UNIX - SZTUCZKI I CHWYTY

Grzegorz Blinowski
G.Blinowski małpka ii pw edu pl

"Shaw's Principle:
Build a system that even a fool can use, and only a fool will want to use it."

Niby-przedmowa do wydania elektronicznego

"Sztuczki i chwyty" powstały 10 lat temu, jaka jest obecnie wartość merytoryczna tej książki trudno ocenić, sądzę, że jakieś 50% materiału jest "w miarę" aktualne. Jako ciekawostki polecam wzmianki o Linuxie i FreeBSD :-) Czy książka jest pomocna w nauce Unixa, czy też nadaje się tylko dla kustoszów muzeów informatyki – niech ocenią czytelnicy.

O stronie edytorskiej wydania elektronicznego

W stosunku do wersji oryginalnej (tej przed składem – innej nie posiadam) wprowadziłem tylko minimalne zmiany redakcyjne – wyróżniłem rozdziały i podrozdziały, dodałem spis treści. W sensie redakcyjnym książka jest bardzo siermiężna, niestety nie mam czasu, a może i ochoty na jej dalsze polerowanie...

O oryginalnym wydaniu

"Sztuczki i chwyty" zostały wydane przez "PLJ" w okolicach lata 1993 roku. Poziom edytorski tego wydania był niestety przerażająco niski, nie chodzi tu nawet o brak jakiejkolwiek korekty! Stosowany w owych czasach przez w.w. wydawnictwo system składu najwyraźniej popadł w konflikt z dość egzotycznymi sekwencjami znaków występującymi w uniksowych poleceniach przytaczanych w książce. Efekt - "połknięte" znaki i całe sekwencje, błędne formatowanie wcięć etc. W sumie (zanim zemdlałem) naliczyłem kilkadziesiąt większych i mniejszych błędów.

O ile się orientuję Wydawnictwo PLJ już nie istnieje, zaś niezależnie od tego faktu nadal jestem właścicielem prawa autorskich "Sztuczek" - stąd to elektroniczne wydanie.

Grzegorz Blinowski, Warszawa, styczeń 2004

SPIS TREŚCI

1.WSTĘP

System operacyjny UNIX zrobił oszołamiającą karierę od czasu swego powstania w 1969 roku na "mało używanym komputerze PDP 8 stojącym w kącie laboratorium". Obecnie jest dostępny dla olbrzymiej ilosci typów komputerów bardziej i mniej rozpowszechionych: od IBM AT po super-komputery firmy Cray. W ostatnich latach UNIX rozpowszechnił się również w naszym kraju i choć doleko mu do popularnosci MS-DOS-u to grono jego zwolenników ciągle rosnie. Jednoczesnie system ten uchodzi wsród większosci polskich użytkowników za trudny do nauczenia się i niewdzięczny w codziennych zastosowanich. Opinie te są po częsci wynikiem braku na naszym rynku książek wychodzących poza ramy "wstępu do wstępu" lub krótkiego kompendium. Osoby, które swoją wiedzę wyniosły z książek dla "bardzo początkujących" lub z kursów często czują się zagubione gdy stają przed konkretnym, nawet nieskomplikowanym problemem.

Książka ta ma być pomocą dla osób znających podstawy użytkowania systemu, a więc dla srednio zaawansowanych, z niewielkim lub srednim doswiadczeniem. Zawarto w niej odpowiedzi na konkretne pytania pojawiające się w codziennej pracy. Lektura pogrupowanych tematycznie przykładów i omówień mniej lub bardziej typowych problemów powinna być pouczająca nie tylko dla osób poszukujących konkretnych "recept" ale także dla tych, którzy jeszcze nie wiedzą, że dany problem istnieje! Tak więc np. rozdział poswięcony programowi rsh przeznaczony jest zarówno dla osób, które mają trudnosci z uruchomieniem tego programu jak i dla tych, które nigdy o takim programie nie słyszały.

Częsć materiału zawartego w "Sztuczkach i Chwytach" jest oparta o listę FAQ (Frequently Asked Questions) UNIX-owej grupy dyskusyjnej sieci Usenet. Tłumaczenia tych informacji wydaje się być celowe ze względu na ciągle małą dostępnosć w naszym kraju usług oferowanych przez Internet. Lista FAQ poswięcona systemowi UNIX stoi na bardzo wysokim poziomie merytorycznym, zas jaj tematyka dobrze pokrywa się z zapotrzebowaniem srednio zaawasnowanego użytkownika systemu.

Nadmieńmy też, że "Sztuczki i Chwyty" są przeznaczone dla tzw. zwykłych użytkowników, jednak początkujący i srednio zaawansowani administratorzy systemów powinni znaleźć tu wiele ciekawych informacji.

Innym problemem często bagatelizowanym przez autorów książek ze słowem "UNIX" w tytule jest mnogosć odmian tego systemu operacyjnego. Czytelnik nie rzadko przekonuje się, że opisywany program lub usługa jest w jego systemie nieobecny, lub też funkcjonuje zupełnie inaczej niż wynika to z napisanej niefrasobliwie publikacji. W niniejszej książce zwrócono dużą uwagę na uniwersalnosć omówień i przykładów, tam gdzie było to możliwe zasygnalizowano różnice wynikające z odmiennosci wersji UNIX-a. Wersji tych jest bardzo dużo, a więc z koniecznosci nie wszystkie mogły być tu opisane. Najważniejsze zostały wymienione w dodatku A.

Kłopoty z doborem własciwego słownictwa zawsze towarzyszyły polskojęzycznym publikacjom informatycznym. Istotne jest znalezienie złotego srodka pomiędzy skrajnosciami polonizowania wszystkiego na siłę a tworzeniem angielskawych potworków językowych. W "Sztuczkach i chwytach" starano się zminimalizować ilosć specjalistycznych słów angielskich. Dosć zaskakujący może się wydać fakt, że prawie wszystkie słowa i wyrażenia z żargonu UNIX-owego znalazły już swoje polskie tłumaczenie - pozytywnym przykładem jest tłumaczenie książki P. Silvestra.

Autor stoi na stanowisku, że polski przekład nie musi być tak krótki i elegancki jak dany zwrot angielski, powinien jednak przede wszystkim oddawać sedno danego wyrażenia.

Zakres "Sztuczek i chwytów" obejmuje następującą tematykę:

Powłoki:

UNIX-owym powłokom poswięcono dwa rozdziały: w pierwszym przedstawiono bardzo dokładny opis powłoki Bournea wraz z licznymi przykładami, w następnym znaleźć można rozwiązania problemów często pojawiających się w użytkowaniu powłoki Bournea oraz powłoki csh. Omówiono między innymi zasady tworzenia skryptów, pliki inicjalizacyjne powłok, kłopoty z przełączaniem standardowego wejscia i wyjscia, koncepcję "pracy w tle" i wiele innych zagadnień. Czytelnik znajdzie też nieco informacji o innych powłokach systemu.

Przenoszenie plików między systemami

W pierwszej czesci tego rozdziału przedstawiono proste i bardziej zaawansowane przykłady posługiwania się wymienialnymi nasnikami danych: tasmami i dyskietkami.

W drugiej czesci omówiono wiele ważnych usług sieciowych takich jak NFS, UUCP,FTP korzystanie z odległych stacji tasm i dyskietek. Czytelnik znajdzie tu między innymi informację o programach archiwizujących: tar, cpio i bar; kopiowaniu z wykorzystaniem sieciowego systemu plików, a także o prawach dostępu przy korzystaniu z pewnej grupy poleceń sieciowych.

Polecenia

Omówienie typowych i nietypowych zastosowań ważnych i przydatych programów systemowych: find i rsh. Szczególnie ten ostatni nastręczać może wiele trudnosci.

Pliki

W rozdziale tym zebrano informacje pomocne przy manipulacjach plikami. Wiele z zagadnień dotyczących operacji na plikach formalnie należy do czesci poswięconej powłokom, jednak dla wygody czytelnika zaostały one umieszczone własnie w tej czesci. W niniejszym rozdziale znaleźć można między innymi informację o odtwarzaniu usuniętych plików, wzorcach nazw, atrybutach plików. Omówiono też archiwizację, kompresję, wizualizację i zmianę formatu plików. Czytelnik dowie się między innymi jak przesyłać nie-tekstowe oraz bardzo długie pliki pocztą elektroniczną, jak poddać kompresji cały katalog plików, a także jak dokonać konwersji pliku tekstowego w formacie MS-DOS na format UNIX-a.

Procesy

Wykonywanie w tle, srodowisko procesu i skrypty wykonywalne to zagadnienia przedstawione w tym rozdziale. Czytelnik znajdzie też odpowiedź na pytania dotyczące błędów terminalnych i plików "core".

Dodatki

W dodatkach zamieszczono:

Zestawienie oraz porównanie najważniejszych odmian systemu.

Słowniczek "rozszyfrowujący" ważniejsze skróty z UNIX-owego swiatka. Tu Czytelnik znajdzie odpowiedź na pytania o znaczenie nazw takich jak: NIS, Mach, ONC, Motif, ...

Wykaz polsko- i angielsko- językcznej literatury poswięconej systemowi UNIX. Starano się zebrać informację o wszystkich książkach polskich, oraz o najwartosciowszych publikacjach w języku angielskim.

UNIX-owy słowniczek angielsko-polski podsumowywuje zastosowaną w tej książce konwencję nazewniczą.

Słowo o formie graficznej książki.

Nazwy programów w UNIX-ie składają się prawie zawsze z małych liter i tak też są pisane w "Sztuczkach i Chwytach", od tej zasady uczyniono odstępstwo w wypadkach gdy nazwa programu rozpoczyna zdanie. Nie powinno to jednak wprowadzić czytelnika w błąd gdyż zwykle wczesniej pojawia się poprawnie pisana nazwa programu.

Często po nazwie programu, usługi, formatu danych pojawia się cyfra w nawiasach, podaje ona numer rozdziału komputerowego podręcznika zawierającego opis danego hasła - polecamy lekturę rozdziału poswięconego poleceniom systemu.

Nazwy programów, plików, katalogów, a także znaki specjalne i ciągi takich znaków z reguły ujmowane są w podwójne cudzysłowy. Z cytowania zrezygnowano w miejscach, w których mogło by ono wprowadzić zamieszanie: np. przy omawianiu cytowania słów w powłoce.

Teksty skryptów, informację "wyswietlaną" i "wpisywaną" na terminalu w przykładach zapisane są **tym specjalnym krojem**.

Składnia wywołania programów oraz opcje wywołania wydrukowane są *krojem pochyłym*.

Proszę również zwrócić uwagę na fakt istnienia dwu typów apostrofów: zwykłego - ' i odwróconego - ` mają one zupełnie różne znaczenia dla powłoki i nie powinny być nigdy utożsamiane. Podobnie znaki / oraz \ służą różnym celom: pierwszy oddziela nazwy katalogów w scieżce dostępu do pliku, drugi służy głównie jako znak cytujący

znaki występujące bezposrednio po nim.

2.POWŁOKA sh(1).

Program POWŁOKI, nazywany też interpreterem poleceń, a po angielsku shell, jest prawdopodobnie najczęsciej opisywanym programem systemu UNIX. Różnym jego wariantom poswięcono wiele miejsca we wszystkich dostępnych na naszym rynku książkach opisujących ten system. Wszystkie dotychczasowe publikacje w języku polskim przeznaczone były dla użytkowników początkujących lub mało zaawansowanych, dlatego też, z koniecznosci powłoki systemowe opisywane byly dosć pobieżnie. Celem autorów było zapoznanie czytelników z podstawami użytkowania systemu a nie z programem interpretera.

Rozdział poniższy stanowi dosć dokładny opis powłoki Bournea. Postaramy się wyraźnie rozgraniczyć funkcje systemu i funkcje programów usługowych od funkcji swiadczonych przez powłokę. Z dużą uwagą zostaną potraktowane zagadnienia, które z pozoru wydać się mogą nieistote, a których znajomosć oszczędzić może wiele próżnego wysiłku i poszukiwań przy pierwszej próbie napisania przez czytelnika nietrywialnego programu w sh.

Powłoka sh służy głównie jako język programowania - a więc do pisania skryptów. Pewne jej cechy sprawiają, że nie jest ona preferowana do pracy interakcyjnej: brak jest zarządzania pracami, uzupełniania nazw plików (file name completion) oraz mechanizmu historii. Większosć osób wybiera więc powłokę csh, choć wielu doswiadczonych użytkowników posługuje się programem sh także przy pracy interakcyjnej. W niektórych systemach dostępna jest rozszerzona wersja sh - program ksh, nadaje się on dobrze zarówno do pracy interakcyjnej jak i do wsadowej.

Opis poniższy oparty jest na standardowym opisie sh(1), rozszerzono go jednak o przykłady i dodatkowe wyjasnienia, tam gdzie podstawowy tekst mógł by być niejasny dla srednio zaawansowanego czytelnika.

Zakładamy, że czytelnik zna podstawy użytkowania systemu (choć dla wnikliwego odbiorcy nie musi to być konieczne) i pragnie usystematyzować swoją wiedzę. Dlatego nie przedstawimy elementarnych pojęć pliku, katalogu, procesu itp.

Składnia wywołania:

sh [-acefhiknstuvx] argumenty

Definicje podstawowe:

odstęp:
znak tabulacji - TAB lub SPACJA
nazwa:
ciąg liter, cyfr i znaków podkreslenia nie zaczynający się od cyfry
parametr:
nazwa, cyfra, lub który kolwiek ze znaków: "*", "@", "#", "?", "-", "$", i "!"

Składnia:

Proste_polecenie:
Ciąg niepustych słów oddzielonych odstępami. Pierwsze słowo jest nazwą polecenia/programu do wykonania. Prócz przypadków wymienionych poniżej pozostałe słowa są przekazywane jako argumenty do wykonywanego polecenia. Nazwa polecenia jest przekazywana jako argument

zerowy.

Przyklad:

cat plik1 plik2

"cat" - polecenie; "plik1", "plik2" - argumenty

Potok:
Polecenie lub sekwencja poleceń (nie koniecznie prostych) oddzielonych od siebie znakiem "|" lub "^". Standardowe wyjscie każdego polecenia, za wyjątkiem ostatniego, stanowi standardowe wejscie następnego polecenia. Każde polecenie wykonywane jest jako osobny proces (być może za wyjątkiem ostatniego, jesli jest ono poleceniem wbudowanym). Powłoka czeka aż ostatnie polecenie potoku zakończy działanie.

Kody powrotu:
Jesli polecenie zakończyło się poprawnie jego kod powrotu wynosi zero, w przeciwnym wypadku jest on większy od 128.
Kod powrotu potoku równy jest kodowi powrotu jego ostatniego polecenia.
Kod powrotu procesu jest dostępny poprzez odpowiednią zmienną specjalną powłoki. Dodajmy, że więcej informacji o kodach powrotu procesów i ich interpretacji znaleźć można w opisie funkcji systemowej wait(2).

Przykład:

ls /home | grep abcd | more

ls /home	pierwsze polecenie
grep abcd	drugie polecenie
more	trzecie polecenie

Pierwsze polecenie wywoła program ls, który przekaże do następnego listę plików znajdującym się w katalogu "/home". Polecenie drugie wywoła program grep z jednym argumentem "abcd". Program grep wyprowadzi na swoje standardowe wyjscie te linie (tu: nazwy plików), które zawierają dany wzorzec "abcd". Trzecie polecenie wyswietli otrzymaną z polecenia drugiego, poprzez swoje standardowe wejscie, listę plików, interakcyjnie "stronicując" wydruk.

Lista jest to jedeno, lub więcej prostych_poleceń lub potoków, oddzielonych znakami ";", "&", "&&", "||". Lista jest zakończona znakiem ";", "&" lub znakiem nowej linii. Znaki ";" i "&" mają równe i niższe priorytety niż "||", "&&". Powłoka czeka aż wykonanie poprzedzającego znak ";" potoku zostanie zakończone. Znak "&" powoduje, że poprzedzający go potok zostanie wykonany asynchronicznie (w tle) - powłoka nie będzie czekać na zakończenie pracy potoku. Symbole "&&" i "||" używane są w celu warunkowego wykonania listy po ich prawej stronie: Lista po prawej stronie symbolu "&&" wykonana zostanie tylko wtedy gdy potok po jego lewej stronie zwrócił zerowy kod powrotu. Lista po prawej stronie symbolu "||" wykonana zostanie tylko wtedy gdy potok po jego lewej stronie zwrócił nie zerowy kod powrotu. Dowolna ilosć znaków "NOWALINIA" może zastępować w liscie sredniki.

Przykłady:

./my_job dane | sort -r | store & emacs prog.c

Lista składa się z dwu potoków. Pierwszy potok składa się z poleceń: "./my_job dane", "sort"-r i "store", i wykonywany jest w tle. Drugie polecenie listy "emacs prog.c" wywoływane jest synchronicznie: powłoka czeka na jego zakończenie przed wyswieleniem kolejnego monitu.

test a -gt b && echo "źle, nieprawda, że a > b"
test a -gt b || echo "dobrze, nieprawda, że a > b"

Wykonanie pierwszej listy nie spowoduje wyswietlenia żadnego komunikatu: program test zwróci wartosć niezerową a więc polecenie "echo ..." nie zostanie wykonane. W drugim przykładzie wyswietlony zostanie komunikat "dobrze, nieprawda, że a > b".

Słowa specjalne
Specjalne znaczenie następujących słów jest rozpoznawane gdy są one pierwszym słowem polecenia oraz nie są cytowane:

if then else for case esac for while until do done { }

Komentarze
Słowa zaczynające się znakiem "#" i wszystkie następne, aż do znaku nowej linii są przez powłokę ignorowane.

Polecenia
Polecenie jest to proste_polecenie lub jedna z następujących konstrukcji:

for nazwa [in słowo ...] do lista done %$for% %$done% %$do%

lista będzie wykonana raz dla każdej kolejnej wartosci zmiennej nazwa. Zmienna nazwa będzie przyjmowała wartosci z ciągu "słowo ...". Jeżeli fraza "in słowo ..." zostanie pominięta zmienna nazwa będzie przyjmowała kolejne wartosci parametrów pozycyjnych (czyli argumentów wywołania) powłoki.

Przykład:

```
for litera in a b c; do /bin/echo -n $litera; done

for litera in a b c
  do /bin/echo -n $litera
done
```

Wyswietlony zostanie ciąg znaków "abc", proszę zauważyć, że sredniki po liscie wartosci i liscie poleceń są konieczne, gdyby ich nie było powłoka nie wiedziała by gdzie kończy się lista!

Drugi przykład również wyswietli ciąg "abc"; tu znaki nowej linii zastępują sredniki.

case słowo in [wzorzec [| wzorzec] ...) lista ;;] ... esac %$case% %$in% %$esac%

Wykonana zostanie lista poleceń występującą bezposrednio po pierwszym wzorcu pasującym do słowa. Wzorzec ma taką samą postać jak wzorce nazw plików, z tą różnicą że rozpoczynające słowo znaki "/", "." oraz "/." nie muszą zostać dopasowane eksplicite.

Przykład:

```
case $USER in
 ala | ola)      # wykonaj polecenia dla ali i oli
                 ;;
 *)              # wykonaj polecenia dla pozostałych wartosci USER
                 ;;
esac             # Po prostu "case" od końca!
```

Słowo które próbujemy dopasować to wartosć zmiennej USER, wzorce to "ala" i "ola" dla pierwszej listy i "*" (wzorzec pasujący do wszystkich słów) dla drugiej listy. Jesli zmienna USER ma wartosć "ala" lub "ola" wykonywana jest pierwsza lista, dla pozostałych wartosci tej zmiennej wykonywana jest druga lista. Proszę zwrócić uwagę na podwójne sredniki kończące listę poleceń.

if lista then lista [elif lista then lista] .... [else lista] fi %$if% %$then% %$else% %$fi%

Wykonywana jest lista występująca po słowie "if", jesli jej kod powrotu wyniesie zero wykonywana jest lista po słowie then, w przeciwnym wypadku wykoywana jest lista po słowie elif i jesli jej kod powrotu wyniesie zero wykonana zostanie lista występująca po drugim słowie then. Jeżeli kolejne listy występujące po słowie elif zwrócą zerowy kod powrotu zostanie wykonana lista występująca po słowie else. Jesli nie wykonana się żadna then-lista, ani else-lista kod powrotu polecenia wyniesie zero, w przeciwnym wypadku kod powrotu konstrukcji if równy jest kodowi powrotu wykonanej then-listy lub else-listy.

Przykład:

```
if test -d Katalog; then
        echo "Katalog instnieje"
elif test -d .katalog then
        echo "katalog ukryty istnieje"
else echo "Katalogi nie istnieja";  fi
```

Jesli plik "Katalog" istnieje, i jest katalogiem wyswietlony zostanie komunikat "Katalog istnieje", w przeciwnym wypadku, jesli istnieje katalog ".katalog" zostanie wydrukowany komunikat "katalog ukryty istnieje". Jesli nie istnieją katalogi "Katalog" oraz ".katalog" wyswietlony zostanie komunikat "Katalogi nie istnieją".

while lista do lista done %$while% %$do% %$done%

Powtarzane jest wykonywanie while-listy, jesli kod powrotu jej ostatniego polecenia wynosi zero wykonywana jest do-lista, w przeciwnym wypadku pętla zostaje zakończona. Jesli do-lista nie zostanie wykonana ani razu kod powrotu polecenia wynosi zero. Słowo while może być zastąpione przez until w celu negacji testu zakończenia pętli.

(lista)

Wykonanie listy w pod-powłoce.

Przykład:

```
( cd /tmp; /usr/local/bin/praca )
```

Typowym przykładem zastosowania konstrukcji nawiasowej jest zmiana katalogu aktualnego i wykonanie pewnego zadania. Dalsza część skryptu wykonuje się z oryginalnym katalogiem aktualnym.

{lista;}

Wykonanie listy.

nazwa () {lista;} %$funkcje powłoki sh%

Definicja funkcji o nazwie "nazwa" i ciele "lista". Zobacz też dalej: "Wykonanie polecenia", export i return.

Przyklad:

```
$my_error() {
>echo "błędn$1 $2 w $3"
>}
$my_error(y,znak,pliku)
```

```
błędny znak w pliku
$
```

Zdefiniowano i przetestowano funkcję my_error

```
fun_old() { old=new; }
old=old
fun_old()
if [ $old == "old" ]; then echo "Stara wersja sh"
  else  echo "Nowe sh"
fi
```

Powyższy skrypt sprawdza, czy wartosci zmiennych mogą zostać "przeniesione" z wnętrza funkcji (w starszych wersjach systemu funkcje powłoki wykonywane byly w pod-powłokach).

Nazwy funkcji mogą pokrywać sie z nazwami poleceń wbudowanych, jednak funkcji takiej nie da się wywołać - przykład:

```
$fun echo() { echo 'tra la la' ; }
$echo ha! ha!
ha! ha!
$
```

Przed wykonaniem polecenia powłoka dokonuje szeregu substytucji tekstowych, opiszemy je w kolejnosci wykonywania:

Substytucja poleceń.

Substytucja poleceń polega na wykonaniu poleceń występujących między dwoma odwróconymi apostrofami - "`" i tekstowym zastąpieniu tych poleceń rezultatami ich dzialania tj. ich standardowym wyjsciem.

Z ciągu znaków zawartego między odwróconymi apostrofami usuwane są "\" cytujące znaki: "\", "`" , ","$", NOWALINIA . Znaki "`" mogą być cytowane, możliwe jest więc zagnieżdżanie substytucji poleceń. Pary "\" NOWALINIA są usuwane. Ponieważ znak "\" występujący przed znakiem "$" jest również usuwany konstrukcja "\$parametr" jest traktowana jak "$parametr". Znak cytujący występujący przed innym znakiem niż uprzednio wymienione nie jest usuwany.

Przykłady:

Załóżmy, że w katalogu aktualnym istnieją trzy pliki o nazwach $x, plik1 i x", oto przykładowa sesja z terminalem:

```
$ls
plik1   x"   $x
$cat plik1 'x"' '$x'        # obejrzmy ich zawartosć
to jest plik1
to jest x"
plik
$x=plik1                    # nadajmy zmiennej x wartosć plik1
$cat `ls $x`                # dostaniemy: cat plik1
to jest plik1
$cat `ls \$x`               # dostaniemy: cat plik1
to jest plik1
$cat `ls \\$x`              # dostaniemy: cat $x
plik
```

$cat `ls x\"` # dostaniemy: cat x"
to jest x"
$cat `cat \\$x`1 # dostaniemy: cat plik1
to jest plik1

Substytucja parametrów:

Po wykonaniu substytucji poleceń dokonywana jest substytucja parametrów.

W powłoce sh zaimplementowano dwa typy parametrów: parametry pozycyjne i tzw. zmienne. Parametry pozycyjne ozaczane są cyframi (od 0 do 9), wartosci tych parametrów zmienia się poleceniem "set". Do zmiennych odwołujemy sie poprzez ich identyfikator, który jest nazwą. Wartosci nadajemy zmiennym znakiem "=":

nazwa_zmiennej=wartość

Po znaku "=" nie powinien wystąpić odstęp

Nazwa zmiennej nie może kolidować z nazwą funkcji.

Do wartosci parametru odwołujemy się pisząc $nazwa_parametru lub $n (n =0...9).

Przykłady:

$x='zmienna x'
$echo $x
zmienna x
$echo $0
sh
$

Istnieje też kilka dodatkowych "rozszerzeń" mechanizmu substytucji parametrów ułatwiających programowanie w shellu:

${parametr}
podstawiana jest wartosć parametru, nawiasy klamrowe są potrzebne gdy chcemy "skleić" wartość parametru z następnym słowem.

