linie tekstu oraz plik B zawierający te same trzy linie tekstu ale zapisane w odwrotnej kolejności. Wówczas wydając polecenie:

```
diff A B > R
```

otrzymamy plik różnicowy pokazany w trzeciej kolumnie poniższej tabeli:

Plik A	Plik B	Plik różnicowy R
a b c	c b a	1,2d0 < a < b 3a2,3 > b > a

Zapisy w pliku różnicowym należy odczytać następująco: aby plik A stał się identyczny jak plik B należy „nałożyć” na plik A łatę. Po wykonaniu poniższych dwóch czynności otrzymamy plik A identyczny jak plik B (dla przypomnienia „naczelna idea” jest taka: nie posiadamy pliku B – posiadamy za to plik A oraz plik różnicowy R i z ich pomocą chcemy „zbudować” plik B). Wszystkie niezbędne dane znajdziemy w pliku różnicowym R. Znakiem < oznaczony jest wiersz „pierwotny”, a znakiem > wiersz „ostateczny”. Należy więc wykonać następujące czynności:

1. w pliku A wykasować wiersze „pierwotne” 1-2, które znajdują się pod wierszem 0, czyli u nas są to dwie pierwsze linie, bo nie ma przecież linii „zero” (zapis `1,2d0`)
2. aby uzyskać wiersze wynikowe 2-3 należy pod wierszem 3 wstawić dwa wiersze „ostateczne” (czyli „b” oraz „a”; zapis: `3a2,3`).

Uwaga 1: wszystkie numery wierszy zapisywane w pliku różnicowym są oryginalnymi numerami wierszy w plikach **przed** ich modyfikacją.

Uwaga 2: ponieważ plik źródłowy A jest bardzo niewielki, więc rozmiar pliku różnicowego R jest kilkakrotnie większy od pliku źródłowego A. Jednak w praktycznych zastosowaniach rozmiary plików źródłowych są o wiele większe, natomiast z drugiej strony pliki poddawane operacji

porównywania nie różnią się zbytnio od siebie i dlatego wynikowy plik różnicowy (który rejestruje tylko różnice w plikach a nie ich jednakowe elementy wspólne) jest wielokrotnie mniejszy niż rozmiar plików źródłowych. Dopiero wtedy tworzenie pliku różnicowego ma sens.

Powyższe realizowane jest po wydaniu następującego polecenia:

```
patch A R
```

W wyniku wykonania tego polecenia zawartość pliku A stanie się **identyczna** z zawartością plik B.

Zwróćmy również uwagę, że zawartość pliku różnicowego nie jest jednoznaczna. Wynika to z algorytmu, według którego działa program `diff`. Stara się on po prostu utworzyć plik różnicowy o możliwie jak najmniejszym stopniu złożoności, a tym samym wielkości. Dla powyższego przykładu inna wersja pliku różnicowego może wyglądać tak jak poniżej. Czynności do wykonania są inne, ale efekt końcowy (czyli plik A identyczny z plikiem B) będzie taki sam.

Plik różnicowy R
1c1 < a --- > c 3c3 < c --- > a

5.3. Przykład 2

Kolejny przykład jest nieco bardziej złożony:

Plik C	Plik D	Plik różnicowy R
a b c d e	c D x e f g	1,2d0 < a < b 4c2,3 < d --- > D > x 5a5,6 > f > g

Zapisy w pliku różnicowym należy odczytać następująco: aby plik C stał się identyczny jak plik D należy „nałożyć” na plik C łatę, czyli wykonać następujące czynności:

1. w plik C wykasować linie 1-2, które znajdują się pod wierszem 0, czyli u nas są to dwie pierwsze linie, bo nie ma przecież linii „zero” (zapis: `1,2d0`),
2. aby uzyskać wiersze wynikowe 2-3 należy wiersz 4 („pierwotny” – u nas „d”) zastąpić dwoma wierszami „ostatecznymi” („D” oraz „x”; zapis: `4c2,3`),
3. aby uzyskać wiersze wynikowe 5-6 należy pod wierszem 5 wstawić dwa wiersze „ostateczne” (czyli „f” oraz „g”; zapis: `5a5,6`).

