Wprowadzenie do systemu UNIX

Grzegorz Mucha*
<grzes@twins.pk.edu.pl>

1995

1 Operacje na zbiorach

1.1 Nazwy zbiorów

W UNIXie istnieje praktyczna dowolność nazywania plików – nie ma istotnych ograniczeń na długość nazwy, jakiś jej podział, czy też rodzaje znaków użytych w nazwie. Istotne jest to, że system rozróżnia duże i małe litery, w związku z czym nazwy **plik**, **Plik** i **PLIK** oznaczają trzy *różne* zbiory. Nazwa zbioru stanowi jedną całość, nie ma tutaj dobrze znanego z DOSa podziału na nazwę i rozszerzenie. Oczywiście można nazwać plik np. **dane.out**, ale oznacza to tylko i wyłącznie, że nazwa zbioru składa się z 8 znaków. Kropka jest traktowana jak każdy inny znak, co w szczególności oznacza, że można nadać plikowi nazwę typu **abc...def**. Jedyna sytuacja, gdzie kropka ma jakieś znaczenie to jest przypadek "**.**" (bieżąca kartoteka) i "**..**" (kartoteka nadrzędna). Zwyczajowo przyjęte jest, że np. programy w C kończą się na "**.c**", w Fortranie na "**.f**" itd., ale wynika to tylko z umowy. W nazwie zbioru mogą się znajdować praktycznie wszystkie znaki, choć nie jest wskazane, żeby początkujący użytkownicy nadawali plikom nazwy z minusami, gwiazdkami, spacjami itp. Trzeba po prostu napisać tak, żeby system mógł to jednoznacznie zinterpretować. Jeżeli zatem będziemy próbowali odwołać się do zbioru o nazwie "**ab#1*?**–", a system zareaguje na to nie po naszej myśli, znaczy to, że źle wprowadziliśmy dane polecenie.

1.2 Nazwy uogólniające

Czasami zachodzi potrzeba odwołania się do kilku plików naraz, chociażby w celu ich skasowania. Do tego celu służy cała gama różnych znaków specjalnych. Podstawowe to dobrze znane z DOSa[1] '*****' i '**?**'. '*****' oznacza dowolną ilość znaków, w tym i zero. Zatem **a*b** jest to odwołanie do pliku, który zaczyna się na '**a**' i kończy na '**b**'.

*Przejrzane i uzupełnione przez A. Matuszaka <max@twins.pk.edu.pl>

[1]Znakomita większość przypadków wskazuje na to, że porównywanie UNIXa z DOSem prowadzi do wielu błędów i nieporozumień. Jeśli tylko to możliwe (choć trudne) zapomnij o DOSie i poznawaj UNIX jako zupełnie nowy system, a w żadnym wypadku nie traktuj go jako pewne rozszerzenie DOSa! Znajdzie się kilka miejsc, gdzie takie porównanie będzie miało miejsce - ale tylko wtedy, gdy pomoże to zwrócić uwagę na coś sprzecznego ze starymi nawykami.

'**?**' oznacza jeden dowolny znak. Można również stosować bardziej wyrafinowane konstrukcje, np. **prog[1-9].c** dotyczyć będzie zbiorów **prog1.c**, **prog2.c**, ... , **prog9.c**, ale już nie **proga.c**. Analogicznie [**abc**]**xyz** oznacza **axyz**, **bxyz** i **cxyz**.

1.3 Polecenia do operacji na zbiorach

1.3.1 Przeglądanie zawartości zbiorów

Jest kilka możliwości przeglądnięcia zawartości zbioru. Pierwsza z nich to polecenie `cat`. Jego składnia jest następująca:

`cat` *nazwa nazwa2 nazwa3 ....*

Taka postać spowoduje wypisanie na ekran zawartości wszystkich plików podanych powyżej. Jeżeli zawartość pliku (plików) będzie większa niż ilość linii terminala, to część tekstu się "przewinie".