${parametr:-słowo}
podstawiany jest parametr jesli jest on zdefiniowany, w przeciwnym wypadku podstawiane jest słowo

${parametr:=słowo}
jesli parametr nie został zdefiniowany, lub jest pusty, nadawana mu wartosć słowo. Podstawiana jest wartosć parametru.

${parametr:?słowo}
jesli parametr jest zdefiniowany podstawiana jest wartosć parametru, w przeciwnym wypadku wyswietlane jest słowo i wykonanie powłoki zostaje zakończone.

${parametr:+słowo}
jesli parametr jest zdefiniowany i niepusty podstawiane jest słowo, w przeciwnym wypadku podstawiane jest słowo puste

Przykład:

Przedstawiamy rezultaty działania skryptu dokonującego przykladowych substytucji parametrów:

Skrypt:

```
echo ${1:-"param #1 pusty"}
echo ${1:+"param #1 nie pusty"}
echo 'ls:' ${ls:-`ls`}
echo ${2:?"param pusty, koniec"}
```

Oto przykładowe rezultaty działania powyższego skryptu, zakładamy, że katalog aktualny zawiera pliki "pattern" oraz "pattern.res", a zmienna "ls" nie jest zdefinowana:

```
$pattern
param #1 pusty

ls: pattern pattern.res
./pattern: 2: param pusty, koniec

$pattern a
a
param #1 nie pusty
ls: pattern pattern.res
./pattern: 2: param pusty, koniec

$pattern a b
a
param #1 nie pusty
ls: pattern pattern.res
b
```

Interpretacja odstępów:

%$interpretacja odstępów w powłoce sh%
Po wykonaniu substytucji poleceń oraz parametrów powłoka dzieli otrzymany tekst na słowa. Jako separatory słów przyjmuje się znaki zawarte w zmiennej IFS (Internal Field Separator). Jawne słowa puste: " lub "" są zachowywane, niejawne słowa puste (pojawiające się np. w wyniku substytucji parametrów nie zdefiniowanych) są usuwane.

Wejscie/Wyjscie:

%$wejscie/wyjscie w powłoce sh%
Wejscie i Wyjscie polecenia może zostać skojarzone z dowolnym plikiem, lub zamknięte przy pomocy specjalnej notacji powłoki. Wymienione niżej konstrukcje dołączania wejscia/wyjscia maga pojawić się w dowolnym miejscu prostego polecenia oraz na końcu polecenia, NIE są one przekazywane jako argumenty wywoływanych poleceń.

Zanim przedstawimy wspomniane wyżej konstrukcje niezbędna jest krótka dygresja dotycząca deskryptorów plików.

Każdy nowo powołany proces w systemie UNIX posiada tzw. tablicę otwartych plików. Proces identyfikuje plik jako indeks tej tablicy. Indeks ten nazywany jest deskryptorem pliku. Proces

potomny dziedziczy tablicę otwartych plików swego procesu macierzystego. Proces powoływany przez powłokę posiada trzy otwarte pliki tzw. dojscia: standardowe wejscie - deskryptor nr. 0, standardowe wyjscie - deskryptor nr. 1, standardowa diagnostyka - deskryptor nr. 3. Pliki te w procesie potomnym są domyslnie takie same jak w powłoce - dlatego też możemy porozumiewać się z uruchomionym programem tą samą drogą (z tego samego terminala) co z powłoką. Nic nie stoi jednak na przeszkodzie by związać standardowe dojscia procesu z innymi plikami, oraz by otworzyć dla procesu nowe pliki i związać je z arbitralnymi deskryptorami. Służą temu następujące konstrukcje:

< nplik
Użyj pliku o nazwie "nplik" jako standardowego wejscia

> nplik
Użyj pliku o nazwie "nplik" jako standardowego wyjscia. Jesli plik nie istnieje zostanie utworzony; jesli plik istnieje jego poprzednia zawartość zostanie skasowana

>> nplik
Użyj pliku o nazwie "nplik" jako standardowego wyjscia. Jesli plik nie istnieje zostanie utworzony; jesli plik istnieje standardowe wyjscie procesu zostanie dołączone do jego końca.

<&n
Skopiuj dojscie numer n na dojscie nr 0 - otwarty plik o deskryptorze nr. n staje się również standardowym wejsciem.

>&n
Skopiuj dojscie numer n na dojscie nr 1 - otwarty plik o deskryptorze nr. n staje się również standardoweym wyjsciem.

<&-
Zamknij standardowe wejscie. Proces czytający z zamkniętego w ten sposób pliku zobaczy znak EOF.

>&-
Zamknij standardowe wyjscie.

Każdy z poprzednich symboli może zostać poprzedzony cyfrą, interpretacja takich konstrukcji jest następująca:

n>&m
Skopiuj dojscie numer m na dojscie nr n

n<&m
Skopiuj dojscie numer m na dojscie nr n

n< nplik
Otwórz plik o nazwie "nplik" do czytania i skojarz go z deskryptorem nr. n

n> nplik
Otwórz plik o nazwie "nplik" do pisania i skojarz go z deskryptorem nr. n

n<&-
Zamknij plik o deskryptorze nr n.

n>&-
Zamknij plik o deskryptorze nr n.

Kolejnosć występowania symboli podłączenia plików jest istotna. Powłoka ewaluuje je od lewej do prawej, np. "... 1>xxx 2>&1" oznacza, że deskryptor nr. 1 zostanie skojarzony z plikiem o nazwie xxx, a następnie skopiowany na deskryptor nr. 2 (standardowa diagnostyka). W ten

sposób standardowe wyjscie i standardowa diagnostyka zostaną skierowane do tego samego pliku. Proszę zauważyć, że gdyby kolejnosć była odwrotna standardowa diagnostyka została by skojarzona z terminalem (oczywiscie jeżeli deskryptor nr. 1 był z nim skojarzony).

Przykłady:

Skojarzenie standardowego wyjscia i standardowej diagnostyki z tym samym plikiem "rezultaty":

./prog > rezultaty 2>&

Wyswietlona zostanie zawartosć pliku "./dane":

cat 9< ./dane 8<&9 <&8 # dziwnie i niepotrzebnie, ale dobrze

Należy pamiętać, że niedozwolone jest czytanie z pliku otwartego do zapisu, oraz zapis do pliku otwartego do czytania. Polecenie:

cat 9> ./dane <&9

Kopiujące na deskryptor standardowego wejscia deskryptor pliku otwartego do zapisu spowoduje wyswietlenie komunikatu o błędzie:

cat: read error on standard input: Bad file number

Podobnie polecenie:

cat 9< ./plik >&9

Wygeneruje błąd:

cat: write error: Bad file number

Dokumenty włączone (here documents):

%$dokumenty włączone w powłoce sh%

Mechanizm dokumentu wlączonego umożliwia związanie standardowego wejscia poleceń wykonywanych przez powłokę ze standardowym wejsciem powłoki, aż do wystąpienia w strumieniu wejsciowym słowa oznaczającego koniec dokumentu włączonego, lub do wystąpienia końca pliku.

Składnia:

<< [-]słowo

Po wykonaniu substytucji poleceń i parametrów w słowie powłoka przekazuje kolejne linie standardowego wejscia do polecenia. Jesli słowo, lub jego część jest cytowana to we wczytywanym tekscie nie są dokonywane żadne substytucje, w przeciwnym wypadku dokonywana jest substytucja poleceń i parametrów, cytowane znaki nowej linii (\NOWALINIA) są usuwane, należy więc cytować znaki \, $, ' aby uniknąć ich interpretacji przez powłokę.

Jesli słowo poprzedzone jest znakiem "-" to:
1) Znaki tabulacji rozpoczynające słowo są usuwane przed dokonaniem substytucji na słowie.
2) Znaki tabulacji rozpoczynające linię tekstu włączonego są usuwane.

Przykłady:

Rezultatem skryptu,:

```
cat > here-tekst << EOTEOT
$USER
 tekst
        tekst po znaku tabulacji
EOTEOT
cat >> here-tekst << -'EOTEOT'
$USER
 tekst
        tekst po znaku tabulacji
EOTEOT
```

przy wartosci zmiennej USER "/home/staff/gjb" będzie zapisanie do pliku o nazwie "here-tekst" następujących linii tekstu:

```
/home/staff/gjb
 tekst
        tekst po znaku tabulacji
$USER
 tekst
tekst po znaku tabulacji
```

Substytucja nazw plików:

$%substytucja poleceń w powłoce sh%
Przed wykonaniem polecenia każde jego słowo zawierające jeden ze znaków "*", "?", "[" jest interpretowane jako wzorzec nazwy pliku. Wzorzec nazwy pliku zastępowany jest alfabetycznie posortowaną listą nazw plików do niego pasującą, jesli do wzorca nie pasuje nazwa żadnego pliku powłoka pozostawia wzorzec bez zmian. Znak "." występujący na początku nazwy pliku bezposrednio po znaku "/" oraz sam znak "/" muszą być dopasowane jawnie.
Interpretacja wzorca nazwy pliku:

*	Pasuje do każdej nazwy, także do pustej
?	Pasuje do każdego znaku
[...]	Pasuje do każdego znaku zawartego między nawiasami
[x-y]	Pasuje do każdego znaku z zakresu x-y
[!...]	Pasuje do wszystkich znaków z wyjątkiem występujących po "!"

Przykłady:

Wyswietlone zostaną nazwy wszystkich plików z aktualnego katalogu nie rozpoczynających się od "." i kończących sie na "o"

```
echo *o
```

Wyswietlone zostaną nazwy wszystkich plików z aktualnego katalogu

```
echo .* *
```

Cytowanie znaków:

%$cytowanie w powłoce sh%
Podsumujmy reguły cytowania znaków specjalnych:

Następujące znaki mają specjalne znaczenie dla powłoki i kończą słowo (chyba, że są cytowane):

; & ( ) | ^ < > NOWA_LINIA SPACJE* ZNAKITABULACJI**

Gwiazdki ozaczają dowolną ilosć poprzedzających je znaków specjalnych. Znak może zostać zacytowany poprzez poprzedzenie go znakiem \, lub ujęcie w pojedyncze lub podwójne apostrofy: '', "". Przypomnijmy jednak, że znak \ nie cytuje wewnątrz odwróconych apostrofów. Cytowany znak nowej linii jest usuwany z wnętrza słowa przed dokonaniem wszystkich substytucji.

Wszystkie znaki zawarte między pojedynczymi apostrofami (''), prócz pojedynczego apostrofu, są cytowane. Znak ' można zacytować ujmując go między znakami cudzysłowa: """.

Wewnątrz cudzysłowów ("") dokonywane są substytucje poleceń i parametrów, rezultaty są cytowane - tak, że nie zachodzi substytucja nazw plików i interpretacja odstępów. Po dokonaniu wszystkich substytucji słowa zawarte w cudzysłowie przekazywane są do polecenia jako jeden argument (ale, patrz dalej). Znak "\" cytuje wewnątrz cudzysłowów znaki "\", "`", ", "$", NOWALINIA w pozostałych przypadkach znak "\" sam jest cytowany (pozostaje w słowie). Wyjątkiem od zasady przekazywania cytowanych słów jako pojedynczego argumentu jest $@ cytowane podwójnym cudzysłowem co interpretowane jest jak "$1" "$2" ... - w ten sposób możemy zacytować osobno wszystkie parametry pozycyjne.

Przykłady:

Skrypt "cnt" wyswietla liczbę argumentów wywołania, oto on:

```
#!/bin/sh
echo $#
```

I rezultaty jego wywołania:

```
$./cnt "a b"
1
$./cnt 'a b'
1
$./cnt a b
2
```

Cytowanie a substytucja nazw plików:

```
$a='*'
$echo $a
.....                <<- lista plików
$echo "$a"
*
$echo '$a'
$a
```

Jesli katalog aktualny był by pusty pierwsze polecenie "echo" wyswietliło by znak "*".
W drugim przykładzie zaszła substytucja parametrów, lecz nie była dokonana substytucja nazw plików.

Wykonanie polecenia:

Po dokonaniu substytucji poleceń, parametrów i nazw plików dane polecenie jest wykonywane:

1) Jesli polecenie jest poleceniem wbudowanym powłoki (takim jak np. cd), lub jesli jego nazwa zgodna jest z nazwą jednej ze zdefiniowanych funkcji jest ono wykonywane w aktualnej powłoce (w starszych wersjach sh funkcje wykonywane były w pod-powłoce).

2) Jesli polecenie nie spełnia powyższych warunków powłoka tworzy nowy proces, i próbuje znaleźć, a następnie uruchomić w nim - poprzez funkcję systemową execve(2) - dane polecenie:

W tym celu przeszukiwane są katalogi wymienione w zmiennej srodowiskowej PATH, chyba że nazwa polecenia zawiera znak "/". Po odnalezieniu program zostaje uruchomiany, pod warunkiem, że: ma pozwolenie na wykonanie i jest w formacie binarnym wykonywalnym (a.out(5)) lub jest tzw. wykonywalnym skryptem zaczynającym się znakami #! . W wypadku wykonywalnego skryptu powłoka uruchamia interpreter, którego nazwa występuje po #! - np. #!/bin/sed - podłączając jednoczesnie skrypt jako plik z poleceniami dla uruchomionego procesu; nie wszystkie wersje systemu implementują skrypty wykonywalne (zob. exec(2) by zorientować się czy Twój system zapewnia ten mechanizm). Skrypty wykonywalne zostały dokładnie omówione w rozdziale poswięconym procesom.

Przeszukanie wielu katalogów może być stosunkowo długotrwałą operacją, dlatego powłoka pamięta położenie uruchamianych uprzednio programów. Może to być przyczyną nieoczekiwanego zachowania powłoki: Załóżmy, że zmienna PATH ma postać: ...:/moje/bin:/inne/bin. Katalog "/inne/bin" zawiera program "prog". Po pierwszym uruchmieniu jego powłoka zapamięta położenie programu "prog". Jesli utworzymy w katalogu /moje/bin program o tej samej nazwie - "prog" powłoka nadal będzie wykonywała program z katalogu /inne/bin. W podobnych wypadkach należy nakazać powłoce uaktualnienie informacji o położeniu plików - zob. polecenie hash.

Uwagi:

Polecenie wbudowane "type" informuje o tym jak interpretowane będzie konkretne polecenie.

Jesli chcemy by wykonywane były programy z katalogu aktualnego musi być on wymieniony w zmiennej PATH. Pusta nazwa katalogu w PATH - tj. :: - jest traktowana jako katalog aktualny. Umieszczenie nazwy katalogu aktualnego w zmiennej PATH jest jednak odradzane - zob. uwagi w kolejnym rozdziale.

Środowisko (environment)

Podobnie jak każdy proces powłoka posiada srodowisko (zob. environ(5)) będące listą par "nazwa wartosć". Przed rozpoczęciem interpretacji poleceń powłoka sh kopiuje zmienne srodowiskowe do swojej przestrzeni parametrów. Oznacza to, że operacje na parametrach (zmiennych) powłoki nie mają wpływu na srodowisko powłoki, podobnie nowo utworzony parametr nie jest przekazywany do srodowiska. Do związania parametrów powłoki z jej zmiennymi srodowiskowymi służy polecenie "export" przywiązujące paramter powłoki do srodowiska (zob. też set -a). Zmienna srodowiskowa może zostać usunięta poleceniem wbudowanym "unset". Środowisko procesu powołanego przez powłokę zawiera więc: oryginalne srodowisko odziedziczone przez powłokę, minus zmienne usunięte przez "unset", plus eksportowane (!) parametry zmienione i dodane.

Środowisko każdego prostego_polecenia może zostać rozszerzone o nowe zmienne za pomocą poprzedzenia prostego_polecenia parami: "zmienna=wartosć".

Przykład:

EDITOR=vi prog
(export EDITOR; EDITOR=vi; prog)

Dwie powyższe konstrukcje są równoważne.

Polecenia wbudowane powłoki.

Przedstawimy tu tylko niektóre, najprzydatniejsze polecenia wbudowane ([...] oznacza ciąg argumentów):

: %$polecenie puste powłoki sh%
Polecenie puste

. nazwapliku

Wykonuje w aktualnej powłoce polecenia z podanego pliku

break [n] %$break%

Wyjscie z n otaczających pętli for lub while (n domyslnie = 1). Zobacz ostatni przykład w tym rozdziale.

continue [n] %$continue%

Rozpoczyna kolejną iterację n-tej otaczjącej pętli for lub while (n domyslnie = 1)

cd[argument] %$cd%

Zmienia katalogu aktualny na podany. Domyslnym argumentem jest $HOME. Jesli argument nie zaczyna się znakiem "/" (a więc nie jest bezwzględną nazwą katalogu) w celu znalezienia katalogu "argument" przeszukiwane są katalogi okreslone w zmiennej CDPATH. %$zmienna $CDPATH%

echo [...] %$echo%

Przesyła argumenty do standardowego wyjscia, zauważmy, że istnieje też bardziej uniwersalny program echo(1).

eval [...] %$eval%

Argumenty są wczytywane, interpretowane i wykonywane tak jak standardowe wejscie powłoki.

Przykład:

Skrypt "cnt" podający liczbę swoich argumentów (patrz wyżej) zwróci następujace rezultaty jesli wywołamy go przez eval:

$eval ./cnt 'a b'
2
$eval ./cnt '"a b"'

exec [...] %$exec%

Polecenie okreslone przez argumenty jest "nakładane" na powłokę, oznacza to że po zakończeniu wykonania polecenia nie powracamy do powłoki. Polecenia exec możemy też użyć do zmiany plików wejsciowych powłoki - zob. przykłady w następnym rozdziale.

exit [n] %$exit%

Zakończenie działania powłoki, kod powrotu wyniesie n, jesli n nie zostanie podane kod powrotu równy będzie kodowi powrotu ostatniego wykonanego polecenia. Zauważmy, że znak EOF (ctrl-D) wprowadzony jako pierwszy w linii również zakończy działanie powłoki.

export [nazwa ...] %$export%

Zmienne o podanych nazwach będą obecne w srodowisku następnych wykonywanych poleceń. Funkcje nie mogą być eksportowane. Polecenie eksport bez argumentów wyswieli nazwy eksportowanych zmiennych, jednak nazwy zmiennych eksportowanych z powłoki macierzystej nie będą wyswietlone.

read [nazwa ...] %$read%

Ze standardowego wejscia wczytywana jest linia tekstu, po podzieleniu jej na słowa (odstępy okreslone są w zmiennej IFS) kolejnym zmiennym "nazwa" przypisywane są kolejne słowa; zmiennej ostatniej przypisywane są wszystkie pozostałe słowa.

return [n] %$return%

Wyjscie z funkcji (zob. funkcje) Kod powrotu wyniesie n, lub jesli n nie zostało podane, będzie on równy kodowi powrotu ostatniego wykonanego polecenia.

set [-fntvx- [argument ...]] %$set%

-f
Wyłączenie substytucji nazw plików
-n
Powłoka będzie wczytywać polecenia bez ich wykonywania
-t
Wyjscie z powłoki po wykonaniu jednego polecenia
-v
Wyswietlanie wczytywanych linii
-x
Wyswietlanie poleceń i ich argumentów przed wykonaniem (opcja -x wraz z opcją -n mogą służyć do lokalizacji błędów w skryptach)
-
Oznacza koniec opcji. Użyteczne, gdy chcemy nadać zerowemu parametrowi pozycyjnemu wartosć "-": "set - -".
Argumenty występujące za opcjami zostają przypisane kolejnym parametrom pozycyjnym powłoki - zobacz przykład na końcu tego rozdziału.

Znaczenie opcji można zanegować znakiem "+". Opcje podane wyżej mogą być użyte przy wywołaniu powłoki. Zmienna $- zawiera aktualnie aktywne opcje. Zobacz też "Uruchomienie i Opcje Powłoki" niżej.

shift [n] %$shift%

Parametry pozycyjne zostaną przesunięte o n w lewo, tak, że np. dla n=1: $1=$2, $2=$3, ..., a stara wartość $1 zginie. Domyslna wartość n wynosi 1.

trap [arg] [n] %$trap%

Polecenie arg zostanie wykonane gdy powłoka otrzyma sygnał numer n, zob. pytanie dotyczące umieszczenia nazwy katalogu aktualnego w znaku zachęty jako przykładu zastosowania. Zastosowane polecenia trap obłożone jest licznymi zastrzeżeniami, zob. sh(1).

type [nazwa ...] %$type%

Dla każdej wymienionej nazwy podawane jest jak została by ona zinterpretowana w wypadku użycia jej jako nazwy polecenia.

umask [ooo] %$umask%

Zmiana maski pozwoleń dostępu do tworzonych plików.

Jak wiadomo każdy plik posiada pozwolenia odczytu, zapisu i wykonania dla właściciela, grupy oraz innych użytkowników. Pozwolenia te zapisać można w postaci trzy cyfrowej liczby oktalnej. Kolejne cyfry tej liczby okreslają prawa dostępu dla właściciela pliku (cyfra najstarsza), użytkowników z grupy do której plik należy oraz dla pozostałych - zob chmod(1). Domyslnie pozwolenia dla nowo tworzonego pliku to: 666 dla zwykłego pliku i 777 dla katalogu. Zmienna umask ogranicza prawa dostępu do nowo tworzonego pliku: wartość ustalona poleceniem umask jest odejmowana od pozwoleń domyslnych. Jesli umask=022 to nowo tworzony plik otrzyma pozwolenia (666-022 =) 644 - grupa i inni nie uzyskają praw zapisu i wykonania (przeszukania gdy plik jest katalogiem). Inne dobre wartosci umask to 077 (tylko właściciel ma prawa dostępu), oraz 037 (grupa może czytać).

unset [nazwa ...] %$unset%

Wszystkie podane zmienne i funkcje zostaną usunięte ze srodowiska. Zmiennych PATH, PS1, PS2, MAILCHECK i IFS nie można jednak usunąć.

Uruchomienie i Opcje Powłoki:

Powłoka uruchomiona z zerowym argumentem zaczynającym się znakiem "-" wykonuje polecenia z plików /etc/profile oraz $HOME/.profile przed przystąpieniem do dalszej pracy.

Jesli nie podano opcji -c lub -s pierwszy argument (jesli istnieje) traktowany jest jako nazwa pliku z poleceniami, pozostałe argumenty przekazywane są jako argumenty pozycyjne.

Jesli obecna jest opcja -i, lub jesli standardowe wejscie i wyjscie powłoki związane są z terminalem powłoka będzie pracować interkacyjnie. Oznacza to że:

1) Sygnał TERMINATE będzie ignorowany (kill 0 nie zabije powłoki)
2) Sygnał INTERRUPT będzie przechwytywany i ignorowany
3) Sygnał QUIT będzie ignorowany.

Wywołanie postaci:

sh ... -c ciąg_znaków

Spowoduje zinterpretowanie ciągu znaków jako pojedynczego polecenia.

Wywołanie:

sh ... -s ...

Spowoduje uruchomienie powłoki czytającej polecenia ze standardowego wejscia i przesyłającej komunikaty do standardowej diagnostyki, argumenty występujące za '-s' potraktowane będą jako parametry pozycyjne.

Pozostałe opcje zostały opisane wraz z poleceniem set.

Przykład:

Skrypt wykonywalny uruchomiony z opcjami -x i -n oraz jednym argumentem -x:

```
#!/bin/sh -x -n - -x
# - oznacza koniec opcji, tak że -x będzie już argumentem
#
....
```

Bardziej skomplikowany przykład:

Na zakończenie podamy bardziej rozbudowany przykład dotyczący wielu aspektow programowania w sh:

Dosć często chcemy uzyskać wartosć ostatniego parametru pozycyjnego. Dla powłok zgodnych ze standardem POSIX jest to stosunkowo proste:

```
eval ostatni=\${$#}
```

Kolejne polecenie działa we wszystkich powłokach sh:

```
for ostatni   # ostatni przyjmuje wartosci kolejnych argumentów
do
    :         # polecenie puste
done
```

W następnym przykładzie pokażemy jak uzyskać dostęp do n-tego argumentu, a także jak odwrócić listę argumentów. Jest to też dobry przyklad podsumowujący rożne konstrukcje powłoki sh:

```
t0= u0= cyfry='1 2 3 4 5 6 7 8 9' argv=

for h in '' $cyfry
do
    for t in "$t0" $cyfry
    do
        for u in $u0 $cyfry
        do
            case $# in
            0)
```

```
                    break 3                     # (1)
                esac
                eval argv$h$t$u=\$1             # (2)
                argv="$argv \"\$argv$h$t$u\""   # (3)
                shift                           # (4)
            done
            u0=0
        done
        t0=0
    done

    # Odtwórzmy oryginalne argumenty
    eval set x "$argv"                  # (5)
    shift
```

Podany przykład działa poprawnie dla 999 pierwszych argumentów. Instrukcja (1) zapewnia wyjscie z pętli po przetworzeniu wszystkich argumentów (w przeciwnym przypadku pętle zawsze wykonywały by 999 iteracji!). Polecenie (2) tworzy zmienną argv"xyz" równą parametrowi nr. xyz. (3) zapewnia, że argv będzie zawierać kolejne parametry, i że ewentualne odstępy zostaną poprawnie zacytowne. Argumenty przesuwane są w lewo o jedną pozycję (4). Parametry pozycyjne są odtwarzane w poleceniu (5).

Uzyskanie wartosci n-tego argumentu wygląda teraz następująco:

```
eval argN=\$argv$N
```

Jesli chcemy odwrócić kolejnosć argumentów linię (3) musimy zastąpić linią:

```
argv="\"\$argv$h$t$u\" $argv"
```

Powyższy przykład zawdzięczamy: Martinowi Weitzelowi <@mikros.systemware.de:martin@mwtech.uucp> i Maartenowi Litmaath <maart@nat.vu.nl>

3.POWŁOKI - PYTANIA I ODPOWIEDZI

Jak wykonać skrypt z wnętrza innego skryptu bez tworzenia nowego procesu ?