Po wykonaniu powyższych trzech czynności otrzymamy w wyniku plik C taki sam jak plik D. Wszystkie niezbędne dane znajdziemy oczywiście w pliku różnicowym R

Powyższe realizowane jest po wydaniu następującego polecenia:

```
patch C R
```

W wyniku wykonania tego polecenia zawartość pliku C stanie się **identyczna** z zawartością plik D.

Poszczególne etapy zamiany można przedstawić w poniższej tabeli:

plik pierwotny C	etap 1	etap 2	etap 3
a b c d e	c d e	c D x e	c D x e f g

5.4. Przykład 3

Chcąc z kolei odzyskać oryginalną zawartość pliku C należy również użyć do tego celu pliku różnicowego. Realizowane jest to po wydaniu następującego polecenia (uwaga na przełącznik `-R`):

```
patch –R C R
```

Tym razem jednak plik różnicowy będzie interpretowany będzie nieco inaczej, a mianowicie:

- aby uzyskać wiersze wynikowe 1-2 należy na początku pliku (pod wierszem „zerowym") wstawić dwa wiersze „pierwotne" (czyli „a" oraz „b"; zapis: `1,2d0`),
- aby uzyskać wiersz wynikowy 4 należy wiersze „ostateczne" 2-3 zamienić na wiersz „pierwotny 4 (czyli „d"; zapis: `4c2,3`),
- aby uzyskać wiersz wynikowy 5 należy wykasować wiersze „ostateczne" 5-6 („f" oraz „g"; zapis: `5a5,6`).

W wyniku wykonania tego polecenia plik C powróci do swojej pierwotnej zawartości. Wszystkie niezbędne dane znajdziemy oczywiście w pliku różnicowym R.

Poszczególne etapy zamiany można przedstawić w poniższej tabeli:

plik wynikowy C	etap 1	etap 2	etap 3
c D x e f g	a b c D x e f g	a b c d e f g	a b c d e

5.5. Fragment oryginalnej dokumentacji

Poniżej zamieszczono oryginalny fragment z dokumentacji polecenia `diff` (polecenie: `info diff`). Lekturę tego tekstu potraktuj jako ćwiczenie językowe i uzupełnienie informacji na temat formatu normalnego polecenia `diff`.

```
Detailed Description of Normal Format
-------------------------------------

   The normal output format consists of one or more hunks of
```

```
differences; each hunk shows one area where the files differ.  Normal
format hunks look like this:

     CHANGE-COMMAND
     < FROM-FILE-LINE
     < FROM-FILE-LINE...
     ---
     > TO-FILE-LINE
     > TO-FILE-LINE...

   There are three types of change commands.  Each consists of a line
number or comma-separated range of lines in the first file, a single
character indicating the kind of change to make, and a line number or
comma-separated range of lines in the second file.  All line numbers are
the original line numbers in each file.  The types of change commands
are:

`LaR'
     Add the lines in range R of the second file after line L of the
     first file.  For example, `8a12,15' means append lines 12-15 of
     file 2 after line 8 of file 1; or, if changing file 2 into file 1,
     delete lines 12-15 of file 2.

`FcT'
     Replace the lines in range F of the first file with lines in range
     T of the second file.  This is like a combined add and delete, but
     more compact.  For example, `5,7c8,10' means change lines 5-7 of
     file 1 to read as lines 8-10 of file 2; or, if changing file 2 into
     file 1, change lines 8-10 of file 2 to read as lines 5-7 of file 1.

`RdL'
     Delete the lines in range R from the first file; line L is where
     they would have appeared in the second file had they not been
     deleted.  For example, `5,7d3' means delete lines 5-7 of file 1;
     or, if changing file 2 into file 1, append lines 5-7 of file 1
     after line 3 of file 2.
```

Literatura

1. Dokumentacja systemowa `man` oraz `info`