Jeżeli mamy do obejrzenia plik o dużym rozmiarze, możemy użyć polecenia `more`. Wywołanie jest bardzo podobne do powyższego:

`more` *nazwa nazwa2 ...*

more pokazuje zawartość pliku po jednym ekranie, po naciśnięciu <Enter> przewija 1 linię, zaś spacja powoduje przewijanie w dół o cały ekran[2].

Można wreszcie do obejrzenia zbioru użyć edytora, wybierając z całej gamy dostępnych, a więc `vi`, `joe`[3] czy `xedit`[4]. Spośród wyżej wymienionych najprostszy dla początkujących jest `joe`, ale najważniejszy jest `vi`, który choć na początku może wzbudzić pewne sensacje, jest jednakże najbardziej standardowym edytorem UNIXowym, i jako taki znajdzie się na każdej stacji UNIXowej, czy to na IBMach, czy na Sunach, czy na Sony'ch, czy wreszcie na Alphie.

1.3.2 Kopiowanie plików

Aby skopiować plik należy użyć polecenia `cp`. Można to zrobić w dwojaki sposób:

`cp` *nazwa_źródłowa nazwa_docelowa*

np. `cp plik1 plik2` spowoduje skopiowanie zbioru `plik1` na zbiór `plik2`.

`cp` *nazwa nazwa2 nazwa3 ... docelowa_kartoteka*

co z kolei umożliwi skopiowanie grupy plików do żądanej kartoteki.

1.3.3 Przesuwanie plików

Do tego celu służy polecenie `mv` w formie:

`mv` *nazwa nazwa2 nazwa3 ... docelowa_kartoteka*

co spowoduje przesunięcie plików o wyszczególnionych nazwach do żądanej kartoteki.

1.3.4 Zmiana nazwy pliku

W tym celu również wykorzystamy polecenie `mv`. Konstrukcja

`mv` *stara_nazwa nowa_nazwa*

zmieni nazwę pliku ze starej na nową.

---

[2] Nieco wygodniejszym narzędziem jest `less`, ale nie jest on standardowy, a więc nie w każdym systemie dostępny.

[3] edytor `joe` jest opisany w dalszym ciągu pracy.

[4] `xedit` pracuje w środowisku XWindows.

1.3.5 Linkowanie plików

Czasami istnieje potrzeba posiadania wiernej kopii jakiegoś (niekoniecznie swojego) pliku, przy czym chcemy, aby wszelkie zmiany w prawdziwym pliku natychmiast implikowały identyczne zmiany w naszej kopii. Oczywiście można co pewien czas sprawdzać czy coś się zmieniło i robić bieżące kopie, ale jest to dość męczące i szybko się nudzi. Poza tym, jeżeli oryginalny plik zajmuje dużo miejsca, każdorazowe jego kopiowanie przez jakiegoś użytkownika powoduje zmniejszanie się wolnego miejsca na dysku. W UNIXie istnieje polecenie `ln` (od link). Za jego pomocą możemy sobie w dowolnym miejscu tworzyć odwołanie do rzeczywistego pliku. Efekt jest taki, że każdy z zainteresowanych użytkowników tworzy sobie w kartotece nazwę pliku, której może normalnie używać np. do przeglądania jego zawartości, zaś praktycznie odwołuje się do tego samego miejsca na dysku. Jeżeli fizycznie odwołuje się do tego samego pliku, to jego zmiana jest od razu widoczna dla wszystkich. Robi się to tak:

`ln` *nazwa_oryginalna nazwa_własna*

Uwaga! Powyższa operacja jest możliwa tylko i wyłącznie wtedy, gdy obie lokacje znajdują się na tym samych urządzeniu fizycznym (np. dysku). Wyobraźmy sobie przykład: oryginał znajduje się na dysku 1, zaś użytkownik tworzy sobie dowiązanie w swojej domowej kartotece na dysku 2. Nagle z jakiś powodów dysk 1 zostaje zdemontowany. Niczego nie świadom użytkownik próbuje zobaczyć zawartość swojego zlinkowanego zbioru, system próbuje się odwołać do zbioru na dysku 1 i... głupieje, bo dysku 1 nie ma.