Często zachodzi potrzeba wykonania poleceń zapisanych w innym pliku niż aktualnie wykonywany skrypt bez powoływania nowego procesu; dobrym przykladem może być zarejestrowanie aliasów lub funkcji przechowywanych w osobnym pliku z plików inicjalizacyjnych ".cshrc" lub ".profile".

Dla wszystkich powłok typu Bournea (sh, ksh, bash, rc, zsh) należy użyć polecenia ".". Dla wszystkich powłok typu csh (csh, tcsh) należy użyć polecenia "source"%$source$%.

Przykłady:

sh:

. ./.funkcje

csh:

source ./.aliases

Jakie są zasady definiowania zmiennych w różnych powłokach ?

%$set%

Zwróćmy uwagę na fakt występowania dwu typów zmiennych: zmiennych "prywatnych" powłoki i zmiennych srodowiskowych - dziedziczonych przez procesy potomne.

Powłoki typu csh używają składni "set zmienna=wartość" dla zmiennych lokalnych i "setenv zmienna wartość" dla zmiennych srodowiskowych (proszę zauważyć brak znaku "=" w drugim przypadku.

Powłoki typu sh używają składni "zmienna=wartość" zarówno dla zmiennych lokalnych, jak i srodowiskowych. Zmienna powinna zostać "eksportowana" jesli chcemy by jej wartość przeszła do srodowiska, czynimy to poleceniem "export ZMIENNA". Zmienne raz zaznaczone jako eksportowane będą przekazywane do srodowisk powoływanych przez powłokę procesów.

W powłoce typu csh pewne zmienne są automatycznie przekazywane do srodowiska. Np. zmiana zmiennej lokalnej "path" spowoduje zmianę zmiennej srodowiskowej PATH.

Nazywanie zmiennych eksportowanych (srodowiskowych) dużymi literami jest wyłącznie konwencją.

Czy powłoka sh posiada aliasy ?

Powłoka ta nie posiadaja polecenia alias typowego dla powłoki csh, ma natomiast zaimplementowany mechanizm funkcji. Funkcje mogą być definiowane i wywoływane zarówno w skryptach jak i interakcyjnie.

Przykład definicji i użycia funkcji w powłoce sh:

$ff ()

```
> {
> find . -name $1 -print
> }
$ ff phile
```

Zdefiniowano funkcję o nazwie ff. Funkcja ta wywołuje program find z argumentem, który interpretowany jest jako nazwa poszukiwanego pliku.

Polecenie "ff phile" rownoważne będzie poleceniu:

```
find . -name phile -print
```

Jak widać parametrów formalnych funkcji używa się podobnie jak parametrów pozycyjnych powłoki.

Powłoki wywodzące się od powłoki sh: "ksh" i "zsh" posiadają zarówno funkcje jak i aliasy.

Jakich plików inicjalizacyjnych (dot files) używając popularne powłoki ?

%$.cshrc% %$.login% %$.profile% %$.logout% %$.history% %$.cshdirs% %$.tcsh% %$.bashrc% %$.inputrc% %$.zshrc% %$.rcrc% %$.zshenv% %$.zprofile% %$.zshrc% %$.zlogin%

Oto lista plików inicjalizacyjnych i pomocniczych większosci używanych obecnie powłok:

csh:

Niektóre wersje używają globalnych .cshrc i .login. Przechowywane są one w różnych katalogach (w zależnosci od wersji).

Wykonywane (w tej kolejnosci):
.cshrc - zawsze.
.login - powłoki główne (login shell).
(BSD: /etc/.login jesli nie istnieje ~/.login)
Przy zakończeniu powłoki:
.logout - powłoki główne.

Inne:
.history - zapisuje historię poleceń (jesli zmienna savehist zdefiniowana).

tcsh:
Wykonywane (w tej kolejnosci):
/etc/csh.cshrc - zawsze.
/etc/csh.login - powłoki główne.
.tcshrc - zawsze.
.cshrc - jesli .tcshrc nie był obecny.
.login - powłoki główne

Przy zakończeniu:
.logout - powłoki główne.

Inne:
.history - zapisuje historię poleceń (jesli $savehist zdefiniowane).
.cshdirs - zapisuje stos katalogów

sh:
Wykonywane (w tej kolejnosci):
/etc/profile - powłoki główne.
.profile - powłoki główne.

Przy zakończeniu:
każde polecenie lub skrypt okreslony uprzednim poleceniem: "trap polecenia 0"

ksh:
Wykonywane (w tej kolejnosci):
/etc/profile - powłoki główne.
.profile - powłoki główne.
$ENV - zawsze, jesli zdefiniowane.

Przy zakończeniu:
każde polecenie lub skrypt okreslony uprzednim poleceniem "trap polecenie 0"

bash:
Wykonywane (w tej kolejnosci):
/etc/profile - powłoki główne.
.bash_profile - powłoki główne.
.profile - powłoki główne jesli ".bash_profile" nie istnieje
.bashrc - powłoki interakcyjne, ale nie główne.
$ENV - zawsze, jesli zdefiniowane

Przy zakończeniu:
.bash_logout - powłoki główne.

Inne:
.inputrc

zsh
Wykonywane (w tej kolejnosci):
.zshenv - zawsze, chyba że zastosowano flagę "-f"
.zprofile - powłoki główne.
.zshrc - powłoki interakcyjne, chyba że zastosowano flagę "-f"
.zlogin - powłoki główne.

Przy zakończeniu:
.zlogout - powłoki główne.

rc
Wykonywane:
.rcrc - powłoki główne

Jak uzyskać nazwę aktualnego katalogu w tekscie zachęty?

%$pushd% %$popd% %$alias% %$chdir% %$trap%

Odpowiedź zależy oczywiscie od powłoki, której używamy. Zacznijmy od trudniejszych przypadków:

C Shell - csh:
Do pliku .cshrc dodajemy:

```
alias setprompt 'set prompt="${cwd}% "'
setprompt          # aby zainicjalizować tekst zachęty
alias cd 'chdir \!* && setprompt'
```

Musimy jeszcze uwzględnić inne metody zmiany katalogu aktualnego przez użytkownika:

```
alias pushd 'pushd \!* && setprompt'
alias popd 'popd \!* && setprompt'
```

Jesli nasza wersja csh nie posiada zmiennej "cwd" możemy użyć "`pwd`".

Możemy też w tekscie zachęty umiescić tylko część nazwy aktualnego katalogu, np. tylko ostatni składnik:

```
alias setprompt 'set prompt="$cwd:t% "'
```

Powłoka Bournea - sh:

Dla większosci wersji programu sh wystarczy wprowadzić funkcję np. ccd, zastępującą standardowe polecenie cd:

```
ccd () { cd $* ; PS1="`pwd` $ "; }
```

Jesli jednak posługujemy się starszą wersją systemu (SVR2, lub starszą) sh nie udostępnia funkcji! Umieszczenie nazwy aktualnego katalogu w tekscie zachęty staje się prawdziwym wyzwaniem, oto przykładowe rozwiązanie:

Do pliku .profile dodajemy:

```
LOGIN_SHELL=$$ export LOGIN_SHELL
CMDFILE=/tmp/cd.$$ export CMDFILE
PROMPTSIG=16 export PROMPTSIG
trap '. $CMDFILE' $PROMPTSIG
```

A oto zawartosć pliku ccd (powinien znaleźć się w jednym z katalogów wymienionych w zmiennej PATH):

```
; ccd - zmień katalog aktualny i tekst zachęty
cat > ${CMDFILE?"ccd: error"} <<EOF
cd $1
PS1="\`pwd\`$ "
EOF
kill -${PROMPTSIG?"ccd: error"} ${LOGIN_SHELL?"ccd: error"}
```

Zanalizujmy ten majstersztyk programowania w shellu:

Skrypt ccd tworzy skrypt o nazwie "cd.$$" w katalogu /tmp. Utworzony skrypt zawiera: polecenie zmiany katalogu oraz zmianę tekstu zachęty. Następnie ccd wysyła do powłoki sygnał (wybrano rzadko używany sygnał SIGURG o numerze 16), na który powłoka reaguje wykonaniem dopiero co utworzonego skryptu "cd.$$".

Korn shell - ksh:

W pliku .profile wystarczy umiescić:

```
PS1='$PWD $ '
```

lub

```
PS1='${PWD##*/} $ '
```

Jesli chcemy uzyskać wyłącznie ostatni składnik aktualnej scieżki.

tcsh (ulepszona csh):

%~	oznacza aktualny katalog, ~ zastępuje $HOME
%d lub %/	pełna scieżka aktualnego katalogu
%d or %.	ostatni składnik aktualnego katalogu

możemy wiec do pliku .cshrc dodać:

set prompt='%~ '

bash czyli "Bourne Shell Again" - jak sama nazwa wskazuje jest to ulepszona wersja sh:

$PS1 zawierające "\w" daje pełną scieżkę aktualnego katalogu skracając $HOME do znaku "~"; "\W" daje pełną sieżkę aktualnego katalogu.

Jak skonstruować wzorzec powłoki (shell glob-pattern) pasujący do wszystkich plików oprócz "." i ".." ?

%$sed%

Nie jest to tak proste, jak by się zdawało na pierwszy rzut oka:

*	nie dopasuje się do plików ukrytych (zaczynających się znakiem ".")
.*	pasuje do "." i ".."
.[^.]*	nie pasuje do "..", ale także do "..a"
.??*	nie pasuje do "." i "..", ale także do ".a"

Wzorzec "* .??*" jest na tyle dobrym przybliżeniem, że wiele osób na nim poprzestaje. Oto rozwiązanie dla purystów wykorzystujące substytucję poleceń w powłoce:

`ls -a | sed -e '/^\.$/d' -e '/^\.\.$/d'`

Program sed usuwa "." i "..", kopiując pozostałe nazwy bez zmian. Zauważmy, że ta substytucja nie poradzi sobie z plikami, których nazwy zawierają znaki nowej linii.

Jak rozdzielić standardowe wyjscie (stdout) i standardową diagnostykę (stderr) procesu?

W sh jest to banalne:

program > stdout_file 2> stderr_file

W csh nieco trudniejsze gdyż >& przełącza stderr i stdout!

sh -c 'program > stdout_file 2> stderr_file'

Powyższy przykład sprowadza problem do rozwiązania dla powłoki sh. Można jednak postąpić inaczej: stdandardowe wyjscie przełączamy w pod-powłoce, a standardową diagnostykę (i stdout, ale ono już zostało przełączone) w głównej powłoce:

(program >stdout_file) >& stderr_file

Jak proces potomny (program, skrypt) może wpłynąć na srodowisko, lub katalog aktualny procesu macierzystego?

%$source%

Problem ten pojawia się zwłaszcza przy pisaniu złożonych skryptów, wywołujących inne skrypty lub programy.

Proces potomny posiada własne srodowisko i własny katalog aktualny, nie jest więc możliwe bezposrednie odziaływanie na proces macierzysty. Proces potomny może zmieniać swoje srodowisko i katalog aktualny, nie ma to jednak żadnego wpływu na proces macierzysty posiadający również własne srodowisko i własny katalog aktualny.

Jesli chcemy by proces potomny zmienił srodowisko procesu macierzystego, musi odbyć się to za zgodą tego ostatniego. Np. proces potomny może zapisać pewne zmienne srodowiskowe do pliku, zas proces macierzysty odpowiednio ten plik zinterpretować.

Często lepszym rozwiązaniem jest wykonanie skryptu bez powoływania nowego procesu. W powłoce Bournea i Korna możemy użyć polecenia ".":

```
. jakis_skrypt
```

W powłoce C polecenia source:

```
source jakis_skrypt
```

Przed pisaniem skryptów dobrze uswiadomić sobie jakie polecenia wykonywane będą w pod-powłoce, unikniemy w ten sposób licznych nieprzyjemnych błędów. Zwróćmy też uwagę na to, że w starszych wersjach sh funkcje wykonują się w pod-powłokach. Niemożliwe jest zatem napisanie funkcji zmieniającej np. katalog aktualny; zobacz też następne pytania.

Jak w skrypcie wczytać znaki z terminala ?

%$read% %$stty%

W powłoce sh możemy użyć polecenia read. Oto przykład pętli wczytującej i przetwarzającej znaki wprowadzone z terminala:

```
while read line
do
        ...
done
```

W csh używamy specjalnego symbolu "$<"

```
while ( ... )
set line = "$<"
if ( "$line" == "") break
...
end
```

Możemy też wczytać jeden znak - skrypt dla powłoki sh:

```
echo -n "Wprowadż znak"
stty cbreak
znak='dd if=/dev/tty bs=1 count=1 2>/dev/null'
stty -cbreak
echo "Nacisnąles $znak"
```

Krótkie objasnienie: terminal wprowadzamy w tryb "surowy" (raw) poleceniem stty(1) z opcją cbreak ("stty raw" w niektórych systemach). Wczytujemy jeden znak bezposrednio z terminala naszego procesu - polecenie dd(1) (if - input file), na wszelki wypadek przełączamy standardową diagnostykę do "/dev/null". Ponownie wprowadzamy terminal w zwykly tryb pracy, i drukujemy wprowadzony znak.

Jak z poziomu .cshrc okreslić czy jest to powłoka główna (login shell) ?

%$setenv%

Zadając to pytanie możemy mieć na mysli trzy różne rzeczy:
1) czy csh wykona polecenia z pliku .login po zakończeniu interpretacji pliku .cshrc ?
2) czy jest to powłoka interakcyjna (a nie np. uruchomiona przez rsh) ?
3) czy jest to pierwsza (tzn. nie wywolana przez inną) powłoka użytkownika w systemie ?

W przypadku 1) możemy na podstawie pseudo zmiennej $$ i wyniku polecenia ps zbadać czy nasza powłoka została wywołana ze znakiem "-" rozpoczynającym nazwę, jesli tak to polecenia z pliku .login zostaną wykonane.

Przedstawiony niżej szkieletowy plik .cshrc pozwala wyodrębnić sytuacje z punktów 2 i 3.

```
#
# Wprowadzamy pomocniczą zmienną CSHLEVEL
#
if (! $?CSHLEVEL) then
#
# jestesmy w powłoce nr. 0, a więc uruchominej przez
# login lub rsh
# Ustawimy srodowisko dla tej i następnych powłok:
#
        setenv CSHLEVEL 0
        set ...                 # wg. gustu
        source ~/.env           # srodowisko
else
#
# Jestesmy w powłoce potomnej, nie warto więc ponownie
# inicjalizować srodowiska
#
        set tmp = $CSHLEVEL
        @ tmp++
        setenv CSHLEVEL $tmp
endif
#
# Jesli $prompt pusty to powłoka jest nieinterakcyjna:
#
if (! $?prompt) exit
#
# Tu zainicjalizujemy to co potrzebuje powłoka interakcyjna:
#
source ~/.aliases  # aliasy warto trzymać osobno
set prompt= ...         # własny prompt wg. gustu
...                     # inne polecenia
```

Jak okreslić czy wykonywana powłoka jest interakcyjna?

%$case%

Dla csh zobacz poprzedni punkt, w którym szczegółowo omówiono ten i pokrewne zagadnienia. W powłokach typu sh można badać zmienne specjalne $PS1, lub (lepiej) $-. Jesli ta ostatnia ma wartosć 'i' to powłoka jest interakcyjna. Oto przykładowy skrypt:

```
case $- in
*i*)    # Interakcyjna
        ;;
*)      # Nieinterakcyjna
        ;;
esac
```

W rozdziale poswięconym procesom Czytelnik znajdzie odpowiedź na pytanie jak arbitralny proces lub skrypt może okreslić czy wykonywany jest interakcyjnie.

Jak przełączyć standardowe wejscie (wyjscie) w pętlach powłoki sh ?

%$while% %$exec% %$wejscie/wyjscie w powłoce sh%
Rozpatrzmy poniższy przyklad:

```
foo=klops
#
while read line
do
        # cos robimy z line ...
        foo=dobrze
done < /etc/passwd
#
echo "foo wynosi: $foo"
```

W wielu implementacjach systemu, otrzymamy komunikat: "foo wynosi: klops". Powodem jest nieudokumentowany mechanizm sh: przełączenie standardowego wejscia/wyjscia dla poleceń złożonych (while, if, etc.) powoduje wykonanie tego fragmentu skryptu w pod-powłoce. Jak było powiedziane uprzednio - zmiany srodowiska w procesie potomnym nie mają wpływu na srodowisko procesu macierzystego.

Nowsze wersje sh powinny jednak radzić sobie ze skryptem podobnym do powyższego bez tworzenia pod-powłoki. Jest to ustalone przez standard POSIX 1003.2 (Powłoki i Narzędzia).

Następny skrypt funkcjonuje poprawnie z powłokami "historycznymi" jak i ze zgodnymi ze standardem 1003.2:

```
#
foo=klops
#
# Deskryptor nr. 9 jest duplikatem deskryptora nr. 0 (czyli stdin)
# stdin podłączymy do pliku /etc/passwd
# Po wykonaniu pętli ponownie przełączymy stdin na deskryptor nr 0
# zobacz też dup(2)
#
# exec pozwala na samo przełączenie, i nie jest wykonywany
```

```
# w pod-powłoce
#
exec 9<&0 < /etc/passwd
#
while read line
do
        # cos robimy z line ...
        foo=dobrze
done
#
# Wracamy do stanu z przed przełączenia, oraz zamykamy
# deskryptor nr.9
#
exec 0<&9 9<&-
echo "foo wynosi: $foo"
```

Ten skrypt nigdy nie wydrukuje "foo wynosi: klops" !

Dlaczego wartosci parametrów zmienianych w potoku nie są poprawnie zapamiętywane ?

Skrypt powłoki sh:

```
foo=klops
echo "hau! hau!" | read foo
echo "foo wynosi: $foo"
```

Wypisze "foo wynosi: klops" w niektórych systemach a "foo wynosi hau! hau!" w innych. Dlaczego? Każdy składnik potoku wykonywany jest we własnej pod-powłoce, jednak w niektórych implementacjach sh ostatni składnik potoku jest wyjątkiem: jesli jest to polecenie wbudowane (takie jak read) pod-powłoka nie jest potrzebna, a polecenie może być wykonane przez główną powłokę. Standard POSIX 1003.2 nie wypowiada się jednoznacznie, dopuszczając obydwa rozwiązania.

Zobacz poprzedni punkt oraz opis powłoki sh.

Jak wykonywać w tle takie programy jak ftp, passwd, telnet,... ?

%$ftp% %$passwd% %$telnet% %$expect%
Dokładnie: czy możliwa jest automatyczna, przeprowadzana przez skrypt powłoki, konwersacja z tego typu programami?

Nie. Powłoka nie potrafi porozumieć się z interakcyjnym programem używającym terminala bezposrednio. Istnieją jednak programy pomocnicze - "opakowania", przy pomocy których możemy osiągnąć cel. Oto przykładowy skrypt w którym powłoka porozumiewa się z programem "passwd" za posrednictwem pakietu "expect":

```
set password [index $argv 2]
spawn passwd [index $argv 1]
expect "*password:"
send "$password\r"
expect "*password:"
send "$password\r"
```

expect eof

Pakiet expect nie jest częścią systemu, można go otrzymać z archiwum durer.cme.nist.gov, plik pub/expect.shar.Z.

Pakiet "pty" (dostępny w katalogu pub/flat/pty-* w archiwum stealth.acf.nyu.edu) spełnia podobną rolę, współpracując z programem interakcyjnym za posrednictwem pseudo-terminala, z którym z kolei może się już porozumieć skrypt.

Dlaczego umieszczenie nazwy katalogu aktualnego "." w zmiennej PATH może być niebezpieczne?

%$PATH%

Możliwość "automatycznego" wykonywania programów z aktualnego katalogu jest z całą pewnoscią wygodna dla użytkownika: nie musi on pisać "./prog" - wystarczy "prog". Istnieją jednak dwa dobre powody by nie umieszczać nazwy katalogu aktualnego w PATH:

Jesli "." jest pierwszą nazwą katalogu w PATH ryzykujemy wykonaniem niewłasciwych programów. Załóżmy, że "/tmp" jest katalogiem aktualnym, wydajemy polecenie "ls", jesli "/tmp" zawiera czyjs program/skrypt o nazwie "ls" zostanie on wykonany zamiast programu systemowego! W najgorszym przypadku "/tmp/ls" może być programem złosliwie podłożonym (tzw. "kretem"), którego celem jest zniszczenie plików, lub uzyskanie dla włamywacza naszych praw dostępu.

Znacznie częściej natrafimy na niespodziewane problemy wynikające z kolizji nazw naszych własnych programów z programami systemowymi. Klasycznym przykładem jest program test (kto nie nazwał tak kiedys swojego programu?).

Nieco lepszym rozwiązaniem jest umieszczenie "." na końcu zmiennej PATH, np. w ten sposób jesli używamy sh:

PATH=$PATH:.

Dzięki takiemu rozwiązaniu unikamy kolizji z nazwami programów systemowych, redukujemy też prawdopodobieństwo uruchomienia kreta. Załóżmy jednak, że program włamywacza nosi nazwę "sl" - kiedys ktos pracując w katalogu /tmp napisze niechcący "sl" zamiast "ls" - na to własnie liczy autor kreta! Podany przykład nie brzmi może zbyt wiarygodnie, stanowi jednak klasyczny przykład zdobywania dostępu do systemu przez osoby niepowołane. Dodajmy, że bardzo wyrafinowani włamywacze również, jednak w nieco inny sposób, korzystają z niefrasobliwosci osób umieszczających "." w zmiennej PATH, szczegóły przemilczymy.

Podsumujmy ten punkt Dobrą Radą: najlepiej nie umieszczać nazwy katalogu aktualnego w zmiennej PATH. Doswiadczony użytkownik swietnie sobie bez tego ułatwienia radzi.

Dlaczego powłoka czasami nie może uruchomić programu znajdującego się w katalogu wymienionym w zmiennej PATH?

%$PATH%

Odpowiedź dla sh: Powłoka pamięta nazwy katalogów zawierających uprzednio wykonywane programy. Jesli w katalogu wymienionym w zmiennej PATH umiescimy nowy program lub skrypt (np. kopiując go z innego katalogu, lub kompilując pliki źródłowe) i nazwa tego programu będzie identyczna z nazwą programu, którego położenie powłoka zapamiętała, wykonany zostanie stary program. Zobacz polecenia "type" i "hash" w rozdziale poswięconym powłoce sh.

Odpowiedź dla powłoki csh: W celu przyspieszenia dostępu do plików powłoka pamięta położenie plików wykonywalnych w wewnętrznej strukturze danych. Struktura ta jest uaktualniana po uruchomieniu powłoki, po zmianie zmiennej PATH, oraz po wydaniu polecenia "rehash". Jesli więc umiescimy plik wykonywalny w któryms z katalogów wymienionych w zmiennej PATH powłoka nie zauważy jego istnienia póki nie zajdzie jedna z wymienionych wyżej sytuacji.

4.PRZENOSZENIE PLIKÓW MIĘDZY SYSTEMAMI.

W rozdziale tym omawiamy problem praktyczy, przed którym staje wielu użytkowników UNIX-a: przenoszenie plików między systemami. Jest to dosć szerokie zagadnienie obejmujące zasady korzystania z wymienialnych nosników danych takich jak tasmy i dyskietki, współpracę z programami archiwizującymi, a przedewszystkim tematykę sieciową. Rozdział niniejszy podzielony jest na dwie częsci. W pierwszej omówiono współpracę systemu z nosnikami wymienialnymi oraz programy archiwizujące. Część druga w całosci poswięcona jest problematyce sieciowej. Ta ostatnia potraktowana została dosć szeroko. Omówiono zarówno usługi o charakterze lokalnym: NFS, jak i typowe dla sieci rozległych: UUCP i FTP.

Osobnym zagadnieniem jest kompresja plików (także archiwalnych) oraz zmiana formatu plików tekstowych przy przenoszeniu plików między systemem UNIX a systemem MS-DOS. Problemy te zostały omównione w rozdziale poswięconym plikom.

Gdy jeden z systemów nie jest podłączonych do sieci:

Musimy użyć fizycznego nosnika danych: tasmy lub deskietki.

Nosnikami egzotyczymi, takimi jak wymienne dyski optyczne, lub wymienne dyski twarde nie będziemy się zajmować.

)Pliki Archiwalne

Archiwum to pojedynczy plik, na który składają się zawartosci innych plików. Archiwum nie jest po prostu zbiorem wielu plików - znajdują się w nim niezbędne informacje pomocnicze w postaci tzw. nagłówków. Nagłówek pliku zawarty w archiwum zawiera informacje takie jak: nazwa i scieżka składowanego pliku, jego typ, pozwolenia dostępu, identyfikator własciciela itp. Korzysci płynące ze skomasowania wielu plików w jednym są dosć oczywiste: zyskujemy na łatwosci i czasie przenoszenia, składowania i zarządzania (zauważmy że jeden duży plik kopiuje się szybciej niż wiele małych!).

Podkreslmy, że w przeciwieństwie do popularnych pakietów archiwizujących z otoczenia systemu MS-DOS archiwizatory UNIX-owe NIE dokonują kompresji danych. Jest jednak możliwe poddanie uprzednio sporządzonego archiwum kompresji przy pomocy specjalnie do tego celu przeznaczonych programów.

Archiwa w formacie tar(5)

%$tar%

Do współpracy z tasmami służy polecenia tar(1) (Tape ARchive). Poleceniem tar można tworzyć archiwa na dowolnym nosniku: tasmie streamera 1/4", tasmie 8mm, a także archiwa będące zwykłymi plikami i znajdujące się w standardowym systemie plików.