Na szczęście istnieje na to rada - jest to tzw. linkowanie symboliczne[5]. Różni się od poprzedniego poleceniem:

`ln -s` *nazwa_oryginalna nazwa_własna*

Przy tak zlinkowanym zbiorze system jest gotowy na to, że coś się dzieje z urządzeniem, na którym jest oryginał i w razie czego po prostu napisze, że zbioru, do którego się odwołujemy np. nie ma.

1.4 Operacje na kartotekach

W systemie UNIX określenie "pliki" niekoniecznie oznacza to, co zwykliśmy tym mianem określać, a więc zbiory tekstowe, programy itp. Istnieją bowiem trzy rodzaje plików UNIXowych - pliki zwykłe, kartoteki i pliki specjalne. Znaczenie kartotek jest intuicyjnie zrozumiałe - w nich są przechowywane informacje o strukturze plików na dysku.

1.4.1 Zakładanie nowej kartoteki

Do założenia nowej kartoteki służy komenda `mkdir`, której używa się w następujący sposób:

`mkdir` *nazwa_kartoteki*

1.4.2 Poruszanie się między kartotekami

Do poruszania się miedzy kartotekami służy komenda `cd`. Można jej użyć w dwojaki sposób:

[5] UWAGA: w starszych wersjach UNIXa linkowanie symboliczne nie istnieje.

`cd` *nazwa_kartoteki*

np. `cd /usr/local/bin` spowoduje zmianę bieżącej kartoteki na tą podaną w wywołaniu komendy. Istnieją 3 specjalne nazwy kartotek:

`.` - oznacza bieżącą kartotekę;

`..` - poprzednia kartoteka w hierarchii;

`~` - kartoteka domowa użytkownika[6].

Druga forma wywołania polecenia cd to wywołanie go bez argumentów:

`cd`

co spowoduje przejście do domowej kartoteki (a zatem jest równoznaczne `cd ~`).

Jeżeli chcemy się dowiedzieć, w jakiej aktualnie kartotece jesteśmy, możemy wykonać komendę *pwd*.

1.4.3 Przeglądanie zawartości kartoteki

Do przeglądnięcia zawartości kartoteki służy polecenie `ls`. Standardowa postać tego polecenia jest następująca:

`ls` *[-opcje] [lista plików/kartotek]*

np. `ls -lg /usr/lib/lib*`

Pominięcie listy plików spowoduje wypisanie zawartości bieżącego katalogu. Podanie polecenie bez opcji pokazuje tylko listę plików. Są jednak opcje pozwalające na uzyskanie większej ilości informacji, np. `-l`. Przykładowy wydruk po `ls -l` może wyglądać następująco:

```
total 15
drwxr-xr-x   3 grzes       512 Oct 11 19:31 C
-rw-------   1 grzes     25437 Jun  7 11:13 book51
```

Znaczenie pierwszej kolumny omówimy za chwilę. Pozostałe parametry to liczba dowiązań do pliku lub katalogu, właściciel pliku, jego rozmiar, data ostatniej modyfikacji, wreszcie nazwa. W niektórych UNIXach opcja `-l` podaje także nazwę grupy, do której należy plik, w niektórych aby uzyskać tę informację, należy podać dodatkowo opcję `-g`:

```
$ ls -lg

total 15
drwxr-xr-x   3 grzes    prac     512 Oct 11 19:31 C
```

Istnieje wiele innych opcji, zainteresowanych odsyłam do manuala - polecenie `man ls`.

W znakomitej większości przypadków podstawowe pojęcie o danej komendzie daje przeczytanie manuala, czyli systemowej dokumentacji. Na początku najczęściej używanym poleceniem powinno być: `man`.

[6] znak $\sim$ nie posiada tego specjalnego znaczenia w powłoce sh.