Tar jest obecnie najpopularniejszym programem archiwizującym (zob. pax i ustar dalej). Format archiwów tar(5) został uznany za standard POSIX.

Uproszczona składnia:

tar -c [bBefFhivw]
[urządzenie] [rozmiar_bloku]
nazwa_pliku ... -C directory nazwa_pliku

tar -t [BefFhiv]
[urządzenie] [nazwa_pliku ...]

tar -u [bBefFhivw]
[urządzenie] [rozmiar_bloku] nazwa_pliku ...

tar -x [BefFhilmopvw]
[urządzenie] [nazwa_pliku ...]

Pierwsza opcja oznacza typ operacji na archiwum:

-c Tworzenie archiwum
-t Wypisanie zawartosci archiwum w formacie podobnym do "ls -l"
-u Uaktualnienie archiwum, zapisane zostaną pliki nieobecne, lub modyfikowane po utworzeniu archiwum
-x Ekstrakcja z archiwum. Odtwarzane są wszystkie, lub wyspecyfikowane pliki.

W wielu odmianach systemu znak minus poprzedzający główną opcję jest opcjonalny lub wręcz zbędny.

Specyficzne opcje polecenia tar dotyczą współpracy z tasmami, a więc parametrów takich jak gęstosć zapisu itp., zwykle nie trzeba ich używać. Oto przykład utworzenia archiwum:

tar -cvf /dev/rst1 /home/lukrecja/gjb

Archiwum tworzone jest na urządzeniu "/dev/rst1" - streamerze połączonym do szyny SCSI w komputerze pracującym pod kontrolą systemu SunOs. Pierwszy parametr "c" okresla typ operacji. Następne dwie opcje nakazują: "v" (verbose) - podawanie nazw archiwizowanych plików, "f" - następny argument jest nazwą pliku/urządzenia na którym archiwum ma być umieszczone.

Następny przykład pokazuje jak "rozpakować" archiwum:

tar -xvf /dev/rst1

Opcja "x" - (extract) spowoduje odtworzenie wszystkich plików znajdujących się w archiwum. Archiwum oczywiscie nie zostaje zmodyfikowane.

Nazwy plików w archiwum mogą być zapisane w postaci pełnej scieżki dostępu - jak w pierwszym przykładzie, jak i w postaci względnej:

cd ~ ; tar -cvf /dev/rmt0 .

Polecenie to składuje pliki z katalogu głównego użytkownika na tasmie rmt0. Pliki będą nazwane względem katalogu "~".

Wydruk nazw (i ewentualnie atrybutów) plików zawartych w archiwum możemy obejrzeć stosując opcję "t" - table of contents:

tar -tvf /dev/rmt0

Tworzenie archiwum zawierającego dwa pliki: "foo" i "bar" z katalogu aktualnego:

tar -cvf /dev/rmt0 ./foo ./bar

Polecenie z kolejnego przykładu wykona dokladnie to samo co pierwsze polecenie, z jedną różnicą - tasma nie zostanie przewinięta, gdyż odwołujemy się do "urządzenia nieprzewijającego" "/dev/nrst1":

tar -cvf /dev/nrst1 /home/lukrecja/gjb

Możemy teraz zapisywać kolejne archiwum, które znajdzie się bezposrednio za ostatnio utworzonym.

#) Polecenie mt(1)

%$mt%

Jak poruszać się po tasmie zawierającej wiele archiwów, umieszczonych jedno za drugim i odzielonych znacznikami końca pliku (EOF) ?

Służy do tego polecenie mt(1) (Magnetic Tape). Jego składnia:

mt [-f nazwa_jednostki_tasm] polecenie [licznik]

Niektóre polecenia akceptowane przez mt:

fsf	- przewiń do przodu przez "licznik" znaków EOF
rewind	- przewiń tasmę
eom	- przewiń do przodu do końca zapisanej tasmy
erase	- skasuj zawartosć tasmy

Tak więc ciąg poleceń:

mt -f /dev/nrst1 rewind; mt -f /dev/nrst1 fsf 2

Przewinie tasmę za drugie archiwum (o ile oczywiscie na tasmie znajdują się dwa archiwa utworzone poleceniem tar). Możemy teraz odczytać trzecie archiwum (o ile istnieje), lub je zapisać.

Jesli chcemy odczytać kilka kolejnych archiwow używamy urządzenia nieprzewijającego jako źródła danych polecenia tar.

Zwróćmy uwagę, że zapisanie n-tego archiwum niszczy wszystkie następne, nawet jesli nie zostały one fizycznie zamazane!

#) Więcej o opcjach polecenia tar.

Oto niektóre, bardziej zaawansowane możliwosci programu tar:

Istnieje możliwosć archiwizowania plików znajdujących się w różnych katalogach bez archiwizowania całego wspólnego dla nich pod-drzewa katalogów. Służy temu opcja:

-C katalog plik

Przykład:

tar -cf /dev/rmt0 -C /usr/include stdio.h /home/gjb .

Na tasmie rmt0 umieszczone zostanie archiwum zawierające plik "stdio.h" z katalogu "/usr/include" oraz katalog "/home/gjb" (oczywiscie wraz z pod-katalogami).

b rozmiar_bloku

Okresla ilosć rekordów tworzących blok. Użycie większych bloków przyspiesza transmisję.

Użycie tej opcji jest niezbędne przy tworzeniu archiwum na tasmie, niepotrzebne lub nawet szkodliwe w innych wypadkach.

Przykład:

tar cvfb /dev/rst2 20 ...

Użyto dwudziesto rekordowych bloków.

B *wielokrotne odczyty*

Opcja to nakazuje programowi tar wczytywanie pełnych bloków, nawet jesli będzie to wymagało wielokrotnych odczytów. Opcję "B" stosuje się gdy tar nie jest w stanie odczytać "na raz" pełnego bloku.
Sytuacja taka ma miejsce przy odczycie archiwum poprzez sieć z odległego komputera - usługi sieciowe nie zapewniają jednorazowych i niepodzielnych transferów dużych bloków danych. Podobnie, gdy tar pobiera lub przesyła archiwum z/do potoku konieczne jest zastosowanie opcji B.

Oto przykład ekstrakcji danych z tasmy znajdującej się na odległym komputerze "lukrecja":

rsh -n lukrecja dd if=/dev/rst2 bs=20b | tar -xvBfb - 20 pliki

Na odległej maszynie uruchmiono proces blokowego transferu danych (zob. opisy programów rsh i dd dalej). Odległym urządzeniem jest "/dev/rst2", wspólczynnik blokowania wynosi 20. Na lokalnym komputerze tar przeprowadza ekstrakcję z archiwum - "x"; podając informacje o znalezionych plikach - "v"; tar poinformowano o tym, że dane wejsciowe pochodzić będą z potoku - "B"; plikiem wejsciowym jest standardowe wejscie - "f -", współczynnik blokowania wynosi dwadziescia - "b 20".

Ostatni przykład pokazuje jak składować dane z maszyny nie zaopatrzonej w stację tasm lub dyskietek a podłączonej do sieci.

m *dot. czasu modyfikacji pliku*

Czasy ostatniej modyfikacji plików nie będą odtworzone z archiwum -czas modyfikacji zostanie zmieniony na czas ekstrakcji pliku. Ta opcja może być użyta tylko z poleceniem ekstrakcji.

o *dot. identyfikatora wlasciciela pliku*

Identyfikator właściciela pliku nie będzie odtworzony z archiwum - zostanie zmieniony na identyfikator użytkownika dokonującego ekstrakcji. Ta opcja może być użyta tylko z poleceniem ekstrakcji.

p *zachowaj oryginalne pozwolenia*

Pliki odtwarzane z archiwum zachowają archiwalne pozwolenia, pozwolenia odtwarzanych plików NIE będą zależeć od aktualnej wartosci zmiennej "umask". Tylko dla super-użytkownika opcja ta będzie honorowana w odniesieniu do wszystkich plików.

) Archiwa w formacie cpio(5).

%$cpio%

Polecenie cpio(1) podobnie jak tar służy do tworzenia archiwów. Archiwa mogą być umieszczone na tasmie, dyskietkach, mogą mieć też postać zwykłego pliku. Cpio jest nieco przestarzałym programem, opiszemy go jednak ponieważ w niektórych wersjach systemu jest to jedyne polecenie tego typu.

Składnia:

Tworzenie archiwum:

cpio -o [acv] < NazwaPliku > NazwaArchiwum

Odtworzenie plików z archiwum:

cpio -i [bmvdrts] [Wzorzec ...] < Wejscie

Przy tworzeniu pliku archiwalnego plik wejsciowy zawiera listę plików i katalogów do zarchiwizowania. Archiwum wysyłane jest do standardowego wyjscia.

Opcje przy wywołaniu "cpio -o":

a	modyfikuje czas dostępu do pliku archiwizowanego
c	nagłówek archiwum jest znakowy
v	wyswietlana jest lista archiwizowanych plików

Przy rozpakowywaniu archiwum wprowadzane jest ze standardowego wejscia; tworzone są pliki, których nazwy pasują do wzorca. Wzorzec ma podobną postać jak wzorce nazw plików z powłok sh i csh (meta znaki "?","*","[","]" itp), z tą rożnicą, że meta-znaki pasują także do znaku "/".

Opcje przy wywołaniu "cpio -i":

b	zamienia miejscami bajty w pół-słowach i pół-słowa w słowach
d	tworzy katalogi jesli jest to potrzebne
r	pozwala na zmianę nazwy pliku
t	wyłącznie wyswietla listę nazw plików znajdujących się w archiwum
m	Zachowuje poprzednie czasy modyfikacji plików
s	zamienia miejscami bajty
v	wyswietla nazwy oraz atrybuty przetwarzanych plików

Przykłady:

1) Utworzenie archiwum na dyskietce (tu: "/dev/rfd0"), nazwy składowanych plików zapisano w pliku "lista-plików".

cpio -ov < lista-plików > /dev/rfd0

Dyskietka powinna być sformatowana i niezamontowana.

2) Składowanie plików z rozszerzeniem ".trm", z aktualnego katalogu do pliku "/tmp/trm.files":

ls *.trm | cpio -ov > /tmp/trm.files

3) Składowanie aktualnego katalogu, wraz ze wszystkimi jego pod-katalogami na dyskietkę:

find . -print | cpio -ov > /dev/rfd0

4) Odtworzenie zawartosci archiwum, z dyskietki:

cpio -idmv </dev/rfd0

Opcja "d" powoduje, że odpowiednie katalogi są tworzone, jesli jest to potrzebne; opcja m zachowuje ostatni czas modyfikacji plików zapisanych w archiwum. Nazwy plików będą wyswietlane w czasie ekstrakcji - flaga "v".

5) Odtworzenie wybranych plików z dyskietki:

cpio -i "*.c" "*.o" </dev/rfd0

Odtworzone będą tylko pliki z rozszerzeniami "*.c" i ".o". Cudzysłowów użyto by zapobiec interpretacji meta-znaków "*" przez powłokę.

6) Wypisanie zawartosci pliku w formacie cpio znajdującego się na dyskietce:

cpio -itv </dev/rfd0

Zawartość zostanie wypisana w postaci podobnej do wydruku zawartosci katalogu poleceniem "ls -l".

Skrócony wydruk zawartosci archiwum:

cpio -it </dev/rfd0

#) Archiwa w formacie bar(5)

%$bar%

Do przenoszenia stosunkowo małych ilosci danych (nie więcej niż kilkunastu megabajtów) najlepiej nadają się dyskietki. Dyskietek możemy używać w dwu zasadniczo odrębnych reżimach: archiwum blokowego lub systemu plików.

Archiwum blokowe tworzymy i odczytujemy poleceniem bar(1) (Block ARchive). Polecenie to tworzy archiwa mogące zawierać zarówno pojedyncze pliki jak i hierarchie katalogów. Archiwum przechowuje także informacje o typie, pozwoleniach dostępu, właścicielu i datach utworzenia/modyfikacji/dostępu zawartych w nim plików. Plik archiwalny może być tworzony na dowolnym zapisywalnym nosniku (dyskietce, tasmie streamera, tasmie 8mm itp.), może to też być zwykły plik.

Polecenia bar nadaje się jednak szczególnie do składowania plików na dyskietkach. Pliki w formacie bar(5) są przenosne pomiędzy różnymi wersjami systemu. Istnieje także możliwość tworzenia i odczytu archiwów na komputerach pracujących pod kontrolą innego systemu operacyjnego (np. MS-DOS). Archiwum może zawierać pliki nietypowe: np. typu b lub c, nie można jednak zapisać do niego pewnych typów plików specjalnych np. plików typu "named pipe" (FIFO).

Duże pliki formatu bar mogą być automatycznie umieszczane na kilku fizycznych dyskietkach, nic nie stoi więc na przeszkodzie tworzenia b. dużych archiwów (za wyjątkiem czasu tworzenia i ekstrakcji).

Składnia polecenia bar jest dosć podobna do uprzednio przedstawionej składni polecenia tar(1) nie będziemy jej tu prezentować w całosci - pokażemy jednak kilka typowych zastosowań.

Przykłady:

Tworzenie archiwum wszystkich plików, wraz z pod-katalogami katalogu /home/lukrecja/gjb, archiwum na dyskietce (urządzenie /dev/fd0a):

bar -cvf /dev/fd0a /home/lukrecja/gjb

Opcja "c" - create - polecenie tworzenia archiwum, "v" - verbose - Podawane będą informacje o nazwach i rozmiarze archiwizowanych plików, "f" - nazwa pliku - tu: urządzenia na którym tworzone będzie archiwum. Uwaga: W pewnych wersjach programu bar nie umieszcza się znaku "-" przed głównymi opcjami.

Odczytanie archiwum z dyskietki:

bar -xvf /dev/fd0a /home/lukrecja/gjb

Opcja "x" - extract.

Nazwy plików w archiwum mogą być zapisane w postaci pełnej scieżki dostępu - jak w poprzednim przykładzie, jak i w postaci względnej - zob. polecenie tar.

Zawartosć archiwum możemy obejrzeć stosując opcję "t" - table of contents:

bar -tvf /dev/fd0a

Ostatni przykład, tworzenie archiwum zawierającego dwa pliki: "foo" i "bar" z katalogu aktualnego:

bar -cvf /dev/fd0a ./foo ./bar

Archiwum zaczyna się zwykle od pierwszego bloku dyskietki. Należy więc pamiętać, że jej poprzednia zawartosć z dużym prawdopodobieństwem zostanie całkowicie i nieodwracalnie przez nas zniszczona!

Formaty archiwów bar(5) i tar(5) są dosć podobne, NIE są jednak jednakowe. Archiwum zapisane poleceniem bar nie może być odtworzone przy pomocy polecenia tar i odwrotnie.

#) Systemy plików na dyskietce

Wiele wersji UNIX-a dopuszcza istnienie systemu plików na dyskietce. Często może to być zarówno system rodzimy tzn. UNIX-owy jak i popularny system plików systemu MS-DOS-u.

Unix-owy system plików na dyskietce tworzymy poleceniem mkfs(1) lub newfs(1). Dyskietka musi być zaformatowana. Program formatujący dyskietkę nazywa się najczęsciej fdformat (System V).

Dyskietka z systemem plików musi być przed użyciem zamontowana, jest to czynnosć w zasadzie zarezerwowana dla super-użytkownika, jednak w większosci systemów istnieją odpowiednie skrypty dostępne dla użytkowników nieuprzywilejowanych.

Po zamontowaniu, do plików na dyskietce odwołujemy się w standardowy sposób. Dyskietka jest zwykle zakotwiczona w ustalonym przez administratora katalogu np. "/pcfs" (dyskietka MS-DOS-owa w SunOs) lub "/floppy" (System V). Należy oczywiscie pamiętać o ograniczeniach narzuconych na częsć systemu plików reprezentowanego przez dyskietkę DOS-ową - np. o ograniczeniu długosci nazw plików.

Istnieją też odmiany UNIX-a (np. starsze wersje SunOs, HP-UX), w których z dyskietek zapisanych w formacie MS-DOS korzystać można wyłącznie przy pomocy odpowiednich programów, będących odpowiednikami poleceń DOS-u.

Na zakończenie warto wspomnieć, że możliwosci wykorzystania dyskietek zapisanych w formacie MS-DOS jest ograniczona. Można zwykle używać tylko pewnych formatów (np. tylko 1.4MB na dyskietce 3.5"). System może nie tolerować błędnych sektorów, nawet jesli zostały one porawnie oznaczone - w skrajnych przypadkach (które zdarzają się zaskakująco często) może dojsć do załamania się lub zawieszenia systemu.

) Inne programy archiwizujące - ustar(1) i pax(1).

%$ustar% %$pax%

Zasygnalizujmy jeszcze istnienie dwu innych, mniej popularnych programów archiwizujących. Programy pax (Portable archive eXchange) i ustar (USENIX tar) są zwykle dostarczane wraz z UNIX-em System V, oraz z systemami oferującymi zgodnosć na poziomie programów usługowych z Systemem V (np SunOs). Programy te służą do tworzenia, odczytu i manipulacji archiwów zapisanych w standardzie zgodnym ze standardem IEEE 1003.1-1988 (POSIX.1) -

czyli po prostu tar(5). Pax obsługuje również archiwa w formacie cpio(5).

Oba komputery podłączone są do sieci i zapewniają protokoły TCP/IP i NFS.

#) Kopiowanie z wykorzystaniem NFS (Network File Service).

Najszybszą i najprostrzą w użyciu metodę możemy zastosować gdy komputer, na który chcemy przenieść dane eksportuje swoje katalogi, a komputer "źródłowy" montuje je za posrednictwem NFS.

Załóżmy, że komputer lukrecja eksportuje katalog "/home", zawierający podkatag użytkownika gjb. Komputer z plikami do przeniesienia: falstaf montuje ten katalog jako "/home/lukrecja". Przeniesienie pliku "foo" z katalogu aktualnego na falstafie odbywa się tak samo jak na systemie lokalnym:

cp ./foo /home/lukrecja/gjb

Plik foo trafia do katalogu "/home/gjb" na komputerze lukrecja.

Informację o zamontowanych lokalnych i odległych systemach plików daje polecenie df.

#) Polecenie rcp(1).

%$rcp%

Jesli odpowiednie katalogi nie są eksportowane, lub nie zastały zamontowane możemy posłużyć się poleceniem rcp:

rcp ./foo lukrecja:/home/gjb

W drugą strone: z lukrecji do falstafa:

rcp lukrecja:/home/gjb/foo .

Rcp ma skladnię podobną do cp(1):

rcp [-p] [-r] plik-źródłowy plik-docelowy
rcp [-] [r] plik-źródłowy [...] katalog-docelowy

Plik okresla się nie tylko poprzez scieżkę, ale i poprzez nazwę komputera na którym się on znajduje.

Załóżmy, że pracujemy na lukrecji i chcemy przenieść wszystkie pliki z katalogu / home/gjb/Doc do katalogu /tmp na komputerze falstaf:

rcp -r '/home/gjb/Doc/*' falstaf:/tmp

Apostrofy zapobiegną niepotrzebnemu tu rozwinięciu wzorca "*", opcja -r oznacza "przenies wraz z pod-katalogami".

Przykład na tzw. "third party copies". Pracujemy na komputerze oberon, i wykonujemy tę samą operację co w ostatnim przykładzie:

rcp -r 'lukrecja:/home/gjb/Doc/*' falstaf:/tmp

Można też wykonywać operacje "na konto" innego użytkownika:

rcp 'kazio@lukrecja:.profile' falstaf:/tmp/profile-kazia

Skopiuje plik ".profile" z katalogu głównego użytkownika "kazio" na maszynie "lukrecja" do katalogu "/tmp" na komputerze falastaf.

Kopiowanie z podaniem nazwy użytkownika oznacza nie tylko interpretowanie scieżek relatywnie do katalogu głównego danego użytkownika, oznacza to też sprawdzanie praw dostępu nie dla użytkownika lokalnego, a dla podanego użytkownika.

Reasumując, plik dla polecenia rcp może być okreslony na trzy sposoby:

nazwa-komputera:scieżka
nazwa-użytkownika@nazwa-komputera:scieżka
nazwa-użytkownika@nazwa-komputera.domena:scieżka

W ostatnim przypadku występuje pełen adres internetowy.

UWAGA: Nie należy używać polecenia rcp do kopiowania pliku "na samego siebie", może to spowodować zniszczenie jego zawartosci.

Polecenie rcp może się zakończyć niepowodzeniem gdy oddalony system odmówi współpracy. Zdażyć się to może w dwu przypadkach: Po pierwsze, tak jak przy zwyklym kopiowaniu, gdy użytkownik nie ma praw odczytu/zapisu do katalogu do którego chce uzyskać dostęp. Po drugie użytkownik wywołujący rcp musi posiadać zezwolenia na zdalny dostęp. Zezwolenia te okresla zawartosć pewnych globalnych plików administracyjnych (być może innych na każdej maszynie), a także lokalnego pliku "~/.rhosts" każdego użytkownika. Plik "~/.rhosts" może zakazać lub zezwolić na dostępu okreslonym użytkownikom z okreslonych komputerów, wszystkim użytkownikom z pewnych komputerów itd.

Uwaga: Przypominamy, że "~" oznacza katalog główny użytkownika.

) Prawa zdalnego dostępu i plik ~/.rhosts.

Plik "~/.rhosts" ma formę tekstową i zawiera linie postaci:

[-]nazwa-komputera [[-][nazwa-użytkownika]] [...]

Interpretacja pliku ".rhosts" jest następująca: nazwa-komputera jest nazwą odległego komputera po której występują nazwy odległych użytkowników, którym można zaufać, użytkownicy ci mają prawo dostępu - tj. prawo wykonania poleceń rsh, rcp oraz polecenia rlogin (bez podania hasła) na konta lokalnego użytkownika.

Minus przed nazwą komputera/użytkownika oznacza odebranie praw dostępu.

Pliki systemowe są sprawdzane przed lokalnymi. Tak więc jesli zakażemy dostępu z konkretnej maszyny, to będzie on zakazany, nawet jesli globalnie jest to dozwolone. Odwrotnie, jesli zezwolimy na dostęp, a został on globalnie zakazany, nasze zezwolenie nie będzie honorowane. Jesli prawa dostępu dla danej maszyny i użytkownika nie są okreslone w plikach systemowych oraz nie są okreslone w lokalnym pliku .rhosts to odległemu użytkownikowi odmawia się dostępu.

Globalnym plikiem przechowującym informację o prawach zdalnego dostępu jest zazwyczaj "/etc/hosts.equiv" - plik zawierający nazwy komputerów równoważnych - "equivalent hosts". Plik ten ma dokładnie taki sam format jak pliki ".rhosts" poszczególnych użytkowników. Należy jednak zwrócić uwagę, że informacje te mogą być przechowywane w innych plikach

administracyjnych.

UWAGA: Brak pliku "~/.rhosts" jest najczęstrzą przyczyną odmówienia prawa zdalnego dostępu do konta danego użytkownika.

Inne metody przenoszenia plików - program dd(1)

%$dd%

Mało znanym, lecz pożytecznym programem jest dd(1). Kopiuje on wyspecyfikowany plik wejsciowy na wyspecyfikowany plik wyjsciowy dokonując zadanych konwersji.

Polecenie dd służy głównie do odczytu i zmiany formatu danych na tasmach magnetycznych (np. zapisanych w formacie EBCDIC), jest też pomocne wszędzie tam gdzie mamy do czynienia z nosnikiem o nie znanym formacie danych. Przykłady:

dd if=/dev/fd0a ibs=512 iseek=10 count=1 | od -c | more

odczytuje zawartosć dyskietki (tu: /dev/fd0a), ustalono wielkosć bloku na 512 bajtów (w zasadzie jest to zbędne, gdyż wartosć ta jest przyjmowana domyslnie), odczyt rozpoczyna się od dziesiątego bloku (iseek=10), odczytywany jest jeden blok. Ponieważ nie podano nazwy pliku wyjsciowego wyniki przesyłane są do standardowego wyjscia i filtrowane poprzez od(1) (zob. dalej) oraz more(1).

Polecenie dd ma ogólną postać:

dd [opcja=wartosć] ...

Niektóre opcje:

if=nazwa_pliku	plik wejsciowy, domyslnie stdin
of=nazwa_pliku	plik wyjsciowy, domyslnie stdout
ibs=n	wielkosć bloku wejsciowego, domyslnie 512 bajtów
obs=n	wielkosć bloku wyjsciowego, domyslnie 512 bajtów
cbs=n	wielkosć bufora konwersji, domyslnie 512 bajtów
iseek=n	omiń pierwsze n bloków wejsciowych
oseek=n	omiń pierwsze n bloków wyjsciowych
count=n	kopiuj n bloków wejsciowych
conv=	okresla konwersję
lcase	zamień wszystkie litery na małe
ucase	zamień wszystkie litery na duże
swab	zamień sąsiednie bajty

Liczby okreslające wielkosć (bloku, bufora) mogą być opcjonalnie zakończone znakami "k", "b" lub "w", co oznacza odpowiednio mnożniki 1024, 512 i 2.

Przyklad:

dd if=/dev/rmt1 of=./tape ibs=800 obs=8k cbs=80 conv=ascii,lcase

To polecenie odczyta tasmę (tu /dev/rmt1) zawierającą dziesięć osiemdziesięcio bajtowych rekordów (ibs=800, cbs - conversion buffer size) w formacie EBCDIC. Plik wynikowy w formacie ASCII, z wielkimi literami zastępionymi małymi ma nazwę tape, i umieszczony zostanie w katalogu aktualnym.

Korzystanie z odległych stacji tasm i dyskietek.

%$rsh% %$dd% %$tar% %$bar%

Przy omawianiu opcji polecenia tar zademonstrowalismy przykład ekstrakcji z jednostki tasm fizycznie podłączonej do odległego komputera. Przedstawilismy również polecenie "dd", a w kolejnym rozdziale czytelnik znajdzie informacje dotyczące programu rsh. Możemy więc teraz usystematyzować naszą wiedzę dotyczącą obsługi nosników danych zamontowanych na odległych komputerach.

Aby skorzystać z odległego nosnika musimy przede wszystkim znać nazwę maszyny na której jest on fizycznie zamontowany. Na odległej maszynie musimy posiadać prawa dostępu - tak aby możliwe było wykonanie zdalnej powłoki poprzez program rsh.

Przyjmijmy, dla ustalenia uwagi, że używać będziemy jednostki tasm i stosować do archiwizacji program tar. Główna zasada wykorzystania odległego urządzenia polega na uruchomieniu lokalnego programu archiwizującego, oraz odległego programu fizycznego zapisu/odczytu na nosnik danych. Program odległy uruchamiany jest przez program lokalny rsh i poprzez ten własnie program porozumiewa się z lokalnym programem tar.

Tworzenie archiwum będzie więc miało ogólną postać:

tar ...f - ... | rsh nazwa-odleglej-maszyny dd ...

Zas odczyt z archiwum:

rsh nazwa-odległej-maszyny dd ... | tar ...f - ...

Program tar pisze/czyta ze standardowego wejscia/wyjscia - nakazalismy mu to podając znak "-" jako nazwę pliku archiwalnego. Przypominamy też o koniecznosci użycia z poleceniem tar opcji "B" - wielokrotnych odczytów bloku.

Oto dokładne wersje powyższych potoków:

tar -cvBfb - 20 | rsh -n lukrecja dd of=/dev/nrst2

rsh -n lukrecja dd if=/dev/rst2 bs=20b | tar -xvBfb - 20 pliki

W powyższych przykładach odległy komputer nazywa się lukrecja, korzystamy ze stacji tasm "/dev/nrst2", stosując dwudziesto rekordowe bloki. Nazwy odczytywanych/zapisywanych plików są wyswietlane - opcja "v".

Przenoszenie plików usługą uucp(1) (Unix to Unix Copy):

%$uucp%

Nieco archaiczną, lecz bardzo rozpowszechnioną metodą przenoszenia plików jest uucp. Uucp to pakiet programów użytkowych, administracyjnych i systemowych (demonów). Funkcjonować może w systemach połączonych kablem, bez udziału jakich kolwiek urządzeń posredniczących (np. przez łącza RS-232C); w systemach komunikujących się poprzez modemy (a więc w szczególnosci przez linie telefoniczne) oraz w systemach połączonych siecią pracującą w protokole TCP/IP.

Komunikacja pomiędzy odległymi systemami odbywa się na zasadzie sesji. Program obsługujący sesję uucp (uucico) porozumiewa się ze swoim partnerem na odległej maszynie (też uucico) wykonując transfery plików, oraz inne niezbędne czynnosci wymagane przez protokół komunikacji. Zauważmy, że z punktu widzenia systemu nie różni się to od współpracy

użytkownika pracującego na terminalu z programem powłoki. Użytkownika zastępuje program inicjujący transmisję, a powłokę program obsługujący sesję uucp na wywołanym komputerze.

Usługi uucp wykonywane są w dwu fazach. W pierwszej następuje rejestracja usługi (kopiowania plików, wykonania odległego polecenia) na lokalnym komputerze, w drugiej następuje przekazanie niezbędnych plików na odległy komputer.

Wykonywanie poleceń transmisji uucp obywa się asynchronicznie - jest to szczególnie widoczne w wypadku połączeń poprzez linię telefoniczną - zlecenia są gromadzone na lokalnym komputerze, o wyznaczonej godzinie wybierany jest właściwy numer i następuje sekwencja transferów. Na odległym komputerze odebrane pliki są wstępnie umieszczane w katalogu spulera uucp (zwykle /var/spool/uucppublic), a następnie przenoszone do katalogów docelowych użytkowników lub ich skrzynek pocztowych (uucp może współpracować z pocztą elektroniczną).

Zestawienie systemu uucp jest dosć skomplikowaną procedurą administracyjną i nie będziemy jaj tutaj omawiać. Ograniczymy się do niektórych poleceń użytkownika.

Nazwy plików i adresy w uucp:

Specyfikacja pliku obejmuje nazwę (nazwy) systemu, scieżkę oraz ewentualnie nazwę użytkownika.

Specyfikacja pliku źródłowego:

nazwa-systemu!scieżka

Znakiem wykrzyknika można też poprzedzić nazwę pliku lokalnego.

Specyfikacja pliku docelowego (tzw. bang path):

nazwa-systemu!nazwa-systemu!...!nazwa-systemu!scieżka

W tym przypadku możemy okreslić "trasę" po której dokonana zostanie transmisja, możemy więc porozumieć się z systemami, z którymi nie mamy bezposredniego połączenia fizycznego. Zwróćmy jednak uwagę, że w przypadku "przeskakiwania" przez wiele z systemów na każdym z nich musimy posiadać odpowiednie prawa dostępu.

Scieżka to:

- pełna nazwa scieżki
- nazwa scieżki poprzedzona nazwą użytkownika: *~nazwa*
- nazwa scieżki poprzedzona znakiem ~, w takim przypadku scieżka interpretowana będzie względem katalogu "/var/spool/uucppublic"

Należy pamiętać o zwykłych regułach cytowania znaków mających specjalne znaczenie dla powłoki. Znaki-wzorce nazw plików mogą występować w specyfikacji plików lokalnych, nie powinny jednak pojawić się w specyfikacji plików odległych gdyż może to spowodować wystąpienie błędów na odległym systemie.

#Polecenie uucp:

Kopiowanie:

uucp [-c] [-j] [-m] [-s plik] [-xn] plik-źródłowy

plik-docelowy

Opcje:

-c
plik nie jest buforowany w lokalnym katalogu uucp.

-C
plik będzie buforowany lokalnie przed wysłaniem.

-j
Drukowany jest numer zlecenia. Może on służyć do pytania o stan wykonania zlecenia (uustat), lub do anulowania danego zlecenia. Numer zlecenia jest aktualny dopóki zlecenie pozostaje w lokalnej kolejce zleceń.

-m
Zleceniodawca informowany jest o wykonaniu zlecenia pocztą elektroniczną.

-r
Transfer nie jest wykonywany, a jedynie lokalnie kolejkowany.

-s
Informacje o transferze zapisywane są do pliku o podanej nazwie (w niektórych wersjach uucp opcja ta jest ignorowana).

-x n
n=0...9, okresla poziom informacji diagnostycznych przesylanych na standardowe wyjscie. Poziom 9 oznacza najbardziej szczegółową diagnostykę.

Przykłady (zakładamy pracę interakcyjną z powłoką csh):

uucp /home/gjb/dane lukrecja\!/tmp/dane.gjb

Lokalny plik "/home/gjb/dane" zostanie przeniesiony do systemu "lukrecja" i umieszczony w katalogu "/tmp" pod nazwą "dane.gjb".

uucp -j dane falstaf\!bolek\!sosna\!\~/dane.gjb

Lokalny plik o nazwie "dane" zostanie przeniesiony na system "sosna" poprzez systemy "falstaf" oraz "bolek" i umieszczony w katalogu publicznym uucp jako "dane.gjb". Podany zostanie numer zlecenia - opcja "-j".

#) Inne polecenia usługi uucp.

%$cu% %$uuto% %$uux% %$uustat%

Pozostając przy tematyce uucp omówimy najważniejsze programy tej usługi.

cu(1)
Rozpoczyna sesję na odległym komputerze, istnieje możliwosć transferu plików oraz wykonywania poleceń na oddalonym i lokalny komputerze.

uuto(1)
Kopiuje do katalogu publicznego na odległym komputerze: "/var/spool/uucppublic/receive". Odebrany plik może zostać pobrany poleceniem uupick(1).

uux(1)

Tworzy pliki niezbędne do zdalnego wykonywania poleceń na oddalonym komputerze i przesyła je na odległy komputer. Uux udostępnia więc usługę podobną do rsh, różniącą się jednak od niej asynchronicznoscią.

Składnia polecenia uux jest podobna do składni programu uucp:

uux [...] Polecenie

Opcje programu uux niewiele różnią się od opcji programu uucp(1) - uux używa tego samego mechanizmu transferu plików. Zasadnicza różnica między uux i uucp polega na wykonaniu plików z poleceniami przez demona "uuxqt" na odległym systemie.

Należy pamiętać o zwykłych regułach cytowania znaków mających specjalne znaczenie dla powłoki. Znaki-wzorce nazw plików mogą występować w specyfikacji plików lokalnych, nie powinny jednak pojawić się w specyfikacji plików odległych.

Podobnie jak w poleceniu uucp nazwy plików (tu też programów!) mogą być lokalne, lub globalne. Obowiązuje ten sam co w programie uucp sposób zapisu nazwy komputera(ów) i nazwy pliku (programu).

Przykłady:

uux "/usr/bin/diff /home/gjb/plik1 lukrecja!/home/gjb/plikx >
\~/plik.diff"

Lokalnym programem diff(1) porównamy plik "/home/gjb/plik1" z plikiem "/home/gjb/plikx" znajdującym się na komputerze "lukrecja". Informacje o ewentulanych różnicach zostaną umieszczone w lokalnym pliku "/var/spool/uucppublic/plik.diff". Proszę zauważyć, że znaki ">" i "~" są cytowane w celu uniknięcia interpretacji przez lokalną powłokę.

Nazwa pliku docelowego może też być umieszczona w nawiasach klamrowych, poniższe polecenie jest równoważne poprzedniemu:

uux /usr/bin/diff /home/gjb/plik1 lukrecja\!/home/gjb/plikx
\{ \~/plik.diff \}

Znak "!" jest cytowany aby uniknąć jego interpretacji przez powłokę csh.

W kolejnym przykładzie wykonamy to samo zadanie używając programu diff z odległego komputera, oraz umieszczając rezultaty na odległym komputerze "otello":

uux "falstaff!/usr/bin/diff !/home/gjb/plik1
lukrecja!/home/gjb/plikx > otello!\~/plik.diff"

W ostatnim przykladzie "plik1" znajduje się na komputerze falstaff:

uux "falstaff!/usr/bin/diff /home/gjb/plik1
lukrecja!/home/gjb/plikx > otello!\~/plik.diff"

uustat(1)

Informuje o stanie zleconych transmisji (uucp, uuto, uux).

Usługa FTP.

Protokół FTP.

Usługa FTP umożliwia przenoszenie plików między odległymi systemami. Wykorzystywany jest protokół pochodzący z sieci ARPANET - File Transfer Protocol. Protokół ten jest standardem swiatowym, niezależnym od architektury konkretnej maszyny lub systemu operacyjnego. FTP jest obecnie dominującym protokołem przenoszenia plików w sieciach rozległych.

FTP jest przykładem klasycznego systemu klijent/serwer. Klijent nawiązuje połączenia z serwerem, po sprawdzeniu praw dostępu serwer obsługuje polecenia wydawane przez klijenta dotyczące kopiowania plików do/z serwera, wyswietlania zawartosci katalogów itp. Oznacza to, że w celu uzyskania połączenia FTP z konkretnym systemem muszą być spełnione następujące warunki: system klijenta musi odostępniać odpowiedni program komunikacyjny, serwer musi posiadać program obsługujący żądania dostępu FTP, klijent musi posiadać prawa dostępu do serwera (podobne do systemu kont w UNIX-ie).

Interfejsem użytkownika - klijentem usługi FTP jest program ftp(1). Dalsza część tego podrozdziału będzie poswięcona temu programowi.

Program ftp(1)

%$ftp%

Współpraca z programem ftp odbywa się w sposób interakcyjny, i jest nieco podobna do pracy w UNIX-owej powłoce. Typowa sesja ftp składa się z trzech etapów: nawiązania połączenia z systemem odległym, uzyskania odpowienich praw i właściwej sesji.

Składnia wywołania programu ftp:

ftp [-dgintv] [adres-serwera]

Adres odległego systemu (symboliczny lub numer internetowy) może być zadany jako argument "adres-serwera" w takim wypadku ftp natychmiast próbuje nawiązać łączosć. Po udanym połączeniu serwer żąda przestawienia się, dla serwerów UNIX-owych posiadanie konta użytkownika jest równoważne możliwosci otwarcia sesji ftp. Wiele systemów udostępnia też tzw. "anonimowy" dostęp FTP, oczywiscie z pewnymi ograniczeniami.

Przykładowy początek sesji:

papcio% ftp archie.au
Connected to archie.au.
220 plaza.aarnet.EDU.AU FTP server (Version 2.1aWU(1) Tue Jun 1 10:53:58 EST 1993) ready.
Name (archie.au:pfmuw): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-
230- This is the AARNet Archive Server, Melbourne, Australia.
...

Niezależnie od tego czy połączenia zostało nawiązane czy nie, program przechodzi do interakcyjnego trybu pracy. Przedstawmy najważniejsze polecenia interakcyjne:

open nazwa [nport]
Nawiąż łączność z serwerem "nazwa" wykorzystując port serwera numer "nport". Za wyjątkiem nietypowych przypadków numer portu nie musi być podawany.

dir [odległy-katalog] [nazwa-pliku]
Drukuje zawartość "odległego-katalogu". Zawartość katalogu podawana jest w formacie typu "ls -l". Jesli dana jest "nazwa-pliku" spis zawartosci odległego katalogu jest umieszczany w lokalnym pliku o zadanej nazwie. Jesli nie podano argumentów wyswietlana jest zawartość aktualnego katalogu. Podobnie jak powłoka program ftp posiada koncepcje katalogów aktualnych, przy czym okreslony jest zarówno lokalny jak i odległy katalog aktualny. Nazwy lokalnych/globalnych plików mogą być podawane w odniesieniu do katalogów aktualnych (można też oczywiscie podać ścieżkę dostępu rozpoczynającą się znakiem "/").

ls [odległy-katalog] [nazwa-pliku]
Drukuje zawartość katalogu. Wydruk ma postać skróconą - podawane są tylko nazwy plików. Interpretacja argumentów jest taka sama jak w poleceniu dir.

cd odległy-katalog
Zmienia odległy katalog aktualny.

pwd
Wyswietla nazwę odległego katalogu aktualnego.

lcd katalog-lokalny
Zmienia lokalny katalog aktualny.

get plik-odległy [plik-lokalny]
Kopiuje do systemu lokalnego plik z systemu odległego. Jesli nie podano nazwy lokalnej użyta będzie nazwa odległa - parametry "case", "ntrans", "nmap" mogą jednak okreslić dodatkowe reguły odwzorowania nazw. Plik nie musi być przeniesiony "dosłownie" - mogą zajść pewne zmiany formatu: zob. "tryby transferu" dalej. Proszę również zwrócić uwagę, że nazwa pliku odległego nie może zawierać znaków-wzorców. Poleceniem tym można przenieść jednorazowo tylko jeden plik.

put plik-lokalny [plik-odległy]
Umieść plik-lokalny na odległej maszynie. Zob uwagi dotyczące polecenia "get" wyżej.

quit
Zamyka sesję (jesli jest aktywa) i kończy działanie programu ftp.

close
Zamka sesję z odległym serwerem.

help [polecenie]
Wyswietla listę poleceń, lub informację o funkcji danego polecenia.

Przesyłanie wielu plików

Istnieją też pelecenia, umożliwiające przesłania grup plików, których nazwy pasują do danego wzorca. Nazwy plików zawierające znaki-wzorce rozwijane są na lokalnym lub na odległym komputerze. Można się jednak spodziewać poprawnego rozwijania podstawowych znaków takich jak "*". Nazwy katalogów są również rozwijane, jednak w sposób silnie zależny od konkretnego serwera FTP.

mput lokalne-pliki
Rozwija nazwy "lokalnych-plików" tak jak robi to csh. Następnie dla każdego pliku z uzyskanej listy plików wykonywana jest operacja "put". Zob. też "glob".

mget odległe-pliki
Rozwija nazwy "odległych-plików" na odległym komputerze. Następnie dla każdego pliku z uzyskanej listy plików wykonywana jest operacja "get", tak więc znaczenie mają parametry case, ntrans i nmap. Zob. opis polecenia "get" i "glob". Pliki odebrane umieszczone zostaną w

lokalnym katalogu aktualnym.

mdelete odległe-pliki
Usuwa "odległe-pliki" na odległej maszynie.

glob
Włącza i wyłącza rozwijanie znaków-wzorców nazw plików (takich jak "*").

mdir odległe-pliki [lokalny-plik]
Wyswietla (lub zapisuje do "lokalnego-pliku") zawartosci odległych katalogów - podobnie jak polecenie "dir".

mls odległe-pliki [lokalny-plik]
Wyswietla (lub zapisuje do "lokalnego-pliku") zawartosci odległych katalogów - podobnie jak polecenie "ls". Poleceniem tym można zbadać jak rozwijane są na odległej maszynie nazwy katalogów zawierające znaki-wzorce.

prompt
Włącza i wyłącza potwierdzanie przez użytkownika transferów indywidualnych plików. Gdy potwierdzanie jest wyłączone operacje na wielu plikach - takie jak "mget", "mput", "mdelete" - wykonywane są bez interakcji z użytkownikiem.

Tryby transmisji i formaty plików.

FTP może przesyłać pliki w dwu podstawowych trybach - tekstowym i binarnym. W tym ostatnim, pliki przesylane są "dosłownie". Tryb binarny powinien więc być ustawiany gdy kopiujemy programy, archiwa, pliki skompresowane itp. Tryb tekstowy jest trybem domyslnym, możliwe są w nim pewne konwersje formatu pliku.

ascii
Ustawia tekstowy tryb transmisji.

binary
Ustawia binarny tryb transmisji.

UWAGA: Nie ustawienie binarnego trybu pracy programu ftp powoduje niepoprawną transmisję programów, plików typu ".Z" i ".tar", oraz wszelkich plików nie-tekstowych!

cr
Włącza i wyłącza translację znaków RETURN podczas transferów tekstowych. Gdy opcja "cr" jest aktywna pary znaków RETURN/LINEFEED tłumaczone są na pojedyncze znaki LINEFEED.

Wywoływanie poleceń lokalnych i inne polecenia ftp.

Dowolne polecenie lokalne wykonać można poprzedzając je znakiem "!". Przykład interakcyjnego uruchmienia powłoki sh:

```
ftp> !sh
$
$^D
ftp>
```

W zwykłej sesji FTP, natychmiast po uzyskaniu połączenia z odległym systemem następuje otwarcie sesji: odległy system domaga się identyfikacji użytkonika, tzn. jego nazwy, hasła, oraz (rzadziej) nazwy konta. Jesli domyslna opcja "auto-login" jest wyłączona, lub jesli nie powiodła się poprzednia próba identyfikacji, sesja może być otwarta poleceniem "user":

user nazwa-użytkownika [hasło] [nazwa-konta]

Odległy system zapyta się o hasło i nazwę konta jesli nie będą one wymienione a okażą się konieczne.

Przykładowa sesja ftp.

Poniżej przytoczona została autentyczna sesja FTP z serwerem "archie.au", komentarze wyróżniono inną czcionką, pewne informacje od serwera zostały zastąpione przez "...".

```
papcio% ftp archie.au
Connected to archie.au.
220 plaza.aarnet.EDU.AU FTP server (Version 2.1aWU(1) Tue Jun 1
10:53:58 EST 1993) ready.
Name (archie.au:pfmuw): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
```

Nawiązano połączenie, serwer przesyła komunikat powitalny

```
230-
230-    This is the AARNet Archive Server, Melbourne, Australia.
230- ...
230-Local time is Wed Jul 14 22:33:05 1993
230-
230-Please read the file /info/welcome-ftpuser
230-  it was last modified on Tue Jun  8 12:59:54 1993 - 36 days ago
230 Guest login ok, access restrictions apply.
```

Oglądamy zawartosć katalogu głównego serwera

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 66
...
drwxrwsr-x  11 gnu      ftpers      11776 Jul 13 11:16 gnu
drwxr-sr-x  19 owngoph  ftpers        512 Jul  4 07:50 gopher
drwxrwsr-x  11 graphics staff          512 Jun 29 10:10 graphics
drwxrwsr-x   3 info     staff          512 Jul 13 19:43 info
drwx------  2 root     unix           512 Apr  8  1992 lost+found
lrwxrwxrwx   1 root      ftpers           22 Aug  4  1992 ls-lRt.Z ->
info/listings/ls-lRt.Z
drwxrwsr-x   8 root     staff          512 Jan 30 04:58 micros
drwxr-sr-x   4 root     staff          512 Mar 27  1992 projects
...
226 Transfer complete.
1752 bytes received in 3.85 seconds (0.44 Kbytes/s)
```

Zmiana odległego katalogu aktualnego

```
ftp> cd micros/pc/simtel-20/c
250-Questions about or comments on this SIMTEL20 MS-DOS collection
should be
250-sent to w8sdz@TACOM-EMH1.Army.Mil (Keith Petersen).
250-
250-Please note that SIMTEL20 is now using the ZIP 2.x format for
most
250-new files, including the simibm.zip and simlist.zip indexes.  You
250-will need pkz204g.exe or unz50p1.exe from the zip directory to
unzip
250-these files.
250-
...
250 CWD command successful.
```

Z katalogu "/micros/pc/simtel-20/c" pobierzemy kilka plików, wyswietlamy uprzednio

jego zawartość:

```
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 12687
-r--r--r--   1 pc       unix        16204 Jul  2 20:00 00-index.txt
-rw-r--r--   1 pc       daemon      12382 May  6  1990 64colors.zip
...
-rw-r--r--   1 pc       daemon     265204 Feb  3  1991 boss02b.zip
-rw-r--r--   1 pc       daemon      85641 Feb  3  1991 boss03.zip
-r--r--r--   1 pc       ftp         14689 Feb  6  1989 bplus11.zip
-r--r--r--   1 pc       ftp          4921 Jan 26  1989 btoa.zip
-r--r--r--   1 pc       ftp         43998 Dec 31  1987 c-flow.zip
-r--r--r--   1 pc       ftp         33490 Aug  5  1987 c4window.zip
-rw-r--r--   1 pc       daemon       6823 Jun 22  1991 c60probs.zip
^C-r--r--r--   1 pc       ftp          1
```

Transmisja zastaje przerwana przez cntrl-C

```
426 Transfer aborted. Data connection closed.
226 Abort successful
1536 bytes received in 11.55 seconds (0.13 Kbytes/s)
```

Pobieramy plik "00-index.txt", proszę zwrócić uwagę, że tryb transmisji jest tekstowy.