1.5 Atrybuty plików

Każdy plik, niezależnie od tego, czy jest to plik zwykły, kartoteka, czy plik specjalny, posiada swoje atrybuty. Ich wartość można zobaczyć wydając polecenie `ls -l`. Pojawia nam się tajemniczy napis głoszący np:`-rwxr-x---`. Pierwszy znak oznacza typ pliku - '`-`' oznacza plik zwykły, '`d`' - katalog, '`l`' - link symboliczny, '`b`', '`c`', '`f`' - pliki specjalne. Pozostałe dziewięć znaków możemy podzielić na trzy grupy po trzy znaki każda. Pierwsza oznacza uprawnienia dla właściciela pliku, druga dla wszystkich z danej grupy, zaś trzecia prawa dostępu dla wszystkich innych. We wszystkich grupach '`-`' oznacza brak zezwolenia na daną operację, '`r`' prawo do czytania (a więc i kopiowania) zbioru, '`w`' prawo do zapisywania, a '`x`' do wykonywania. Należy tu trochę rozróżnić prawa dla plików zwykłych i kartotek. Dla pliku zwykłego '`w`' oznacza prawo do modyfikacji zbioru, zaś '`x`' prawo do wykonania programu. Dla kartoteki z kolei '`r`' oznacza prawo do obejrzenia zawartości kartoteki, '`w`' - prawo do zmiany czegoś w kartotece, np. dodania lub usunięcia dowolnego pliku z kartoteki, zaś '`x`' jest prawem do "wejścia" do kartoteki. Ale podeprzyjmy się przykładem. Załóżmy, że polecenie `ls -lg` dało nam jako rezultat:

```
-rwxr-x---   3 grzes    prac   12732 Oct 11 19:31 prog
```

Oznacza to, że plik o nazwie "`prog`" jest zbiorem zwykłym, użytkownik "`grzes`" ma prawo go czytać, modyfikować i uruchomić (wykonać), wszyscy użytkownicy będący w grupie "`prac`" mogą go przeczytać i wykonać (prawo do wykonania wcale nie oznacza prawa do czytania), ale już nie mogą go modyfikować, zaś pozostali nie mają do tego zbioru żadnych praw.

1.5.1 Zmiana atrybutów pliku

Zmiany atrybutów pliku może dokonać tylko jego właściciel, i to niezależnie od tego, kto ma prawa do pisania po pliku! Zmiany tej dokonuje się poleceniem

`chmod` *kto w_jaki_sposób jakie_uprawnienia nazwa_pliku*

kto – tutaj możemy mieć dowolną kombinację znaków '`u`' (user – właściciel), '`g`' (group – grupa), '`o`' (others – pozostali) lub '`a`' (all – połączenie `u`', '`g`' i '`o`', czyli dla wszystkich);
w_jaki_sposób – możemy uprawnienia dodawać (znak '+'), odejmować ('-') lub ustawiać niezależnie od poprzednich ('=');
jakie_uprawnienia – dowolna kombinacja znaków '`r`', '`w`' i '`x`'.
Np. polecenie `chmod go-w` dane spowoduje odebranie prawa do pisania grupie i innym.

1.5.2 Grupa

Mechanizm dostępu grupy do pliku został wymyślony przede wszystkim po to, aby umożliwić sprawną pracę kilku osób nad tym samym projektem. Różne prawa do operacji na pliku dla grupy i pozostałych dają możliwość "dzielenia" się plikiem tylko z wybranymi osobami. **Pozwala to na udostępnianie zawartości pliku wybranym osobom bez przekazywania im hasła.**

Mniej prostym przypadkiem używania tego mechanizmu jest przypisanie plikowi grupy innej niż własna użytkownika. Np. prowadzący zajecia należący do grupy `prac` poleceniem `chgrp` może plikowi przypisać grupę `mes`. W ten sposób plik:

```
-rw-r----- 1 max mes 1662 Sep 18 18:15 zaliczenia.oceny
```
może być czytany tylko przez członków grupy `mes` oraz właściciela. (Poprawki może nanosić tylko właściciel).

2 Wydawanie poleceń

2.1 Składnia poleceń

Wszystkie polecenia mają taką samą składnie ogólną:

nazwa_polecenia argument argument argument...