```
ftp> get 00-index.txt
200 PORT command successful.
150 Opening ASCII mode data connection for 00-index.txt (16204
bytes).
226 Transfer complete.
local: 00-index.txt remote: 00-index.txt
16430 bytes received in 16.88 seconds (0.95 Kbytes/s)
```

Zmieniamy tryb transmisji na binarny

```
ftp> binary
200 Type set to I.
```

Pobieramy plik binarny

```
ftp> get btoa.zip
200 PORT command successful.
150 Opening BINARY mode data connection for btoa.zip (4921 bytes).
Transfer complete.
local: btoa.zip remote: btoa.zip
4921 bytes received in 5.98 seconds (0.8 Kbytes/s)
```

I kończymy sesję

```
ftp> quit
221 Goodbye.
papcio%
```

Automatyczna zmiana nazw plików - współpraca z serwerami nie UNIX-owymi.

Standardowo nazwa pliku nie jest zmieniana podczas transmisji zarówno od jak i do serwera. Istnieje jednak możliwość wyznaczenia sposobu konwersji nazw plików. System UNIX akceptuje zwykle pliki o dość arbitralnych nazwach, jednak w pewnych okolicznosciach może zajść konieczność (np. przy współpracy z serwerem nie UNIX-owym) systematyczej zmiany nazw przesyłanych plików. Służą temu poniższe polecenia.

case
Włącza i wyłącza translację dużych liter przy przesyłaniu z odległego komputera poleceniem "mget". Gdy "case" jest aktywne nazwy plików pisane w całosci dużymi literami zostaną zmienione na pisane w calosci literami małymi.

ntrans [znaki-z [znaki-do]]
Bez argumentów wyłącza translację znaków. Argumenty okreslają odwzorowanie znaków z ciągu "znaki-z" na znaki z ciągu "znaki-do". Gdy pierwszy ciąg jest dłuższy znaki nie mające swoich odpowiedników w ciągu "znaki-do" są usuwane. Dla poleceń "put" i "mput" zmieniane są nazwy plików umieszczanych w systemie odległym. Dla poleceń "get" i "mget" zmieniane są nazwy plików umieszczanych na lokalnym komputerze. Translacja zachodzi tylko w wypadku nie podania nazwy pliku docelowego (lokalnego lub odległego).

nmap [wzorzec-z wzorzec-do]
Polecenie bez argumentów wyłącza mechanizm translacji nazw plików. Argumenty wyznaczają sposób translacji nazw plików przy trasmisjach z i do odległego systemu.

We wzorcach mogą wystąpić meta-symbole: $1, $2, ... $9 oznaczające pole wzorca, meta symbol $0 oznacza całą nazwę. Specjalne znaczenie ma też konstrukcja: "[wzorzec1,wzorzec2] " zastępowana przez "wzorzec1" gdy jest on niepusty, oraz przez "wzorzec2" gdy "wzorzec1" jest pusty. Tak więc polecenie:

nmap $1.$2 $2-$1

Spowoduje przetłumaczenie nazwy "to.nazwa" na "nazwa-to". A polecenie:

nmap $0 $0.ftp

Przetłumaczy nazwę "perl.tar.Z" na "perl.tar.Z.ftp".

Załóżmy że okreslono następującą translację nazw:

nmap $1.$2.$3 [$1,$2].[$2,quux]

Spowoduje to następujące translacje:

moje.dane	**moje.dane**
moje.dane.bak	**moje.dane**
moje	**moje.quux**
.moje	**moje.moje**

Jak zwykle do cytowania specjalnych znaków "$", "[", "]", "," służy znak "\".

Podsumowanie sposobu interpretacji nazw lokalnych plików

Jesli nazwa pliku zaczyna się znakiem "-" ftp używa standardowego wejscia (do odczytu) lub standardowego wyjscia (do zapisu).

Jesli pierwszym znakiem nazwy pliku jest "|" reszta nazwy interpretowana jest jako polecenie powłoki. Ftp rozwidla się tworząc proces powłoki (funkcją popen(3)), a następnie czyta (pisze) ze standardowego wejscia (do standardowego wyjscia) utworzonej powłoki. Przykład: "dir |more". Jesli polecenie zawiera znaki odstępu musi być cytowane.

Gdy nie zachodzą dwa powyższe przypadki i włączona jest opcja "glob" lokalne pliki rozwijane są wg. konwencji powłoki csh(1) - zob. "glob". Jesli ftp potrzebuje tylko jednej nazwy pliku (np. przy poleceniu put) wykorzystywana jest pierwsza nazwa z listy.

Dla poleceń "mget" i "get", w których nie podano lokalnej nazwy pliku przyjmuje się nazwę odleglą, przetłumaczoną wg. opcji: "case", "ntrans", "nmap" - w tej kolejnosci.

Dla poleceń "mput" i "put", w których nie podano nazw plików odległych przyjmuje się nazwy lokalne, przetłumaczone ewentualnie przez opcje "ntrans" i "nmap" - w tej kolejnosci. Uzyskane po translacji nazwy mogą też być zmienione na odległej maszynie.

Opcje wywołania programu ftp.

-g Wyłącza rozwijanie nazw plików (globbing)

-i Dezaktywuje opcję interakcji z użytkownikiem przy operacjach mget, mput mdelete.

-n Nie otwiera automatycznie sesji. Program ftp może automatycznie otworzyć sesję korzystając z
pliku inicjalizacyjnego użytkownika.

5.UŻYTECZNE PROGRAMY USŁUGOWE.

Co oznaczają numery podawane za nazwami haseł, np. rm(1), tar(4) ?

%$man% %$whatis% %$apropos%

Są to numery rozdziałów "elektronicznego" podręcznika zainstalowanego w systemie: zob. program man(1). Aby dowiedzieć się więcej o programie man piszemy:

man 1 man

albo (AIX)

man -s 1 man

lub, po prostu

man man

Podanie numeru sekcji jest istotne gdy to samo haslo występuje w kilku rozdziałach np. tar(1)- program i tar(5) - format plików.

Najczęsciej spotykana tematyka rozdziałów:

1	Programy i polecenia dla użytkownika
2	Wywołania systemowe
3	Funkcje biblioteczne
4	Urządzenia zewnętrzne i sterowniki urządzeń
5	Formaty plików
6	Gry
7	To co nie pasuje do tematyki pozostałych rozdziałów
8	Programy administracyjne i systemowe.

W niektórych wersjach Unix-a rozdziały oznaczane są literami, np. w Xenix-ie rozdział "C" zawiera informacje o poleceniach, a "S" funkcjach systemowych.

Każdy rozdział posiada wstęp, który możemy obejrzeć pisząc:

man n intro

Rozdziały mogą dzielić się na pod-rozdziały, np. 3m, 3n, 3yp w Ultrix-ie

O rozdziały w których występuje dane hasło możemy zapytać poleceniami whatis albo apropos:

whatis man
apropos man

Wymaga to jednak uprzedniego zainstalowania przez administratora odpowiedniej indeksowej bazy danych.

Program find.

%$find%

Program find(1) jest uniwersalnym narzędziem służącym do wyszukiwania plików. Plik może być wyspecyfikowany poprzez nazwę lub/i poprzez atrybuty. Przeszukiwane jest zadane pod-drzewo katalogów.

Składnia:

find scieżka [...] predykat [...]

Program find przeszukuje zadane katalogi, wraz z pod-katalogami, szukając plików spełniających zadane warunki (predykaty). Wartosci warunków ewaluowane są metodą "short-circuit", tak więc jesli plik nie spełnia któregos z predykatów kolejne warunki nie są już sprawdzane, ma to dosć istotne znaczenie ponieważ celem pewnych predykatów jest uzyskanie pewnego efektu ubocznego np: podanie nazwy pliku lub wykonanie zadanego polecenia.
Find standardowo nie podąża za dowiązaniami symbolicznymi.

Najważniejsze predykaty programu find:

-name nazwa-pliku
Prawda, jesli plik posiada zadaną nazwę. Nazwa może być wzorcem typu wyrężenia regularnego - regexp(5). Należy pamiętać o cytowaniu specjalnych znaków wzorca, w przeciwnym wypadku zostaną one niepotrzebnie zinterpretowane przez powłokę.

-path nazwa-scieżki
To samo co "-name", z tą rożnicą, że dopasowywana jest nazwa scieżki. Znaki "/" nie są traktowane w sposób specjalny: "*/plik" będzie pasować np: do "./users/gjb/plik".

-type c
Prawda, jesli c plik jest typu okreslonego przez c. Oto dozwolone wartosci c:

f	zwykły plik
d	katalog
c	specjalny plik znakowy
p	Plik FIFO (named pipe)
l	symboliczne dowiązanie
s	"socket"

-user wnazwa
Prawda, jesli właściciel pliku nazywa się "wnazwa"

-group gnazwa
Prawda, jesli plik jest własnoscia grupy o nazwie "gnazwa"

-perm liczba-oktalna
Prawda, jesli pozwolenia pliku równe są *liczbie-oktalnej* - zob chmod(1).

-atime n
Prawda, jesli w ciągu ostatnich n dni nie było do pliku dostępu.

-mtime n
Prawda, jesli w ciągu ostatnich n dni plik nie był modyfikowany.

-ctime n
Prawda, jesli w ciągu ostatnich n dni metryczka pliku nie była modyfikowana

-newer npliku
Prawda, jesli znaleziony plik jest nowszy od pliku o zadanej nazwie - "npliku".

-inum n
Prawda, jesli numer metryczki znalezionego pliku równy jest n.

-print
Zawsze prawda. Powoduje wydrukowanie nazwy znalezionego pliku

Wartosci numeryczne (występujące np. po predykatach -mtime i -ctime) mogą być poprzedzone znakiem "-", co oznacza "mniej niż n", lub znakiem "+", co oznacza "więcej niż n".

Zaawansowane predykaty programu find.

-follow
Zawsze prawda. Find będzie padążać za symboliczymi dowiązaniami.

-exec polecenie
Prawda, jesli wykonane polecenie zwróci zerowy kod powrotu. Polecenie musi być zakończone srednikiem. Para znaków {} w tekscie polecenia jest zastępowane przez nazwę znalezionego pliku.

-ok polecenie
To samo co "-exec", z tą różnicą, że użytkownik pytany jest czy wykonać polecenie dla danego pliku.

-prune
Zawsze prawda. Powoduje zaprzestanie przeszukiwania dla pasującego pliku - co ma sens gdy plik ten jest katalogiem.

-fstype typ
Prawda, jesli plik należy do danego systemu plików. Nazwa systemu plików może być zależna od wersji systemu UNIX, oto typowe nazwy: nfs, ufs (SunOs), hfs (HP-UX), hsfs, pcfs (dyskietka PC-towa).

-links n
Prawda, jesli plik posiada n dowiązań.

warunek1 -o warunek2
Jest to pseudo-warunek *lub*, prawdziwy gdy "warunek1" lub "warunek2" są prawdziwe.

!warunek
Pseudo-warunek "!" ozacza negację - jest prawdziwy go następujący po nim warunek nie jest prawdziwy.

(...)
Warunki ujmować można w nawiasy. Mogą się one okazać pomocne gdy zadawane jest wiele alternatywanych warunków. Należy pamiętać o cytowaniu nawiasów w celu uniknięcia ich interpretacji przez powłokę.

Przykłady:

We wszystkich przykładach przez "plik znajdującym się w katalogu" należy rozumieć także należenie pliku do jednego z pod-katalogów danego katalogu.

Znaleź i wydrukuj scieżki i nazwy plików znajdujących się w katalogu "/katalog":

find /katalog -print

Drukuj też nazwy plików z katalogu "/home":

find /katalog /home -print

Drukuj podkatalogi katalogu "/home":

find /home -type d -print

Drukuj nazwy plików znajdujących się w katalogu /home i zawierających tekst "gruba ryba":

find /home -type f -exec grep -l "gruba ryba" '{}' \;

Proszę zwrócić uwagę na cytowania.

Usuń pliki o nazwie "core", znajdujące się w katalogu "/" i nie modyfikowane od dziesięciu dni:

find / -name core -mtime +7 -exec rm {} \;

Drukuj nazwy wszystkich plików zawartych w aktualnym katalogu (i jego pod-katalogach), omijając pod-katalogi o nazwie "SCCS". Proszę zauważyć, że pierwsze dopasowanie nazwy "-name SCCS" służy do odcięcia poszukiwania dla katalogu o nazwie "SCCS", pseudo-warunek "-o" (lub) powoduje, że ostatni predykat - "-print" jest ewaluowany.

find . -name SCCS -prune -o -print

Przykład na zastosowanie nawiasów. Należy usunąć wszystkie pliki o nazwach "a.out" i "core", do których w ciągu tygodnia nie było dostępu, i które nie są montowane za posrednictwem NFS:

find . \(-name a.out -o -name core \) -atime +7 -exec rm {} \; -o -fstype nfs -prune

Inne możliwosci programu find
%$find%

Program find może też oferować inne predykaty dopasowujące pliki, umożliwiać budowanie archiwów, zwykle posiada też opcje wykorzystujące charakterystyczne cechy konkretnego systemu plików.

Kłopoty z symbolem "{}" programu find

Program find posiada opcję "-exec" powodującą wykonanie zadanego polecenia dla wszystkich znalezionych plików. Find zastąpi symbol "{}" nazwą znalezionego pliku.

Załóżmy, że chcemy wykonać program "program" dla każdego pliku z katalogu znalezionego przez find:

find / -type d -exec program {}/* \;

(Znaki "\" wstawiono aby uniknąć interpretacji "*" i ";" przez powłokę)

Chcemy aby wykonany został ciąg poleceń:

program katalog1/*
program katalog2/*
...

Niestety, find interpretuje symbol "{}" tylko wtedy, gdy jest on otoczony znakami odstępów. Symbol "{}/*" pozostanie więc bez zmian, wykone zostaną polecenia:

program {}/*
program {}/*
...

Jak osiągnąć żądany efekt?

1) Można wywołać polecenie "program" posrednio - przy pomocy jedno-linijkowego skryptu "./koperta", oto jego zawartosć:

program "$1"/*

Piszemy:

find / -type d -exec ./koperta {} \;

2) Jako "kopertę" możemy wykożystać powłokę sh:

find / -type d -exec sh -c 'program $0/*' {} \;

Trik polega na tym, że argumentem zerowym "cmd" przy wywołaniu "sh -c 'cmd' A B C" jest A.

3) Możemy "skonstruować" wsadowy plik poleceń przy pomocy programu sed:

find / -type d -exec -print | sed 's:.*:program &/*:" | sh

Program rsh - "remote shell"

%$rsh%

Za pomocą rsh można uruchamiać programy na innej, podłączonej do sieci maszynie. Rsh nawiązuje połączenie z odległą maszyną, nakazując jej wykonanie zadanego polecenia. Polecenie wykonywane jest w powłoce uruchomionej "na konto" danego użytkownika, rsh jest więc podobne do polecenia rlogin. Standardowe wejscie rsh zostaje połączone ze standardowym wejsciem odleglego procesu; podobnie, standardowe wyjscie rsh zostaje połączone do standardowego wyjscia odległego procesu; sygnały interrupt, quit i terminate są propagowane do odleglego procesu.

Składnia:

rsh [-n] [-l nazwa-użytkownika] nazwa-komputera polecenie

Jesli nie podano nazwy użytkownika (opcja "-l") odległe polecenie wykonywane jest "na koncie" użytkownika który wywołał rsh. Konsultowane są pliki zezwoleń dostępu sieciowego: zob. 3.2.3. Opcja -n objasniona zostanie w następnym punkcie.

Przykłady:

falstaf% rsh lukrecja cat moje.dane > lokalny.plik

Plik "moje.dane" znajdujący się na komputerze lukrecja (w katalogu głównym użytkownika) zostaje skopiowany do pliku "lokalny.plik" na komputerze z którego wywołano rsh - tzn. z falstaffa.

falstaf% rsh lukrecja cat moje.dane ">" moje.dane.bak

Tu poleceniem jest "cat moje.dane ">" moje.dane.bak", plik "moje.dane" na komputerze lukrecja zostanie skopiowany do pliku "moje.dane.bak" na tym samym komputerze.

) W jakich wypadkach wykonanie programu rsh kończy się niepowodzeniem ?

%$rsh%

Przyczyn może być kilka:

1) Użytkownik wywołujący rsh musi posiadać odpowiednie prawa dostępu na odległym komputerze. Warunkiem koniecznym jest posiadanie konta (chyba, że opcją -l wywołujemy odległe polecenie na konto innego użytkownika), zob. też uwagi dotyczące polecenia rcp(1) i pliku ".rhosts" w poprzednim rozdziale. W wypadku braku odpowiednich praw dostępu rsh zgłasza błąd "execute permission denied" (lub inny o podobnej tresci).

2) Mniej oczywistą przyczyną może być zakłócenie zdalnego wywołania przez polecenia zapisane w pliku ".cshrc" (o ile powłoką użytkownika na odległym komputerze jest csh). Polecenia te, przeznaczone do ustawienia srodowiska pracy interakcyjnej mogą "ogłupić" rsh. Rozwiązanie jest proste: wszystkie takie polecenia powinny być wykonywane tylko wtedy gdy powłoka będzie rzeczywiscie pracować interakcyjnie. Wykonujemy je więc warunkowo. Przykładowy plik .cshrc:

```
...
if ( $?prompt ) then
        polecenia
        ...
endif
```

W powłoce nieinterakcyjnej, czyli np. uruchomionej przez rsh zmienna "prompt" nie będzie zdefiniowana, tak więc odpowiednie polecenia nie będą wywoływane. Zobacz też omówienie powłok interakcyjnych i nieinterakcyjnych w rozdziale poswięconym powłokom.

#) Jak używać rsh w tle ?

%$rsh%

Najprostrze rozwiązania:

```
        rsh komputer polecenie &
```

oraz

```
        rsh komputer 'polecenie &'
```

są błędne ponieważ nie rozwiązują problemu standardowego wejscia, wyjscia i diagnostyki zarówno procesu lokalnego (rsh), jak i odległego polecenia. Oto poprawne rozwiązanie:

```
rsh -n komputer 'polecenie >&/dev/null </dev/null &'
rsh -n komputer 'polecenie >/dev/null 2>&1 </dev/null &'
```

Pierwszy przykład odnosi się do przypadku, gdy na odległej maszynie używamy powloki csh, drugi gdy odległą powłoką jest sh.

Opcja -n powoduje podłączenie lokalnego standardowego wejscia (tzn. wejscia polecenia rsh) do "/dev/null", tak że rsh może się wykonywać w tle na lokalnym komputerze. Związanie standardowych plików na odległym komputerze powoduje, że rsh może zakończyć sesję (nie ma przepływu danych między lokalną a odległą maszyną). Polecenie jest wykonywane w tle

tylko na odległej maszynie. Oczywiscie nie musimy łączyć standardowych plików odleglego polecenia z "/dev/null", może to być dowolny plik.

Przypominamy, że "/dev/null" to specjalny plik: dla zapisu jest on "czarną dziurą" - można do niego pisać bez ryzyka wystąpienia jakich kolwiek błędów, oczywiscie informacja zapisana do /dev/null nie jest nigdzie przechowywana. Dla odczytu "/dev/null" jest plikiem pustym - próba odczytu powoduje wystąpienie błędu konica pliku.

6.PLIKI - PYTANIA I ODPOWIEDZI

Jak odtworzyć usunięty plik?

%$alias% %$rm%

W każdym poprawnie zarządzanym systemie administrator powinien wykonywać regularnie (tj. co najmniej codziennie) składowanie systemu plików. Jesli jednak od ostatniego składowania wpisales 100KB tekstu, lub jesli backupy nie są wykonywane regularnie problem pozostaje.

Można powiedzieć zdecydowanie że rm(1) naprawdę USUWA plik, a odtworzenie pliku w systemie wielodostępnym jest praktycznie niemożliwe. Wraz z wykonaniem polecenia "rm" system gubi informację o tym jakie bloki dyskowe tworzyły usunięty plik, co gorsza bloki te zostaną prawdopodobnie wykorzystane (i zamazane!) jako pierwsze gdy zajdzie potrzeba alokacji miejsca na dysku. Skomplikowana struktura systemu plików, zwłaszcza w nowszych wersjach UNIX-a, skutecznie zniechęca do prób "ręcznej" interwencji. Być może natychmiastowe odmontowanie partycji z utraconymi plikami pozwoli uratować część danych, musimy się jednak liczyć z tym, że nawet bardzo doswiadczonemu znawcy systemu "operacja ratunkowa" może zając kilka dni.

Dla początkujących użytkowników dobrym zabezpieczeniem przed pochopnym skasowaniem pliku będzie stały alias "rm rm -i". Inne rozwiązanie to utworzenie nowego polecenia usunięcia pliku, przenoszącego dany plik do katalogu "smietnika".

Przyklad dla csh, umieszczamy co następuje w ~/.login:

```
if ( ! -d ~/.zsyp ) mkdir ~/.zsyp              # tworzy ~/.zsyp
alias r   'mv \!* ~/.zsyp'                     # zastępuje rm
alias zsyp          'rm -f ~/.zsyp/*'  # naprawdę usuwa
```

w pliku ~/.logout umieszczmy:

```
# usuń smieci przy wyjsciu z systemu
rm -f ~/.zsyp/*
```

Nie poleca się zastępowania rm jego aliasem lub skryptem z dwu powodów: Po pierwsze może się okazać, że zarządzanie smietnikiem kosztuje nas więcej wysiłku niż uważne stosowanie polecenia rm. Po drugie, przyzwyczajenie się do "łagodnego" rm może być tragiczne w skutkach gdy przesiądziemy się na system z prawdziwym rm!

Jak usunąć plik, którego nazwa zaczyna się na "-" ?

%$rm%

Generalizując: jak uniknąć interpretowania znaku "-" występującego na początku nazwy pliku jako opcji wywołania programu, którego plik ten jest argumentem.

Należy tak wyspecyfikować plik do usunięcia, aby jego nazwa nie zaczynała się znakiem minus. Najprosciej napisać:

polecenie ./-nazwa_pliku

dla naszego konkretnego przypadku:

rm ./-nazwa_pliku

(Oczywiscie zakładając, że plik znajduje się w aktualnym katalogu).

Dla wielu poleceń, zwłaszcza tych które używają popularnej funkcji getopt(3) do rozbioru i interpretacji argumentów, opcja "--" (lub "-") oznacza ostatnią opcję wywołania, wszystkie argumenty występujące po tym znaku nie są już interpretowane jako opcje.

Tak więc pisząc:

polecenie -- -nazwa_pliku

W naszym przykladzie:

rm -- -nazwa_pliku **lub:**
rm - nazwa_pliku

Osiągniemy poprawny efekt.

Jak usunąć plik z "dziwnymi" znakami w nazwie?

%$rm% %$find%

Pliki o dziwnych nazwach magą pojawiać się dosć często jako efekt niepoprawnego działania programów użytkownika, lub błędów przy pracy z terminalem. Nietypowe znaki mogą ukazywać się jako "?" przy wyswietleniu zawartosci katalogu, co dodatkowo utrudnia usunięcie pliku lub zmianę jego nazwy.

Najprostrze rozwiązanie to oczywiscie:

rm -i jakis_wzorzec_który_pasuje_do_nazwy_danego_pliku

Program "rm" zapyta nas, czy chcemy usunąć pliki, których nazwy pasują do podanego wzorca. Niestety, jesli interpreter poleceń zeruje ósmy bit znaków, a nazwa natrętnego pliku zawiera znak o wartosci większej od 127 operacja nie powiedzie się.

Możemy spróbować inaczej:

rm -ri .

Rm zapyta nas teraz o każdy plik. Wystarczy, że odpowiemy twierdząco tylko przy nazwie interesującego nas pliku, a przecząco dla wszystkich innych. I tu, niestety, pojawiają się problemy. Po pierwsze, nie wszystkie wersje polecenia rm poprawnie zinterpretują opcję "-i". Po drugie, rm "przejdzie" po wszystkich pod-katalogach katalogu ".". Można temu zaradzić tymczasowo zakazując dostępu do pod-katalogów programem chmod(1). Po trzecie: opcja "-r" i wzorce zawierające "*" dla polecenia rm to potencjalne niebezpieczeństwo utraty cennych plików. Zawsze upewnij się czy wiesz co robisz!

Najpewniejsze, ale i najbardziej skomplikowane rozwiązanie odwołuje się do bardzo pomocnego programu find:

find . -type f ... -ok rm '{}' \;

Co oznacza: dla każdego pliku z katalogu aktualnego (i jego pod-katalogów), będącego plikiem zwyklym (-type f), i spełniającego pewne dodatkowe warunki "..." wykonaj polecenie rm z nazwą pliku jako argumentem, pytając przedtem o zgodę (-ok). Jak dobrać predykaty "..." aby jednoznacznie wskazywały one na złosliwy plik? Najlepiej posłużyć się numerem metryczki pliku (polecenie "ls -i" poda nam pary: numer_metryczki nazwa_pliku). Znamy więc numer

metryczki, powiedzmy że jest to 12345, piszemy:

```
find . -inum 12345 -ok rm '{}' \;
```

jesli jestesmy ostrożni, lub nie chcemy utracić zawartosci pliku:

```
find . -inum 12345 -exec mv '{}' nowa_nazwa_pliu \;
```

W ostatnim przypadku nie użylismy predykatu "-ok". Może być to wskazane gdy spodziewamy się że nazwa pliku zawiera znaki "szkodliwe" dla terminala, np: znak czyszczenie ekranu, lub nawet sekwencje które mogą przeprogramować terminal!

Find to program użyteczny w wielu sytuacjach, proponuję czytelnikowi zapoznanie się z jego opisem w rozdziale poswięconym programom usługowym.

Pozostaje najbardziej przykry przypadek: plik, którego nazwa zawiera "/". Plik taki może zostać utworzony w katalogu zamontowanym poprzez NFS w systemie nie UNIX-owym. Zauważmy, że nie pomogą nam żadne sztuczki z find-em i rm gdyż funkcja systemowa unlink(2) zawsze błędnie zinterpretuje nazwę pliku jako nazwę pliku w pod-katalogu, np. jesli plik ma nazwę "a/b" system zawsze będzie ją interpretował jako plik "b" w katalogu "a". Plik taki możemy spróbować usunąć tą samą drogą jaką powstał tj. poprzez NFS. Jesli operacja ta się nie powiedzie musimy prosić o pomoc administratora systemu. Jak poradzi sobie z tym administrator? To pytanie z nie istniejącej jeszcze ksiażki "Sztuczki i Chwyty dla Administratora Systemu UNIX".

Jak systematycznie zmienić nazwy grupy plików np. "*.foo" na "*.bar"?

%$mv% %$foreach% %$basename%

Pierwsze, narzucające się rozwiązanie: "mv *.foo *.bar" jest oczywiscie błędne. Załóżmy, że w katalogu aktualnym mamy pliki "a.foo" i "b.foo", powłoka csh rozwinie wzorzec "*.foo" na "a.foo b.foo" i poda komunika "no match" gdyż nie ma plików, których nazwy pasują do *.bar. Powłoka Bournea wykona polecenie: "mv a.foo b.foo *.bar", które powiedzie się tylko w wypadku istnienia katalogu o nazwie "*.bar", pozostaje tylko pytanie, czy naprawdę o to nam chodziło?

Do wykonania operacji "uniwersalnego mv" użyjemy prostego skryptu:

csh:

```
foreach f ( *.foo )
   set base='basename $f .foo'
   mv $f $base.bar
end
```

sh:

```
for f in *.foo; do
   base='basename $f .foo'
   mv $f $base.bar
done
```

Polecenie "basename" usuwa sufix z ciągu znaków będącego nazwą pliku.

Zamiast "basename" możemy zastosować specjalne mechanizmy powłoki:

csh:

```
foreach f ( *.foo )
   mv $f $f:r.bar
end
```

ksh:

```
for f in *.foo; do
   mv $f ${f%foo}bar
```

A oto inne podejscie wykorzystujące edytor strumieniowy sed:

```
ls -d *.foo | sed -e 's/.*/mv & &' -e 's/foo$/bar/' | sh
```

Ten, z pozoru skomplikowany, ciąg poleceń ma prostą interpretację: Lista plików pasująca do wzorca "*.foo" jest przetwarzana przez program sed. Ten ostatni dla każdej nazwy pliku wykonuje dwie substytucje tekstowe: "nazwa.foo" zamienia na "mv nazwa.foo nazwa.foo", a następnie zamienia "foo" kończące linię na bar. Tak utworzony ciąg poleceń mv kierowany jest do powłoki sh, która je wykonuje.

Jak zmienić nazwy plików na pisane małymi literami ?

%$tr% %$echo% %$mv% %$eval%

Zasada postępowania jest podobna do zastosowanej w poprzednim punkcie. Do zmiany dużych liter na małe użyjemy polecenia tr(1) (transliterate). Działa ono jak filtr, zmieniający wyspecyfikowane znaki na inne. Składnia polecenia "tr" pozwala na zastąpienie znaków z pewnego zakresu na znaki należące do innego zakresu. Oto przykladowe skrypty działające na wszystkich plikach z aktualnego katalogu:

csh:

```
foreach f ( * )
   mv $f `echo $f | tr '[A-Z]' '[a-z]'`
end
```

sh:

```
for f in *; do
  mv $f `echo $f | tr '[A-Z]' '[a-z]'`
done
```

Zasadnicza częsć obu skryptów jest taka sama. Drugi argument polecenia "mv" konstruowany jest poprzez "przefiltrowanie" nazwy pliku przez tr zmieniające duże litery na małe.

W kolejnym przykładzie użyjemy dodatkowych możliwosci powloki Korna:

```
typeset -l l
for f in *; do
  l="$f"
  mv $f $l
done
```

Niestety, podane przykłady nie poradzą sobie z plikami o złosliwych nazwach, np. z takimi których nazwy zawierają spacje. Rozwiążemy ten problem wymuszając podwójnym cudzysłowem potraktowanie kilku ciągów znaków odzielonych spacjami, znakami tabulacji, lub znakami nowej linii jako jednego argumentu. Przyklad dla sh:

```
for f in *; do
  eval mv '"$f"' \"`echo "$f" | tr '[A-Z]' '[a-z]'`\"
done
```

Jak dowiedzieć się o czas utworzenia pliku?

Jest to niemożliwe - czas taki nie jest nigdzie przechowywany. Metryczka pliku posiada trzy pola związane z czasem: czas utworzenia metryczki: pokazuje go "ls -lc", czas ostatniej modyfikacji pliku: "ls -l", czas ostatniego dostępu do pliku: "ls -lu". Czas utworzenia metryczki jest najlepszym przybliżeniem "czasu utworzenia pliku", jest on jednak uaktualniany przez operacje, które nie zmieniają nawet zawartosci pliku np: chgrp, chmod, chown, mv, ln. Programistę będzie najbardziej interesował czas ostatniej modyfikacji pliku, to na jego podstawie programy takie jak make(1) decydują o tym czy plik jest "aktualny" w stosunku do innych plików.

Jakie znaczenie mają zezwolenia dostępu dla symbolicznego dowiązania?

Nie mają one żadnego znaczenia, i zwykle ustawione są na 777. Znaczenie mają jedynie zezwolenia dostępu dla pokazywanego przez symboliczne dowiązanie pliku. Z poziomu powłoki próba zmiany zezwoleń dostępu dla symbolicznego dowiązania spowoduje zmianę zezwoleń dla pliku, na który pokazuje dowiązanie.

Jak podzielić plik na mniejsze częsci ?

%$split% %$csplit%

Dosć często jestesmy zmuszeni do podziału dużych plików, najczęsciej tekstowych, na kilka (nascie) mniejszych. Może to okazać się konieczne np. gdy chcemy przesłać plik pocztą elektroniczną (e-mail radzi sobie tylko z plikami mniejszymi od 100 lub 300 kB).

W wielu systemach możemy posłużyć się programem split(1), o następującej składni wywołania:

split [-Nnnn] [Plik [Prefix]]

lub:

split [-nnn] [Plik [Prefix]]

Program ten dzieli plik "Plik" na częsci, z których każda (za wyjątkiem być może ostatniej), zawiera dokladnie nnn linii (domyslna wartosć wynosi zwykle 1000). Kolejne pliki bedą miały nazwy: "Prafixaa", "Prefixab", itd. Domyslny prefix to "x".

Jesli chcemy podzielić plik na $k równych częsci możemy napisać:

split -`expr \`wc -l Plik | awk '{print $1}' \` /$k` Plik Prefix

Polecenie "wc -l" oblicza ilosć linii w pliku, a expr dzieli otrzymaną liczbę przez k.

W niektórych systemach dostępne jest polecenie csplit(1). Przy jego pomocy możemy podzielić plik w dowolnie wybranych miejscach. Punkty podziału okreslane są na podstawie dopasowania wyrażeń regularnych, można też podać numer linii.

Poniższy przykład dzieli plik o nazwie "quux" na częsci po 100 linii każda, przy czym ostatnia - setna część może być dłuższa.

csplit -k quux 100 {99}

Następny przykład dzieli program napisany w języku C: "prog.c" na pliki, z których każdy zawiera jedną funkcję (zakładamy, że funkcje są zawsze kończone znakiem "}" występującym jako pierwszy w linii):

csplit -k prog.c '/^}/+1' {99}

Oto pełna składnia polecenia csplit:

csplit [-f prefix] [-k] [-s] nazwa_pliku argument [...]

Csplit wczytuje plik o nazwie "nazwa_pliku" i dzieli go na n+1 pod-plików, gdzie n jest liczbą argumentów. Pod pliki mają nazwy xx0 do xxn (chyba, że podano prefix). Argumenty mają następującą postać i interpretację:

/wyrażenie_regularne/

Utwórz plik od bieżącej linii do linii pasującej do danego wyrażenia regularnego (bez niej). Bieżącą linią staje się linia pasującą do wyrażenia. Argument może być zakończony znakiem "+" lub "-" po którym występuje liczba linii.

%wyrażenie_regularne%

Jak wyżej, z tym że nie tworzony jest nowy plik.

numer_linii

Utwórz plik od bieżącej linii do lini "numer_linii" (bez niej). Bieżącą linią staje się linia o numerze "numer_linii".

{liczba}

Powtarza poprzedni argument będący wyrażeniem regularnym. Jesli poprzednim argumentem jest "numer_linii" plik będzie dzielony co "numer_linii" linii.

Opcje programu csplit:

-f prefix	ustal prefix nazw tworzonych plików
-k	nie usuwaj utworzonych plików w przypadku wystąpienia błędu.
-s	nie drukuj liczby wczytanych znaków.

Jesli nie mamy programu csplit lub split możemy posłyżyć się nowym awk - nawk(1), nie polecam natomiast poleceń "head" i "tail" - zazwyczaj źle radzą sobie one z plikami o dużej liczbie linii. Najprostrzym rozwiązaniem może okazać się po prostu krótki program w C.

Prosty podgląd plików - polecenie od.

%$od%

Program od(1) (Octal Dump) służy do wyprowadzania zawartosci plików w zadanej postaci:

oktalnej, heksadecymalnej, ASCII, itp.

od [-bcCDdFfOoSsvXx] [nazwa_pliku]

Opcje:

- *-b* Wyprowadź bajty oktalnie
- *-c* Wyprowadź znakowo, znaki niedrukowalne pojawiają się jako liczby
- *-d* Wyprowadź jako 16 bitowe liczby dziesiętnie
- *-f* Wyprowadź jako 64 bitowe liczby zmiennoprzecinkowe
- *-o* Wyprowadź jako 16 bitowe liczby oktalne
- *-x* Wyprowadź jako 16 bitowe liczby heksadecymalnie
- *-s* Wyprowadź jako 16 bitowe liczby diesiętnie ze znakiem

Domyslnie dane wprowadzane są ze standardowego wejscia.

Przykład: wyswietlenie zawartosci pustego katalogu potraktowanego jak zwykły plik:

mkdir katalog; cat katalog | od -c

Kompresja plików:

%$compress% %$uncompress% %$pack% %$unpack% %$zcat%

Podobnie jak inne systemy operacyjne UNIX udostępnia zwykle co najmniej jeden program dokonujący kompresji danych. Najpopularniejszym jest program compress(1) dostępny we wszystkich nowszych wersjach systemu.

Składnia:

compress [nazwa-pliku]

Program ten zmniejsza wielkosć pliku stosując metodę Lempel-Ziv. Tworzony jest zkompresowany plik o nazwie "nazwa-pliku.Z". Jesli pakowanie powiodło się oryginalny plik jest usuwany. Pozwolenia dostępu, czasy dostępu i modyfikacji nie są zmieniane. Jesli nie podano nazwy pliku wejsciowego przyjmuje się standardowe wejscie. Program nie pozwoli na kompresję pliku już upakowanego.

Do rozpakowania służy program uncompress(1). Składnia:

uncompress [nazwa-pliku]

Jesli nie podano nazwy pliku wejsciowego przyjmuje się standardowe wyjscie. Jesli rozpakowanie powiodło się plik spakowany jest usuwany. Pliki specjalne (katalogi etc.) nie ulegają kompresji.

Format compress jest przenosny między różnymi wersjami systemu.

Innym, mniej popularnym programem jest pack(1) (zob. też unpack(1) i zcat(1)). Program ten stosuje kodowanie Huffmana. Składnia jest podobna do składni compress. Tworzone pliki mają

rozszerzenie ".z".

Jak zarchiwzować plik (pliki) poddając je jednoczesnie kompresji?

%$tar% %$compress%

Programy tar i bar nie pakują tworzonego archiwum, przeciwnie, długość tworzonego pliku jest większa od sumy długosci plików składowych.

Spakowane archiwum można uzyskać poddając kompresji standardowe wyjscie polecenia tar.

Przykład. Kompresja zawartosci katalogu aktualnego, tworzony jest plik kat.tar.Z:

tar cfv - . | compress > kat.tar.Z

Jak przesyłać pocztą elektroniczną (e-mail) pliki nie tekstowe?

%$uuencode% %$uudecode% %$mail%

Poczta elektroniczna poprawnie obsługuje jedynie "listy" będące plikami tekstowymi. Nie można przesyłać plików binarnych - a więc: programów w postaci wykonywalnej, bibliotek, archiwów w formacie tar, itp.

W celu przesłania pliku binarnego należy go uprzednio zakodować, tak aby stał się on plikiem tekstowym. Plik taki można już bezpiecznie przesłać korzystając ze standardowych usług typu mail. Odbiorca musi oczywiscie zdekodować otrzymany plik tekstowy - tzn. przenieść go ponownie do formatu binarnego. Należy się liczyć z tym, że tekstowa postać pliku binarnego będzie dwa razy dłuższa od niezakodowanego oryginału.

Do kodowania i dekodowania plików binaranych służy para programów uuencode(1) i uudecode(1).

Składnia:

uuencode [nazwa-pliku] etykietka-pliku

Plik o nazwie "nazwa-pliku" kodowany jest znakami tekstowymi, plik zakodowany przekazywany jest zawsze do standardowego wyjscia. Jesli nie podano nazwy-pliku przyjmuje się standardowe wejscie. Zakodowany obraz pliku binarnego poprzedza krótki nagłówek mający postać:

begin nnn etykietka-pliku

Gdzie nnn - pozwolenia dostępu (np 644), a etykieta-pliku jest taka sama jak odpowiedni argument programu.

Do rozkodowania stosuje się program uudecode.

Składnia:

uudecode [nazwa-pliku-zakodowanego]

Polecenie to zamienia zakodowany plik tekstowy na plik binarny. Tworzony jest plik o nazwie i pozwoleniach dostępu zapisanych w nagłówku pliku zakodowanego. Jesli nie podano nazwy pliku za plik wejsciowy przyjmuje się standardowe wejscie, w takim wypadku wynik jest

kierowany do standardowego wyjscia.

Przyklad: chcemy zakodować i przesłać pocztą plik o nazwie "my.prog":

uuencode my.prog program > my.prog.UU
mail kazio < my.prog.UU

Zakodowany tekstowo plik "my.prog" zapisano do pliku "my.prog.UU", w nagłówku zakodowanego pliku umieszczono etykietkę "program". Tak zakodowany plik wysłalismy do użytkownika kazio. Kazio odebrał naszą przesyłkę i zapisał ją w pliku list. Aby zdekodować ten plik do postaci binarnej pisze:

uudecode list

W katalogu bieżącym Kazia tworzony jest plik binarny o nazwie program i będący wierną kopią oryginalnego pliku "my.prog".

Co robić gdy uudecode nie potrafi zdekodować pliku przesłanego nam pocztą ?

%$uudecode% %$awk%

Czy plik rzeczywiscie jest w postaci UU ? Powinien on wyglądać
mniej więcej tak:

From root Mon Feb 1 22:40:10 1993
Received: from inet-gw-2.pa.dec.com by viki.ii.pw.edu.pl
Date: Mon, 1 Feb 93 13:36:46 -0800
Message-Id: <9302012136.AA14020@inet-gw-2.pa.dec.com>
From: "ftpmail service on inet-gw-2.pa.dec.com" <nobody@Pa.dec.com>
To: blinowsk@ii.pw.edu.pl
.
. Reszta nagłowka pocztowego
.

Status: R

begin 644 ftpmail.uu
M4$L#!`H`!@`&`,"$=Q2U-/XS!1```$@G```(````1$5#3TU0+D-A"GL'!AL&
.
. Zakodowany plik
.
M%-K;8W+,`@``T0,```8`````````0`@````0$```%)%041-15!+!08`````
.`P`#`*(````P0P`````
`
end

Program uudecode powinien zdekodować plik zawierający nagłówek poczty, ale możemy spróbować usunąć nagłówek przed dekodowaniem.

Zakodowany tekst musi kończyć się słowem "end". Jesli przesyłka składa się z kilku częsci musimy je przed rozkodowaniem scalić. W takim wypadku konieczne jest usunięcie wszystkich nagłówków pocztowych (za wyjątkiem pierwszego).

Inną przycznyną niepowodzenia może być specyficzny "błąd" w oprogramowaniu pocztowym. Niektóre mailer-y kasują kończące linię spacje. Jesli dokładniejsza analiza pliku wykaże, że nie wszystkie linie występujące między nagłówkiem ("begin") ,a końcem pliku ("end") są tej samej długosci musimy dodać do skróconych linii brakujące spacje. Najprosciej wykonać to przy

pomocy prostego skryptu sed lub awk. Takiego jak poniższy:

```
# skrypt dla awk - poprawUU
start==1 \
{       len = length($0);
        if ( len<61 )  {
           printf("%s",$0);
           for (i=0;i<61-len;i++)
              printf(" ");
           printf("\n");
        }
          else
            print $0;
}
/begin/ { start=1;}
/^end/ { start=0; }
```

Uruchomienie skryptu:

```
awk -f poprawUU < kiepski.UU > dobry.UU
```

Transfer plików tekstowych między systemem UNIX a MS-DOS.

%$unix2dos% %$dos2unix%

Pliki teksowe mają nieco inny format w systemach MS-DOS i UNIX. W UNIX-ie nie występuje znak końca pliku, a linie kończone są znakiem CR (powrotu karetki); w DOS-ie występuje znak końca pliku - EOF (jest to cntrl-Z), a linie kończone są parą znaków CR,LF (powrót karetki i znak nowej linii). W większosci przypadków przy przenoszeniu plików miedzy tymi dwoma systemami niezbędne jest wykonanie konwersji pliku (chyba że wykonuje ją automatycznie program, którym pliki przenosimy - np. ftp).

Do konwersji plików tekstowych służą dwa programy: dos2unix(1) i unix2dos(1).

Składnia:

unix2dos [-ascii] [-iso] źródłowy docelowy
dos2unix [-ascii] [-iso] źródłowy docelowy

Pierwszy program dokonuje konwersji z formatu tekstowego systemu UNIX do formatu tekstowego systemu DOS, drugi odrotnie.

Opcje:

-*iso* konwersja CR <-> CR/LF, usunięcie/dodanie znaku końca pliku, oraz zamiana znaków w standardzie rozszerzonym ISO na znaki "DOS extended", jest to opcja domyslna.

-*ascii* konwersje jw. ale bez konwersji "ISO extended" <-> "DOS extended"

PROCESY

W rozdziale tym omówione są zagadnienia ogólno systemowe, które najwygodniej ująć pod wspólną tematyką "procesów". Czytelnik znajdzie tu informacje poswięcone współpracy aplikacji z systemem, a więc takim zagadnieniom jak uruchamianie programu, obsługa sygnałów, błędy terminalne w procesie.

Jak tworzyć programy i skrypty z pozwoleniem SUID ?

%$SUID% %$chmod%

Przypominamy, że programy typu SUID (Set User ID) działają z efektywnym identyfikatorem ich właściciela, a nie użytkownika, który je uruchomił. Dzięki mechanizmowi SUID możemy stworzyć program/skrypt pozwalający innym użytkownikom na modyfikację oraz dostęp do pewnych wybranych plików należącym do nas, nie zmieniając jednoczesnie żadnych praw dostępu. Zauważmy, że wiele programów systemowych jest typu SUID: np. program passwd - zwykły użytkownik nie ma pozwolenia zapisu do plików przechowujących informację o kontach, jednak może uruchomić program passwd, który modyfikuje te pliki.

Cechę SUID nadajemy programowi poleceniem chmod(1).

Przykład:

chmod u+s ./program

Załóżmy że w katalogu aktualnym posiadamy skrypt "moje.skrypt", któremu chcemy nadać pozwolenie SUID:

chmod u+s ./skrypt

Inny użytkownik uruchomi go pisząc:

sh -f skrypt

Nie przyniesie to spodziewanego rezultatu - pozwolenie SUID dla skryptu nie ma tu żadnego znaczenia, gdyż wykonywanym programem jest sh a nie skrypt, zas "/bin/sh" nie ma oczywiscie pozwoleń SUID.

Nasz system musi zapewniać specjalny mechanizm "wykonywalnych skryptów". Skrypty te nazywane są wykonywalnymi ponieważ podobnie jak wykonywalne programy binarne (w formacie a.out(5)) rozpoczynają się tzw. liczną magiczną (magic number) - jest to para znaków "#!". Bezposrednio po liczbie magicznej w skrypcie wykonywalnym umieszczona jest scieżka interpretera danego skryptu i (ewentualnie) opcje.

Przykład:

Pierwsze linie skryptu wykonywalnego przeznaczonego dla sh(1):

```
#!/bin/sh -
#
# To jest skrypt wykonywalny suid.scr
.....
```

Jesli teraz użytkownik wywoła nasz skrypt pisząc po prostu:

suid.scr arg1 arg2

System wykona w rzeczywistosci polecenie równoważne poleceniu:

/bin/sh -f suid.scr - arg1 arg2

Z jedną rożnicą: jesli "suid.scr" miał pozwolenie SUID to powołany proces interpretera sh będzie działał z efektywnym UID właściciela pliku "suid.scr" !

Opisany mechanizm skryptów wykonywalnych obecny jest w nowszych wersjach Systemu V, oraz w 4.3BSD (np. SunOs).

Co dokładnie oznacza "wykonanie w tle"?

Zależy to od tego czy nasza wersja systemu zapewnia sterowanie pracami - "job control", czy też nie.

Koncepcja "pracy w tle" i "pracy na pierwszym planie" związana jest z powłoką, może być jednak oparta na pewnych mechanizmach systemowych.

Jesli system nie zapewnia sterowania pracami, to proces pracujący w tle ignoruje sygnały SIGINT (przerwanie, typowo cntrl-C) oraz SIGQUIT (typowo cntrl-\). Standardowe wejscie takiego procesu związane jest z plikiem "/dev/null".

Przykładem powłoki nie wykorzystującej sterowania pracami jest powłoka Bournea - sh, przykładem powłoki, która implementuje ten mechanizm jest csh.

W nowszych wersjach UNIX-a istnieją mechanizmy systemowe ułatwiające zarządzanie grupami procesów i umożliwiające tym samym zrealizowanie pełnego sterowania pracami. Wprowadzono pojęcie Grupy Procesów (process group). Grupa procesów może być związana z konkretnym terminalem, w takim przypadku pracuje "na pierwszym planie", nadając grupie procesów numer nie związany z terminalem umieszcza się ją w tle.

W wersji systemu implementującej grupy procesów mogą istnieć powłoki nie wykorzystujące tego mechanizmu. W takim wypadku proces zostaje umieszczony w tle w podobny sposób jak w systemie nie implementującym grup procesów.

W jaki sposób proces może okreslić czy wykonywany jest w tle?

%$isatty%

Jesli chcemy stwierdzić czy powłoka pracuje interakcyjnie czy też nie, np. gdy musimy podjąć decyzję czy można wyswietlić znak zachęty, możemy zastosować następujące testy:

sh:

if [-t 0]; then ... interakcyjna ... fi

csh:

if (isatty(0)) { ... interakcyjna ... }

W obydwu przypadkach sprawdzamy czy standardowe wejscie (deskryptor numer 0) jest terminalem, w pierwszym przypadku przy pomocy programu test, w drugim poleceniem "isatty".

Nie da się arbitralnie stwierdzić czy proces pracuje w tle, ponieważ pojęcie to oznacza co

innego w różnych wersjach systemu, a nawet, może mieć różne interpretacje w konkretnym systemie. Zobacz poprzedni punkt.

Czy program może okreslić swoją scieżkę wywołania ?

W ogólnym przypadku nie. Przyjęte jest, że zerowy argument przekazany procesowi stanowi pełną nazwę, lub przynajmniej nazwę jego programu. Jesli zerowy argument wywołania - argv [0] - zaczyna się znakiem "/" jest to zapewne pełna scieżka, jesli nie, proces może przeszukać katalogi zmiennej srodowiskowej PATH, szukając pliku wykonywalnego o nazwie takiej samej jak argv[0]. Zauważmy jednak że proces macierzysty może przekazać procesowi potomnemu dowolną wartosć argv[0], przekazanie pełnej scieżki, lub nazwy programu jest czystą konwencją.

Co oznaczają komunikaty "segmentation fault (core dumped)", "bus error (core dumped)" itp. ?

%$dbx% %$core%

Komunikaty te pochodzą od systemu operacyjnego i oznaczają wystąpienie w jednym z wykonywanych procesów błędu uniemożliwiającego dalszą pracę procesu. Przyczyną wystąpienia tego typu błędów jest najczęsciej próba odwołania do nielegalnego adresu, niewłasciwe odwołanie do pamięci - np. próba pobrania podwójnego słowa spod adresu nieparzystego itp.

W każdym takim przypadku w katalogu aktualnym zapisywany jest obraz pamięci procesu: plik o nazwie "core". Plik ten może mieć bardzo duże rozmiary - rzędu megabajtów, dlatego często korzystne jest wymuszenie jego maksymalnego rozmiaru, lub wręcz ograniczenie długosci plików core do 0 - patrz np. polecenie "limit", parametr "coredumpsize" powłoki csh.

Obraz pamięci procesu zapisany w pliku "core" analizować można przy pomocy programu uruchomieniowego (symbolicznego debugera) - dbx, lub w starszych wersjach systemu - sdb.

Przykład:

Załóżmy, że wykonanie programu p nazwie "a.out" spowodowało wystąpienie terminalnego błędu i zrzucenie obrazu pamięci do pliku "core". Chcąc poznać przyczynę załamania wywołujemy program dbx, oto przykladowa sesja:

```
$dbx a.out core
warning: no source compiled with -g
reading symbolic information ...
[using memory image in core]

segmentation violation in main at 0x10000284
0x10000284 (main+0xc) 80630000          l   r3,0x0(r3)
(dbx) quit
$
```

Programowi dbx podajemy tu nazwę programu, którego wykonanie zakończyło się błędem oraz nazwę pliku z obrazem pamięci. Dbx ostrzega o niedostępnosci pełnej informacji symbolicznej (i braku plików źródłowych). Po wczytaniu wyspecyfikowanych plików podawane jest miejsce wystąpienia terminalnego blędu (tu funkcja main programu), oraz informacje dodatkowe: adres instrukcji, której wykonanie zakończyło się niepowodzeniem, zawartosci niektórych rejestrów, etc. Program rozpoczyna teraz pracę interakcyjną: w oczekiwaniu na polecenia użytkownika

wyswietlany jest znak zachęty "(dbx) " .

Szczegóły funkcjonalne programu dbx siłą rzeczy zależne są od systemu (architektury procesora!). Powyższy przykład pochodzi ze stacji roboczej R6000 (IBM).

Program dbx jest narzędziem programistycznym i służy głównie do analizy, oraz wykrywania błędów w programach, których kod źródłowy jest dostępny. W przypadku wystąpienia błędu w aplikacji, której kodu źródłowego nie posiadamy przydatnosć programu uruchomieniowego dbx jest ograniczona.

7.DODATEK A: WERSJE SYSTEMU UNIX

Unix dostępny jest w wielu odmianach na wiele różnych typów komputerów posiadających często bardzo odmienne architektury. Wymieńmy tu stacje robocze firm IBM (procesor R6000), Silicon Graphics (procesory MIPS), SUN (procesory SPARC, wczesniej Motorola 680x0 i 80386); komputery typu IBM AT (procesory Intel 80x86); komputery VAX firmy DEC; systemy wieloprocesorowe np. Sequent Symetry; systemy transputerowe.

Na szczęscie podstawowych wersji systemu jest zdecydowanie mniej niż różnych jego odmian przeznaczonych na poszczególne maszyny - w chwili obecnej (1993 r.) rynek zdominowany jest przez trzy podstawowe wersje UNIX-a:

BSD
System V
OSF/1

Wersje te różnią się dosć istotnie zarówno konstrukcją jądra jak i systemu plików, dostępnymi programami usługowymi i administracyjnymi. W trakcie rozwijania systemu następował jednak transfer rozwiązań programowych między wersjami, tak że wiele mechanizmów pochodzących np. z BSD znaleźć można w Systemie V.

W poniższym omówieniu będziemy często wymieniać skrótowe nazwy mechanizmów systemowych i usług np. STREAMS, NFS, itp. Wszystkie te hermetyczne skróty są objasnione w słowniczku umieszczonym w następnym dodatku.

Powiedzmy teraz nieco więcej o wymienionych wyżej wersjach systemu:

BSD (Berkeley Software Distribution):
Typowy dla komputerów "VAX-owych", komputerów firmy SUN - system SunOs i niektórych stacji roboczych. Najintensywniej rozwijająca się do niedawna odmiana systemu, z niej pochodzi większość istotnych ulepszeń.
Kolejne odmiany i najważniejsze nowosci w nich wprowadzone:

2.x BSD	-	1978: znaczenie czysto historyczne
4.2 BSD	-	1983: TCP/IP, eternet, UFS (długie nazwy plików), dowiązania symboliczne.
4.3 BSD	-	1986: Z tej wersji wywodzi się SunOS.
4.3 Reno	-	1990: VAX, HP 9000/300: NFS (firmy SUN), MFS (Memory file system), Kerberos (protokół bezpieczeństwa), rozbudowane usługi sieciowe.
4.4 BSD	-	Wersja alfa 1992: nowy system pamięci wirtualnej, bardzo duże pliki (nie ograniczone rozmiarem do 2GB), system plików typu "log", zintegrowany Kerberos, zgodny z wieloma standardami POSIX.

System V (V - pięć rzymskie):
Twórca: AT&T. Najpopularniejsza wersja, najwięcej odmian. Posiada wiele zapożyczeń z BSD (szczególnie w SVR4).
Odmiany:

SVR2	-	(System V Release 2): SVID - System V Interface Definition.
SVR3	-	1986: STREAMS, TLI, RFS, liczne zapożyczenia z XENIX a w SVR3.2
SVR4	-	1988: obecnie najważniejsza - przyszłosciowa odmiana

systemu! Zapożyczono wielu mechanizmów z BSD, SunOS

i Xenixa. Najważniejsze cechy: protokoły NFS i TCP/IP; OpenLook, X11/News, powłoka ksh, internacjonalizacja, ANSI C, spełnia standardy POSIX, X/OPEN, SVID3

OSF/1:
OSF - Open Software Fundation jest zrzeszeniem wielu firm komputerowych, m.in.: Apollo, DEC, HP, IBM. Wersja systemu OSF/1 jest zgodna z SVR2 i SVID2 (SVID3 ?) i pokrewna AIX-owi firmy IBM. Standard OSF/1 zapewnia zgodnosć ze standardami POSIX oraz X/OPEN, symetryczną wieloprocesorowość, wielowątkowe jądro (bazujące na jądrze Mach 2.5), systemy plików UFS, S5, NFS, bezpieczeństwo na poziomie B1, STREAMS, TLI/XTI, interfejs graficzny Motif. W chwili obecnej Open Software Fundation przeżywa pewien schyłek, jednak ustalone przez nią standardy będą z całą pewnoscią wywierać wpływ na rozwój systemu UNIX przez kilka kolejnych lat.

Ze względów historycznych istotne są też dwie "arhaiczne" wersje: System 7 będący ostatnim "klasycznym" UNIX-em, bezposrednim potomkiem pierwszej wersji systemu i jednoczesnie protoplastą zarówno Systemu V jak i BSD, oraz System III (trzy rzymskie) - protoplasta Systemu V.

Komercyjne i darmowe odmiany UNIX-a:

Na wyżej wymienionych wersjach "podstawowych" będących pewnym kanonem bazują różne wersje komercyjne. Oto najważniejsze z nich:

AIX:
UNIX firmy IBM, bazuje na SVR2, z pewnymi mechanizmami BSD (sieć). Różni się dosć istotnie od wszystkich pozostałych odmian systemu.

A/UX:
Odmiana stworzona przez firmę Apple na komputery MacIntosh. Oparty na Systemie V i BSD (sieć, system plików).

386BSD, BSD/386:
Nowe, eksperymentalne systemy na procesory Intel 386, oparte oczywiscie o UNIX BSD.

HP-UX:
Mieszanka starego SVR2 i 4.2BSD przeznaczona na stacje robocze HP-9000 i RISC (800/700).

Linux:
"Public Domain" - czyli darmowy, przeznaczony na komputery zgodne z IBM AT (386), obecnie dostępna jest wersja beta.

SCO Unix:
Po prostu SVR3.2 na procesory 80x86.

Solaris:
Wersja 1.1 i niższe zob. SunOS. Wersje 2.x: Nowy system firmy SunSoft przeznaczony na komputery wyposażone w procesory SPARC lub 80386, i486. Solaris oferuje symetryczną wieloprocesorowość, OpenWindows 3.0, OpenLook, ONC, NIS(+), Kerberos. Kompilatory nie są dostarczane wraz z systemem!

SunOs:
Przeznaczony na stacje robocze firmy Sun wykorzystujące procesory 680x0, 80386 i SPARC.

Oparty na 4.3BSD. Wniósł bardzo wiele nowosci będących obecnie standardami: np. NFS. Ostatnia warsja: 4.1.3.

Ultrix:
4.2BSD firmy DEC.

Xenix:
Odmiana UNIX-a firmy Microsoft przeznaczona na maszyny z procesorem 80x86. W zasadzie oparty na SVR2, lecz starsze wersje na Systemie III oraz V 7. Historycznie pierwsza implementacja systemu UNIX na procesorze firmy Intel. Różni się dosć istotnie od pozostałych odmian, tak że w zasadzie mógł by być wymieniony jako czwarta główna wersja systemu.

System V - SVID2 a BSD 4.3 - SunOS:

Pozostańmy na chwilę przy różnicach dzielących dwie najpopularniejsze gałęzie rozwojowe systemu UNIX.

System V reprezentowany jest na naszym rynku przez produkty następujących firm: DELL, IBM (system AIX), Hewled Packard (system HP-UX), SCO, Sun Soft (system Solaris 2.x); BSD praktycznie wyłącznie przez SunOS 4.x firmy SUN i w bardzo małym stopniu przez odmiany przeznaczone na komputery zgodne z IBM AT (BSD386) oraz Ultrix.

Użytkownika systemu najbardziej interesują różnice w dostępnosci i składni typowych poleceń i usług. Postaramy się więc zestawić je skrótowo:

Cecha:	**typowo w SVRx**	**typowo w BSD**
nazwa jądra	/unix	/vmunix
typowe powłoki	sh, ksh	sh, csh
system plików	S5	UFS
max. długosć nazwy pliku	<= 14 znaków (*)	< 255 znaków
drukowanie	lp, lpstat, cancel	lpr, lpq, lprm
zarządzanie pracami	od wersji SVR4	tak
NFS	od SVR4 lub opcja	tak
Sieć (TCP/UDP)	od SVR3	tak

(*) Większosć obecnie dostępnych wersji Systemu V nie posiada tak drastycznego ograniczenia długosci pliku.

Polecenia w istotny sposób różniące się składnią opcji i argumentów: ar, banner, cat, cc, cflow, chmod, col, cxref, date, df, diff3, dircmp, du, echo, expr, grep, lint, ln, ls, m4, nohup, od, pax, pg, pr, ps, sed, sort, stty, su, sum, tabs, tic, time, tr.

System V różni się od BSD szczególnie w warstwie administracji i operacji zarezerwowanych dla super-użytkownika, jednak te zagadnienia nie będą w niniejszej książce poruszane.

8.DODATEK B: NAJWAŻNIEJSZE TERMINY I SKRÓTY:

CDE (Common Desktop Environment)
Próba ujednolicenia graficznego interfejsu użytkownika poprzez firmy zrzeszone w X/Open. Częsci składowe CDE to: model CUA (Common User Access) firmy IBM, VUE firmy HP i protokół ToolTalk firmy SunSoft. CDE nie zaistniał jeszcze jako standard.

Chorus:
Mikro-jądro systemu oparte na nowoczesnej koncepcji przekazywania komunikatów (message-passing microkernel). Być może kolejna wersja Systemu V będzie oparta o Chorus. Obecnie Chorus zapewnia binarną zgodnosć z SVR4

DCE (Distributed Computing Environment):
Zbiór standardów przetwarzania rozproszonego obejmujący: RPC, DNS, bezpieczeństwo, wątki, rozproszony system plików. Pochodzi z OSF (Open Software Fundation, zrzeszająca m.in. firmy IBM, HP, DEC). Nie związany z konkretnym systemem operacyjnym.

DNS (Domain Name Service)
Protokół sieciowy warstwy aplikacji, częsć standardu TCP/IP. Służy do gromadzenia i udostępnia informacji o powiązanich adresów symbolicznych z adresami internetowymi. To DNS zawdzięczamy możliwosć odwołania się do komputera poprzez nazwę Internetową np: "papcio.ii.pw.edu.pl" zamiast mało mówiącego adresu numerycznego: 148.81.64.6.

FFS (Fast File System) - pochodzi z uniwersytetu w Berkeley,
System plików oryginalnie zaimplementowany w UNIX-ie BSD a następnie w systemie SunOS, gdzie występuje pod nazwą UFS. W FFS zminimalizowano czasy dostępu do dysku poprzez strategie optymalnej alokacji miejsca dla plików. Wpowadzono koncepcje grupy cylindrów i fragmentów (cylinder groups, fragments). FFS przeniknął również do Systemu V.

HSFS (High Sierra File System)
System plików stosowany na CDROM-ach. Standard UNIX-a jak i MS-DOS-u

Mach:
Grupa nowoczesnych jąder systemu stworzonych na Uniwersytecie Carnegi Mellon. Baza dla wielu odmian UNIX-a : OSF/1, MACMach. Zawiera mechanizmy umożliwiające pracę wieloprocesorową.

Motif
"Nakładka" na system X11 definjująca sposób tworzenia i wygląd aplikacji graficznych. Jest produktem OSF konkurencyjnym dla OLIT. Zob. X11, OLIT.

NeWS (Network extensible Window System):
Twórca: SUN. PostScriptowy, sieciowy pakiet graficznego interfejsu: okna, narzędzia, architektura klijent/serwer. Obecnie praktycznie wyparty przez system X11; firma SUN wycofała się z rozbudowy NeWS.

NFS (Network File System):
Protokół sieciowego dostępu do plików. Obecnie będący standardem. Wprowadzony przez firmę SUN. W przeciwieństwie do starszych koncepcji rozproszonego systemu plików (np.

RFS) serwer NFS jest "bezstanowy", tzn. nie przechowuje informacji o stanie udostępnionych innym maszynom plików (deskryptorów) co zwiększa odporność na błędy i załamania systemu. NFS jest protokołem "przezroczystym" - tzn. nie ma praktycznie różnicy w sposobie dostępu do plików lokalnych a odległych.

NIS (Network Information Servis)

Usługa bazo-danowa typu klijent/serwer służąca do zarządzania zasobami sieci lokalnej. NIS umożliwia np. przechowywanie globalnej tablicy adresów internetowych odpowiadających nazwom komputerów, globalne zarządzanie kontami użytkowników. Bazy danych NIS mogą być przechowywane i propagowane z wielu komputerów, tak więc system odporny jest na awarie poszczególnych maszyn. NIS jest częścią systemu SunOS, jest też dostępny jako opcja w niektórych wersjach systemu V.

NIS+ (Network Information Service +):

Usługa podobna do NIS, jednak nie ograniczona do sieci lokalnych. Dostępna w systemie Solaris 2.x (SunSoft).

ONC (Open Network Computing):

Grupa serwisów sieciowych : NFS - zapewnia "przezroczysty" dostęp do plików na odległych komputerach. RPC - wywołania procedur na odległych komputerach. XDR uniwersalny format danych będący bazą dla RPC.

OLIT (OPEN LOOK Intrinsics Toolkit):

"Nakładka" na X11 oparta o Xtoolkit (Xt) z MIT definjująca wygląd i sposób współpracy z aplikacjami graficzymi. Twórcy: Sun i AT&T. Zob. też. X11, MOTIF.

RFS (Remote File System):

Protokół sieciowego dostępu do plików, pochodzi z Systemu V. Obecnie wyparty przez NFS.

RPC (Remote Procedure Call):

Protokół umożliwiający wywołanie procedury na odległym komputerze, przekazanie jej parametrów oraz uzyskanie rezultatów. Stanowi podstawę dla sieciowych usług wyższych warstw (np. NFS). Sam RPC może być oparty na protokołach TCP lub UDP. Zobacz też XDR.

S5 FS:

Rodzimy system plików Systemu V.

sockets:

Programowy interfejs do warstwy transportowej sieci. Zaimplementowane w BSD (SunOS). Zobacz też TLI.

STREAMS:

Uniwersalny protokół wymiany komunikatów, implementowany w jądrze systemu. Może służyć do komunikacji pomiędzy procesami na jednej maszynie, jak i komunikacji w sieci. Po raz pierwszy pojawił się w SVR4, obecny również w SunOS.

TLI (Transport Library Interface):

Analogiczne do "sockets" z BSD. Zaimplementowany w Systemie V, oparty na protokołach TCP oraz OSI.

UFS: (Unix File System)

Zobacz FFS.

XDR (External Data Representation):
Standardowy format danych umożliwiający ich wymianę między komputerami o rożnych architekturach. Stosowany przy protokole RPC.

XTI (X/Open Transport Interface):
Rozszerzony TLI (zob TLI).

X11:
System interfejsu graficznego z MIT. Obecnie standard na komputerach typu stacja robocza. Zapewnia: szeroko pojęte okna, pracę w sieci, architekturę klient/serwer. X11 jest systemem uniwersalnym, nie definjującym np. wyglądu graficznego aplikacji lub sposobu współpracy z użytkownikiem, dlatego też na bazie X11 skonstruowano wiele szeroko pojętych "narzędzi" okreslających sposób tworzenia i wygląd aplikacji: zob. OLIT, MOTIF. Cechą charakterystyczną X11 odróżniającą ten system od innych srodowisk okienkowych jest możliwosć interakcji z aplikacją (mysz, konsola) wykonywaną na odległym komputerze: stacji roboczej, lub specjalnie przeznaczonym do interakcji graficznej X-terminalu.

SVID (System V Interface Definition):
Standard obejmujący różne aspekty Systemu V, stworzony i rozwijany przez AT&T. Kolejne wydania mają następujacą zawartosć:

SVID1	1985 r.	Tom1: Wywołania systemowe i i biblioteki.
SVID2	1986 r.	Tom1: j.w. rozszerzony. Tom2: polecenia i narzędzia z zakresu podstawowego, rozszerzonego, administracji i tworzenia oprogramowania. Tom3: opis wspólpracy z terminalem, STREAMS, TLI, RFS.
SVID3	Standaryzacja w oparciu o wersję 4.2 Systemu V.	

VUE (Visual User Environment)
Interfejs graficzny użytkownika opracowany przez firmę Hewlett-Packard, oparty o X11 i Motif. Ostatnio wstępnie zaakceptowany jako przyszły standard w CDE przez firmy IBM, SCO, SunSoft, Univel i UNIX System Laboratories (konsorcjum X/Open).

9.DODATEK C: KSIĄŻKI POŚWIĘCONE UNIX-OWI:

Lista poniższa zawiera informacje o większosci książkek wydanych po polsku. Z literatury angielsko-języcznej starano się wybrać najwartosciowsze pozycje.

Wymieniono literaturę mającę jakis związek z tą publikacją, tak więc na liscie nie znalazły się książki poswięcone administracji, bezpieczeństwu systemu, programowaniu, aplikacjom UNIX-owym etc.

Książki w języku polskim:

System operacyjny UNIX
Peter P.Silvester
WNT 1990
ISBN 83-204-1086-x
Wprowadzenie dla początkujących, obejmuje wszystkie
podstawowe zagadnienia (pliki, powłoki, jądro, programy usługowe, edytory, języki
i kompilatory).

po prostu UNIX
Maciej Kaniewski, Krzysztof Wiermiejczyk
MIKOM 1992
ISBN 83-85625-06-2
Krótkie kompendium wiedzy o systemie przeznaczone dla osób posiadających
podstawową wiedzę z dziedziny informatyki.

Od PC do Workstation czyli jak zostać użytkownikiem systemu UNIX
Andrzej Dyrek
doctor Q press, 1992
ISBN 83-900371-3-0
Wprowadzenie do systemu oparte na systemie SunOs.

Łagodne wprowadzenie do systemu UNIX
praca zbiorowa
Wydawnicwto Politechniki Warszawskiej 1991
ISBN 83-900413-0-8

Książki w języku angielskim:

#Ogólne:

The Unix Operating System
Kaare Christian
Wiley 1998
ISBN: 0-471-84781-X
Klasyczna pozycja poswięcona poleceniom systemu.

Peter Norton's Guide to Unix
Peter Norton and Harley Hahn
Bantam Computer, 1991

ISBN: 0-553-35260-1
Książka przeznaczona dla osób początkujących, lecz znających dobrze MS-DOS.

A Student's Guide to Unix
Harley Hahn
McGraw Hill, 1993
ISBN: 0-07-025511-3
Bardzo dobra książka przeznaczona dla początkujących, obejmuje praktycznie wszystkie zagadnienia, w tym także sieciowe.

Mastering SunOS
Brent Heslop and David Angell
Sybex, 1990
ISBN: 0-89588-683
Dobry tekst poswięcony systemowi SunOS i srodowisku OpenWindows.

Unix for the Impatient
Paul Abrahams, Bruce Larson
Addison Wesley, 1992
ISBN: 0-201-55703-7
Wysoce polecana pozycja, podręcznik i jednoczesnie przewodnik po systemie. Przeznaczona dla osób srednio i dobrze zaawansowanych.

Unix Power Tools
Jerry Peek, Tim O'Reilly, Mike Loukides
O'Reilly / Bantam, 1993
ISBN: 0-553-35402-7
Bardzo dobra książka. Zawiera tysiące przykładów, wskazówek, skryptów, etc. Jest też dostępny CD-ROM z przykładami. Przeznaczona dla zaawansowanych.

The Design of the Unix Operating System
Maurice Bach
Prentice Hall, 1986
ISBN: 0-13-201799-7
Wyspecjalizowana książka poswięcona wewnętrznej konstrukcji Systemu V.

#Powłoki:

Unix Shell Programming
Lowell Arthur
Wiley, 1990
ISBN: 0-471-51821-2
Powłoki i narzędzia.

Unix Shell Programming
Stephen Kochan, Patrick Wood
Hayden, 1990
ISBN: 0-672-48448-X
Klasyczna pozycja poswięcona powłokom Bournea i Korna.

#Edytory:

Learning GNU Emacs
Debra Cameron, Bill Rosenblatt
O'Reilly, 1992
ISBN: 0-937175-84-6
Prawdopodobnie najlepsza książka o popularnym (i darmowym!) edytorze Emacs.

The Ultimate Guide to the vi and ex Text Editors
Hewlett-Packard
Addison-Wesley, 1989
ISBN: 0-8053-4460-8
Wszystko o edytorze vi.

#Usługi sieciowe, Internet

Zen and the Art of the Internet
Brendan Kehoe
Prentice Hall
ISBN: 0-13-010778-6
Klasyczna pozycja, do uzyskania poprzez anonimowe ftp (np. world.std.com plik /obi/Internet/zen-1.0).

The Waite Group's Unix Communications
Bart Anderson, Brian Costales and Harry Henderson
Sams, 1991
ISBN: 0-672-22773-8
Poczta elektroniczna, UUCP, i wiele innych pożytecznych informacji.

10.DODATEK D INDEKS PROGRAMÓW, POLECEŃ POWŁOK I PLIKÓW INICJALIZACYJNYCH

SŁOWNIK

A:

abnormal termination	-	awaryjne zakończenie się programu
alias (in a shell)	-	alias
appending (to a file)	-	dołączanie do pliku
arguments (to a prog.)	-	argumenty (programu)
argument list	-	lista argumentów (programu)

B:

background	-	tło
background execution	-	wykonanie w tle
backup	-	archiwizacja, składowanie
batch processing	-	przetwarznie wsadowe
blank	-	odstęp (biały znak: spacja, tab, nowa linia)
block device	-	urządzenie blokowe
brace	-	nawias klamrowy "{","}"
bracket	-	nawias kwadratowy "[","]"

C:

carridge return	-	znak końca linii (powrotu karetki)
character	-	znak
catching signals	-	przechwytywanie sygnałów
child process	-	proces potomny; potomek
close	-	zamknąć (plik)
command	-	polecenie
command line	-	wiersz polecenia
comment	-	komentarz
control	-	sterować, sterujące
conversion	-	konwersja
core file	-	plik z obrazem pamięci procesu
current directory	-	katalog aktualny
cursor	-	kursor; znacznik

D:

daemon	-	demon
death (of a process)	-	smierć, zakończenie procesu
default	-	domyslny
deleting (files)	-	usuwanie plików
device	-	urządzenie
directory	-	katalog
driver	-	program sterujący

E:

e-mail	-	poczta elektroniczna
echoing (of chars)	-	wyswietlanie/drukowanie (znaków)
effective (UID, GID)	-	efektywny identyfikator użytkownika/grupy
end of file char. (EOF)	-	znacznik końca pliku
environment	-	srodowisko
erase	-	kasować
erase (char)	-	znak kasowania
escape sequences	-	sekwencje znaków (sterujące)
execution	-	wykonanie
exit status	-	kod powrotu
expression	-	wyrażenie

F:

facility	-	usługa
file	-	plik

file access - dostęp do pliku
file creation - tworzenie/utworzenie pliku
file creation mask - maska pozwoleń pliku
file deletion - usunięcie pliku
file descriptor - deskryptor pliku
file hierarchy - hierarchia plików
file modes - pozwolenia dostępu (do pliku)
filename completion - uzupełnianie nazw plików
filename globbing - substytucja nazw plików
filesystem - system plików
file transfer - przenoszenie plików
foreground - proces pierwszoplanowy
fork - rozwidlenie (procesu)

G:
group id. - identyfikator grupy (użytkowników)

H:
hangup - rozłączenie
hidden - ukryty
home directory - katalog główny użytkownika
host - komputer główny, ale też po prostu jeden z komputerów w sieci

I:
input - wejscie
interactive - interaktywnie
I/O redirection - przełączenie strumieni wejscia/wyjscia
inode - metryczka pliku
invoke - wywołać

K:
kernel - jądro
keypad - klawiatura dodatkowa
kill (a process) - zabić proces

L:
layer - warstwa (zwykle w odniesieniu do hierarchizacji usług sieciowych, np.
transport layer - warstwa transportowa sieci)
library - biblioteka
library functions - funkcje biblioteczne
library routines - j.w.
line discipline - tryb pracy (terminala)
line editor - edytor liniowy
link - dowiązanie
(to) link - konsolidować moduły
linker - konsolidator - program ld(1)
link count - liczba dowiązań
local area network - sieć lokalna
(file) locking - blokowanie (dostępu do pliku)
log in - rozpoczęcie sesji
login shell - powłoka główna
log out - zakończenie sesji
loop - pętla

M:
magic number - liczba magiczna
mandatory locking - blokowanie obowiązkowe
manual - podręcznik

message - komunikat
message passing - przekazywanie komunikatów
minor device number - drugorz_^_^dny identyfikator urządzenia
mode - tryb
mount - montowanie
move (a file) - przeniesienie (pliku)
multiprocessing - wielo-przetwarzający
multiprocessor - wielo-procesorowy
multitasking - wielo-zadaniowy
multiuser - wielo-dostępny
mutual exclusion - wzajemne wykluczanie

N:
named pipe - plik specjalny FIFO
normal termination - poprawne zakończenie się programu
node - węzeł (zwykle komputer w sieci, rzadziej procesor w systemie
wieloprocesorowym)

O:
output - wyjscie
owner - właściciel

P:
padding - uzupełnianie
parameter - parametr
paranthesis - nawias zwykly: "(", ")"
parent directory - katalog macierzysty
parent process - proces macierzysty
password - hasło
path - scieżka
pathname - scieżka
pattern - wzorzec
period - kropka "."
permisions - zezwolenia
pipeline, pipe - potok
positional parameter - parametr pozycyjny
proces group - grupa procesów
prompt - tekst zachęty, monit
process group leader - lider grupy procesów

Q:
quoting - cytowanie

R:
random access - dostęp swobodny
raw - surowy (np. tryb pracy terminala)
real UID/GID - rzeczywisty UID/GID
(file) redirection - skojarzenie pliku (deskryptora)
relative - względny
remote - odległy (komputer)
root directory - katalog główny, korzeń

S:
screen editing - edycja pełno-ekranowa
screen handling - obsługa ekranu
script - skrypt

search permisions - pozwolenia przeszukania
semicolon - srednik ";"
sending signals - przekazywanie sygnałów
server - serwer; komputer (proces) usługodawca
set-group-id - zachowaj-identyfikator-grupy
set-user-id - zachowaj-identyfikator-użytkownika
sharable programs - programy (współ)dzielone
shell - powłoka; interpreter poleceń
shell script - skrypt powłoki
signal handling - obsługa sygnałów
standard error - standardow diagnostyka
standard input - standardowe wejscie
standard output - standardowe wyjscie
string - łańcuch
sub-directory - podkatalog
super block - superblok
super user - super-użytkownik (użytkownik uprzywilejowany)
suspend - zawiesić (proces)
swapping - wymiatanie (procesów)
swap, swap area - obszar wymiany
symbolic link - dowiązania symboliczne
system function - funkcja systemowa
system call - wywołanie systemowe

T:
tape units: - jednostki tasm
terminal modes - tryby pracy terminala
terminate - zakończyć
text - zwyczajowa nazwa kodu (!) programu
tool - narzędzie
toolset - zestaw (biblioteka) narzędzi

U:

underscore - znak podkreslenia "_"
unmount - odmontowanie
user - użytkownik
user-id (UID) - identyfikator użytkownika

V:
valid - legalny
value - wartosć
variable - zmienna
void - pusty, nieważny
volume - wolumin

W:
wait - oczekiwanie
walking a directory tree - przejscie po drzewie katalogów
wide are network - sieć rozległa
wildcard - znak-wzorzec (np. "*" w powłoce)
word - słowo
working directory - katalog roboczy
workstation - stacja robocza

Z:
zombie - (proces) zambi