Nazwa polecenia może być samą nazwą programu, jeżeli znajduje się on gdzieś na ścieżce dostępu, lub pełną nazwą ścieżkową, np. `/bin/rm`. Argumenty są przeważnie opcjonalne i składają się na opcje (poprzedzone znakiem '`-`') i nazwy zbiorów. Opcje można przeważnie zapisywać razem po jednym znaku '`-`' lub dzielić i pisać każdą osobno. Dlatego równoważne będą polecenia: `ls -l -g` oraz `ls -lg`. Podobnie przeważnie nie ma znaczenia kolejność opcji (`ls -lg` i `ls -gl`).

2.2 Redyrekcja wejścia/wyjścia

2.2.1 Pliki standardowe

Każdy program ma tzw. standardowe wejście i standardowe wyjście. Np. program sort czyta swoje standardowe wejście, sortuje podane linie i podaje posortowane na standardowe wyjście. Wywołanie tego programu bez żadnych opcji powoduje, iż będzie czytał linie z klawiatury, a po zakończeniu wpisywania wypisze na ekranie już posortowane linie. Jest to jednak dość niewygodne i wolelibyśmy np. przygotować zbiór do posortowania i kazać programowi posortować zawartość tego zbioru i wypisać na ekran. W tym celu wystarczy zmienić standardowe wejście programu z klawiatury na zbiór:

```
sort < dane
```

Polecenie to uruchomi program `sort`, przekazując mu jednocześnie, że swoje wejście ma czerpać ze zbioru dane. W analogiczny sposób można np. wyjście polecenia `ls -l` zamiast wypisywać na ekran skierować do jakiegoś zbioru:

```
ls -l > listing
```

Można łączyć jedno i drugie. Przypuśćmy, że mamy program o nazwie `prg`, który czyta dane ze standardowego wyjścia i po obliczeniu wypisuje wyniki na standardowe wyjście. Nie ma jednak sensu za każdym uruchomieniem wpisywać wszystkich danych od nowa, a i chcielibyśmy mieć wyniki w jakimś miejscu, a nie przepisywać z ekranu. Możemy zatem np. przygotować dane w zbiorze dane, i po uruchomieniu naszego programu poleceniem

```
prg < dane > wyniki
```

otrzymamy wyniki w zbiorze wyniki.

Uwaga: posłużenie się skierowaniem '`>`' do jakiegoś zbioru powoduje zniszczenie jego poprzedniej zawartości. Ominąć ten problem możemy używając skierowania '`>>`', które oznacza dopisywanie wyjścia do istniejącego pliku.

2.2.2 Potoki

Bardzo wygodnym sposobem przesyłania danych są potoki (pipelines). Wyobraźmy sobie sytuację, gdy np. `prog1` produkuje wyniki, które mogą potem służyć jako dane dla programu `prog2`. Aby połączyć jedno z drugim, moglibyśmy się posłużyć zbiorem pomocniczym i wykonać sekwencję rozkazów:

```
prog1 > zbior
prog2 < zbior
```

Jest jednak dużo wygodniejsza metoda. Możemy bowiem bezpośrednio nawiązać komunikację pomiędzy programami, łącząc standardowe wyjście jednego ze standardowym wejściem drugiego, używając do tego celu znaku '|', np.

```
prog1 | prog2
```

Należy zauważyć, że `prog2` nie jest nazwą zbioru, lecz programu.

3 Edytor Joe.

Podstawowym edytorem dla systemu operacyjnego Unix jest edytor vi. Tą szczególną pozycję zawdzięcza on dostępności w każdej implementacji systemu. Jednak nie jest to narzędzie zbyt lubiane przez początkujących adeptów Unixa, dlatego istnieje ogromna liczba innych edytorów łatwiejszych w pierwszym kontakcie. W komputerach laboratorium L–5 takimi edytorami są: `joe` i `xedit`. Szczególnie pierwszy z nich godny jest polecenia, ponieważ jego obsługa bardzo przypomina analogiczne programy DOSowe.

joe nie wymaga praktycznie żadnej nauki przed użyciem. Edytor osługuje się w sposób "naturalny" czyli po naciśnięciu klawisza pojawia się odpowiadająca mu litera w miejscu gdzie znajduje się kursor. Do przemieszczania kursora po ekranie służą strzałki, a klawisz `Backspace` powoduje usunięcie znaku znajdującego się przed kursorem. W każdej chwili dostępna jest ściąga z informacjami na temat bardziej skomplikowanych opcji edycyjnych. Po wywołaniu edytora w prawym górnym rogu pojawia się napis: `Ctrl-K H for Help`. Oznacza to, że należy nacisnąć klawisz `Control` (oznaczany też skrótowo `Ctrl`) i jednocześnie nacisnąć klawisz `k`, a następnie nacisnąć `h`. Ukazuje się wtedy menu z pytaniem, którą część helpu potrzebujemy, zazwyczaj potrzebna jest podstawowa czyli BASIC. Wybranie jej powoduje ukazanie się okna z informacją w górnej części ekranu. Znajdują się tam informacje o sekwencjach klawiszy służących poszczególnym operacjom edycyjnym. Znak "`^`" należy rozumieć jako przyciśnięty klawisz `Control`. Dodać należy, że są to typowe sekwencje stosowane w edytorze WordStar, a następnie przejęte przez środowisko Borlanda. Okno pomocy może być stale włączone w trakcie pisania, gdyż nie zakłóca ono pracy edytora, a jedynie zmiejsza dostępną część ekranu.

4 Kompilacja programów

Podstawowymi kompilatorami używanymi pod UNIXem są kompilator języka C i kompilator fortranu. Zazwyczaj istnieją w systemie dwa kompilatory C, jeden pochodzący od dostawcy sprzętu – zoptymalizowany dla jego potrzeb i kompilator pochodzący od grupy GNU, nie tak optymalny, ale łatwiej przenośny między różnymi platformami. Wywołanie wszystkich kompilatorów jest podobne:

kompilator opcje zbiór_źródłowy biblioteki
Każdy kompilator ze zbioru źródłowego z końcówką .c (język C), lub .f (fortran) tworzy zbiór pośredni z końcówką .o, a następnie wywołuje program łączący (linker), który łączy to ze standardowymi lub jawnie zadeklarowanymi bibliotekami tworząc program wykonywalny. Jeśli użytkownik nie poda jaka ma być nazwa programu wykonywalnego, domyślnie jest przyjmowana `a.out`.

4.1 Ważniejsze opcje kompilacji i łączenia

Generalna zasada jest następująca: wszystkie podane opcje, które nie dotyczą kompilatora, są traktowane jako opcje łaczenia (linkowania). Z tego też powodu opisu właściwych opcji trzeba szukać nie tylko w opisie kompilatora, ale także w opisie programu łączącego `ld`.

Kompilacja

-c tylko kompilacja, bez łączenia.

-g zachowanie informacji symbolicznej dla debuggera.

Łączenie

-o nazwa zbioru wykonywalnego

-l symboliczna nazwa biblioteki, podanie `-lx` oznacza żądanie łączenia z modułami zawartymi w bibliotece `libx.a`

Przykład

Podanie komendy:

```
f77 -g plik1.f plik2.f plik3.o -lmetnum -o program
```

zostanie zrozumiane jako wywołanie kompilatora fortranu (`f77`), który ma skompilować teksty procedur zawarte w plikach `plik1.f` i `plik2.f`. Do odpowiednich plików pośrednich `.o` zostanie dołączona informacja symboliczna. Plik `plik3.o` nie będzie kompilowany, bo już jest w postaci kodu pośredniego, więc jego nazwa zostanie przeniesiona do modułu łączącego, który połączy `plik1.o`, `plik2.o` i `plik3.o` z procedurami zawartymi w bibliotece `libmetnum.a`. Wynikowy program wykonywalny zostanie umieszczony w pliku `program`.